neptune.ai

# Building a Machine Learning Platform [Definitive Guide]

Stephen Oladele

# Table of Content

# Intruduction

Moving across the typical machine learning lifecycle can be a nightmare. From gathering and processing data to building models through experiments, deploying the best ones, and managing them at scale for continuous value in production—it's a lot.

As the number of ML-powered apps and services grows, it gets overwhelming for data scientists and ML engineers to build and deploy models at scale.

Supporting the operations of data scientists and ML engineers requires you to reduce—or eliminate—the engineering overhead of building, deploying, and maintaining high-performance models. To do that, you'd need to take a systematic approach to MLOps—enter platforms!

Machine learning platforms are increasingly looking to be the "fix" to successfully consolidate all the components of MLOps from development to production. Not only does the platform give your team the tools and infrastructure they need to build and operate models at scale, but it also applies standard engineering and MLOps principles to all use cases.

But here's the catch: understanding what makes a platform successful and building it is no easy feat. With so many tools, frameworks, practices, and technologies available, it can be overwhelming to know where to start. That's where this guide comes in!

In this comprehensive guide, we'll explore everything you need to know about machine learning platforms, including:

- Components that make up an ML platform.
- How to understand your users (data scientists, ML engineers, etc.).

- Gathering requirements from your users.
- Deciding the best approach to build or adopt ML platforms.
- Choose the perfect tool for your needs.

This guide is a result of my conversations with platform and ML engineers and public resources from platform engineers in companies like Shopify, Lyft, Instacart, and StitchFix.

* * *

# What is a machine learning platform?

An ML platform standardizes the technology stack for your data team around best practices to reduce incidental complexities with machine learning and better enable teams across projects and workflows.

Why are you building an ML platform? We ask this during product demos, user and support calls, and on our ML Platform podcast. Generally, people say they do MLOps to make the development and maintenance of production machine learning seamless and efficient.

Machine learning operations (MLOps) should be easier with ML platforms at all stages of a machine learning project's life cycle, from prototyping to production at scale, as the number of models in production grows from one or a few to tens, hundreds, or thousands that have a positive effect on the business.

The platform should be designed to orchestrate your machine learning workflow, be environment-agnostic (portable to multiple environments), and work with different libraries and frameworks.

> *Data scientists only have to think about the where and when to deploy a model in a batch, not the how. The platform handles that.*
>
> — Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free; A Machine Learning Platform for Stitch Fix's Data Scientists

→ **Related post: MLOps: What It Is, Why It Matters, and How To Implement It**

# MLOps principles that ML platform should solve

Understanding MLOps principles and how you can implement them can govern how you build your ML platform. The principles of MLOps can actively help you frame how you define the goals of your machine learning platform.

## What do you want to prioritize?

Reproducible workflows? Seamless deployment? More effective collaboration? Here is how Airbnb's ML infrastructure team defined theirs:

> *The four goals we had when we built out our machine learning infrastructure were: keeping it seamless, versatile, consistent, and scalable... For seamless, we wanted to make it easy to prototype and productionize, and use same workflow across different frameworks. Making it versatile by supporting all major frameworks... Consistent environment across the entire stack... Keeping it horizontally scalable and making it elastic.*
>
> — **Andrew Hoh, Former Product Manager for the Machine Learning Infrastructure team at Airbnb at a presentation on "Bighead: Airbnb's end-to-end Machine Learning Platform"**

This section talks about the **MLOps principles** that can help ML platforms solve different kinds of problems:

- Reproducibility
- Versioning

- Automation
- Monitoring
- Testing
- Collaboration
- Scalability

These principles are standard requirements your ML platform should embody, but you might want to adopt principles specific to your organization and technical needs. For example, if your engineering team's culture involves using open source tools, you might want to consider a culture of using open source and open standard tools. Others may include developing your platform to include a culture of ownership.

Let's take a look at these principles in-depth.

## Reproducibility

Typically, machine learning models are designed to be unique. The core reason is that data usually has more than one type of structure. What is right for one business would not work for another; the data will give different insights.

Because ML projects are inherently experimental, your data scientists will always try out new ways to:

- Work with data,
- Build models,
- And set up parameters.

The challenge is tracking what worked and what didn't and maintaining reproducibility while maximizing code reuse.

To make that possible, your data scientists would need to store enough details about the environment the model was created in and the related metadata so that the model could be recreated with the same or similar outcomes.

You need to build your ML platform with experimentation and general workflow reproducibility in mind. Reproducibility ensures you are dealing with a process and not simply an experiment.

By storing all model-training-related artifacts, your data scientists will be able to run experiments and update models iteratively. From design to development to deployment, these attributes are an important part of the MLOps lifecycle.

## Versioning

Your data science team will benefit from using good MLOps practices to keep track of versioning, particularly when conducting experiments during the development stage. They would always be able to go back in time and closely repeat the results of previous experiments (without taking into account things like the fact that hardware and libraries can be non-deterministic).

Version control for code is common in software development, and the problem is mostly solved. However, machine learning needs more because so many things can change, from the data to the code to the model parameters and other metadata. Your ML platform must have versioning in-built because code and data mostly make up the ML system.

Versioning and reproducibility are important for improving machine learning in real-world organizational settings where collaboration and governance, like audits, are important. These may not be the most exciting

parts of model development, but they are necessary for progress.

It should be able to version the project assets of your data scientists, such as the data, the model parameters, and the metadata that comes out of your workflow.

## Automation

You want the ML models to keep running in a healthy state without the data scientists incurring much overhead in moving them across the different lifecycle phases. Automation is a good MLOps practice for speeding up all parts of that lifecycle. It would make sure that all development and deployment workflows use good software engineering practices.

Continuous integration and deployment (CI/CD) are crucial for effective MLOps and can enable automation. CI/CD lets your engineers add code and data to start automated development, testing, and deployment, depending on how your organization is set up.

Through CI/CD, automation would also make it possible for ML engineers or data scientists to track the code running for their prediction service. This would let you roll back changes and inspect potentially buggy code.

Depending on your use case, building a fully automated self-healing platform may not be worth it. But machine learning should be seen as an ongoing process, and it should be as easy as possible to move models from development to production environments.

A platform without automation would likely waste time and, more importantly, keep the development team from testing and deploying often. This can make it harder for ML engineers to find bugs or make design choices that make it impossible to deliver ML applications and enable new use cases.

An automated platform can also help with other processes, such as giving your data scientists the freedom to use different libraries and packages to build and run their models.

> *... This just shows the flexibility that we allow our users to have because each project can use a different set of requirements, images, and dependencies based on what they need for their use case.*
>
> — Isaac Vidas, **Shopify's ML Platform Lead, at** Ray Summit 2022

## Monitoring

Monitoring is an essential DevOps practice, and MLOps should be no different. Checking at intervals to make sure that model performance isn't degrading in production is a good MLOps practice for both teams and platforms.

In machine learning, performance monitoring isn't just about technical performance, like latency or resource utilization, but also about the predictive performance of the model, especiallywhen production data may change over time, which is more important.

This is crucial for data scientists because you want to make it easy for them to know if their models are providing continuous value and when they aren't, so they can know when to update, debug, or retire them.

An ML platform should provide utilities for your data scientists or ML engineers to check the model's (or service's) production performance and also ensure it has enough CPU, memory, and persistent storage.

→ **Read more: A Comprehensive Guide on How to Monitor Your Models in Production**

## Testing

Quality control and assurance are necessary for any machine learning project. Working together, the team's data scientists and platform engineers should conduct tests.

Data scientists would want to be able to test how well the model works on unseen data and understand the risks so they can design the right validation tests for both offline and production environments.

On the other hand, the platform team would want to make sure the data scientists have the tools they need to test their models while they are building them, as well as the system surrounding their models. That includes the following:

- Pipelines (the data into the pipeline and the model out of the training pipeline),
- Infrastructure,
- And other production services.

MLOps tests and validates not only code and those components but also data, data schemas, and models. When tests fail, a good testing component should make it as easy as possible for team members to figure out what went wrong.

In traditional software engineering, you will find that testing and automation go hand-in-hand in most stacks and team workflows. Most of these tests should be done automatically, which is an important practice for effective MLOps.

Lack of automation or speed wastes time, but more importantly, it keeps the development team from testing and often deploying, which can

make it take longer to find bugs or bad design choices that halt deployment to production.

## Collaboration

The principles you have learned in this guide are mostly born out of DevOps principles. One common theme between DevOps and MLOps is the practice of collaboration and effective communication between teams.

> *When it comes to what seems to work for organizing data teams, there are a couple of overall structures that seem to work quite well. First off, there is the "embedded approach," where you embed a machine learning engineer on each team... The "centralized machine learning engineer approach," where you separate the MLE team that refactors code for data scientists, seems to be more common... Clearly enforced standard operating procedures are the key to having effective handoffs across teams.*
>
> — **Conor Murphy, Lead Data Scientist at Databricks, in "Survey of Production ML Tech Stacks" at the Data+AI Summit 2022**

Your team should be motivated by MLOps to show everything that goes into making a machine learning model, from getting the data to deploying and monitoring the model.

It is very easy for a data scientist to use Python or R and create machine learning models without input from anyone else in the business operation. That might be fine when developing, but what happens when you want to put it into production and there needs to be a unified use case?

If the teams don't work well together, workflows will always be slow or models won't be able to be deployed. Machine learning platforms must incorporate collaboration from day one, when everything is fu -lly audited.

Organizational-wide permissions and visibility will ensure the strategic deployment of machine learning models, where the right people have the right level of access and visibility into projects.

Learn from the practical experience of four ML teams about some collaboration best practices.

## Scalability

Compute power is fundamental to the machine learning lifecycle. Data scientists and machine learning engineers need an infrastructure layer that lets them scale their work without having to be networking experts.

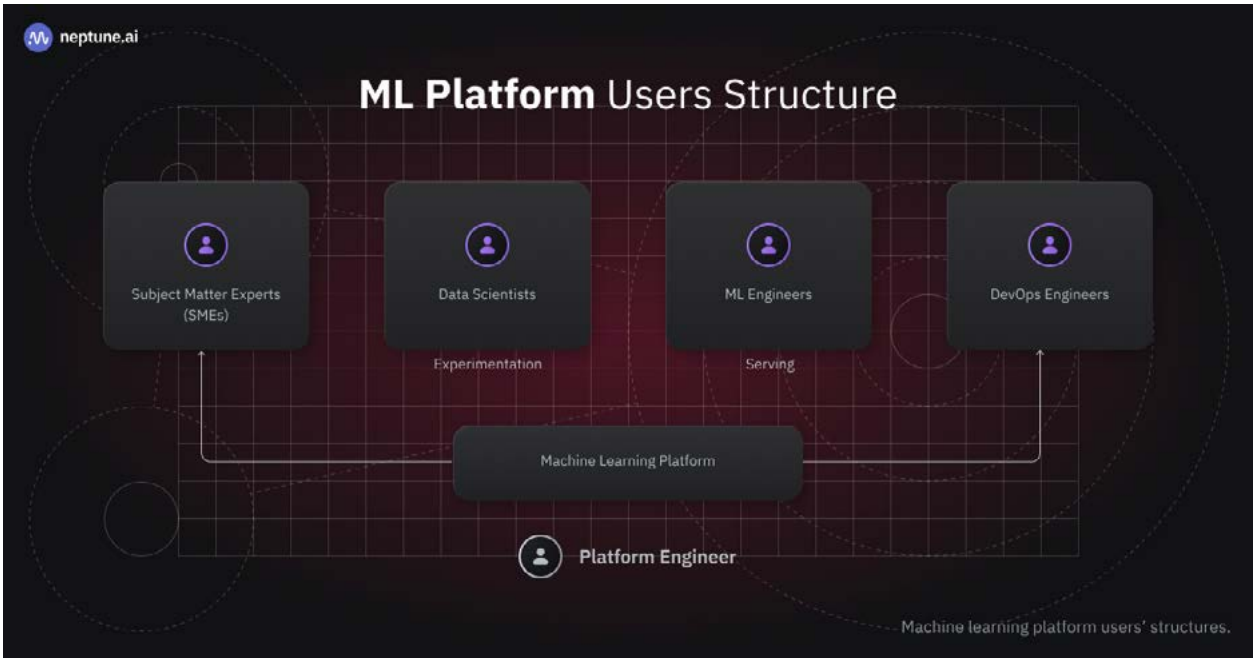
Volumes of data can snowball, and data teams need the right setup to scale their workflow and processes. ML platforms should make it easy for data scientists and ML engineers to use the infrastructure to scale projects.

* * *

# Understanding users of machine learning platforms

## What's your role as a Platform Engineer?

Your role as a platform engineer, or in most cases, an "MLOps engineer" is to practically architect and build solutions that make it easy for your users to interact with the ML lifecycle while proving appropriate abstractions from the core infrastructure.




Machine learning platform users' structures

## ML Engineers and Data Scientists

### Who they are?

Depending on the existing team structure and processes of the business, your ML engineers may work on delivering models to production, and your data scientists may focus on research and experimentation.

Some organizations hire either person to own the end-to-end ML project and not parts of it. You'd need to understand the roles that both personas currently play in your organization to support them.

#### What do they want to accomplish?

The following are some of the goals they'd like to achieve:

- **Frame business problem:** collaborate with subject matter experts to outline the business problem in such a way that they can build a viable machine learning solution.
- **Model development:** access business data from upstream components, work on the data (if needed), run ML experiments (build, test, and strengthen models), and then deploy the ML model.
- **Productionalization:** this is often really subjective in teams because it's mostly the ML engineers that end up serving models. But the lines between data scientists and ML engineers are blurring with the commoditization of model development processes and workflows with tools like Hugging Face and libraries that make it easy to build models quickly. They are always checking model quality to verify that the way it works in production responds to initial business queries or demands.

→ **Related post: Roles in ML Team and How They Collaborate With Each Other**

### DevOps Engineers

#### Who they are?

Depending on the organization, they are either pure software engineers or

simply tagged “DevOps engineers” (or IT engineers). They are mostly responsible for operationalizing the organization’s software in production.

#### What do they want to accomplish?

Perform operational system development and testing to assure the security, performance, and availability of ML models as they integrate into the wider organizational stack.

They are responsible for CI/CD pipeline management across the entire organizational stack.

### Subject Matter Experts (SMEs)

#### Who they are?

SMEs are the non-developer experts in the business problem that have critical roles to play across the entire ML lifecycle. They work with other users to make sure the data reflects the business problem, the experimentation process is good enough for the business, and the results reflect what would be valuable to the business.

#### What do they want to accomplish?

You would need to build interfaces into your platforms for your SMEs to:

- Contribute to data labeling (if your data platform is not separate from the ML platform),
- Perform model quality assurance for auditing and managing risks both in development and post-production,

- Close feedback stages in production to make sure model performance metrics translate to real-world business value.

Of course, what you prioritize would depend on the company's use case and existing problem sphere.

### Other users

Some other users you may encounter include:

- Data engineers, if the data platform is not particularly separate from the ML platform.
- Analytics engineers and data analysts, if you need to integrate third-party business intelligence tools and the data platform, is not separate.

## Gathering requirements, feedback, and success criteria

There is no one way to gather and elicit requirements for your ML platform because it depends on the business use case and overall organizational structure, but here's how Olalekan Elesin, Director of Data Platforms at HRS Product Solutions GmbH, did it at his previous company, Scout24:

> *We did a couple of "statistics" to determine the most pressing problems for teammates who wanted to go from idea to production for machine learning... What we did was create a survey and run it by 40 people... from that survey, we identified three things as most important:*
> *1. Simplify the time it took to get an environment up and running*
> *2. Simplify the time it took to put ML modelsin production.*
> *3. Increase the knowledge on building ML models.*

Olalekan said that most of the random people they talked to initially wanted a platform to handle data quality better, but after the survey, he found out that this was the fifth most crucial need. So in building the platform, they had to focus on one or two pressing needs and build requirements around them.

Gathering and eliciting requirements for building ML platforms is similar to how you would design traditional software programs. You want to:

1. Define the problem your data scientists are facing and how it contributes to the overarching business objectives.
2. Develop the user stories (in this case, the stakeholders you are building the platform for).
3. Design the platform's structure (relating to architecture and other necessary requirements).

## Define the problem

The problem statement describes the problem your users are facing and also the solution to that problem. For example, a problem for your data scientist may be that they take too long to develop and deploy solutions to their end users. Another reason could be that they waste a lot of time configuring the infrastructure needed for their projects.

Of course, problem statements need to be more detailed than that, but this is just to give you an idea of how you could frame them.

> *Developing our ML platform has been a transformative process that involves building trust with data scientists and digging deep to understand their approaches. Doing so successfully is a challenging journey, but when we succeed, we greatly improve the data scientist and client experience.*

— Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free: A Machine Learning Platform for Stitch Fix's Data Scientists

## Develop the user stories

> *While building an ML platform, it is also important to remember who your users are and their profiles. If it requires it, add CLI or different tools to allow them to migrate easily to your new platform.*

— Isaac Vidas, Shopify's ML Platform Lead, at Ray Summit 2022

Once you understand the problem your data scientists face, your focus can now be on how to solve it. The user stories will explain how your data scientist will go about solving a company's use case(s) to get to a good result.

Given the problem statement you defined earlier, you'd have to work with your data scientists to come up with this process as you write stories about their problems.

**An example of a user story could be**: *"Data scientists need a way to choose the compute resource given to them before starting a notebook kernel to make sure they can launch notebooks successfully on their preferred instance"*.

## Design the platform's structure

At this point, you want to start identifying the agents and core components of the platform. Depending on the architectural pattern you decide on, you'll start figuring out how the agents interact with each other and what the dependency relationships are.

For example, how would the models move from the notebooks to a model serving component, or how would the data component interact with

the development environment for your data scientists? This is, of course, high-level, but you get the idea.

All of the platform's components should correspond to thigs that happen in the data scientists' problem domain, not just things that would be nice to have. That's the only way to prevent the system from descending into chaos because the domain gives coherence to the architecture that you wouldn't otherwise have.

> *At Stitch Fix, data scientist autonomy and quick iteration are paramount to our operational capabilities—above all, we value flexibility. If we can't quickly iterate on a good idea, we're doing our clients (and ourselves) a disservice.*
>
> — Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free; A Machine Learning Platform for Stitch Fix's Data Scientists

Some resources for perusal:

- ML platforms where to start? (in MLOps.community by Olalekan Elesin, Director of Data Platforms at HRS Product Solutions GmbH)
- MLOps Jobs to Be Done (from GitLab)

→ **Recommended podcast episode: Learnings From Building the ML Platform at Stitch Fix With Stefan Krawczyk**

* * *

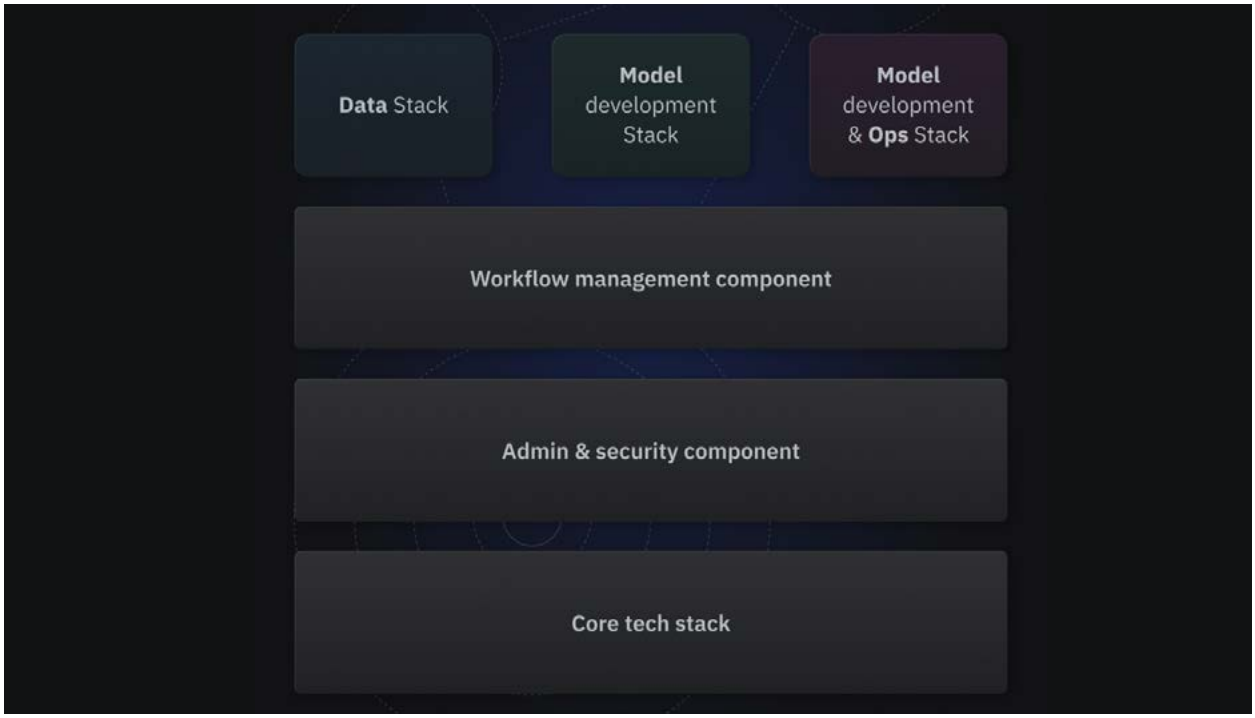
# ML platform architecture

The ML platform architecture serves as a blueprint for your machine learning system. This article defines architecture as the way the highest-level components are wired together.

The features of an ML platform and the core components that make up its architecture are:

1. Data stack and model development stack.
2. Model deployment and operationalization stack.
3. Workflow management component.
4. Administrative and security component.
5. Core technology stack.

Let's take a look at each of these components.




Core components of an ML platform | Source: Modified and adapted from Mary Grace Moesta's presentation at Data+Summit 2022

## 1. Data and model development stacks

Main components of the data and model development stacks include:

- Data and feature store.
- Experimentation component.
- Model registry.
- ML metadata and artifact repository.

### Data and feature store

In a machine learning platform, feature stores (or repositories) give your data scientists a place to find and share the features they build from their datasets. It also ensures they use the same code to compute feature values for model training and inference to avoid training-serving skew.

Different teams may be involved in extracting features from different dataset sources, so a centralized storage would ensure they could all use the same set of features to train models for different use cases.

The feature stores can be offline (for finding features, training models, and batch inference services) or online (for real-time model inference with low latency).

The key benefit that a feature store brings to your platform is that it decouples feature engineering from feature usage, allowing independent development and consumption of features. Features added to a feature store become immediately available for training and serving.

→ **See also: How to Solve the Data Ingestion and Feature Store Component of the MLOps Stack**

## Experimentation component

Experiment tracking can help manage how an ML model changes over time to meet your data scientists' performance goals during training. Your data scientists develop models on this component, which stores all parameters, feature definitions, artifacts, and other experiment-related information they care about for every experiment they run.

Along with the code for training the model, this component is where they write code for data selection, exploration, and feature engineering. Based on the results of the experiments, your data scientists may decide to change the problem statement, switch the ML task, or use a different evaluation metric.

**Check out the resources below to learn more about this component:**

- ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It
- Machine Learning Experiment Management: How to Organize Your Model Development Process
- Experiment Tracking in Kubeflow Pipelines
- ML Metadata Store: What It Is, Why It Matters, and How to Implement It
- How to Build an Experiment Tracking Tool [Learnings From Engineers Behind Neptune]

## Model registry

The model registry component helps you put some structure into the process of productionalizing ML models for your data scientists.

The model registry stores the validated training model and the metadata and artifacts that go with it.

This central repository stores and organizes models in a way that makes it more efficient to organize models across the team, making them easier to manage, deploy, and, in most cases, avoid production errors (for example, putting the wrong model into production).

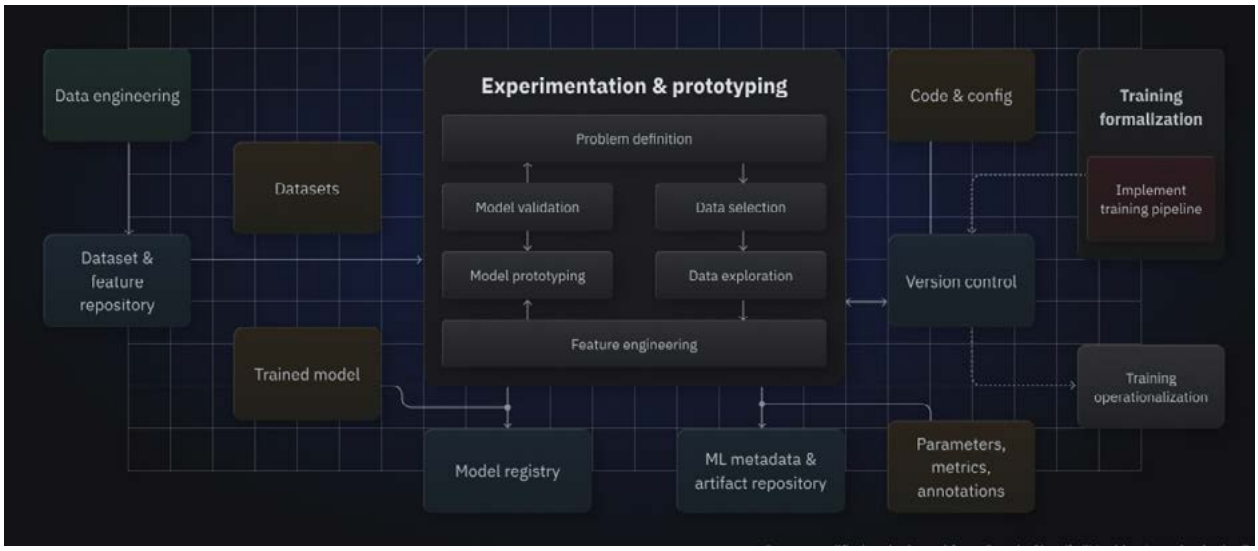
→ **Learn more: Best ML Model Registry Tools**

## ML metadata and artifact repository

You might need the ML metadata and artifact repository to make it easier to compare model performance and test them in the production environment. A model can be tested against the production model, drawing from the ML metadata and artifact store to make those comparisons.

Learn more about this component in this blog post about the ML metadata store, what it is, why it matters, and how to implement it.

Here's a high-level structure of how the data stack fits into the model development environment:



(prev page) Image modified and adapted from Google Cloud's "Machine Learning in the Enterprise" learning resource

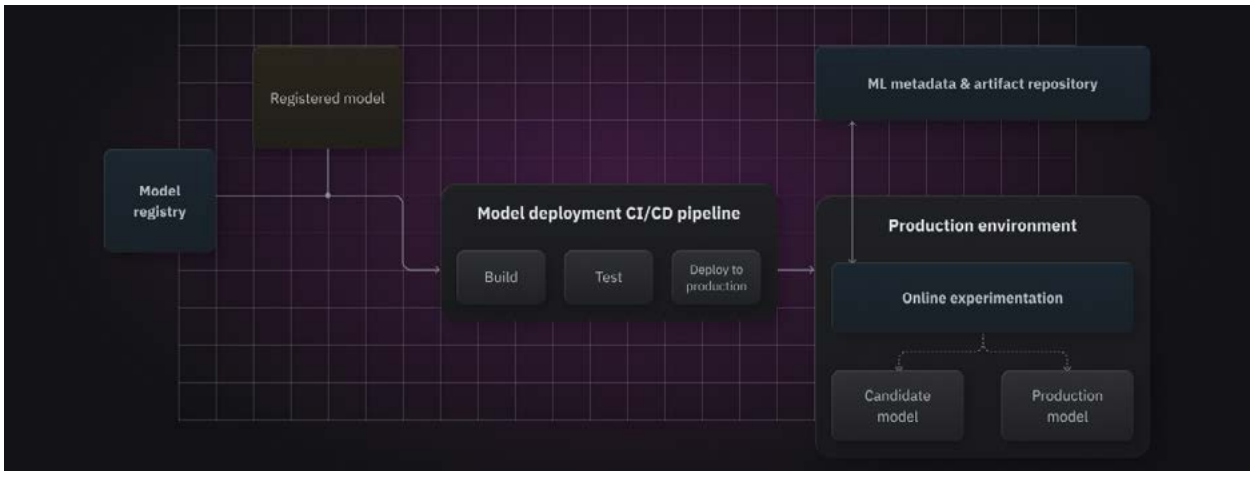
## 2. Model deployment and operationalization stack

The main components of the model deployment and operationalization stack include the following:

- Production environment.
- Model serving.
- Monitoring and observability.
- Responsible AI and explainability.
- ML metadata and artifact repository

Your data scientists can manually build and test models that you deploy to the production environment. In an ideal situation, pipelines and orchestrators take a model from the model registry, package it, test it, and then put it into production.

The production environment component lets the model be tested against the production models (if they exist) by using the ML metadata and artifact store to compare the models. You could also decide to build configurations for deployment methods like canary, shadow, and A/B deployment in the production environment.



(prev page) The high-level structure of a progressive delivery workflow from the model development environment to the production environment | Image modified and adapted from Google Cloud's "Machine Learning in the Enterprise" learning resource

## Model serving component

When your DSs (data scientists) or MLEs (machine learning engineers) deploy the models to their target environments as services, they can serve predictions to consumers through different modalities.

The model serving component helps organize the models in production so you can have a unified view of all your models and successfully operationalize them. It integrates with the feature store for retrieving production features and the model registry for serving candidate models.

You can go through this guide to learn how to solve the model serving component of your MLOps platform.

These are the popular model serving modalities:

- Online inference.
- Streaming inference.
- Offline batch inference.
- Embedded inference.

### Online inference

The ML service serves real-time predictions to clients as an API (a function call, REST API, gRPC, or similar) for every request on demand. The only concern with this service would be scalability, but that's a typical operational challenge for software.

### Streaming inference

The clients push the prediction request and input features into the feature store in real time. The service will consume the features in real time, generate predictions in near real-time, such as in an event processing pipeline, and write the outputs to a prediction queue.

The clients can read back predictions from the queue in real time and asynchronously.

### Offline batch inference

The client updates features in the feature store. An ML batch job runs periodically to perform inference. The job reads features, generates predictions, and writes them to a database. The client queries and reads the predictions from the database when needed.

### Embedded inference

The ML service runs an embedded function that serves models on an edge device or embedded system.

## Monitoring component

Implementing effective monitoring is key to successfully operating machine learning projects. A monitoring agent regularly collects telemetry data, such as audit trails, service resource utilization, application statistics, logs, errors, etc. This makes this component of the system work. It sends the data to the model monitoring engine, which consumes and manages it.

Inside the engine is a metrics data processor that:

- Reads the telemetry data,

- Calculates different operational metrics at regular intervals,
- And stores them in a metrics database.
- The monitoring engine also has access to production data, runs an ML metrics computer, and stores the model performance metrics in the metrics database.

An analytics service provides reports and visualizations of the metrics data. When certain thresholds are passed in the computed metrics, an alerting service can send a message.

→ **Related post:** A Comprehensive Guide on How to Monitor Your Models in Production

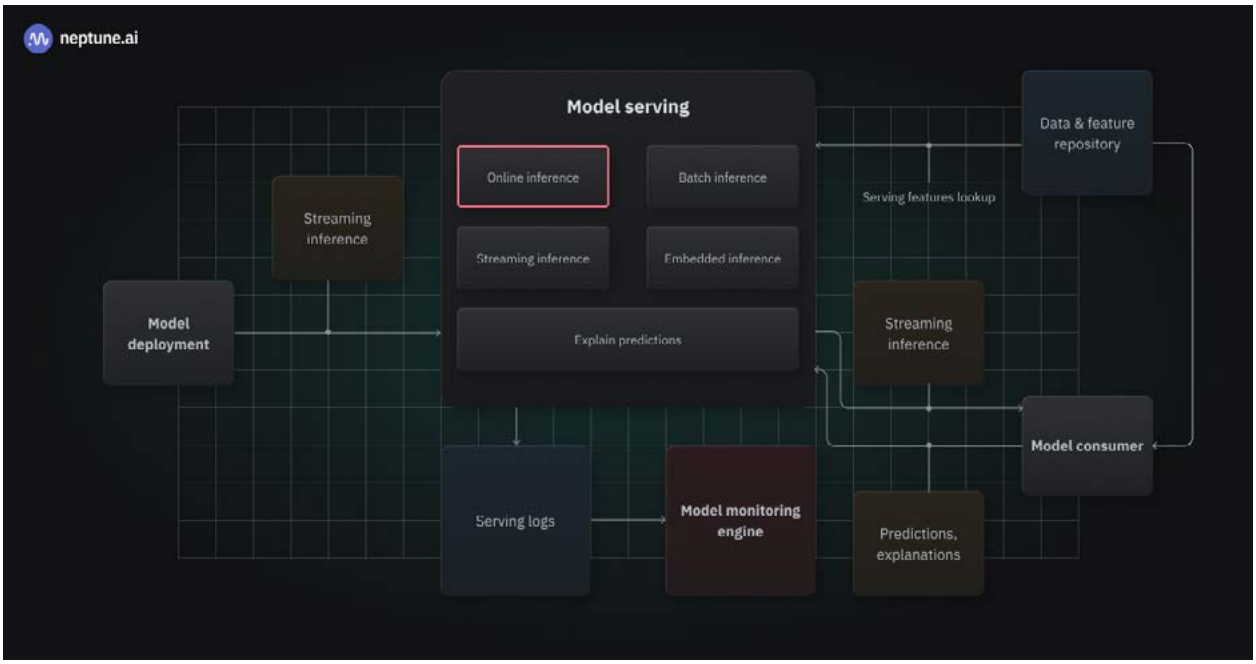
## Responsible AI and explainability component

To fully trust ML systems, it's important to interpret these predictions. You'd need to build your platform to perform feature attribution for a given model prediction; these explanations show why the prediction was made.

**You and your data scientist must implement this part together to make sure that the models and products meet the governance requirements, policies, and processes.**

Since ML solutions also face threats from adversarial attacks that compromise the model and data used for training and inference, it makes sense to inculcate a culture of security for your ML assets too, and not just at the application layer (the administrative compoonent).

→ **Learn more:** Explainability and Auditability in ML: Definitions, Techniques, and Tools




The communication between data and model deployment stack, and the model and operationalization stacks of an ML platform | Image modified and adapted from Google Cloud's "Machine Learning in the Enterprise" learning resource

## 3. Workflow management component

The main components here include:

- Model deployment CI/CD pipeline.
- Training formalization (training pipeline).
- Orchestrators.
- Test environment.

### Model deployment CI/CD pipeline

ML models that are used in production don't work as stand-alone software solutions. Instead, they must be built into other software components to work as a whole. This requires integration with components like APIs, edge devices, databases, microservices, etc.

The CI/CD pipeline retrieves the model from the registry, packages it as executable software, tests it for regression, and then deploys it to the production environment, which could be embedded software or ML-as--a-service.

> *Once users push their Merlin Project code to their branch, our CI/CD pipelines build a custom Docker image.*
>
> — Isaac Vidas, **ML Platform Lead at Shopify, in** "The Magic of Merlin: Shopify's New Machine Learning Platform"

The idea of this component is automation, and the goal is to quickly re-build pipeline assets ready for production when you push new training code to the corresponding repository.

→ **Bookmark for later:** 4 Ways Machine Learning Teams Use CI/CD in Production

## Training formalization (training pipeline)

In cases where your data scientists need to retrain models, this component helps you manage repeatable ML training and testing workflows with little human intervention.

The training pipeline functions to automate those workflows. From:

- Collecting data from the feature store,
- To setting some hyperparameter combinations for training,
- Building and evaluating the model,
- Retrieving the test data from the feature store component,
- Testing the model and reviewing results to validate the model's quality,

- If needed, updating the model parameters and repeating the entire process.

The pipelines primarily use schedulers and would help manage the training lifecycle through a DAG (directed acyclic graph). This makes the experimentation process traceable and reproducible, provided the other components discussed earlier have been implemented alongside it.

→ **Related post: Building ML Pipeline: 6 Problems & Solutions**

## Orchestrators

The orchestrators coordinate how ML tasks run and where they get the resources to run their jobs. Orchestrators are concerned with lower-level abstractions like machines, instances, clusters, service-level grouping, replication, and so on.

Along with the schedulers, they are integral to managing the regular workflows your data scientists run and how the tasks in those workflows communicate with the ML platform.

## Test environment

The test environment gives your data scientists the infrastructure and tools they need to test their models against reference or production data, usually at the sub-class level, to see how they might work in the real world before moving them to production. In this environment, you can have different test cases for your ML models and pipelines.

This article by Jeremy Jordan delves deeper into how you can effectively test your machine learning systems.

If you want to learn how others in the wild are testing their ML systems, you can check out this article focused on ML model testing I curated.

## 4. Administrative and security components

This component is in the application layer of the platform and handles the user workspace and interaction with the platform. Your data scientists, who are your users in most cases, barring other stakeholders, would need an interface to, for example, select compute resources, estimate costs, manage resources, and the different projects they work on.

In addition, you also need to provide some identity and access management (IAM) service so the platform only provides the necessary access level to different components and workspaces for certain users. This is a typical software design task to ensure your platform and users are secured.

## 5. Core technology stack

The main components of this stack include:

- Programming Language.
- Collaboration.
- Libraries and Frameworks.
- Infrastructure and Compute.
- Programming language

The programming language is another crucial component of the ML platform. For one, the language would you use to develop the ML platform, and equally as important, the language your users would perform ML development with.

The most popular language with string community support that would likely ensure you are making your users' workflow efficient would likely be Python. But then again, understand their existing stack and skillset, so you know how to complement or migrate it.

> *One of the areas I encourage folks to think about when it comes to language choice is the community support behind things. I have worked with customers where R and SQL were the first-class languages of their data science community.*
> *They were eager to build all their pipelines in R and SQL... because there is so much community support behind Python and see the field favouring Python, we encourage them to invest time upfront to have our data teams build pipelines in Python...*
>
> — Mary Grace Moesta, **Data Scientist at Databricks, in** Survey of Production ML Tech Stacks" presentation at Data+Summit 2022

## Collaboration

Earlier in this article, you learned that one of the most important principles of MLOps that should be integrated into any platform is collaboration. The collaboration component has to do with how all the platform users can collaborate with each other and across other teams.

The main components here include:

- Source control repository.
- Notebooks and IDEs.
- Third-party tools and integrations.

## Source code repository

During experimentation, this component lets your data scientists share code bases, work together, peer review, merge, and make changes. A source code repository is used to keep track of code artifacts like notebooks, scripts, tests, and configuration files that are generated during experimentation.

> *The big thing to consider here is how your ML teams are structured. If you are setting standards where, for example, your DevOps and Engineering teams work with GitHub, and you have embedded ML teams, it's best to keep that standard. You want to make sure that there's consistency across these teams.*
>
> — Mary Grace Moesta, Data Scientist at Databricks, in Survey of Production ML Tech Stacks presentation at Data+Summit 2022

→ **Recommended: Version Control for ML Models: Why You Need It, What It Is, How To Implement It**

## Notebooks and IDEs

The notebook is the experimentation hub for your data scientists, and there needs to be an agreement on what tools will be ideal for the team long-term—components that will be around in 5–10 years.

> *For the notebook-based tools and IDEs like Jupyter and PyCharm, it's thinking about what is going to be maintainable for the team long-term.*
>
> — Mary Grace Moesta, Data Scientist at Databricks, in Survey of Production ML Tech Stacks presentation at Data+Summit 2022

Using an open source solution like Jupyter Notebook can leave you with flexibility to add direct platform integrations and could also serve as the workspace for your data scientist. From the notebook, you can add the feature for your users to select compute usage and even see cost estimates.

The IDEs, for example, VSCode or Vim, may be how other stakeholders that use the platform interact with it at a code level.

## Third-party tools and integrations

Sometimes, your data team might need to integrate with some external tool that wasn't built with the platform, perhaps due to occasional needs. For example, the data team might want to use an external BI tool to make reports. This component ensures that they can flexibly integrate and use such a tool in their workspace through an API or other communication mechanism.

Through the integrations as well, you'll also have the right abstractions to replace parts of the platform with more mature, industry-standard solutions.

> *Building our own platform did not, however, preclude taking advantage of external tooling. By investing in the right abstractions, we could easily plug into and test out other tools (monitoring and visibility, metrics and analysis, scalable inference, etc.), gradually replacing pieces that we've built with industry standards as they mature.*
>
> — Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free: A Machine Learning Platform for Stitch Fix's Data Scientists

## Libraries and frameworks

This component lets you natively integrate machine learning libraries and frameworks your users mostly leverage into the platform. Some examples are TensorFlow, PyTorch, and so on.

You never want your data scientist to worry about the packages they use to develop. They should be able to know that this is the package version and just import it on their workspace.

Some requirements you want to take into consideration will depend on the features of the library and frameworks:

1. Are they open source or open standard?
2. Can you perform distributed computing?
3. What's the community support behind them?
4. What are their strengths and limitations?

Of course, this is valid if you use an external library rather than building them for your data scientists, which is usually inadvisable.

## Infrastructure and compute

The infrastructure layer of your ML platform is arguably the most important layer to figure out, along with the data component. Your ML platform will run on this layer, and with the different moving parts and components you have seen, it can be quite tricky to tame and manage this layer of the platform.

The infrastructure layer allows for scalability at both the data storage level and the compute level, which is where models, pipelines, and applications are run. The considerations include:

- Are your existing tools and services running on the Cloud, on-prem, or a hybrid?
- Are the infrastructure services (like storage, databases, for example) open source, running on-prem, or running as managed services?

These considerations would help you understand how to approach designing your platform.

But ideally, you want an infrastructure layer that reduces the friction between moving across the different stacks, from data to model development, deployment, and operationalization.

In most cases, that'd arguably be the cloud, but of course, not every use case can leverage cloud infrastructure, so keep that principle of "less friction" in mind regardless of where your infrastructure is located.

## Other resources to learn ML platform design

This section has touched on the most important components to consider when building an ML platform. Find other resources to help you dig deeper into this topic below:

- MLOps Architecture Guide.
- Building a Machine Learning platform (Lemonade).
- Design Patterns in Machine Learning for MLOps (by Pier Paolo Ippolito).
- Machine Learning Operations (MLOps): Overview, Definition, and Architecture (by Kreuzberger, et al., 2022).
- AIIA MLOps blueprints.

* * *

# Considerations when deciding on the scope of the ML platform

## Enterprise machine learning platform vs startup ML platform

Transparency and pure efficiency are two themes for mastering MLOps—especially when meeting data scientists' needs in an enterprise setting because it's mostly speed and agility that matter.

In most startups, data scientists can get away with deploying, testing, and monitoring models ad hoc, especially with a handful of models. However, in the enterprise, they would waste enormous amounts of time reinventing the wheel with every ML model, which may never result in scalable ML processes for the organization.

In this section, you will learn what makes an ML platform for an enterprise different from one for a startup.

Here's a comparison table showing the differences in ML platforms at start-ups compared to enterprises:

| | Startups | Enterprises |
|---|---|---|
| **Number of ML services** | The platform may have a handful of models to support development and production based on a small number of use cases. | The platform could support thousands to hundreds of models in research and production. |

(Continued on the next page)

| | Startups | Enterprises |
|---|---|---|
| **Data volume** | Often in the order of gigabytes or terabytes per day. | On average, most enterprises, especially those with digital products, often deal with data volumes in the order of petabytes per day. |
| **Business impact** | ML services may have ROIs in the hundreds of thousands to tens of millions of US dollars per year. | Enterprise ML services may have a yearly ROI in the hundreds of millions or billions of dollars. |
| **Infrastructure** | The number of use cases supported determines the requirements and how specialized those use cases are. | Due to the scale of models, research, and use cases, the infrastructure needs are often on the high side of the spectrum. |
| **Tooling** | Evaluating tools for ML platforms is frequently simple, with considerations for cloud, open source, or managed services. | Evaluating tools for ML platforms is based on strict criteria involving several stakeholders. |
| **Team sizes** | The platform team is usually a handful of engineers or tens of engineers supporting data scientists. | The platform is typically built by hundreds or thousands of engineers who support many data scientists and researchers from various teams. |
| **Team sizes** | The MLOps maturity level of the platforms is usually at level 0 due to many ad-hoc processes. | Due to the number of engineers and the overhead involved in managing the systems, the maturity levels are usually higher, around level 1 or 2. |

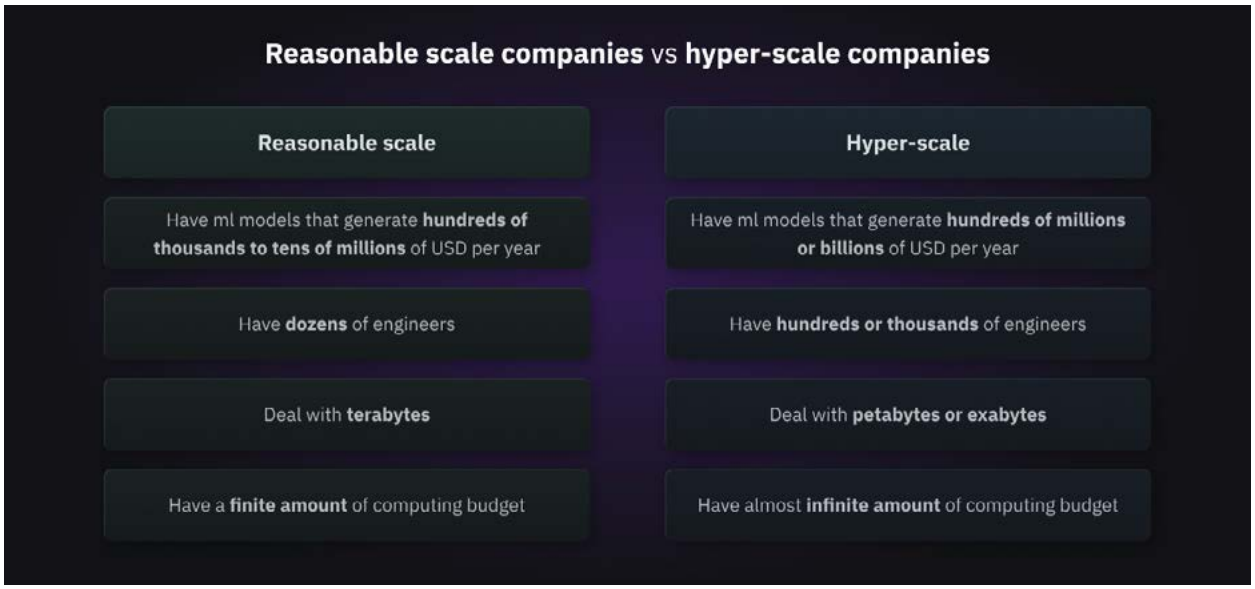
## Reasonable scale ML platform

In 2021, Jacopo Tagliabue coined the term "reasonable scale," which refers to companies that:

- Have ML models that generate hundreds of thousands to tens of millions of US dollars per year (rather than hundreds of millions or billions).
- Have dozens of engineers (rather than hundreds or thousands).
- Deal with data in the order of gigabytes and terabytes (rather than petabytes or exabytes).
- Have a finite amount of computing budget.

A reasonable scale ML platform helps achieve those requirements listed above.

Compared to a reasonable scale platform, a “hyper-scale” platform should support hundreds to thousands of models in production that likely generate millions to billions of dollars in revenue per year. They also have hundreds to thousands of users, support multiple use cases, and handle petabyte-scale data.




Comparing “reasonable scale” to “hyper-scale” companies | Source: Jakub Czakon in MLOps at a Reasonable Scale [The Ultimate Guide]

Learn more about reasonable scale MLOps and related concepts using the resources below:

- MLOps at a Reasonable Scale [The Ultimate Guide]
- Setting up MLOps at a Reasonable Scale with Jacopo Tagliabue
- MLOps without Much Ops. Or: How to build AI companies (by Jacopo Tagliabue in Towards Data Science)
- ML and MLOps at a Reasonable Scale (by Ciro Greco in Towards Data Science)
- The Post-Modern Stack. Joining the modern data stack and the (by Jacopo Tagliabue in Towards Data Science).

## Data-sensitive ML platform

Data privacy is important when building an ML platform, but how should sensitive data be kept from being accidentally or maliciously leaked (adversarial attacks) when your data scientists use it for training or inference?

If your organizational use cases require privacy protections, you need to build your ML platform with customer and user trust and compliance with laws, regulations, and standards in mind. This includes compliance with the data laws of your business vertical.

Consider building the following into your ML platform:

- Access controls to data components specifically at an attribute level.
- Use or build tools to anonymize datasets and protect sensitive information.
- Use or build data aggregation tools to hide individual information.

- Add tools for modeling threats and analyzing data leakages to identify gaps in privacy implementation and fix them.

Some other techniques you may want to learn about include federated learning and differential privacy to make it hard for hackers to reverse-engineer your components.

Useful tools and frameworks to build data-sensitive components as part of your ML platform:

- Tensorflow Privacy.
- ML Privacy Meter.
- PySyft.
- CrypTFlow.

## Human-in-the-loop ML platforms

Human oversight is crucial for every ML platform. As much as you'd like to automate every component of the platform to make your life and that of your data scientists easier, it's most likely not going to be possible.

For example, finding "unknown unknowns" of data quality problems on platforms that use new data to trigger retraining feedback loops might be hard to automate. And when the platform automates the entire process, it'll likely produce and deploy a bad-quality model.

Your components need to have interfaces that relevant stakeholders and subject matter experts can interact with to evaluate data and model quality. You also need to design the platform with them to understand the components that require manual evaluation and those that can be solved with automated monitoring.

Some resources and stories from the wild to learn more about this:

- MLOps is an extension of DevOps. Not a fork
- Greensteam in shipping
- Respo Vision in sports analytics
- Continuum Industries in industrial optimization

## Building an ML platform for special industry verticals

Across different industries and business verticals, the use cases and ML product goals will differ. It can be difficult to build a platform that caters to most use cases—this is one of the reasons why an all-size-fits-all ML platform may not work for most teams.

You need a platform built on components that can be flexible to handle different objects and are reusable across use cases.

The major considerations to make when planning an ML platform across special industry verticals include:

- **Data type:** For the different types of use cases your team works on, what's the most prevalent data type, and can your ML platform be flexible enough to handle them?
- **Model type:** In some cases, you'd be developing, testing, and deploying computer vision models alongside large language models. How would you build your platform to allocate development and production resources effectively?
- **Legal requirements:** What are the legal and governance requirements the platform needs to comply with, from the data component to the predictions that get to the end users?

- **Team structure and collaboration:** How is the team working on this use case structured, and what stakeholders would be actively involved in the projects the platform needs to serve?

Let's look at the healthcare vertical for context.

## Machine learning platform in healthcare

There are mostly three areas of ML opportunities for healthcare, including computer vision, predictive analytics, and natural language processing. For use cases with input data as images, computer vision techniques like image classification, segmentation, and recognition can help augment the work of stakeholders over different modalities (real-time, batch, or embedded inference).

Other use cases may require your data scientists to leverage the ML platform to quickly build and deploy models to predict the likelihood of an event based on large amounts of data. For example, predicting hospital readmissions and emerging COVID-19 hotspots In most cases, these would be batch jobs running on anonymized and de-identified patient data.

With language models and NLP, you'd likely need your data component to also cater for unstructured text and speech data and extract real-time insights and summaries from them.

The most important requirement you need to incorporate into your platform for this vertical is the regulation of data and algorithms. The healthcare industry is a highly regulated field, and you need to make sure that you are enabling your data scientists, researchers, and products to comply with policies and regulations.

> *If I had to reduce what it takes to set up MLOps at a healthcare startup to a 3-step playbook, it would be:*
>
> *1. Find a compelling use case.*
>
> *2. Set up good process.*
>
> *3. Leverage automation.*
>
> — Vishnu Rachakonda, **Senior Data Scientist,** firsthand in Setting Up MLOps at a Healthcare Startup (06:56)

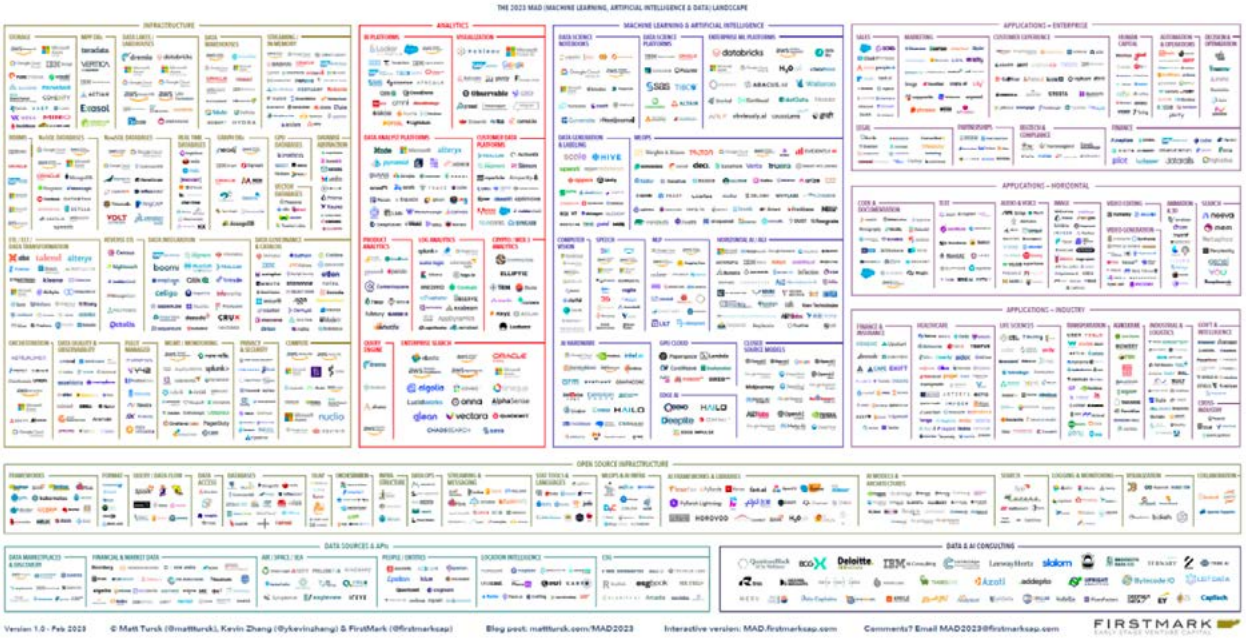
→ **Related podcast episode: Setting Up MLOps at a Healthcare Startup With Vishnu Rachakonda**

* * *

# MLOps tooling landscape

There are quite a lot of MLOps tools out there today, and most times they can turn out to be a mess because it is ridiculously hard to navigate the landscape and understand what solutions would be valuable for you and your team to adopt.

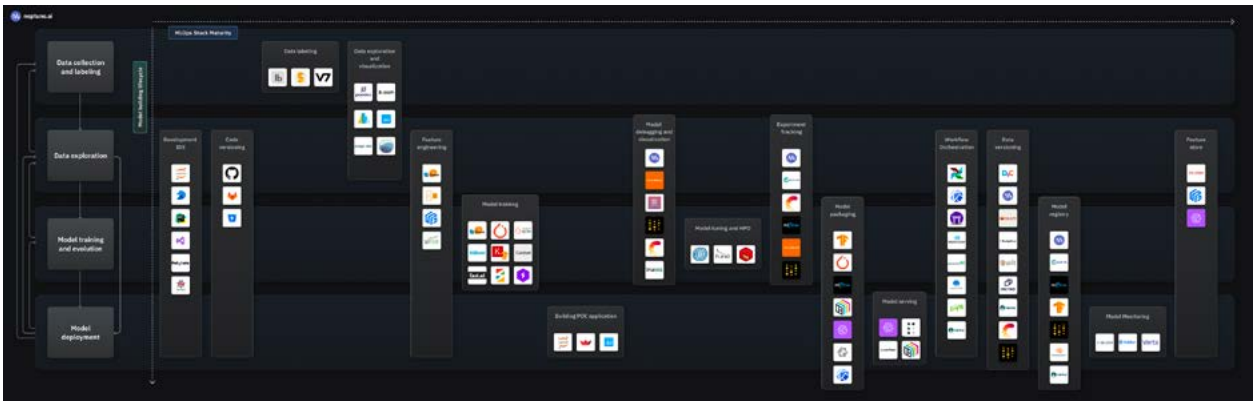
Here's an example of the ML and data tooling landscape in this 2023 review by Matt T:




The 2023 MAD (Machine Learning, Artificial Intelligence & Data) landscape by Matt Turck | See original document and interactive page

Those are a lot of tools that cover not just the scope of your ML platform, but also data components, frameworks, libraries, packages, and solutions your platform may need to integrate with !

Let's take a look at a more simplified diagram that's closer in relevancy to the ML lifecycle you'd likely based your platform on.




You can see that there are quite a number of open-source and vendor-based solutions for each component of the platform—practically a plethora of choices. So how do you make a decision on what tool to evaluate?

* * *

# Making infrastructure and tooling decisions

Your data scientists' workflow should guide the infrastructure and tooling decisions you make. You need to work with them to understand what their overall workflows are for the core business use cases they attempt to solve.

> *One of the big things that I encourage folks to think about when it comes to these ML workflow tools is, first and foremost, how open they are. Are we dealing with open-source technologies? The big reason for that is that we want to make sure that within our technology stacks, we are not going to be bound to certain things.*
> *Because processes and requirements change over time, we want to make sure that we have tools that are flexible enough to support those changes.*
>
> — Mary Grace Moesta, **Data Scientist at Databricks, in** Survey of Production ML Tech Stacks presentation at Data+Summit 2022

Here are some questions you should consider asking:

- What components would be most essential to them for developing and deploying models efficiently, in line with the principles you chose earlier?
- Can your company afford to deploy compute-intensive models over the long term?
- Can the tooling be reliable and allow you to continually innovate upon it?
- Do you want to build or buy these solutions?
- Do you use containers and have specific infrastructure needs?

- Can the tool integrate with open standard, open source, and more importantly, the existing in-house stack (including testing and monitoring ecosystems)?

You might also have noticed that some options provide end-to-end offerings, and you will see that in a section below, while others provide specialized solutions for very specific components of the workflow.

As an example, in our case at neptune.ai, rather than focusing on solving the end-to-end stack, we try to do one thing really well—manage the model development process of your data scientists well. We do this by giving your data scientists an easy-to-use platform for managing experiments and a place to store, test, and evaluate models.

Find resources below to get feature-by-feature comparison tables of MLOps tools and solutions:

- AI Infrastructure Landscape
- MLOps toys
- MLOps Community learn
- Introducing TWIML's New ML and AI Solutions Guide

Other resources to learn more about the tooling landscape perusal:

- Best MLOps Tools and How to Evaluate Them
- Machine Learning Tools articles
- Navigating the MLOps tooling landscape (Part 1: The Lifecycle)
- Navigating the MLOps tooling landscape (Part 2: The Ecosystem)
- Navigating the MLOps tooling landscape (Part 3: The Strategies)
- Exploring the ML Tooling Landscape (Part 1 of 3)
- Exploring the ML Tooling Landscape (Part 2 of 3)

- Exploring the ML Tooling Landscape (Part 3 of 3)
- What I learned from looking at 200 machine learning tools and Machine Learning Tools. Landscape v2 (+84 new tools)

## Build vs buy vs both

Like when you make decisions on any other software, the buy vs. build decision is also a significant one for ML platforms and its components. You need to be strategic on this decision because there are inevitable pros and cons to each choice that require an assessment of the current technology stack in your organization, financial analysis, and stakeholder involvement.

Below are some considerations you want to make when thinking about this decision.

### What stage is your organization at?

In the case where you are super early, perhaps you want to focus on getting to market quickly, and pulling something off the shelf to deliver value may be ideal.

But as you mature as an organization and the number of use cases expands, spending time maintaining vendor charges and services may become prohibitively expensive and may not be as intuitive as building in-house solutions. This is especially the case if you are leveraging only a portion of the vendor's offerings.

### What core business problems willing building or buying solve?

> *To be ML productive at a reasonable scalescale, you should invest your time*

> *in your core problems (whatever that might be) and buy everything else.*
>
> — **Jacopo Tagliabue in MLOps without Much Ops**

You need to consider the capabilities the platform or platform component will provide and how that will affect core business problems in your organization. What do you need to be good at compared to other things?

For example, why would you adopt an ML observability component? The goal of the solution is to prevent service outages and model deterioration. So you'd likely need to consider the relative cost of not monitoring how much money the organization could potentially lose and how it would affect the core business problems.

### What's the maturity of available tools?

There will be some situations where open source and existing vendor solutions may not be fit enough to solve the core business problem or maximize productivity. This is where you try to rank the existing solutions based on your requirements and decide if it's worth it to buy off-the-shelf solutions that meet the mark or build one internally that gets the job done.

### How much resources would it take to build the platform or its components?

Once you understand what the platform—or components of the platform—mean to your core business, the next step is to understand what resources it would take to build them if they are core to your business, or perhaps buy them if they do not solve core business problems but enable engineering productivity regardless. Resources in terms of money, time, and efforts, as well as comparable returns in the form of net present value (NPV).

Software engineering salaries are high, and so is investing time, money, and effort into building systems or components that do not directly contribute to your core business problems or give you some competitive advantage.

### What would it cost to maintain the platform or its components?

In most cases, it's not just about building or buying an ML platform but also maintaining it at the same time. Maintenance can be difficult, especially if the talents and resources involved in the project are transient and do not offer optimal long-term solutions.

You need to make sure that you are clear that whatever decision you make factors in long-term maintenance costs and efforts.

### Build vs buy — final thoughts

Why would you choose to build a component instead of buying one from the market or using an open source tool? If you find in your case that the tools do not effectively meet your engineering requirements, solve your core business problems, or maximize the productivity of your data scientists, it makes sense to focus on building an in-house solution to solve that problem.

For example, in the case of Stitch Fix, they decided to build their own model registry because the existing solutions could not meet a technical requirement.

> *The majority of model registry platforms we considered assumed a specific shape for storing models. This took the form of experiments—each*

> *model was the next in a series. In Stitch Fix's case, our models do not follow a linear targeting pattern.*
> *They could be applicable to specific regions, business lines, experiments, etc., and all be somewhat interchangeable with each other. Easy management of these dimensions was paramount to how data scientists needed to access their models.*
>
> — Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free: A Machine Learning Platform for Stitch Fix's Data Scientists

You have also seen that a lot of argument can be made about buying instead of building your ML platform, and ultimately it's left to you and your stakeholders to make the considerations we listed above.

Here's an opinion from Conor Murphy, Lead Data Scientist at Databricks, in "Survey of Production ML Tech Stacks" at the Data+AI Summit 2022.

> *At the high level, one of the core problems is that not only do we have this diversity of different tools, but also building your ML platform is expensive. Only a handful of companies are heavily investing in building in-house ML platforms. And they are frequently buggy and not optimized as they should be.*

There are also cases where it can be optimal to both build and buy components based on their relevancy to your stack and engineering productivity. For example, buying an experiment tracking tool (that seems to be a solved problem) may make more sense while you focus on building a model serving or data component that may be more ideal for your use case and not necessarily have a solution that's mature enough for the use case.

Find resources below to dig deeper into the “build vs. buy” rabbit hole:

- MLOps: What It Is, Why It Matters, and How to Implement It (Building vs buying vs hybrid MLOps infrastructure)
- Build vs. Buy an End-to-End MLOps Platform (from Valentin)
- ML Ops at Reasonable Scale feat. Jacopo Tagliabue | Stanford MLSys Seminar Episode 35

* * *

# End to end vs canonical stack ML platform

If you are wondering if you should buy an end-to-end ML platform that would seem to solve all your platform needs or build a canonical stack of point solutions based on each component, the following sections have got you covered.

Ultimately, deciding whether to get one platform or mix and match different solutions will be very context-dependent and dependent on understanding where your business is as well as your users.

There are certainly pros and cons to both approaches, and in this section, you will learn when either one could be the better one.

## Sidebar: MLOps templates

If you are thinking about how you could design your ML platform, a good practice would be to check out existing templates. This section contains a list of resources you can use to design your own ML platform and tech stack, so it gives you an idea of what you want to accomplish:

- MyMLOps: Provides a template to design your MLOps stack per component or end-to-end with a list of technologies.
- You don't need a bigger boat: The repository curated by Jacopo Tagliabue shows how several (mostly open-source) tools can be effectively combined together to run data pipelines at scale with very small teams.
- Recs at reasonable Scale: The repository contains a practical implementation of recommendation system on a reasonable scale ML platform.
- MLOps Infrastructure Stack: Provides an MLOps stack template you can use as guard rails to inform your process.

# Choosing an end-to-end ML platform

If you choose to use an open source or vendor-based solution to build your platform, there are several ML platforms that allow you to manage the workflow of your users.

There's often a stance in the MLOps community that one-size-fits-all solutions don't always work for businesses because of the varying use cases and the requirements needed to solve those use cases.

An end-to-end ML platform and the promise that you only need one solution to solve the business use case. But with ML and AI being so general, different domains (vision models, language models, etc.) with varying requirements and service level objectives mean that an end-to--end platform that is not customized to your organization and does not allow flexibility might not cut it.

If you are solving for more than one use case in a different problem domain, it might be worth considering what's most important to your organization and the resources available and asking the following questions:

- Can you build or buy a modular platform, and maintain it as opposed to an end-to-end platform?
- How do you foresee the growing number of use cases in your business?

This guide goes through the various ML platforms that every data scientist needs to know. See the table on the next page highlighting the different end-to-end ML platforms.

| Name | Short Description |
|---|---|
| Algorithmia | Securely govern your machine learning operations with a healthy ML lifecycle. |
| Allegro.io | An end-to-end enterprise-grade platform for data scientists, data engineers, DevOps, and managers to manage the entire machine learning & deep learning product life-cycle. |
| Cnvrg.io | An end-to-end machine learning platform to build and deploy AI models at scale. |
| Dataiku | Platform democratizing access to data and enabling enterprises to build their own path to AI. |
| Datarobot | AI platform that democratizes data science and automates the end-to-end ML at scale. |
| H2O | An open source leader in AI with a mission to democratize AI for everyone. |
| Iguazio | Automates MLOps with end-to-end machine learning pipelines, transforming AI projects into real-world business outcomes. |
| Kubeflow | Dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable, and scalable. |
| Pachyderm | Combines data lineage with end-to-end pipelines on Kubernetes, engineered for the enterprise. |
| Polyaxon | A platform for reproducible and scalable machine learning and deep learning on Kubernetes. |
| Valohai | Takes you from POC to production while managing the whole model lifecycle. |

Some of those platforms are closed, while others have more open standards where you can easily integrate external solutions, swap and replace tooling for specific components, and so on. There are pros and cons to using open platforms compared to closed platforms, and making

a decision will depend on factors such as how regulated the business is, security concerns, existing engineering and technology stacks, and so on.

## Closed vs open end-to-end ML platform

Closed platforms will generally provide an ecosystem that would potentially allow for seamless integrations across your entire stack but would likely confine you to that ecosystem in what's known as a "lock-in."

With open platforms, you can easily swap components and leverage open-standard tools that can provide flexibility for your team and get them to adopt and use the tool without too much overhead. For example, with Kubeflow, an open-source ML platform, you can swap out components easily, whereas that's not the case with a closed solution like DataRobot (or Algorithimia).

When it comes to a managed or open source ML platform, it all depends on the existing infrastructure as well as other build vs. buy decisions you have made. If your organization runs its workloads on AWS, it might be worth it to leverage AWS SageMaker.

* * *

# How to build an ML platform from components

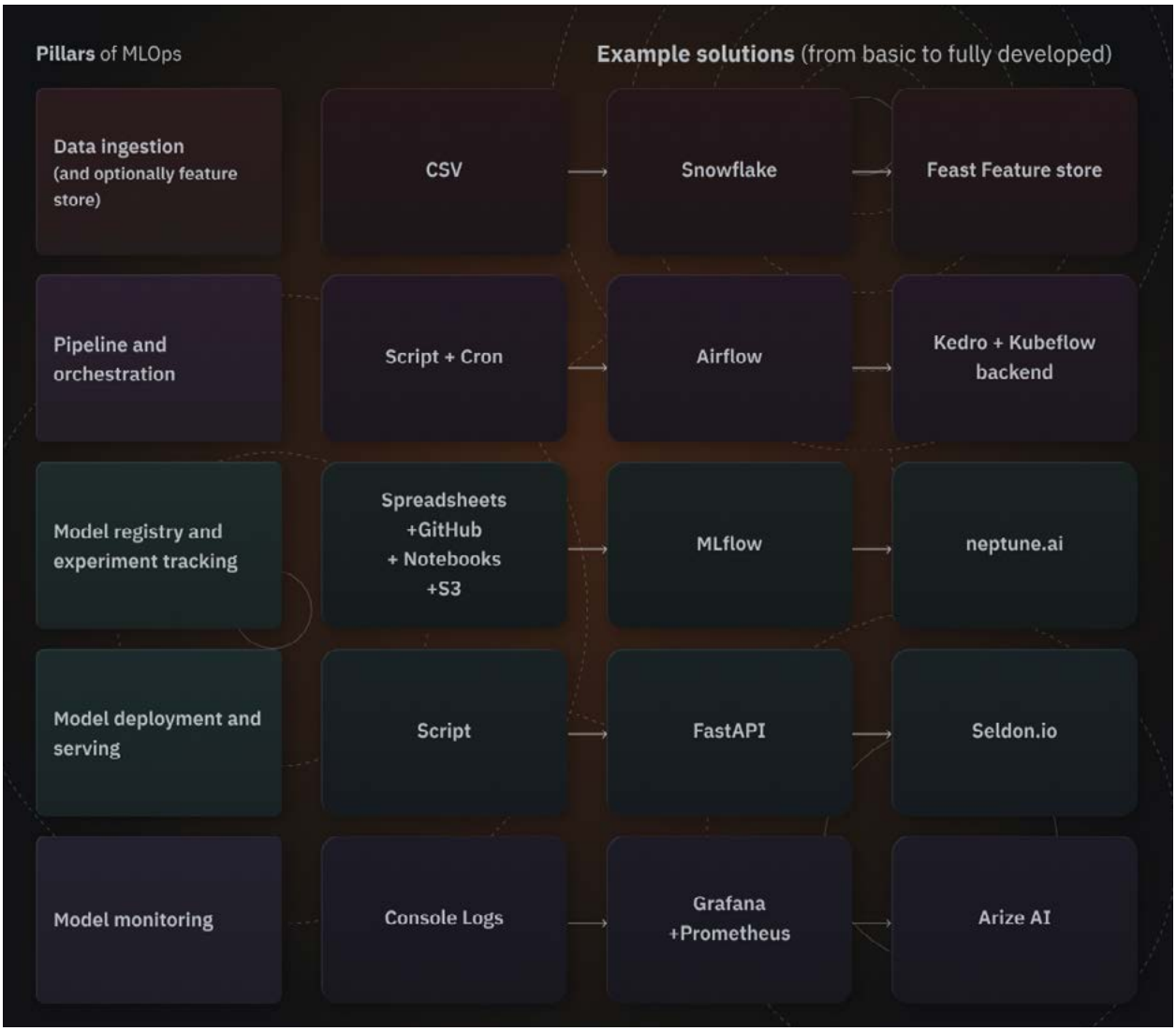
Building your machine learning platform from components is the modular way of putting together a “flexible” machine learning platform where you build or buy stuff as you go. You need to understand the pain points your data scientists and other users are facing and determine if you need that particular component.

For example, do you need a feature store if you just have one or two data scientists developing models now? Do you need a real-time serving component now? Think about what’s important to the business problem you are trying to solve and equally to your users, and start sourcing from there.

When you identify the needs, know that you may not particularly need a full-fledged solution for each component. You can decide to:

- Implement a basic solution as sort of a proof-of-concept,
- See how it solves the pain point,
- Identify gaps in your existing solution,
- Evaluate if it’s worth upgrading to a full-developed solution,
- If needed, upgrade to a fully-developed solution.

See the image on the next page that shows an example implementation of different components, from a basic solution that could be a low-hanging fruit for you and your team to a fully developed solution.




Potential implementation of the components of an ML platform | Source: Jakub Czakon in MLOps at a Reasonable Scale [The Ultimate Guide]

See other resources to learn more about this topic below:

- Building ML Pipeline: 6 Problems & Solutions [From a Data Scientist's Experience]
- Building a Machine Learning platform (by John Roberts in MLOps. community)
- The Ultimate Guide to Building a Scalable Machine Learning Infrastructure
- How to build a MLOps platform

- How to Build a ML Platform Efficiently Using Open-Source
- What I Learned Building Platforms at Stitch Fix
- Building an ML Platform from Scratch: Live Coding Session // Alon Gubkin // MLOps Meetup #67
- Building ML/Data Platform on Top of Kubernetes // Julien Bisconti // MLOps Coffee Sessions #86

* * *

# Data lakes

## Data scientists' pain point

Your data scientists don't have a secure and reliable way to get raw data to process and use for the rest of the machine learning lifecycle.

## Solution

Data lakes and warehouses are the two key components of any data pipeline. The data lake is a platform where any kind or amount of data can be stored, processed, and analyzed. Data engineers are mostly in charge of it.

Ensuring your platform can integrate with a central data lake repository is crucial. This case is very different from a feature store. It gives your data scientists a place to "dump" data that they can use to build features.

Some of the best tools for building a data lake include:

- Snowflake,
- Databricks,
- Cloudera,
- And other cloud options.

* * *

# Data labeling

## Data scientists' pain point

If the raw training data in your data lake or other storage areas doesn't have the required prepopulated labels, your data scientists may have to end their projects early or slow down their work. This is particularly true for projects with unstructured data, just as it is for projects with structured data.

In other cases, if your data scientists have access to the labels, wrong or missing labels could cause problems. They'd likely need additional labels to compensate for those data quality issues.

## Solution

Your data scientists need to add contextual labels or tags to their training data so that they can use it as a target for their experiments.

Making a decision on integrating this component into your platform depends a lot on your use case and isn't entirely black and white. In addition to the human annotator workforce, you can use platforms like:

- SuperAnnotate,
- Amazon Sagemaker Ground Truth,
- Label Studio,
- and others.

→ **Recommended: Deep Learning Guide: Choosing Your Data Annotation Tool**

# Data pipelines

## Data scientists' pain point

Your data scientists find it difficult to collect and merge raw data into a single framework where it can be processed and, if necessary, the labels converted to proper formats. In most cases, they need these steps to be automated because of the volume of data they deal with. And if they work with other teams, they'd want to make sure that modifying the preprocessing steps does not interfere with the quality of their data.

## Solution

The data pipeline encodes a sequence of steps that move the raw data from their producers to a destination: their consumers. It helps your data scientists automate the process of collecting, preprocessing, and managing their datasets.

If you want to learn more about data processing, this a comprehensive article. Some of the best tools for building data pipelines into your platform include:

- dbt,
- Snowflake,
- Airflow,
- Dagster,
- Prefect.

* * *

# Data versioning

## Data scientists' pain point

They cannot trace and audit what data was used to train what model, especially during and after running experiments and developing production-ready models. In most cases, new training data would become available, and they wouldn't be able to keep track of when the data was captured.

## Solution

The data versioning component allows your data scientists to track and manage changes to their data over time. Check out this article for the best methods and practices to do data lineage for your ML platform.

If you are interested in the best tools for data versioning and lineage, see the resources below:

- Best 7 Data Version Control Tools That Improve Your Workflow with Machine Learning Projects
- Best Data Lineage Tools

* * *

# Feature stores

Earlier in this article, you got an overview of what feature stores are. So, when should you think about incorporating a feature store into your platform? Take a look at some of the pain points your users may be experiencing.

## Data scientists' pain point

When your data scientists spend too much time trying to manage the entire lifecycle of a feature, from training to deployment and serving, you most likely need feature stores.

## Software engineers' pain point

You may also run into situations where the software engineers want to know how to integrate the prediction service without having to figure out how to get or compute features from production data.

## DevOps engineers' pain point

In some cases, your DevOps or ML engineers may want to know how to monitor and maintain a feature serving and management infrastructure without a lot of extra work or domain knowledge.

Some of the problems these personas have in common that might require you to build a feature store as part of your platform are:

- Features are hard to share and reuse across multiple teams and different use cases.

- Features are hard to reliably serve in production with low latency or batch inference use cases.
- Likely occrence of training-serving skew mostly because same features and code for data processing is needed across development and production environments, but collaboration is hard.

## Solution

If your users face these challenges, you likely need to build or buy a feature store component to compensate for them. Here are some helpful resources you can use to learn how to build or buy a feature store:

- Building a Feature Store (by Tecton.ai).
- Building a Feature Store (by Weslley Lioba Caldas).
- Building a Feature Store (by Neil Lathia).
- Data Lake Vs. Feature Store.

Tecton, Feast, Hopsworks, and others are examples of feature stores on the market today. You can read this article from the MLOps community to learn more about how to evaluate and choose feature store tools to purchase.

* * *

# Model training component

## Data scientists' pain point

Your data scientists do not understand how to set up infrastructure for their training needs; they often run out of compute during training, which slows down their workflow, and find it difficult to use tools and frameworks that make model training easier.

## Solution

The model training component of your platform allows you to abstract away the underlying infrastructure from your data scientists. It also gives them the freedom to use model training tools without worrying about setting up the training infrastructure for tasks like distributed computing, special hardware, or even managing the complexity of containerization.

If you want to learn about the different libraries and frameworks popular among your data scientists for model training, check out the resources below:

- Top 10 Best Machine Learning Tools for Model Training
- The Best ML Frameworks & Extensions for Scikit-learn
- The Best ML Frameworks & Extensions for TensorFlow
- PyTorch Lightning vs Ignite: What Are the Differences?
- XGBoost: Everything You Need to Know
- XGBoost vs LightGBM: How Are They Different
- When to Choose CatBoost Over XGBoost or LightGBM [Practical Guide]

→ **Related article: Distributed Training: Frameworks and Tools**

# Hyperparameter optimization

## Data scientists' pain point

For your data scientists, the search for the right hyperparameter combination takes a long time and consumes many hours of computational resources.

## Solution

You'd need to integrate a hyperparameter optimization tool as part of your platform to help your data scientists fine-tune parameters and evaluate the effects of different combinations to make their workflow efficient and save compute resources.

Learn more about hyperparameter optimization using the resources below:

- Hyperparameter Tuning in Python: a Complete Guide.
- Kedro Pipelines With Optuna: Running Hyperparameter Sweeps.

Below are some resources for you to learn more about hyperparameter optimization tools:

- Best Tools for Model Tuning and Hyperparameter Optimization
- Scikit-Optimize: Bayesian Hyperparameter Optimization in Python
- Optuna Guide: How to Monitor Hyper-Parameter Optimization Runs
- Keras Tuner: Lessons Learned From Tuning Hyperparameters of a Real-Life Deep Learning Model
- Optuna vs Hyperopt: Which Hyperparameter Optimization Library Should You Choose?

# Experiment tracking, visualization and debugging

## Data scientists' pain point

They run a lot of experiments, every time on different use cases with different ideas and parameters, but they find it difficult to:

- Organize and group the experiments they run,
- Share results with each other,
- Reproduce the results of a past experiment,
- And generally save all experiment-related metadata.

## Solution

If you are trying to figure out the best tools for experiment tracking and metadata storage, here are some resources:

- 15 Best Tools for Tracking Machine Learning Experiments
- Best Tools to Log and Manage ML Model Building Metadata
- Best ML Metadata Store Solutions
- Tools Metadata storage and management (by MLOps.community)
- How to Compare ML Experiment Tracking Tools to Fit Your Data Science Workflow (by DagsHub)

* * *

# Model training operationalization

## Data scientists' pain point

They develop models, deploy them, and a few weeks to months later, they realize the model performs poorly compared to when it was deployed and would need to retrain it without breaking operations in production.

## Solution

Most of the time, your data scientists will develop and deploy models that will need to be retrained over and over. This is where the training operationalization component comes in.  Operationalization may also encompass the development of the training workflow and the components required to successfully deploy a model later in the lifecycle, such as inference scripts or serving to pre-process and post-process components.

Here are some valuable resources to learn about retraining models, training operationalization, and how others implement them:

- Retraining Model During Deployment: Continuous Training and Continuous Testing
- Automatic retraining for machine learning models: tips and lessons learned (by the team at Nubank)

* * *

# Configuration management

## Data scientists' pain point

They don't have an organized way of documenting model training parameters, model and data schema, instructions for training jobs, environment requirements, training tasks, and other metadata that are repeatable and heavily involved in the model development process.

## Solution

The configuration management component enables your data scientists to manage model training experiments. In most cases, they help you build training operationalization into your platform, because your users can use a configuration file like YAML, or a "configuration as code" principle to building their training pipelines and workflow schedulers.

Some examples of configuration management tools include Hydra and Pydantic. This comprehensive article delves into both tools and explains their features.

* * *

# Source control repository

## Data scientists' pain point

They cannot effectively collaborate on code in a central location within the team and across different teams. For example, sharing notebooks, code, tests, and training configuration files to other teammates, teams, and downstream systems (CI/CD pipelines) for hand-off, so the model training or deployment process can be initiated.

## Solution

Consider adopting a source control repository like GitHub, for example and provide standard templates your data scientist can use to bootstrap their projects, commit code, notebooks, and configuration files. See how Shopify implemented their in their ML platform Merlin:

> *From the user's perspective, they use a mono repository that we set up for them. Under that repository, each Merlin project gets its own folder. That folder will contain the source code for the use case, the unit tests, and the configuration that is inside a config.yml file.*
> *Once the user pushes their code to a branch or merges it into the main branch, automated CI/CD pipelines will create Docker images for each project from the artifacts. So they can use that to create the Merlin workspace.*
>
> — **Isaac Vidas, ML Platform Lead at Shopify in Ray Summit 2022**


Adapted from "The Magic of Merlin – Shopify's new machine learning platform" at Ray Summit 2022

* * *

# Model registry

## Data scientists' pain point

Your data scientists running lots of experiments, developing and deploying many models, and working with cross-functional teams may find that managing the lifecycle of these models is often a challenging process.

While the management might be possible with one or a few models, in most cases you will have a lot of models running in production and servicing different use cases.

They might eventually loose track of a model that was deployed a few months ago and needs to be retrained, audited, or "retired", but cannot find the associated metadata, notebooks, and code to go along. And perhaps the project owner already left the company.

## Solution

A model registry will bring the following benefits to your ML platform:

- Faster deployment of your models.
- Simplifies model lifecycle management.
- Production model governance.
- Improve model security by scanning for vulnerabilities, especially when a large number of packages are used to develop and deploy the models.

This is a really central and important component of an ML platform and needs more attention. You can understand more about model registries

in this comprehensive guide. If you are looking to build this component, you can take a page from Opendoor Engineering's approach in this guide.

Considering buying or using an existing open source solution? Some options to consider include neptune.ai, MLflow, and Verta.ai. This article delves into some of the ideal solutions on the market.

* * *

# Model serving and deployment

## Data scientists' pain point

Your data scientists have multiple models that require multiple deployment patterns given the use cases and features they support. The challenge is to consolidate all of these patterns into a single effort without lots of engineering overhead.

## Solution

Incorporate a model serving component that makes it easy to package models into web services and incorporate the major inference serving patterns:

- **Batch inference:** They need the ML service to run batch jobs periodically on production data and store pre-computed results in a database.
- **Online inference:** They need the ML service to provide real-time predictions whenever clients sends input features as requests.
- **Stream inference:** They need the ML service to consume feature requests in a queue, compute predictions in real-time, and push the predictions to a queue. This way, clients watch the predictions queue in real-time and read the results asynchronously (whenever they are available). This would help them manage real-time traffic and handle load spikes.

Below are some resources to learn more about solving this component:

- 5 Model Deployment Mistakes That Can Cost You a Lot

- Serving Machine Learning Models With Docker: 5 Mistakes You Should Avoid
- How to Serve Machine Learning Models with TensorFlow Serving and Docker
- Model Deployment Challenges: 6 Lessons From 6 ML Engineers
- Deploying ML Models: How to Make Sure the New Model Is Better Than the One in Production? [Practical Guide]
- How to Deploy NLP Models in Production
- Deploying Computer Vision Models: Tools & Best Practices

Here are some of the best tools for model serving:

- Best 8 Machine Learning Model Deployment Tools That You Need to Know.
- Best Tools To Do ML Model Serving.

* * *

# POC applications

## Data scientists' pain point

They want to quickly deploy their models into an application that clients can interface with and see what the results look like without too much engineering overhead.

## Solution

When building your platform, ensure you can integrate third-party tools like Streamlit and Gradio that provide interfaces for users to interact with your models even when they have not been fully deployed.

* * *

# Workflow management

## Data scientists' pain point

Your data scientists are working on different use cases, and they find it challenging to manage the workflows for each use case.

## Solution

To tackle the issue of workflow management, you can buy or build a workspace management component so each use case can have its own dedicated component. An example of this would be how the platform team for Shopify's Merlin decided to create and distribute computation workspaces for their data scientists:

> *We coined the term "Merlin workspace"... Each use case can have a dedicated Merlin workspace that can be used for the distributed computation that happens in that Ray cluster. This Merlin workspace can contain both the resource requirements on the infrastructure layer and the dependencies and packages required for the use case on the application layer.*
>
> — Isaac Vidas, **ML Platform Lead at Shopify**

* * *

# Workflow orchestration

## Data scientists' pain point

They manually move their models across each stage of their workflows, and that can be really unproductive for tasks that they deem mundane.

## Solution

You need a workflow orchestration component to add logic to your training and production pipelines in the platform and determine which steps will be executed and what the outcome of those steps will be.

Orchestrators make it possible for model training, evaluation, and inference to be done automatically. All of these things are important because your platform needs to be able to schedule workflow steps, save outputs from the steps, use resources efficiently, and notify project owners if a workflow breaks.

Find resources to learn more about workflow orchestration and pipeline authoring below:

- Building Machine Learning Pipelines: Common Pitfalls
- Building and Managing Data Science Pipelines with Kedro
- Reducing Pipeline Debt With Great Expectations
- Differences Between Shipping Classic Software and Operating ML Models With Simon Stiebellehner (40:20)
- MLOps is an extension of DevOps. Not a fork
- Best Practices When Working With Docker for Machine Learning

Below are the best tools for workflow and pipeline orchestration on your ML platform:

- Best Workflow and Pipeline Orchestration Tools – Machine Learning Guide
- Kedro vs ZenML vs Metaflow: Which Pipeline Orchestration Tool Should You Choose?
- Argo vs Airflow vs Prefect: How Are They Different

* * *

# CI/CD pipelines

## Data scientists' pain point

Your data scientists are often involved in either manually taking models, packaging them, and deploying them to the production environments or just handing off the artifacts to you or the DevOps engineers once they have built and validated the model.

At the same time, whenever they make changes to notebooks to re-build models or update them, they go through the same cycle manually.

## Solution

CI/CD pipelines automate the process of building and rebuilding model packages into executable files that can be automatically deployed to production. Of course, automation may not be possible in all circumstances, as it would depend on the needs of the team and the nature of the business. The goal for you is to identify those manual tasks and see if automating them would not impact the business negatively.

You have already learned this component earlier in this guide. Even though CI/CD is unique to traditional software engineering, some of the ideas can be used in machine learning (ML). You should choose tools and approaches that can at least achieve the following:

- build the model package,
- Move it to a test environment,
- deploy it to a production environment,
- and rollback, if necessary.

Find resources to learn about this topic below:

- Continuous Integration and Continuous Deployment (CI/CD) Tools for Machine Learning
- How to Build MLOps Pipelines with GitHub Actions [Step by Step Guide]

* * *

# Model testing

## Data scientists' pain point

Your data scientists often feel the need to leverage another test environment similar to the production environment to be sure they are deploying the optimal model, but they cannot seem to find the resources to do so.

## Solution

The model testing component would provide a testing environment for your data scientists to test their models on live data without interruptions to the production environment and also test on reference data to determine the segments of data the model does not perform well on or has poor quality in.

This article highlights the 5 tools you could use to solve this component in your platform.

Learn more about leveraging this component using the resources below:

- Automated Testing in Machine Learning Projects [Best Practices for MLOps]
- How to Test a Recommender System
- Build a culture of ML testing and model quality

* * *

# Model monitoring platform

## Data scientists' pain point

They have no visibility into the models they have deployed to production. In some cases, because of the dynamic nature of the datasets involved in production and training, there would be some drift in the distribution of the data and, as a result, some concept drift in the underlying logic of the model they have built and deployed.

## Solution

Implement the observability component with your platform so that the platform, ML service, and model behaviors can be tracked, monitored, and necessary action taken to successfully operationiize the models.

According to Piotr Niedzwiedz's article, the CEO of neptune.ai, you need to be ask the following questions before enabling the continuous and automated monitoring component:

- What do you know can go wrong, and can you set up health checks for that?
- Do you even have a real need to set up those health checks?

Some of the best tools to do ML model monitoring include WhyLabs, Arize AI, Fiddler AI, Evidently, and so forth. You can compare the features of these tools on this MLOps community page.

* * *

# Getting internal adoption of your ML platform

Working with your users to build and test your platform may not be a challenge; on the other hand, gaining internal adoption might be. To understand how you can optimally get internal adoption, you need to understand the challenges they might encounter moving workloads to your ML platform by asking the right questions.

Questions like:

- How can you get your leadership to support a journey to an ML platform visile and meaningfully?
- How will you break down any silos between the business, platform, and other supporting functions?
- What new skills and capabilities will your users need to make the most of the platform?
- How will you win the hearts and minds of those at your organization who are skeptical of the ML platform?

## Leadership: how can you get your leadership to support a journey to an ML platform visile and meaningfully?

Anything that introduces significant organizational changes requires buy-in and adoption from internal and external stakeholders. You will mostly find yourself pitching the importance of the investment in the ML platform to your stakeholders and the entire team.

Buy-in comes in many forms and shapes, such as management approval for budgeting towards developing the platform, creating ownership

in your data and ML team towards the change, and making stakeholders understand the platform's value proposition to the business and engineering productivity.

Getting buy-in does not necessarily mean getting 100% agreement with your vision; it is about having the support of your team, even if they don't wholly agree with you.

## Collaboration: how will you break down any silos between the business, platform, and other supporting functions?

One of the things that makes ML platforms work is that they let teams from different departments work together. You need to communicate clearly how the platform will enable teams to collaborate more effectively and begin to build an "MLOps culture" within the team.

## Capabilities: what new skills and capabilities will your users need to make the most of the platform?

If you have built the platform right, moving workloads and using it should not cause your users too much overhead because the platform is meant to complement their existing skills, or at least standardize them, and not cause them to become obsolete.

You need to be clear on what new skills they need, if any, and properly provide documentation and tutorials for them to leverage those skills.

## Engagement: how will you win the hearts and minds of those at your organization who are skeptical of the ML platform?

In every organization, there are always skeptics who are resistant to change. But the good thing is that in most cases, when you identify the skeptics early on, build the platform, and test it with them, there are chances that you can get them to experiment with the platform and ultimately integrate it.

Your organization and users can only realize the full potential of the ML platform if they:

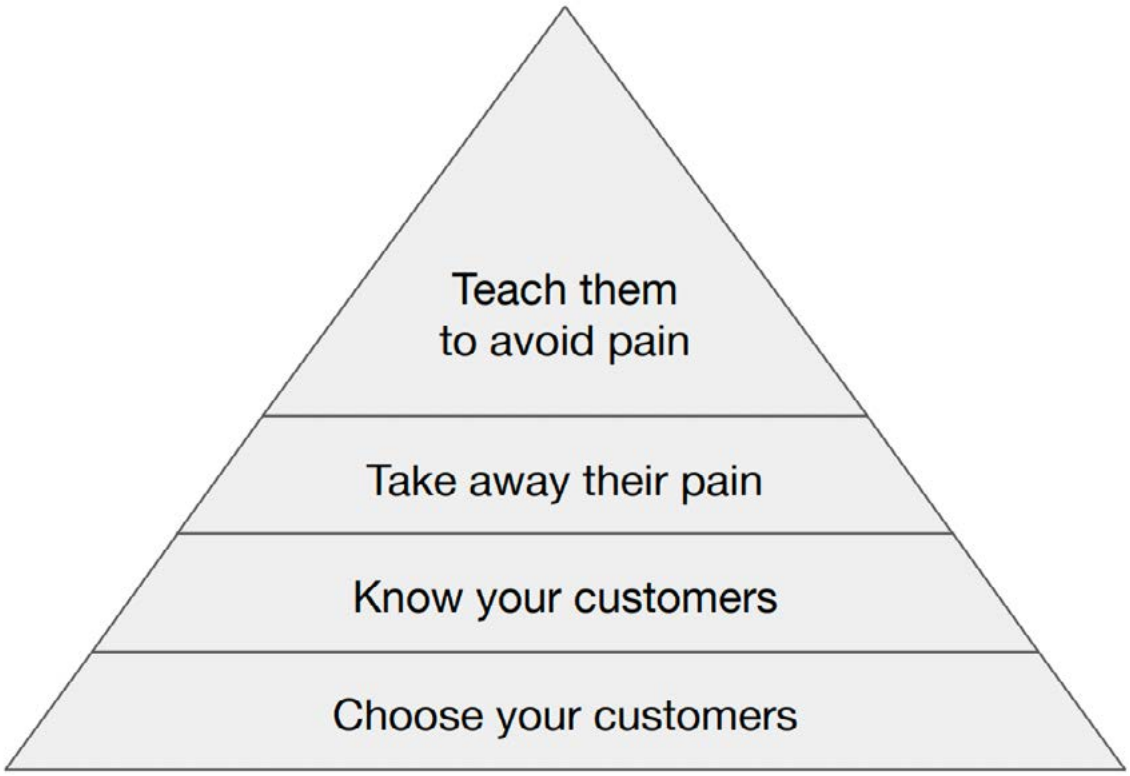
- **Know how to use it;** when you make it operationally simple for them,
- **Know how to make the most of it;** when you make it intuitive to adopt, providing good documentation and tutorials,
- **Are empowered and supported to do things differently than before;** the platform is framework-agnostic and gives them the flexibility to use what tools they are familiar with.

> *The platform we built powers 50+ production services, has been used by 90 unique users since the beginning of the year, and powers some components of tooling for nearly every data science team... By providing the right high-level tooling, we've managed to transform the data scientist experience from thinking about microservices, batch jobs, and notebooks toward thinking about models, datasets, and dashboards.*
>
> — Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free: A Machine Learning Platform for Stitch Fix's Data Scientists

## A real-world case study of gaining internal adoption: Metaflow

To bring those points to life here's a brief and case study of how Metaflow creators managed to get people at Netflix to use their platform.

As told by Julie Amundson during YOW! 2022, Metaflow adopted a hierarchical approach that had customer obsession as its main focus.




Metaflow's Customer Obsession in Action | Source

They had four guiding principles:

1. **Choose your customers:** For Metaflow, this part was easy for obvious reasons, as Data Scientists and ML Engineers were assumed to be the ones who would interact with their tool the most. At the same time, they also kept in mind who were not their customers – Data Engineers, Software Engineers, etc. so as to avoid making any critical decisions based on the feedback of people who will not be using their tool.
2. **Know your customers:** This part focused on having as many conversations as possible with the customers identified in Step

1 to zoom in on specific pain points and weed out generic noise. Once these points were narrowed down, Metaflow paired up their engineers with folks experiencing these difficulties to see these problems firsthand in an end-to-end fashion and gain a deep understanding of their users' world.

3. **Take away their pain:** Knowledge gained in Step 2 allowed them to design effective features and solutions to the identified pain points. Special care was taken to not introduce any additional pains while taking the existing ones away.
4. **Teach them to avoid pain:** The last thing they needed was to teach their user base about these features and the best way around their problems. They essentially treated this education as a part of the product itself and deployed dedicated Slack channels for support and docs in the form of articles and tutorials. The last mile here was completed by word-of-mouth from their users themselves.

**How did Metaflow engineers know they succeeded?**

People at Metaflow knew their tool was having an impact based on the following two heuristics:

1. **Qualitative Metrics:** Strong feedback in the form of buzz and word-of-mouth from industry professionals and leaders gave them confidence that their tool was making strides all around.
2. **Quantitative Metrics:** They had an internal dashboard that kept track of how many new users signed-up, what was the level of adoption in a certain team, and which primary purpose a certain team was using their tool for.

* * *

# MLOps best practices, learnings, and considerations from ML platform experts

We have taken some of the best practices and learnings from the ML platform teams and consolidated them into the following points:

- Embrace iterating on your ML platform.
- Be transparent to your users about true infrastructure costs.
- Documentation is important on and within your platform.
- Tooling and standardization are key.
- Be tool agnostic.
- Make your platform portable.

### Embrace iterating on your ML platform

Like any other software system, building your ML platform should not be a one-off thing. As your business needs, infrastructure, teams, and team workflows evolve, you should keep making changes to your platform.

At first, you likely won't have a clear vision of what your ideal platform should look like. By building something that works and consistently improving it, you should be able to build a platform that supports your data scientists and provides business value.

Isaac Vidas, ML Platform Lead at Shopify, shared at Ray Summit 2022 that Shopify's ML Platform had to go through three different iterations:

> *Our ML platform has gone through three iterations in the past. The first iteration was built on an in-house PySpark solution. The second iteration was built as a wrapper around the Google AI Platform (Vertex AI), which ran as a managed service on Google Cloud.*

> *We reevaluated our machine learning platform last year based on various requirements gathered from our users and data scientists, as well as business goals. We decided to build the third iteration on top of open source tools and technologies around our platform goals, with a focus on scalability, fast iterations, and flexibility.*

Take, for example, Airbnb. They have built and iterated on their platform up to three times over the course of the entire project. The platform should evolve as the number of use cases your team solves increases.

### Be transparent to your users about true infrastructure costs

Another good idea is to make sure that all of your data scientists can see the cost estimate for every job they run in their workspace. This could help them learn how to better manage costs and use resources efficiently.

> *We recently included cost estimations (in every user workspace). This means the user is very familiar with the amount of money it takes to run their jobs. We can also have an estimation for the maximum workspace age cost, because we know the amount of time the workspace will run...*
>
> — Isaac Vidas, **ML Platform Lead at Shopify, in** Ray Summit 2022

### Documentation is important on—and within—your platform

Like any software, documentation is extremely important—documentation on the platform and within the platform. The documentation on the platform should be intuitive and comprehensive so that your users can find it easy to use and adopt.

You can be clear on parts of the platform that are yet to be perfected and make it easy for your users to differentiate between errors due to their own workflows and those of the platform.

Quick-start guides and easy-to-read how-tos can contribute to the successful adoption of the platform. Within the platform as well, you should make it easy for your users to document their workflows. For example, adding a notes section to the interface for the experiment management part could help your data scientist a lot.

Documentation should start right from the architecture and design phases, so you can:

- Have complete design documents explaining all the moving parts in the platform and constraints specific to ML projects.
- Perform regular architectural reviews to fix weak spots and make sure everyone is on the same page with the project.

### Tooling and standardization are key

When you standardize the workflows and tools for them on your platform, you can make the team more efficient, use the same workflows for multiple projects, make it easier for people to develop and deploy ML services, and, of course, make it easier for people to work together. Learn more from Uber Engineering's former senior software engineer, Achal Shah.

### Be tool agnostic

When your platform can integrate with the organization's existing stack, you can help cross-functional teams adopt it faster and work together more proactively. Users would not have to completely relearn new tools

just to move to a platform that promises to “improve their productivity.” If they have to do that from the start, it will almost certainly be a lost cause.

### Make your platform portable

In the same vein, you should also build your platform to be portable across different infrastructures. There may be cases where your platform runs on the organization’s infrastructure layer, and if you build a platform that is native to that infrastructure, you might find it hard to move it to a new one.

Most open-source, end-to-end platforms are portable. You can build your own platform using their design principles as a guide, or you can just use their solutions wherever you can.

Here are some places you can go to learn more about these design principles and best practices:

- MLOps: 10 Best Practices You Should Know
- 5 Must-Do Error Analysis Before You Put Your Model in Production
- Tips for MLOps Setup—Things We Learned From 7 ML Experts
- Your First MLOps System: What Does Good Look Like? With Andy McMahon
- Three Principles for Designing ML-Powered Products | Spotify Design

* * *

# Examples of real-world internal ML platforms

Below, you will find examples of companies and developers that build ML platforms and MLOps tool stacks, so you can take some inspiration from their practices and approaches:

- MLOps at GreenSteam: Shipping Machine Learning [Case Study]
- Setting up a Scalable Research Workflow for Medical ML at AILS Labs [Case Study]
- This Is Our MLOps Tool Stack: Continuum Industries
- Deployment of Data and ML Pipelines for the Most Chaotic Industry: The Stirred Rivers of Crypto
- How These 8 Companies Implement MLOps – In-Depth Guide
- Building a Modern Machine Learning Platform on Kubernetes | Lyft
- Bighead: Airbnb's end-to-end Machine Learning Platform | Airbnb
- Building a Machine Learning platform (Lemonade)
- 3 Principles for Building an ML Platform That Will Sustain Hypergrowth (Doordahs)
- MLOps Architecture: Building your MLOps Pipeline (MODZY)
- Building Voiceflow's Machine Learning Platform From Scratch
- Building Neoway's ML Platform with a Team-First Approach and Product Thinking
- Building ML Platform in Retail and eCommerce
- Using MLOps to Build a Real-time End-to-End Machine Learning Pipeline (Binance)
- Setting up an MLOps pipeline with Microsoft Azure
- Laying the foundation of our open source ML platform with a modern CI/CD pipeline

- The Road to a Serverless ML Pipeline in Production — Part I
- An End-to-End MLOps Platform Implementation using Open-source Tooling

You can also check out this list of ML platform implementations (kudos to Evidently for creating this list!)

## What is next?

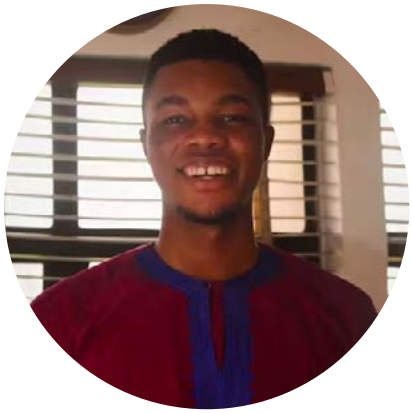
Essentially, go build out that ML platform! If you have problems:

→ **Reach out to us or in the MLOps community (#neptune-ai)**

→ **Subscribe to our podcast**

* * *

# About the author



**Stephen Oladele** - is a Developer Advocate and an MLOps Technical Content Creator at Encord. He has significant experience building and managing data communities, and you will find him learning and discussing machine learning topics across Discord, Slack and Twitter.

Stephen has a background in physical sciences and has worked on a range of machine learning problems from computer vision to recommendation engines, with a focus on accessibility across industries such as agriculture, education, and e-commerce.

In his free time, Stephen loves learning, traveling, spending quality time with my family and friends and volunteering for non-profit causes that make a valuable impact.

→ **Twitter:** https://twitter.com/nerdcyberartist

→ **LinkedIn:** https://www.linkedin.com/in/stephenoladele/

neptune.ai

Did you like this article?

## See more on our MLOps Blog:

https://neptune.ai/blog

See also:

/c/NeptuneAI

/neptune_ai

/company/neptuneai