MLOps: Continuous delivery and automation pipelines in machine learning

This document discusses techniques for implementing and automating continuous integration (CI), continuous delivery (CD), and continuous training (CT) for machine learning (ML) systems.

Data science and ML are becoming core capabilities for solving complex real-world problems, transforming industries, and delivering value in all domains. Currently, the ingredients for applying effective ML are available to you:

- Large datasets
- Inexpensive on-demand compute resources
- Specialized accelerators for ML on various cloud platforms
- Rapid advances in different ML research fields (such as computer vision, natural language understanding, and recommendations AI systems).

Therefore, many businesses are investing in their data science teams and ML capabilities to develop predictive models that can deliver business value to their users.

This document is for data scientists and ML engineers who want to apply DevOps (/devops) principles to ML systems (MLOps). *MLOps* is an ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops). Practicing MLOps means that you advocate for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment and infrastructure management.

Data scientists can implement and train an ML model with predictive performance on an offline holdout dataset, given relevant training data for their use case. However, the real challenge isn't building an ML model, the challenge is building an integrated ML system and to continuously operate it in production. With the long history of production ML services at Google, we've learned that there can be many pitfalls in operating ML-based systems in production. Some of these pitfalls are summarized in Machine Learning: The high-interest credit card of technical debt (https://ai.google/research/pubs/pub43146).

As shown in the following diagram, only a small fraction of a real-world ML system is composed of the ML code. The required surrounding elements are vast and complex.

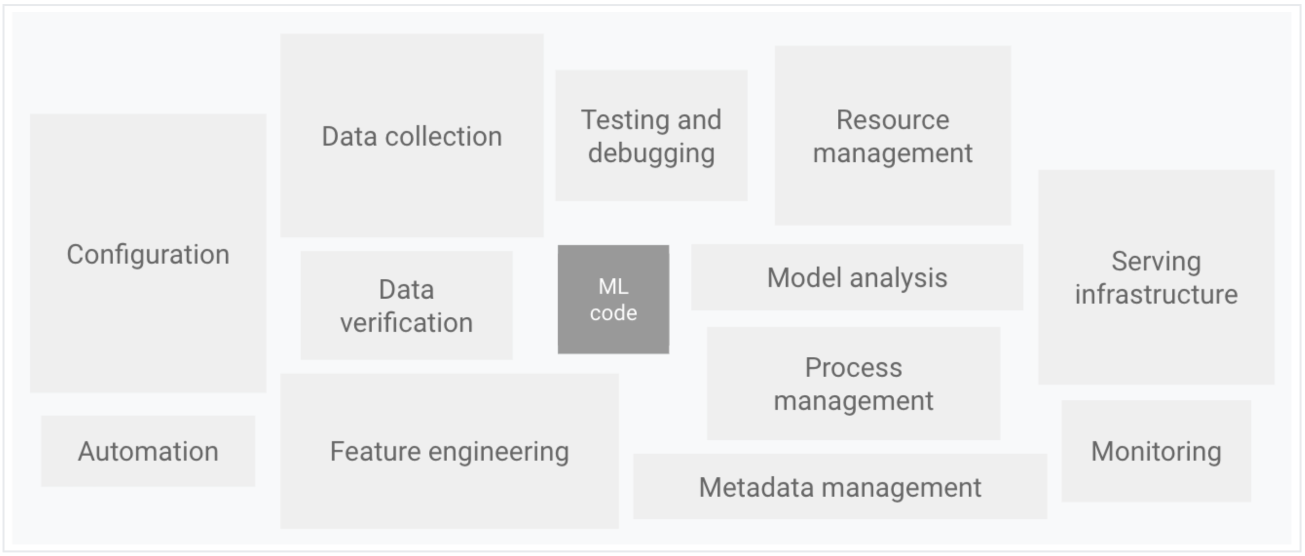

Figure 1. Elements for ML systems. Adapted from Hidden Technical Debt in Machine Learning Systems (https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf).

In this diagram, the rest of the system is composed of configuration, automation, data collection, data verification, testing and debugging, resource management, model analysis, process and metadata management, serving infrastructure, and monitoring.

To develop and operate complex systems like these, you can apply DevOps principles to ML systems (MLOps). This document covers concepts to consider when setting up an MLOps environment for your data science practices, such as CI, CD, and CT in ML.

The following topics are discussed:

- DevOps versus MLOps
- Steps for developing ML models
- MLOps maturity levels

DevOps versus MLOps

DevOps (/devops) is a popular practice in developing and operating large-scale software systems. This practice provides benefits such as shortening the development cycles, increasing deployment velocity, and dependable releases. To achieve these benefits, you introduce two concepts in the software system development:

- Continuous Integration (CI) (https://wikipedia.org/wiki/Continuous_integration)
- Continuous Delivery (CD) (https://wikipedia.org/wiki/Continuous_delivery)

An ML system is a software system, so similar practices apply to help guarantee that you can reliably build and operate ML systems at scale.

However, ML systems differ from other software systems in the following ways:

- Team skills: In an ML project, the team usually includes data scientists or ML researchers, who focus on exploratory data analysis, model development, and experimentation. These members might not be experienced software engineers who can build production-class services.
- Development: ML is experimental in nature. You should try different features, algorithms, modeling techniques, and parameter configurations to find what works best for the problem as quickly as possible. The challenge is tracking what worked and what didn't, and maintaining reproducibility while maximizing code reusability.
- Testing: Testing an ML system is more involved than testing other software systems. In addition to typical unit and integration tests, you need data validation, trained model quality evaluation, and model validation.
- Deployment: In ML systems, deployment isn't as simple as deploying an offline-trained ML model as a prediction service. ML systems can require you to deploy a multi-step pipeline to automatically retrain and deploy model. This pipeline adds complexity and requires you to automate steps that are manually done before deployment by data scientists to train and validate new models.
- Production: ML models can have reduced performance not only due to suboptimal coding, but also due to constantly evolving data profiles. In other words, models can decay in more ways than conventional software systems, and you need to consider this degradation. Therefore, you need to track summary statistics of your data and monitor the online performance of your model to send notifications or roll back when values deviate from your expectations.

ML and other software systems are similar in continuous integration of source control, unit testing, integration testing, and continuous delivery of the software module or the package. However, in ML, there are a few notable differences:

- CI is no longer only about testing and validating code and components, but also testing and validating data, data schemas, and models.
- CD is no longer about a single software package or a service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service).
- CT is a new property, unique to ML systems, that's concerned with automatically retraining and serving the models.

The following section discusses the typical steps for training and evaluating an ML model to serve as a prediction service.

Data science steps for ML

In any ML project, after you define the business use case and establish the success criteria, the process of delivering an ML model to production involves the following steps. These steps can be completed manually or can be completed by an automatic pipeline.

1. Data extraction: You select and integrate the relevant data from various data sources for the ML task.
2. Data analysis: You perform exploratory data analysis (https://wikipedia.org/wiki/Exploratory_data_analysis) (EDA) to understand the available data for building the ML model. This process leads to the following:
   - Understanding the data schema and characteristics that are expected by the model.
   - Identifying the data preparation and feature engineering that are needed for the model.
3. Data preparation: The data is prepared for the ML task. This preparation involves data cleaning, where you split the data into training, validation, and test sets. You also apply data transformations and feature engineering to the model that solves the target task. The output of this step are the *data splits* in the prepared format.
4. Model training: The data scientist implements different algorithms with the prepared data to train various ML models. In addition, you subject the implemented algorithms to hyperparameter tuning to get the best performing ML model. The output of this step is a trained model.
5. Model evaluation: The model is evaluated on a holdout test set (https://wikipedia.org/wiki/Training,_validation,_and_test_sets#Holdout_dataset) to evaluate the model quality. The output of this step is a set of metrics to assess the quality of the model.
6. Model validation: The model is confirmed to be adequate for deployment—that its predictive performance is better than a certain baseline.
7. Model serving: The validated model is deployed to a target environment to serve predictions. This deployment can be one of the following:
   - Microservices with a REST API to serve online predictions.

- An embedded model to an edge or mobile device.
- Part of a batch prediction system.

8. Model monitoring: The model predictive performance is monitored to potentially invoke a new iteration in the ML process.

The level of automation of these steps defines the *maturity* of the ML process, which reflects the velocity of training new models given new data or training new models given new implementations. The following sections describe three levels of MLOps, starting from the most common level, which involves no automation, up to automating both ML and CI/CD pipelines.

MLOps level 0: Manual process

Many teams have data scientists and ML researchers who can build state-of-the-art models, but their process for building and deploying ML models is entirely manual. This is considered the *basic* level of maturity, or level 0. The following diagram shows the workflow of this process.

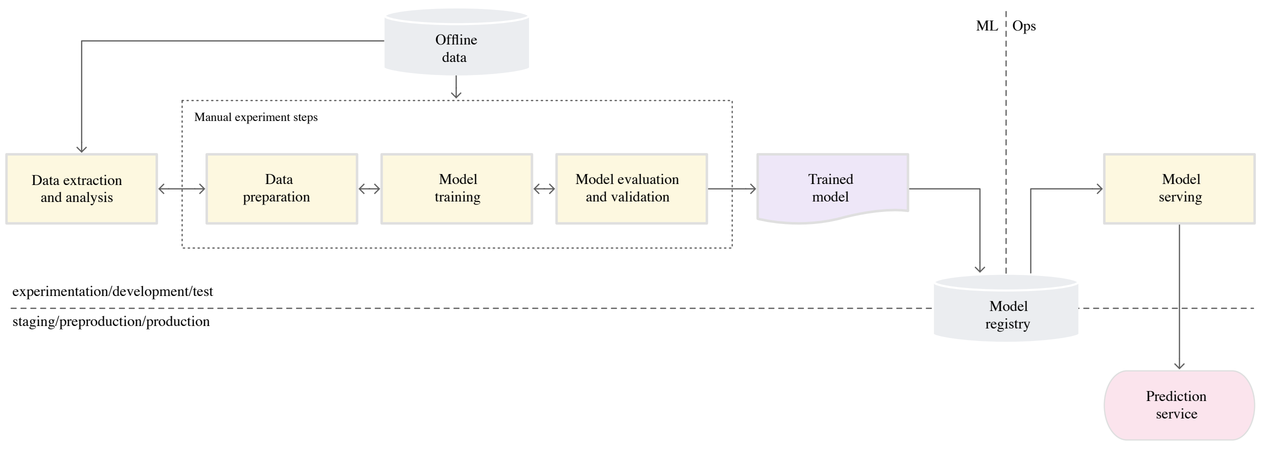

Figure 2. Manual ML steps to serve the model as a prediction service.

Characteristics

The following list highlights the characteristics of the MLOps level 0 process, as shown in Figure 2:

- Manual, script-driven, and interactive process: Every step is manual, including data analysis, data preparation, model training, and validation. It requires manual execution of each step, and manual transition from one step to another. This process is usually driven by experimental code that is written and executed in notebooks by data scientist interactively, until a workable model is produced.

- Disconnection between ML and operations: The process separates data scientists who create the model and engineers who serve the model as a prediction service. The data scientists hand over a trained model as an artifact to the engineering team to deploy on their API infrastructure. This handoff can include putting the trained model in a storage location, checking the model object into a code repository, or uploading it to a models registry. Then engineers who deploy the model need to make the required features available in production for low-latency serving, which can lead to *training-serving skew* (https://developers.google.com/machine-learning/guides/rules-of-ml/#training-serving_skew).
- Infrequent release iterations: The process assumes that your data science team manages a few models that don't change frequently—either changing model implementation or retraining the model with new data. A new model version is deployed only a couple of times per year.
- No CI: Because few implementation changes are assumed, CI is ignored. Usually, testing the code is part of the notebooks or script execution. The scripts and notebooks that implement the experiment steps are source controlled, and they produce artifacts such as trained models, evaluation metrics, and visualizations.
- No CD: Because there aren't frequent model version deployments, CD isn't considered.
- Deployment refers to the prediction service: The process is concerned only with deploying the trained model as a prediction service (for example, a microservice with a REST API), rather than deploying the entire ML system.
- Lack of active performance monitoring: The process doesn't track or log the model predictions and actions, which are required in order to detect model performance degradation and other model behavioral drifts.

The engineering team might have their own complex setup for API configuration, testing, and deployment, including security, regression, and load and canary testing. In addition, production deployment of a new version of an ML model usually goes through A/B testing or online experiments before the model is promoted to serve all the prediction request traffic.

Challenges

MLOps level 0 is common in many businesses that are beginning to apply ML to their use cases. This manual, data-scientist-driven process might be sufficient when models are

rarely changed or trained. In practice, models often break when they are deployed in the real world. The models fail to adapt to changes in the dynamics of the environment, or changes in the data that describes the environment. For more information, see Why Machine Learning Models Crash and Burn in Production
(https://www.forbes.com/sites/forbestechcouncil/2019/04/03/why-machine-learning-models-crash-and-burn-in-production/)
.

To address these challenges and to maintain your model's accuracy in production, you need to do the following:

- Actively monitor the quality of your model in production: Monitoring lets you detect performance degradation and model staleness. It acts as a cue to a new experimentation iteration and (manual) retraining of the model on new data.
- Frequently retrain your production models: To capture the evolving and emerging patterns, you need to retrain your model with the most recent data. For example, if your app recommends fashion products using ML, its recommendations should adapt to the latest trends and products.
- Continuously experiment with new implementations to produce the model: To harness the latest ideas and advances in technology, you need to try out new implementations such as feature engineering, model architecture, and hyperparameters. For example, if you use computer vision in face detection, face patterns are fixed, but better new techniques can improve the detection accuracy.

To address the challenges of this manual process, MLOps practices for CI/CD and CT are helpful. By deploying an ML training pipeline, you can enable CT, and you can set up a CI/CD system to rapidly test, build, and deploy new implementations of the ML pipeline. These features are discussed in more detail in the next sections.

MLOps level 1: ML pipeline automation

The goal of level 1 is to perform continuous training of the model by automating the ML pipeline; this lets you achieve continuous delivery of model prediction service. To automate the process of using new data to retrain models in production, you need to introduce automated data and model validation steps to the pipeline, as well as pipeline triggers and metadata management.

The following figure is a schematic representation of an automated ML pipeline for CT.

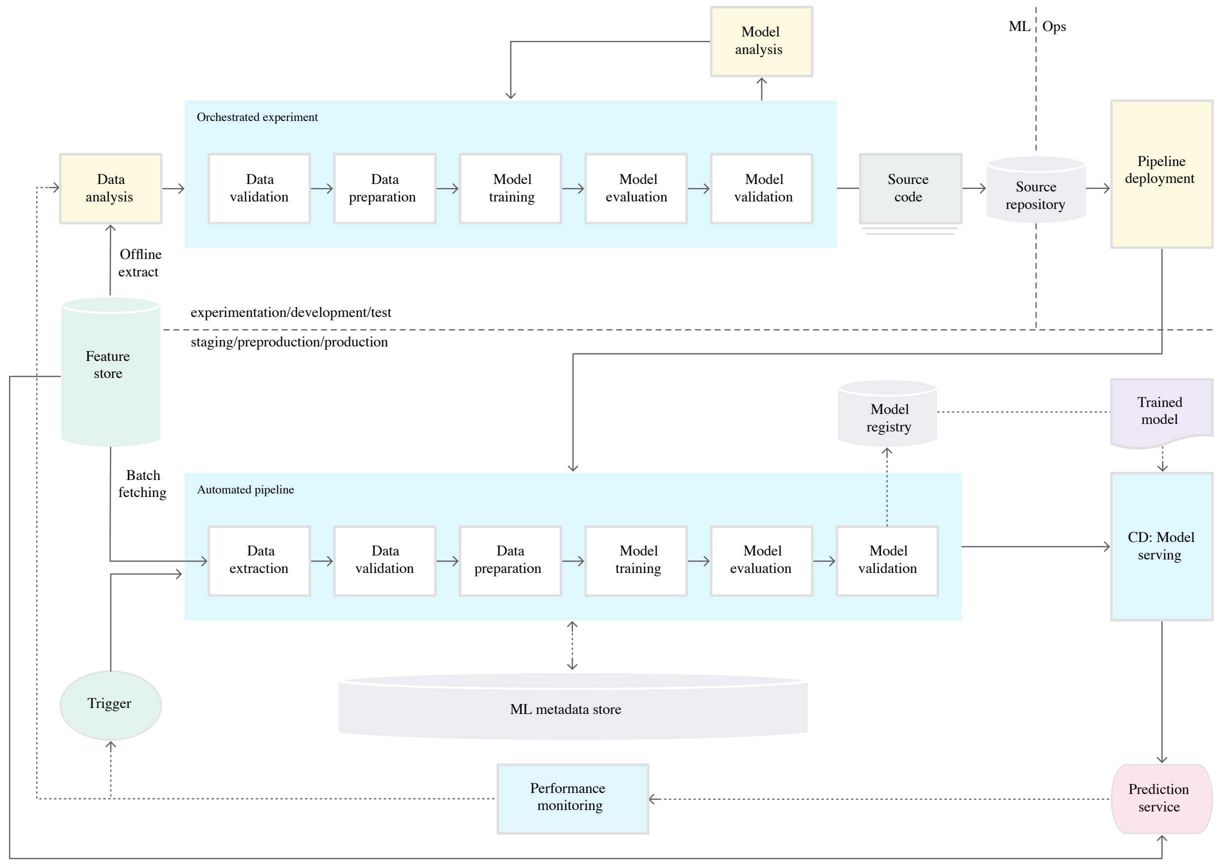

Figure 3. ML pipeline automation for CT.

Characteristics

The following list highlights the characteristics of the MLOps level 1 setup, as shown in Figure 3:

- Rapid experiment: The steps of the ML experiment are orchestrated. The transition between steps is automated, which leads to rapid iteration of experiments and better readiness to move the whole pipeline to production.
- CT of the model in production: The model is automatically trained in production using fresh data based on live pipeline triggers, which are discussed in the next section.
- Experimental-operational symmetry: The pipeline implementation that is used in the development or experiment environment is used in the preproduction and production environment, which is a key aspect of MLOps practice for unifying DevOps.
- Modularized code for components and pipelines: To construct ML pipelines, components need to be reusable, composable, and potentially shareable across ML pipelines. Therefore, while the EDA code can still live in notebooks, the source code

for components must be modularized. In addition, components should ideally be containerized to do the following:

- Decouple the execution environment from the custom code runtime.
- Make code reproducible between development and production environments.
- Isolate each component in the pipeline. Components can have their own version of the runtime environment, and have different languages and libraries.

- Continuous delivery of models: An ML pipeline in production continuously delivers prediction services to new models that are trained on new data. The model deployment step, which serves the trained and validated model as a prediction service for online predictions, is automated.
- Pipeline deployment: In level 0, you deploy a trained model as a prediction service to production. For level 1, you deploy a whole training pipeline, which automatically and recurrently runs to serve the trained model as the prediction service.

Additional components

This section discusses the components that you need to add to the architecture to enable ML continuous training.

Data and model validation

When you deploy your ML pipeline to production, one or more of the triggers discussed in the ML pipeline triggers (#ml_pipeline_triggers) section automatically executes the pipeline. The pipeline expects new, live data to produce a new model version that is trained on the new data (as shown in Figure 3). Therefore, automated *data validation* and *model validation* steps are required in the production pipeline to ensure the following expected behavior:

- Data validation: This step is required before model training to decide whether you should retrain the model or stop the execution of the pipeline. This decision is automatically made if the following was identified by the pipeline.
    - Data schema skews: These skews are considered anomalies in the input data, which means that the downstream pipeline steps, including data processing and model training, receives data that doesn't comply with the expected schema. In this case, you should stop the pipeline so the data science team can investigate. The team might release a fix or an update to the pipeline to handle these changes in the schema. Schema skews include receiving unexpected features,

  not receiving all the expected features, or receiving features with unexpected values.
  - Data values skews: These skews are significant changes in the statistical properties of data, which means that data patterns are changing, and you need to trigger a retraining of the model to capture these changes.
- Model validation: This step occurs after you successfully train the model given the new data. You evaluate and validate the model before it's promoted to production. This *offline model validation* step consists of the following.
  - Producing evaluation metric values using the trained model on a test dataset to assess the model's predictive quality.
  - Comparing the evaluation metric values produced by your newly trained model to the current model, for example, production model, baseline model, or other business-requirement models. You make sure that the new model produces better performance than the current model before promoting it to production.
  - Making sure that the performance of the model is consistent on various segments of the data. For example, your newly trained customer churn model might produce an overall better predictive accuracy compared to the previous model, but the accuracy values per customer region might have large variance.
  - Making sure that you test your model for deployment, including infrastructure compatibility and consistency with the prediction service API.

In addition to offline model validation, a newly deployed model undergoes *online model validation*—in a canary deployment or an A/B testing setup—before it serves prediction for the online traffic.

Feature store

An optional additional component for level 1 ML pipeline automation is a feature store. A feature store is a centralized repository where you standardize the definition, storage, and access of features for training and serving. A feature store needs to provide an API for both high-throughput batch serving and low-latency real-time serving for the feature values, and to support both training and serving workloads.

The feature store helps data scientists do the following:

- Discover and reuse available feature sets for their entities, instead of re-creating the same or similar ones.

- Avoid having similar features that have different definitions by maintaining features and their related metadata.
- Serve up-to-date feature values from the feature store.
- Avoid training-serving skew by using the feature store as the data source for experimentation, continuous training, and online serving. This approach makes sure that the features used for training are the same ones used during serving:
    - For experimentation, data scientists can get an offline extract from the feature store to run their experiments.
    - For continuous training, the automated ML training pipeline can fetch a batch of the up-to-date feature values of the dataset that are used for the training task.
    - For online prediction, the prediction service can fetch in a batch of the feature values related to the requested entity, such as customer demographic features, product features, and current session aggregation features.

Metadata management

Information about each execution of the ML pipeline is recorded in order to help with data and artifacts lineage, reproducibility, and comparisons. It also helps you debug errors and anomalies. Each time you execute the pipeline, the ML metadata store records the following metadata:

- The pipeline and component versions that were executed.
- The start and end date, time, and how long the pipeline took to complete each of the steps.
- The executor of the pipeline.
- The parameter arguments that were passed to the pipeline.
- The pointers to the artifacts produced by each step of the pipeline, such as the location of prepared data, validation anomalies, computed statistics, and extracted vocabulary from the categorical features. Tracking these intermediate outputs helps you resume the pipeline from the most recent step if the pipeline stopped due to a failed step, without having to re-execute the steps that have already completed.
- A pointer to the previous trained model if you need to roll back to a previous model version or if you need to produce evaluation metrics for a previous model version when the pipeline is given new test data during the model validation step.

- The model evaluation metrics produced during the model evaluation step for both the training and the testing sets. These metrics help you compare the performance of a newly trained model to the recorded performance of the previous model during the model validation step.

ML pipeline triggers

You can automate the ML production pipelines to retrain the models with new data, depending on your use case:

- On demand: Ad-hoc manual execution of the pipeline.
- On a schedule: New, labelled data is systematically available for the ML system on a daily, weekly, or monthly basis. The retraining frequency also depends on how frequently the data patterns change, and how expensive it is to retrain your models.
- On availability of new training data: New data isn't systematically available for the ML system and instead is available on an ad-hoc basis when new data is collected and made available in the source databases.
- On model performance degradation: The model is retrained when there is noticeable performance degradation.
- On significant changes in the data distributions (*concept drift* (https://wikipedia.org/wiki/Concept_drift)). It's hard to assess the complete performance of the online model, but you notice significant changes on the data distributions of the features that are used to perform the prediction. These changes suggest that your model has gone stale, and that needs to be retrained on fresh data.

Challenges

Assuming that new implementations of the pipeline aren't frequently deployed and you are managing only a few pipelines, you usually manually test the pipeline and its components. In addition, you manually deploy new pipeline implementations. You also submit the tested source code for the pipeline to the IT team to deploy to the target environment. This setup is suitable when you deploy new models based on new data, rather than based on new ML ideas.

However, you need to try new ML ideas and rapidly deploy new implementations of the ML components. If you manage many ML pipelines in production, you need a CI/CD setup to automate the build, test, and deployment of ML pipelines.

MLOps level 2: CI/CD pipeline automation

For a rapid and reliable update of the pipelines in production, you need a robust automated CI/CD system. This automated CI/CD system lets your data scientists rapidly explore new ideas around feature engineering, model architecture, and hyperparameters. They can implement these ideas and automatically build, test, and deploy the new pipeline components to the target environment.

The following diagram shows the implementation of the ML pipeline using CI/CD, which has the characteristics of the automated ML pipelines setup plus the automated CI/CD routines.

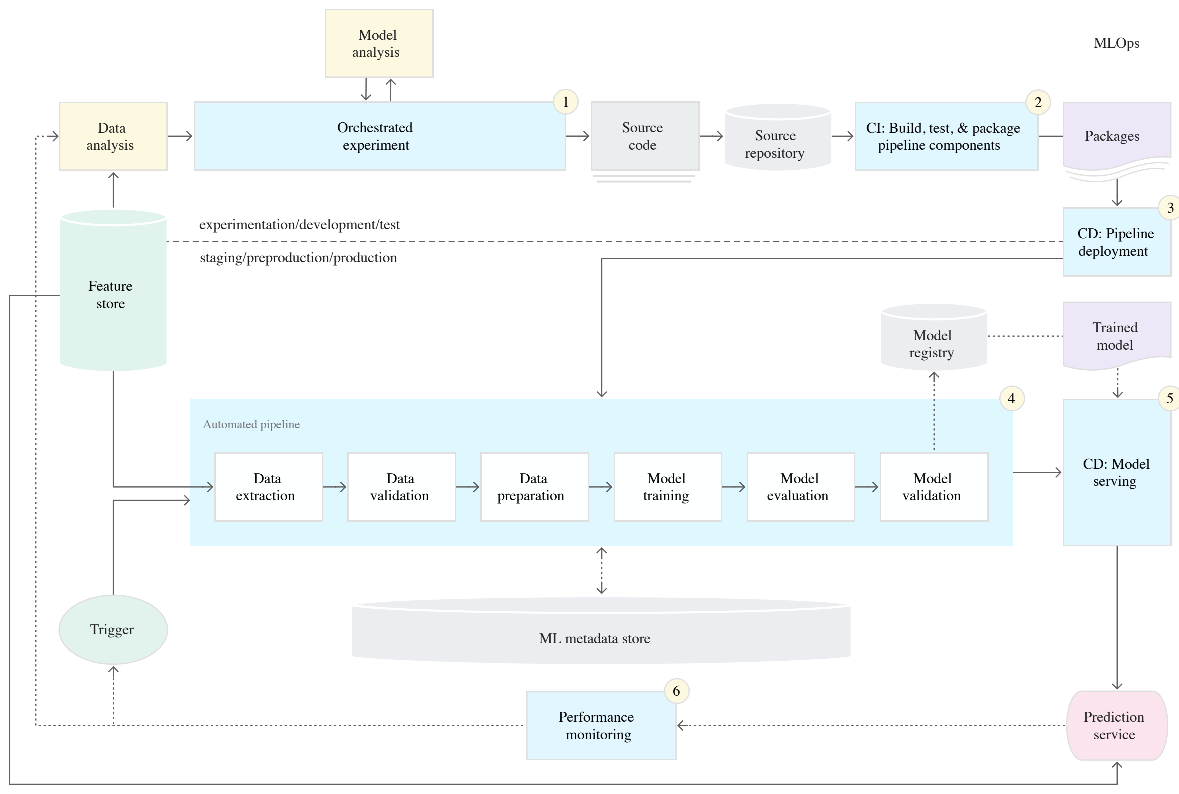

Figure 4. CI/CD and automated ML pipeline.

This MLOps setup includes the following components:

- Source control
- Test and build services
- Deployment services
- Model registry
- Feature store

- ML metadata store
- ML pipeline orchestrator

Characteristics

The following diagram shows the stages of the ML CI/CD automation pipeline:

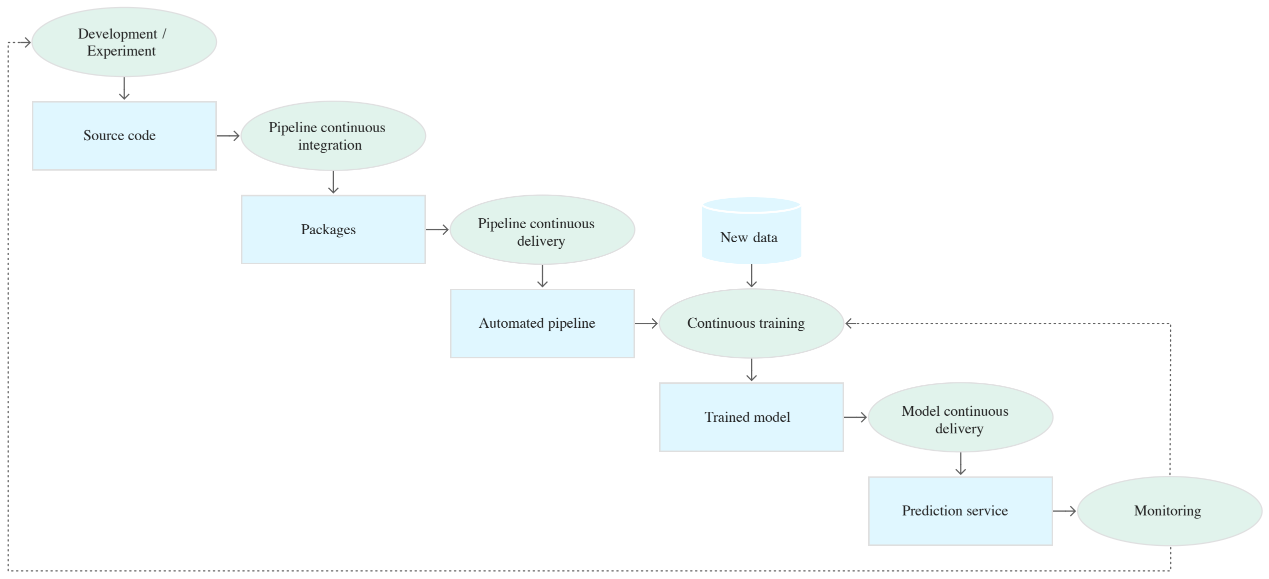

Figure 5. Stages of the CI/CD automated ML pipeline.

The pipeline consists of the following stages:

1. Development and experimentation: You iteratively try out new ML algorithms and new modeling where the experiment steps are orchestrated. The output of this stage is the source code of the ML pipeline steps that are then pushed to a source repository.
2. Pipeline continuous integration: You build source code and run various tests. The outputs of this stage are pipeline components (packages, executables, and artifacts) to be deployed in a later stage.
3. Pipeline continuous delivery: You deploy the artifacts produced by the CI stage to the target environment. The output of this stage is a deployed pipeline with the new implementation of the model.
4. Automated triggering: The pipeline is automatically executed in production based on a schedule or in response to a trigger. The output of this stage is a trained model that is pushed to the model registry.
5. Model continuous delivery: You serve the trained model as a prediction service for the predictions. The output of this stage is a deployed model prediction service.

6. Monitoring: You collect statistics on the model performance based on live data. The output of this stage is a trigger to execute the pipeline or to execute a new experiment cycle.

The data analysis step is still a manual process for data scientists before the pipeline starts a new iteration of the experiment. The model analysis step is also a manual process.

Continuous integration

In this setup, the pipeline and its components are built, tested, and packaged when new code is committed or pushed to the source code repository. Besides building packages, container images, and executables, the CI process can include the following tests:

- Unit testing your feature engineering logic.
- Unit testing the different methods implemented in your model. For example, you have a function that accepts a categorical data column and you encode the function as a one-hot (https://wikipedia.org/wiki/One-hot) feature.
- Testing that your model training converges (that is, the loss of your model goes down by iterations and overfits (https://wikipedia.org/wiki/Overfitting) a few sample records).
- Testing that your model training doesn't produce NaN (https://wikipedia.org/wiki/NaN) values due to dividing by zero or manipulating small or large values.
- Testing that each component in the pipeline produces the expected artifacts.
- Testing integration between pipeline components.

Continuous delivery

In this level, your system continuously delivers new pipeline implementations to the target environment that in turn delivers prediction services of the newly trained model. For rapid and reliable continuous delivery of pipelines and models, you should consider the following:

- Verifying the compatibility of the model with the target infrastructure before you deploy your model. For example, you need to verify that the packages that are required by the model are installed in the serving environment, and that the memory, compute, and accelerator resources that are available.
- Testing the prediction service by calling the service API with the expected inputs, and making sure that you get the response that you expect. This test usually captures problems that might occur when you update the model version and it expects a different input.

- Testing prediction service performance, which involves load testing the service to capture metrics such as queries per seconds (QPS) (https://wikipedia.org/wiki/Queries_per_second) and model latency.
- Validating the data either for retraining or batch prediction.
- Verifying that models meet the predictive performance targets before they are deployed.
- Automated deployment to a test environment, for example, a deployment that is triggered by pushing code to the development branch.
- Semi-automated deployment to a pre-production environment, for example, a deployment that is triggered by merging code to the main branch after reviewers approve the changes.
- Manual deployment to a production environment after several successful runs of the pipeline on the pre-production environment.

To summarize, implementing ML in a production environment doesn't only mean deploying your model as an API for prediction. Rather, it means deploying an ML pipeline that can automate the retraining and deployment of new models. Setting up a CI/CD system enables you to automatically test and deploy new pipeline implementations. This system lets you cope with rapid changes in your data and business environment. You don't have to immediately move all of your processes from one level to another. You can gradually implement these practices to help improve the automation of your ML system development and production.

What's next

- Learn more about Architecture for CI/CD and ML Pipelines using Kubeflow and Cloud Build (/solutions/machine-learning/architecture-for-mlops-using-tfx-kubeflow-pipelines-and-cloud-build)
.
- Learn more about GitOps-style continuous delivery with Cloud Build (/kubernetes-engine/docs/tutorials/gitops-cloud-build).
- Learn more about Setting up a CI/CD pipeline for your data-processing workflow (/solutions/cicd-pipeline-for-data-processing).

- Watch the MLOps Best Practices on Google Cloud (Cloud Next '19) (https://www.youtube.com/watch?v=20h_RTHEtZI) on YouTube.
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).

Last updated 2020-11-16 UTC.