

O'REILLY®

Implementing MLOps

A Production-First Approach



**Early
Release**

**RAW &
UNEDITED**

Yaron Haviv
& Noah Gift

Implementing MLOps in the Enterprise

1ST EDITION

A Production-First Approach

Noah Gift and Yaron Haviv



Beijing • Boston • Farnham • Sebastopol • Tokyo

Implementing MLOps in the Enterprise

by Yaron Haviv and Noah Gift

Copyright © 2022 Yaron Haviv and Noah Gift. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc. , 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com .

- Editors: Corbin Collins and Nicole Butterfield
- Production Editor: FILL IN PRODUCTION EDITOR
- Copyeditor: FILL IN COPYEDITOR
- Proofreader: FILL IN PROOFREADER
- Indexer: FILL IN INDEXER
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Kate Dullea
- August 2023: First Edition

Revision History for the First Edition

- 2022-09-13: First Early Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781098136581> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. Implementing MLOps in the Enterprise, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author(s) and do not represent the publisher's views. While the publisher and the author(s) have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author(s) disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-13658-1

[FILL IN]

Preface

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This element signifies a tip or suggestion.

NOTE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at https://github.com/oreillymedia/title_title.

If you have a technical question or a problem using the code examples, please send email to bookquestions@oreilly.com.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Book Title* by Some Author (O'Reilly). Copyright 2012 Some Copyright Holder, 978-0-596-xxxx-x."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

O'Reilly Online Learning

NOTE

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit <https://oreilly.com>.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway North
- Sebastopol, CA 95472
- 800-998-9938 (in the United States or Canada)
- 707-829-0515 (international or local)
- 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at.

Email bookquestions@oreilly.com to comment or ask technical questions about this book.

For news and information about our books and courses, visit
[*https://oreilly.com*](https://oreilly.com).

Find us on LinkedIn: [*https://linkedin.com/company/oreilly-media*](https://linkedin.com/company/oreilly-media)

Follow us on Twitter: [*https://twitter.com/oreillymedia*](https://twitter.com/oreillymedia)

Watch us on YouTube: [*https://www.youtube.com/oreillymedia*](https://www.youtube.com/oreillymedia)

Acknowledgments

Chapter 1. MLOps - What is it, and why do we need it?

At the root of inefficient systems is an interconnected web of incorrect decisions that compound over time. It is tempting to look for a silver bullet fix to a system that doesn't perform well, but that strategy rarely, if ever, pays off. Let's look at the human body; there is no shortage of quick fixes sold to make you healthy, but the solution to health longevity is more complex because it requires a systematic approach¹. We want to hear that a magic pill will make us healthy instead of developing habits that lead to optimal health.

Similarly, there is no shortage of advice on "getting rich quick." Here again, the data conflicts with what we want to hear. In the book "Don't Trust Your Gut (HarperCollins, 2022)", Seth Stephens-Davidowitz shows that 84% of the top .1 percent receive at least some money from owning a business. Further, the average age of a business founder is about 42, and some of the most successful companies are real estate or automobile dealerships. These are hardly get-rich-quick businesses but businesses that require significant skill, expertise, and wisdom through life experience

Cities are another example of complex systems that don't have silver bullet fixes. Recently WalletHub created a list of best-run cities in America. San Francisco ranked 149 out of 150 despite having many theoretical advantages over other cities, like beautiful weather, being home to the top tech companies in the world, and a 2022-2023 budget of 14 billion dollars for a population of **842 thousand people**. The budget is similar to the entire country of Panama, with a population of **4.4 million people**. As the case of San Francisco shows, revenue or natural beauty alone isn't enough to have a well-run city; there needs to be a comprehensive plan, i.e., execution and strategy matter. No single solution is going to either make or break a city. The WalletHub survey points to extensive criteria for a well-run city,

including infrastructure, economy, safety, health, education, and financial stability.

Similarly, with MLOps, searching for a single answer to getting models into production, perhaps by getting better data or using a specific deep learning framework, is tempting. Instead, just like these other domains, it is essential to have an evidence-based, comprehensive strategy. Next, let's discuss the foundations of MLOps and how to use it effectively to get models into production.

What is MLOps?

At the heart of MLOps is the continuous improvement of all business activity. The Japanese automobile industry refers to this concept as Kaizen, meaning literally “improvement.” For building production machine learning systems, this manifests itself in both the noticeable aspects of improving the model's accuracy as well the entire ecosystem supporting the model.

A great example of one of the non-obvious components of the machine learning system is the business requirements. If the company needs an accurate model to predict how much inventory to store in the warehouse, but the data science team creates a computer vision system to keep track of the inventory already in the warehouse, the wrong problem is solved. No matter how accurate the inventory tracking computer vision system is, the business asked for a different requirement, and it cannot meet the goals of the organization as a result.

So what is *MLOps*? A compound of *Machine Learning* (ML) and *Operations* (Ops), MLOps is the processes and practices for designing, building, enabling, and supporting the efficient deployment of ML models in production, to continuously improve business activity. Similar to *DevOps*, MLOps is based on automation, agility, and collaboration to improve quality. If you're thinking CI/CD, you're not wrong. MLOps supports CI/CD. According to Gartner, “MLOps aims to standardize the deployment and management of ML models alongside the operationalization of the ML pipeline. It supports the release, activation,

monitoring, performance tracking, management, reuse, maintenance, and **governance of ML artifacts.**”

MLOps in the Enterprise

There are substantial differences between an enterprise company and a startup company. Scott Shane, an entrepreneurship expert, wrote in “The Illusions of Entrepreneurship (Yale University Press, 2010)” “that only one percent of people work in companies less than two years old, while 60 percent work in companies more than ten years old”. Longevity is a characteristic of the enterprise company.

He also says, “it takes 43 startups to end up with just one company that employs anyone other than the founder after ten years”. In essence, the enterprise builds for scale and longevity. As a result, it is essential to consider technologies and services that support these attributes.

NOTE

Startups have technological advantages for users, but they also have different risk profiles for the investors versus the employees. Venture capitalists have a portfolio of many companies diversifying their risk. According to **FundersClub**, a typical fund “contains 135 million” and is “spread between 30-85 startups”. Meanwhile, startup employees have their salary and equity invested in one company.

Using the **expected value** to generate the actual equity value at a probability of 1/43, an enterprise offering a yearly 50k bonus returns 200k at year four. A startup produces \$4,651.16 in year four. For most people, on average, startups are a risky decision if judged on finance alone. However, they might offer an excellent reward via an accelerated chance to learn new technology or skills with the slight chance of a huge payout.

On the flip side, if a startup’s life is dynamic, they must pick very different technology solutions than the enterprise. If there is a 2.3% chance a startup will be around in 10 years, why care about vendor lock-in or multi-cloud deployment? Only the mathematically challenged startups build what they don’t yet need.

Likewise, if you are a profitable enterprise looking to build upon your existing success, consider looking beyond solutions that startups use. Other metrics like the ability to hire, enterprise support, business continuity, and price become critical KPIs (Key Performance Indicators). Next, discuss how KPIs can drive ROI (Return on Investment).

Understanding ROI in Enterprise Solutions

The appeal of a “free” solution is that you get something for nothing. In practice, this is rarely the case. In **Figure 1-1** there are three scenarios presented. In the first scenario, the solution costs nothing but delivers nothing, so the ROI is zero. In a second scenario, there is high value at stake, but the cost exceeds the value resulting in a negative ROI. In the third scenario, a value of one million with a cost of 1/2 million delivers 1/2 of a million in value.

Evaluating Technology Platform Solutions		
#1 No Value/No Cost/No ROI	#2 High Value/High Cost/Negative ROI	#3 High Value/High Cost/Negative ROI
Value = 0	Value = 1 Million	Value = 1 Million
Cost = 0	Cost = 2 Million	Cost = 1/2 Million
ROI = 0	ROI = -1 Million	ROI = 1/2 Million

Figure 1-1. Evaluating Technology Platform Solutions ROI

The big takeaway is the cost alone shouldn't be the driver of technology platform solutions. The best choice isn't free, but the solution that delivers the highest ROI since this ROI increases the velocity of the profitable enterprise. Let's expand on the concept of ROI even more by digging into

bespoke solutions, which in some sense are also “free” since an employee built the solution.

In **Figure 1-2**, a genuinely brilliant engineer convinces management to allow them to build a bespoke system that solves a particular problem for the Fortune 100 company. Despite skepticism from some people, the engineer not only delivers quickly, but the system exceeds expectations. It would be tempting to think this is a success story, but it is actually a story of failure. One year later, the brilliant engineer gets a job offer from a trillion-dollar company and leaves. About three months later, the system breaks, and no one is smart enough to fix it. The company reluctantly replaces the entire system and retrain the company on the new proprietary system.

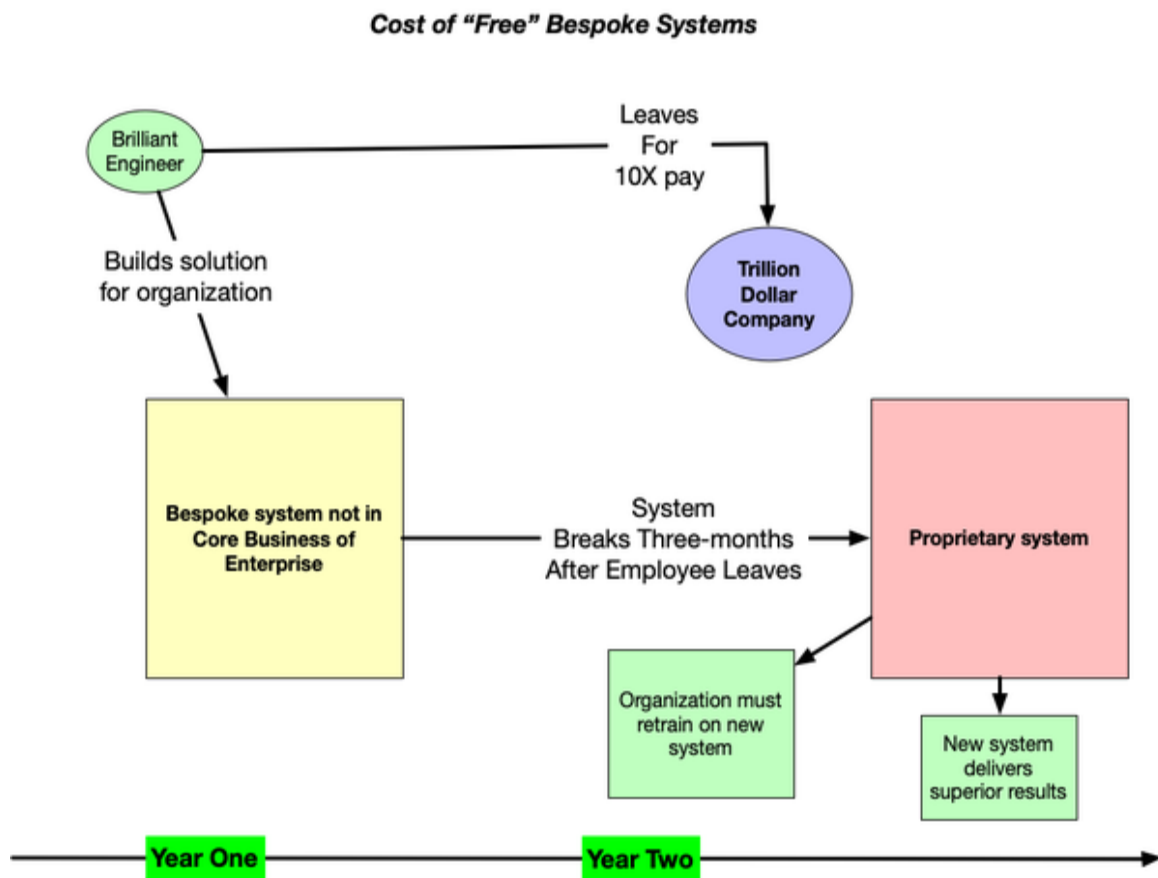


Figure 1-2. Bespoke System Dilemma

The ultimate cost to the organization is the lack of momentum from using a superior system for a year, alongside the training time necessary to switch from the old system to the new system. A “free” solution with positive ROI

can have long-term negative ROI for an organization. This scenario is not only not hypothetical, but you may have seen it yourself².

In the book *Fooled by Randomness: The Hidden Role of Chance in Life and the Markets* (Random House, 2008), the author Nassim Taleb mentions, “it does not matter how frequently something succeeds if failure is too costly to bear.” This statement directly applies to a successful enterprise that wants to implement MLOps. Taking the right kind of strategic risk is of critical importance. In the following section, let’s discuss the concept of risk in more detail.

Understanding Risk and Uncertainty in the Enterprise

Not all risk is the same, just as not all uncertainty is the same. Unlike a startup, an enterprise has made it to the survival phase. There are some risks that enterprises do not need to take. In a book about the enterprise, “Good to Great (Harper Business, 2011)”, Jim Collins asks, “How do good-to-great organizations think differently about technology?”. He found that in every case of a “good-to-great” company, they found technological sophistication and became a pioneer in applying technology. Further, he states that technology is an accelerator, not a creator, of momentum.

NOTE

Mark Spitznagel makes a case for considering the geometric mean in financial investment in the book *Safe Haven* (Wiley, 2021). He states, “Profit is finite. Risk is infinite”. What percentage of your wealth you can lose is more important than the absolute value of the wealth you could lose when investing. This fact is a point well suited to the enterprise. Why take a risk with unbounded loss?

Collins’ key point about technology directly applies to MLOps in the enterprise. The purpose of machine learning is to accelerate the business value that is already there. The reason to use machine learning isn’t to pivot the organization to becoming machine learning researchers competing with

companies that specialize in research; it is to accelerate the strategic advantages of the organization through technology.

The calculated risk of adopting machine learning as a business accelerator is acceptable if done in a manner that allows an organization to limit the downsides of technology change management. There is essentially unbounded risk in a company creating bespoke machine learning solutions and platforms when its core strength is in some other industry, such as manufacturing, hospitality, or financial services.

Many options exist to accelerate technological advancement in the enterprise, including using pre-trained models like [Hugging Face](#) or [TensorFlow Hub](#), computer vision APIs like [AWS Rekognition](#), or open-source AutoML solutions like [Ludwig](#) or MLOps orchestration frameworks like [MLRun](#). Enterprises that adopt MLOps with an approach of using the right level of abstraction give themselves a “good-to-great” advantage over organizations that “hired 15 data scientists” who do “research.” In the latter example, it is often the case that after years of research, in the best case, nothing is done, but in the worst case, a lousy solution creates a worse outcome than doing nothing.

Frank Knight, a professor from the University of Chicago, clearly [articulates the difference](#) between risk and uncertainty. He goes on to state that reward for taking a known risk is much different than a risk where it is immeasurable and impossible to calculate. This form of risk is called “Knightian uncertainty” after Frank Knight. An enterprise doing machine learning should deeply consider which risk they are taking, a regular risk that is knowable, or they are embarking on a path with Knightian uncertainty. In almost all cases, it is better to take knowable risks in machine learning and AI since technology is not the creator of growth; instead, it is the accelerator.

Knowing that acceleration is the crucial insight into great companies that use technology, let’s discuss some of the differences in technology acceleration differences between MLOps and DevOps.

MLOps vs. DevOps

Without DevOps, you cannot do MLOps. DevOps is a foundational building block for doing MLOps, and there is no substitute. DevOps is a methodology for releasing software in an agile manner while constantly improving the quality of both business outcomes and the software itself. A high-level DevOps practitioner has much in common with a master gourmet chef. The chef has deep knowledge of ingredients and years of practical experience creating beautiful and delicious meals, and they can make these meals in an industrialized and repeatable manner. The repeatability allows a restaurant to stay open and earn a profit.

Similarly, with DevOps, an expert in the domain has detailed knowledge of how to build software and deploy it in a high-quality and repeatable manner. One of the biggest challenges for experts in Data Science to transition to MLOps is a lack of experience doing DevOps. There is no substitute for experience; many data science practitioners and machine learning researchers should get experience building and deploying software with the DevOps methodology to get the foundational knowledge and experience necessary to be an expert at MLOps.

NOTE

You can learn more about DevOps from the book “Python for DevOps (O’Reilly, 2019)”, also available on the [O’Reilly platform](#).

There are apparent differences, though, between traditional DevOps vs. MLOps. One clear difference is the concept of Data Drift; when a model trains on data, it can gradually lose usefulness as the underlying data changes. A tremendous theoretical example of this concept comes from Taleb in the book, *Fooled by Randomness*, where he describes how a “naughty child” shown in [Figure 1-3](#) could disrupt the understanding of the underlying distribution of red vs. black balls in a container.

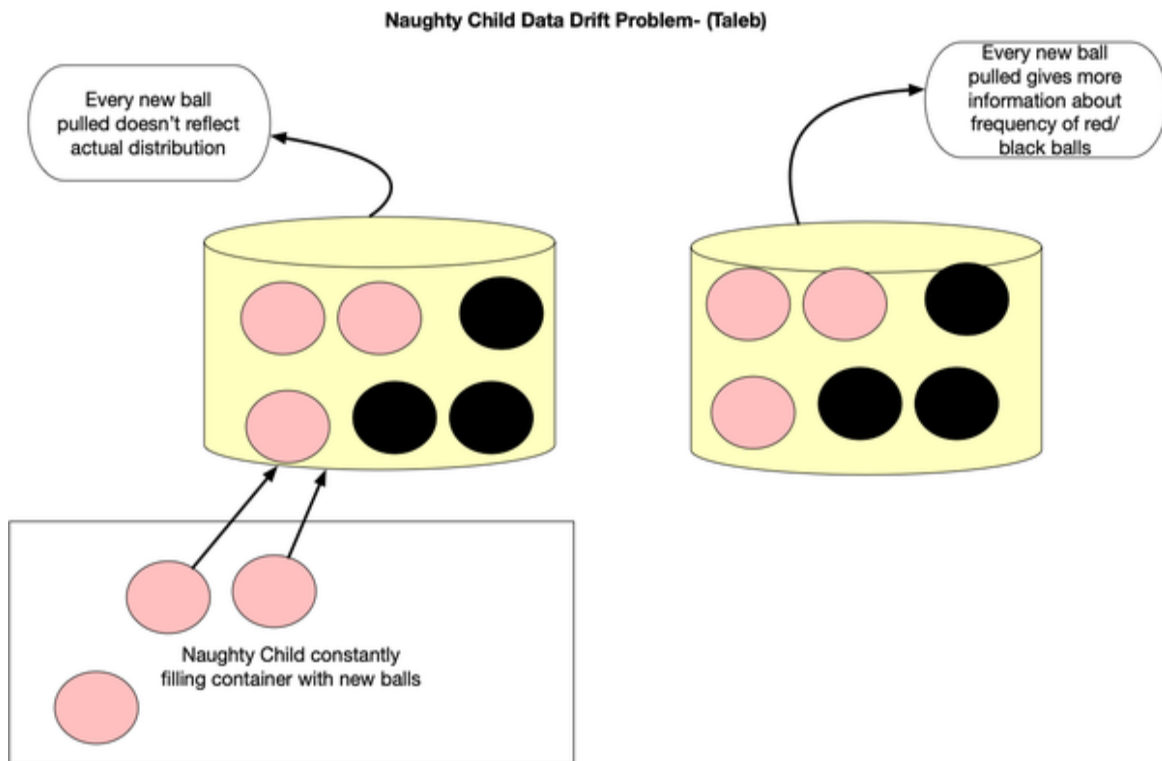


Figure 1-3. Naughty Child Problem by Taleb-Data Drift

In a static condition, the more balls pulled from a container, the more confident a person can be of the underlying distribution of red vs. black balls. In a dynamic condition, if the balls are constantly changed, then a model trained on an older data version won't be accurate. This example effectively captures one of many unique elements specific to MLOps not found in DevOps.

NOTE

Microsoft [notes](#), "Data drift is one of the top reasons model accuracy degrades over time."

The takeaway is that DevOps is a necessary foundation for MLOps, but MLOps' additional requirements, like data drift, don't appear in traditional DevOps. Next, let's talk about what isn't MLOps.

What isn't MLOps?

One way to understand more about MLOps is to define what it is not. Let's discuss some common MLOps anti-patterns:

Hiring a team of data scientists and hoping for the best

Perhaps the most common of the MLOps anti-patterns is hiring a team of data scientists and expecting an excellent solution to appear. Without organization support that understands MLOps and technology infrastructure to support them, there will not be an ideal outcome.

Only building bespoke machine learning solutions

A fundamental problem with only building customized solutions is that they may not be necessary for an organization's business goals. Training a bespoke machine learning model on propriety data for a self-driving company is essential to a competitive advantage. Training a similar model for a Fortune 500 delivery company could be a costly experiment adding no real value to the business.

Dismissing DevOps importance

Teams that work in silos are not following the best practices of DevOps. It is impractical to have a data science team in, say "Texas" that builds models in R, then "throws them over to the DevOps team to put into the software stack in the financial district of San Francisco in Python."

Ultimately, MLOps requires a business and production-first mindset. The purpose of machine learning is to accelerate business value. This fact means the teams building solutions must be agile in their approach to solving machine learning problems. Next, let's discuss mainstream definitions of MLOps.

Mainstream definitions of MLOps

A challenging problem in technology is separating a marketing strategy from a technology strategy. In the case of MLOps, it is not a marketing strategy; it is a specific solution to a severe problem in the enterprise. The bottom line is that models are not making it into production; if they do, they are brittle and fall apart when faced with the complexities of the actual world. Various surveys show that 50-70% of organizations have failed to deliver AI pilots or models to ³production!

With the condition identified, let's discuss the cure. The cure needs to address the following key issues (among others): model deployment and development time, the collaboration between different teams, operational excellence of ML systems, data governance, and finally, enhancing the ROI of the enterprise deploying the model. One minimalist way to then define MLOps is that it supports ML development like DevOps supports software development.

What is ML Engineering

One way to define ML Engineering is to look at popular certifications. **Google Machine Learning Engineer** explains the following criteria for a professional machine learning engineer:

Frame ML problems

Which model to choose depends on business constraints and the context. For example, a business may decide to classify damaged shipped boxes vs. successfully delivered packages. In that context, a classification model would be more appropriate than a regression model.

Architect ML solutions

An ML engineer develops a solution to solve the correctly framed using machine learning alongside other team members.

Design data preparation and processing systems

Two **critical steps** in data preparation and processing are constructing the dataset and then transforming the data.

Develop ML models

The **detailed modeling process** involves a team or individual that creates a model correctly suited to initial model framing.

Automate and orchestrate ML pipelines

A **pipeline** serves to create a process for reproducible and maintainable ML.

Monitor, optimize, and maintain

It is better to be proactive vs. reactive in building complex systems. Building monitoring allows for a **proactive approach** to maintaining ML systems.

NOTE

Several O'Reilly books discuss machine learning engineering, including **Data Science on the Google Cloud Platform**, **Machine Learning Design Patterns** and **Practical MLOps**.

ML Engineering aims to build high-quality ML models that solve specific business problems while creating ROI. Next, let's discuss MLOps in the context of incentives.

MLOps and Business Incentives

A classic problem in business school is incentives. One way to put incentives is “who moved the cheese?”. This scenario refers to a rat in a maze that only moves concerning where the cheese is. Similarly, there are two common incentives worth discussing in MLOps: hiring data scientists without regard for ROI and negative externalities.

Negative Externalities

Negative externalities like a company creating a profit from manufacturing but dumping toxic waste into a river and hurting downstream citizens of the town are classic examples of the fundamental problems in capitalism. In machine learning, the negative externalities could be unfairly biased algorithms that send an innocent person to jail or deny a person credit based on federally protected laws. Enterprises that fail to look into the future could expose themselves to existential risk.

Hiring data scientists without regard for ROI

It has recently been in vogue to “hire data scientists” without regard for what problem they are solving. As discussed earlier, this strategy ultimately doesn’t work because models are not in production at most organizations doing AI and ML.

Technical & Organizational Challenges - Why Most Models Never Make it to production

MLOps in the Cloud

There are several critical advantages of cloud computing that MLOps methodology leverages. First, the cloud is an elastic resource that enables both the efficient use of computing and storage and the ability to scale to meet almost any demand. This capability means that cloud computing has on-demand access to essentially infinite resources.

Second, the cloud has a network effect in that cloud technologies benefit from integrating other cloud technologies. A great example is AWS Lambda, a serverless technology. AWS Lambda is a valuable service to build applications with not because of what it does alone but because of the deep integration with other AWS services like AWS Step Functions, AWS

Sagemaker, or AWS S3. For any active cloud platform, you can assume that the integrated network of services further strengthens its capabilities as the platform develops more features.

Third, all cloud vendors have MLOps platforms. AWS has **Sagemaker**. Azure has **Azure Machine Learning**, and Google has **Vertex AI**; even smaller niche clouds like Alibaba Cloud have platforms like their **Machine Learning Platform for AI**. By using a cloud platform, an organization will likely use some of the offerings of the native ML platform and potentially augment it with custom solutions and third-party solutions.

Fourth, all cloud vendors have Cloud Development Environments. A significant trend is the use of a combination of lightweight CloudShell environments like **AWS CloudShell**, heavier full IDE (Interactive Development Environment) options like **AWS Cloud9**, and notebook environments, both free like **Sagemaker Studio Lab** or **Google Colab** and those with rich IDE integration like **Sagemaker Studio**.

Finally, depending on what a company is doing, they may have no option but to use cloud computing. Some cloud computing components are a hard requirement for organizations specializing in building bespoke Deep learning solutions because Deep learning requires extensive storage and compute capabilities.

In addition to the public cloud vendors, several additional players offer MLOps solutions in the cloud (see below). These vendors can operate on the public cloud or on private clouds. The advantage of using a smaller vendor is the customization level that such a company provides its customers. In addition, an MLOps vendor will have more in-depth expertise in MLOps since that is its only focus. This often ensures a lot more relevant features and a lot more integrations. Finally, by choosing a vendor that isn't tied to a specific cloud provider, you as a customer aren't tied to it either. Rather, you can use the vendor across multiple clouds or on additional infrastructure that you may have (see below).

Forthcoming - The AI Infrastructure Alliance (AIIA) is an AI alliance dedicated to providing data scientists and engineers with clarity and

information about AI/ML tools so they can build robust, scalable, end-to-end enterprise platforms. One of its initiatives is a comprehensive MLOps landscape that maps out all the players in the industry. The AIIA is currently creating an updated MLOps landscape that will map out open source and enterprise solutions for MLOps. The new landscape will encompass multiple categories and hundreds of companies, while detailing the capabilities of each solution.

Temporary: **Here is** the existing AIIA landscape. In **Figure 1-4** notice a typical pattern amongst all clouds in which there is a set of cloud development environments, flexible storage systems, elastic compute systems, serverless and containerized managed services, and 3rd party vendor integration.

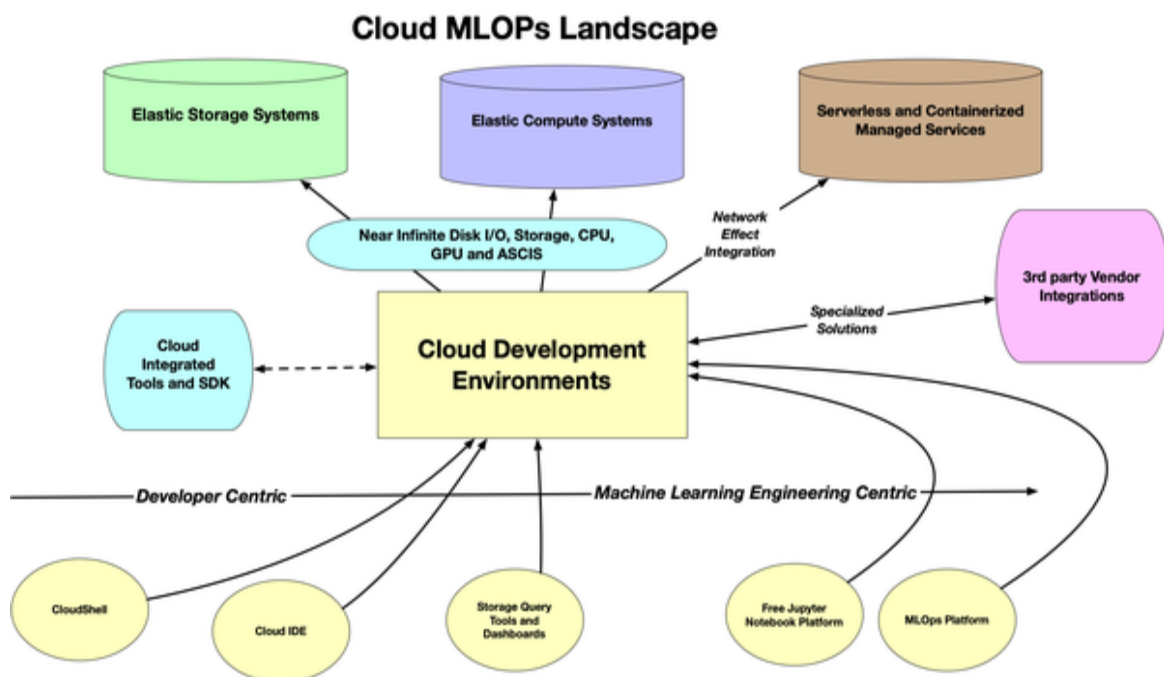


Figure 1-4. Cloud MLOps Landscape

Let's break these categories down into more detail:

Cloud Development Environments

Generally, developer-centric tools like cloud shells and IDEs are on one extreme and machine learning-centric tools on the other. Storage query

tools like [Google BigQuery](#), [Amazon Athena](#), or [Azure Databricks Integration](#) are in the middle.

MLOps Platforms that Operate in the Cloud

MLOps platforms that were built specifically for running MLOps for enterprises on the cloud or across any environment. Solutions like [Iguazio](#), [Valohai](#), [DataRobot](#) and others offer a wide variety of MLOps solutions for the enterprise.

Elastic Storage Systems and Elastic Compute Systems

Deep learning systems thrive on big data, and flexible compute capabilities from GPUs, CPUs, and AI Accelerator ASICs (application-specific integrated circuits) like TPU (Tensor Processing Units). As a result, MLOps platforms, both native and 3rd party, heavily use this elastic capability to provide managed solutions.

Serverless and Containerized Managed Services

Cloud platforms evolve towards more serverless solutions like [AWS Lambda](#) or [Google Cloud functions](#) and solutions with fully managed containerized solutions such as Google [Cloud Run](#) or [AWS Fargate](#). These managed services, in turn, have deep platform integration, which enhances the value proposition of the cloud platform through a network effect.

3rd Party Vendor Integrations

A cloud platform can't have the exact right mix of everything and at the right quality. A trip to a large warehouse store yields a wide variety of offerings at a reasonable price. On the other hand, though they may not have the authentic gourmet food you like or the exact appliance features you need, a cloud provider cannot go deep on everything. As a result, 3rd party integrations handle these specialized or advanced use cases.

With the common aspects of cloud computing for MLOps covered, let's move on now to discuss the cloud environments in more detail.

Key Cloud Development Environments

One of the best new products from Microsoft is [Github CodeSpaces](#). It is a cloud-based development environment with many customizable features and a great place to practice MLOps. In particular, what is helpful about this environment is the deep integration with Github and the ability to customize it with a specialized runtime. Finally, the synergy with [Github Actions](#) allows for a great CI/CD story.

NOTE

You can learn more about Github CodeSpaces with the following O'Reilly videos:

- [GitHub Codespaces and custom dotfiles](#)
- [Compiling Python from scratch with Github Codespaces](#)

There are three different flavors of cloud-based developments available from Google. The first is [Colab Notebooks](#), the second is [Google Cloud Shell](#), and the third is [Google Cloud Shell Editor](#).

In [Figure 1-5](#) there is a full editor available for GCP.

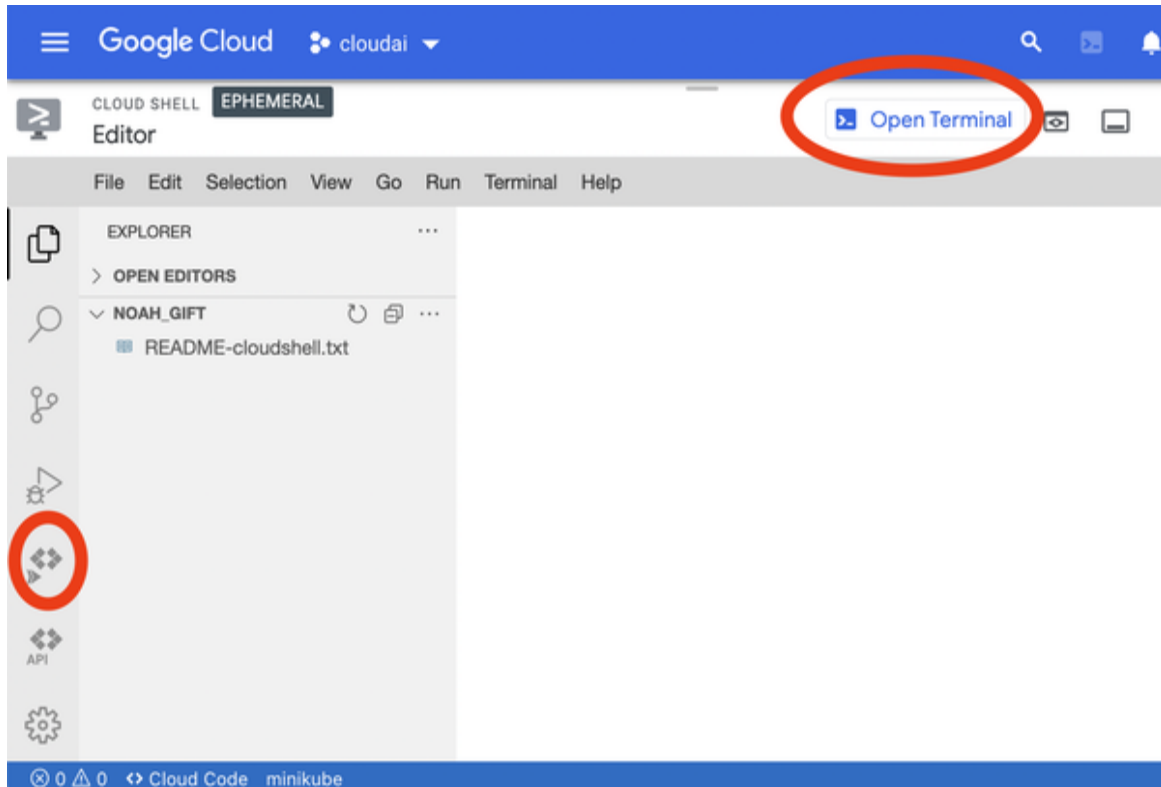


Figure 1-5. Google Cloud Shell Editor

In Figure 1-6 API docs integrate with the development environment.

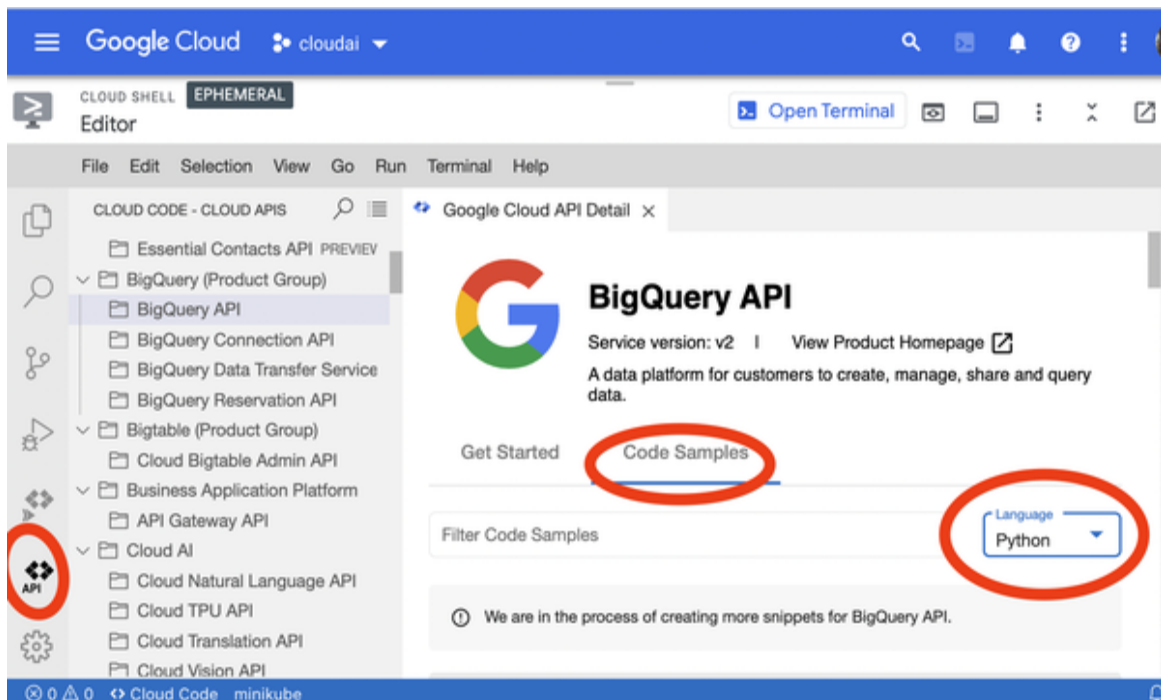


Figure 1-6. Google Cloud Shell Editor API

In **Figure 1-7**, the terminal shows a standard view of the experience using the cloud shell.

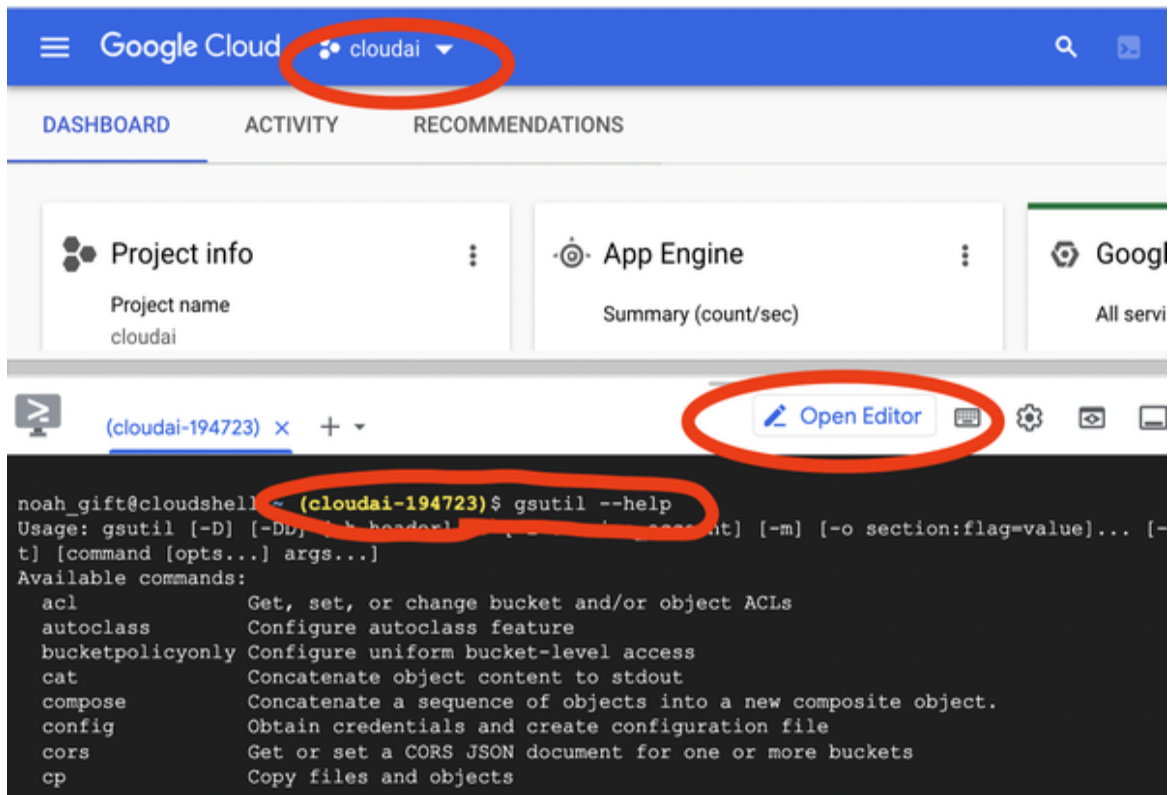


Figure 1-7. Google Cloud Shell Terminal

NOTE

Python for Data Science with Colab and Pandas in One Hour Video Course

What are Google Colab Notebooks, and How do you share them for Data Science projects?

Finally, the AWS platform has cloud shell environments, as shown in **Figure 1-8**.

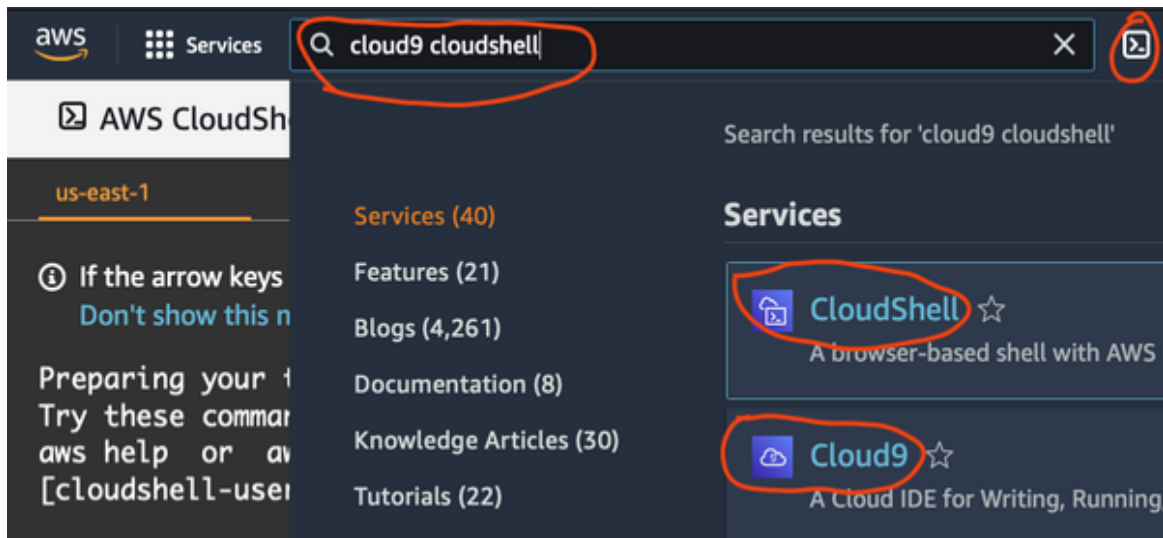


Figure 1-8. AWS Cloud Shell Terminal

NOTE

One quick way to learn about multiple clouds simultaneously is by setup a multi-cloud continuous integration. You can learn how to set this up with [GithubActions in this O'Reilly video](#).

All of this leads to the concept of the cloud-developer workspace advantage, as shown in [Figure 1-9](#). A laptop or workstation is expensive and non-deterministic due to pre-installed software and, by definition, not the deploy target. When you look at a cloud-based workspace, it has many incredible advantages, including power, disposability, pre-loading, and deep integration with advanced tools.

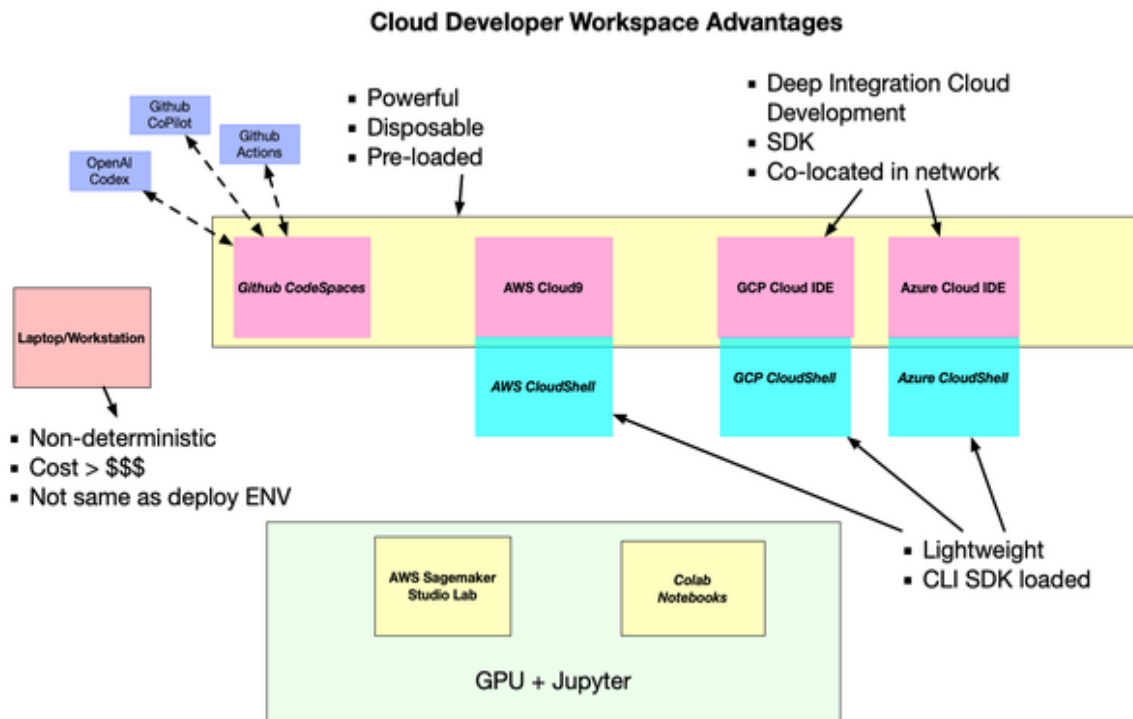


Figure 1-9. Cloud Developer Workspace Advantages

NOTE

You learn more about the cloud-developer workspace advantage in the following O'Reilly video [Cloud Developer Workspace Advantage](#) or on [YouTube](#).

With the importance of cloud computing established, let's look at the leaders in the cloud market.

The Key Players in Cloud Computing

Know someone that wants to earn 200k or more a year? According to the [2022 Cloud Salary Survey](#) by Mike Louikides at O'Reilly, the average salary for certified professionals on AWS, Azure, and GCP is over 200k.

Further backing this up is the data from Statista, as shown in [Figure 1-10](#). As of Q4 2021, there are three key players in the worldwide market. AWS has about 33% of the market share, Azure has about 21%, and Google Cloud has about 10%. Combined, these three vendors have 2/3 of a market

that generates almost 200 billion in revenue. Service revenue increased in size by 37% from the last year.

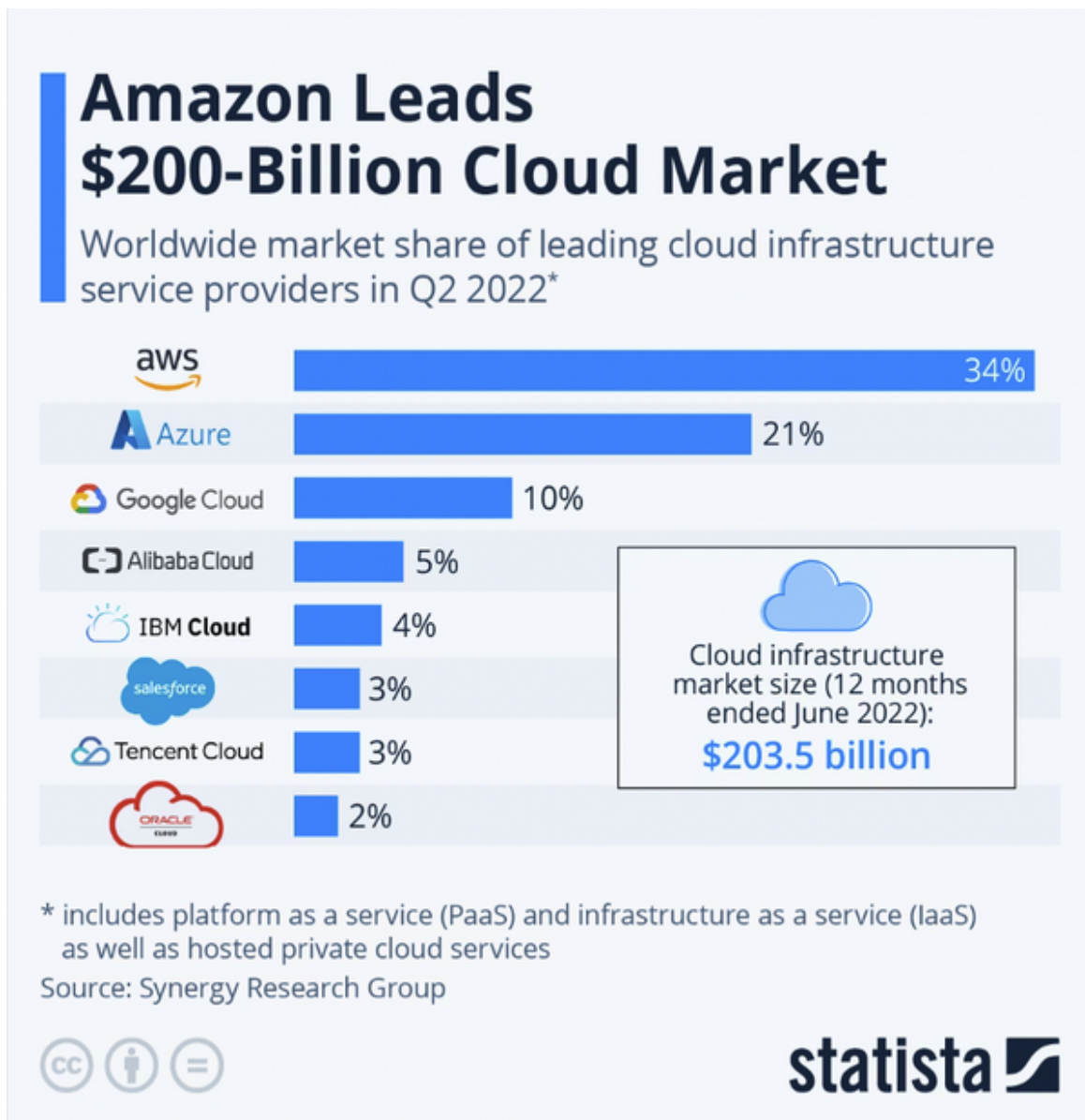


Figure 1-10. Cloud Computing Market

A reasonable strategy for an organization wishing to use cloud computing is to use the platform of the largest providers. The Matthew⁴ effect saying, “the rich get richer, and the poor get poorer,” applies to cloud computing for several reasons.

Available Employees and Vendors to Hire

Leveraging the most prominent cloud platforms makes hiring employers and finding vendors that work with the platform more accessible.

Training material available

The availability of training material for the most prominent platforms makes it easier to train employees.

Services Available

Larger platforms can hire more software engineers and product managers, meaning you can count on a continuation of new features and maintenance in their platform.

Cost of Service

Economies of scale mean that the most significant providers benefit the most from economies of scale. They can leverage pricing advantages by buying in bulk and then passing them down to the customer.

NOTE

You can study for the AWS Cloud Certifications by viewing the [AWS Solutions Architect Professional Course](#) and [AWS Certified Cloud Practitioner Video Course](#) on the O'Reilly platform and authored by [Noah Gift](#).

Now that the top providers in cloud computing are known, let's discuss how each vendor views the world of cloud computing as it relates to MLOps.

AWS View of Cloud Computing as it relates to MLOps

The best place to get a high-level summary of AWS cloud computing is the [Overview of Amazon Web Services AWS Whitepaper](#). In particular, they mention [six advantages of cloud computing](#).

NOTE

You can learn more about AWS in the O'Reilly book “Developing on AWS with C# (O'Reilly 2022)”, also available on the [O'Reilly platform](#).

Trade fixed expense for variable expense

By avoiding large capital expenditures, it encourages agility and efficiency.

Benefit from massive economies of scale

As prices decrease for the supplier, they fall for the customer allowing for lower pricing than if the customer bought the same product. Similarly, managed services on the platform will have a steady schedule of new features.

Stop guessing capacity

There isn't a need to pre-provision resources since systems get built with an elastic ability to scale as needed.

Increase speed and agility

Focusing on an organization's comparative advantage and not building non-essential-to-business IT allows an organization to move faster.

Stop spending money running and maintaining data centers

Cost savings accumulate from outsourcing this component of IT.

Go global in minutes

Going global is a highly challenging problem that goes away with AWS due to its comprehensive offerings.

These features ultimately drive into the core MLOPs offering of Amazon Sagemaker in [Figure 1-8](#) as the project's lifecycle goes from preparation to

building to training, to finally deploying and managing the solution. At the center of the workflow is tight integration with developer tools from Studio and RStudio.

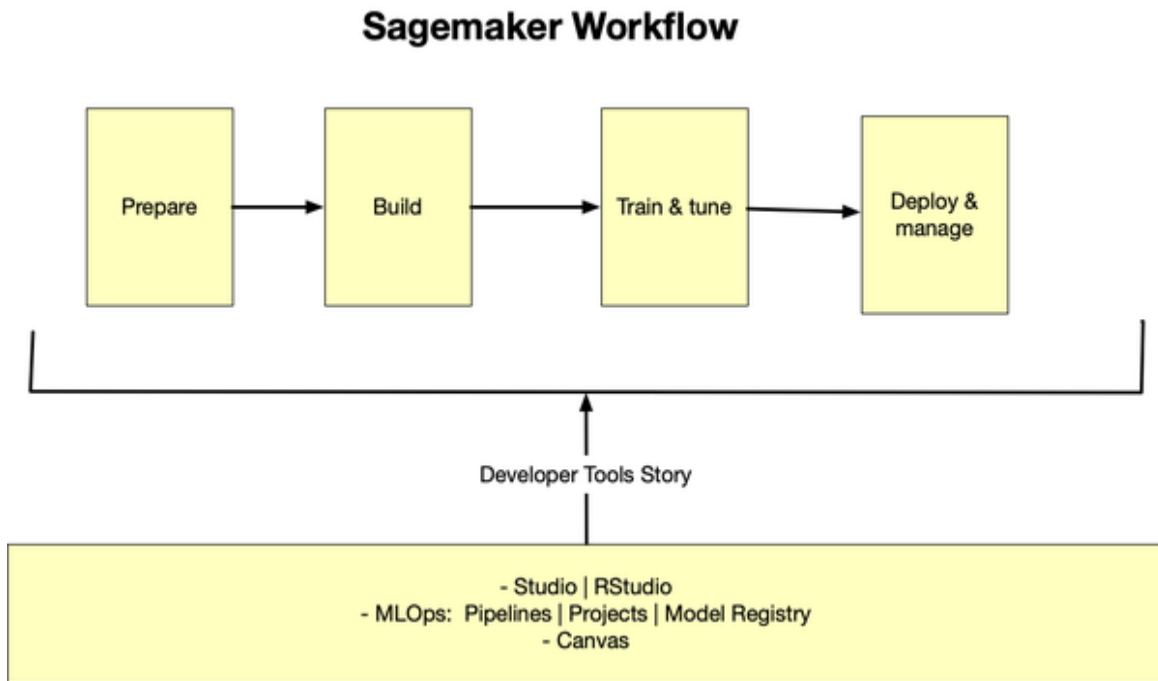


Figure 1-11. AWS Sagemaker MLOPs workflow

NOTE

On the O'Reilly platform video, you can see a complete walkthrough of Sagemaker Studio Lab [Sagemaker Studio Lab](#).

With the AWS view of the MLOPs complete, let's look at Azure next.

Azure View of Cloud Computing as it relates to MLOps

Microsoft Azure **sees the world of MLOPs** as a way to “efficiently scale from a proof of concept or pilot project to a machine learning workload in production.” In the following **Figure 1-12**, the model's lifecycle goes from training, packaging, validating, deploying, monitoring, and retraining.

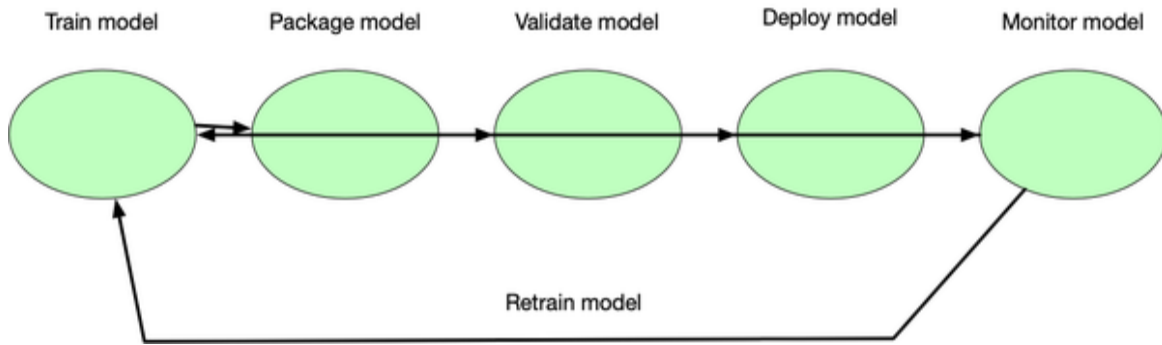


Figure 1-12. Azure MLOps

With Azure MLOps view of MLOps covered, let's next look at how Google views MLOps.

GCP View of Cloud Computing as it relates to MLOps

An ideal place to look at how Google sees the world is by looking through the [production-ml-systems crash course](#). One of the items they point out is how tiny the actual modeling part of the problem is in [Figure 1-13](#) instead, the combination of other tasks, including data collection, serving infrastructure, and monitoring, take up much more of the problem space.

Google's View of MLOps

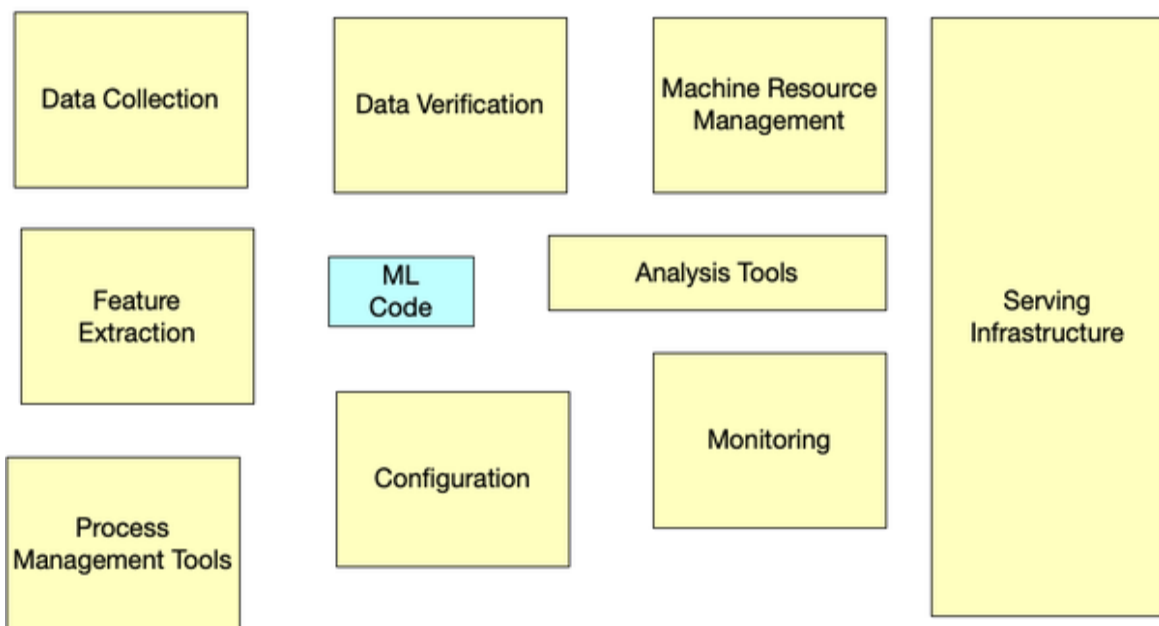


Figure 1-13. Google's view of MLOps

Ultimately this leads to how their **Vertex AI platform** handles the MLOps workflow in **Figure 1-14**: The ML development process occurs, including model framing for the business problem. The data processing phase leads to an operationalized training process that can scale up as needed. Then the model deployment occurs along with a workflow orchestration alongside artifact organization. The model has monitoring baked into the deployment process.

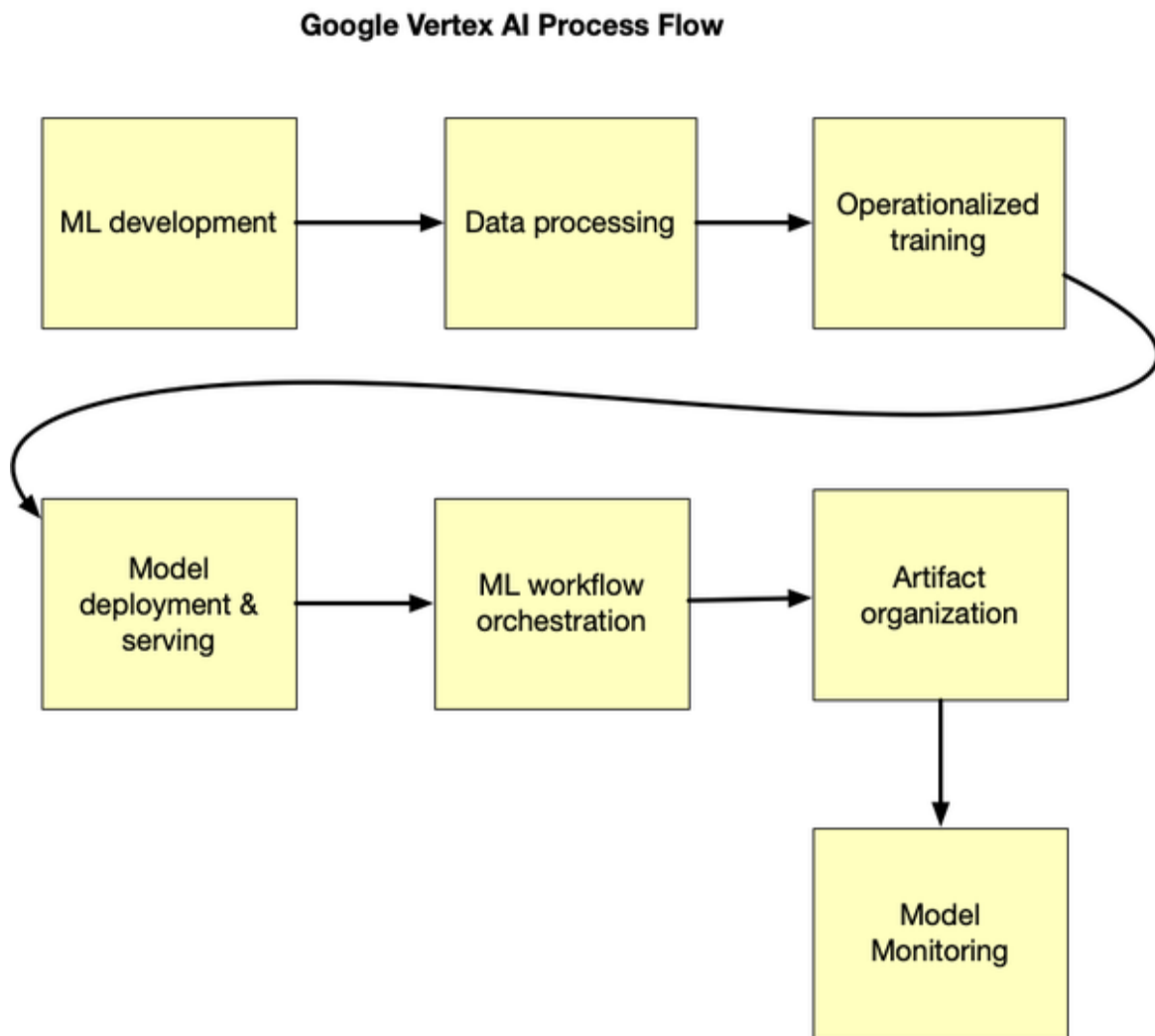


Figure 1-14. Google's view of MLOps

While public cloud providers offer their own solutions, sometimes enterprises might need a solution that is more tailored to their specific needs. Let's look at two more deployment options: on-premises deployment and hybrid cloud deployment.

MLOps On-Premises

In some use cases, enterprises cannot use the public cloud. Business restrictions like the need to secure sensitive data or having to adhere to strict regulations (e.g data localization privacy regulations) require an MLOps solution that can operate on-premises. Many MLOps solutions offer the ability to deploy them either in the cloud or on-premises. The only downside to this approach is that on-premises solutions require the enterprise to provide the servers and equipment that will support the intense computing power needed to run ML algorithms at scale. They will also need to update and maintain the infrastructure.

On the other hand, an on-premises deployment will almost certainly require some sort of customization. This gives enterprise more control over the product and they can make specific requests to tailor it to their needs. More specifically, if the deployed solution is a startup solution they will be attentive and work hard to ensure satisfaction and adoption. If it's a open source product, then not only can enterprises leverage the development power of the community, they can also go inside with their own developers and tinker with the product to ensure it suits their needs.

MLOps in Hybrid Environments

Similar to on-premises deployment, some enterprises might prefer a hybrid cloud deployment. This would involve deploying on the public cloud(s), on-premises, and perhaps even on a private cloud or on edge devices. Naturally, this makes things a lot more complex, since the MLOps solution must enable total separation of the data path from the control path and must be delivered by a highly available, scalable entity that orchestrates, tracks and manages ML pipelines across types of infrastructure deployments. Lest we forget, this has to occur at high speed and with optimal performance. Finally, the solution would ideally provide a single development and deployment stack for engineers across all infrastructure types.

Finding a vendor or open source solution that answers all these requirements might not be as simple, but as mentioned before, your best bet is probably with startups or mature OSS solutions that can be customized to the specific needs of your infrastructure.

Enterprise MLOps Strategy

With a high-level overview of the critical issues involved in MLOPs, it is time to turn to strategy, as shown in **Figure 1-15**. There are four key categories to consider when implementing an MLOPs strategy: cloud, training & talent, vendor, and executive focus on ROI.

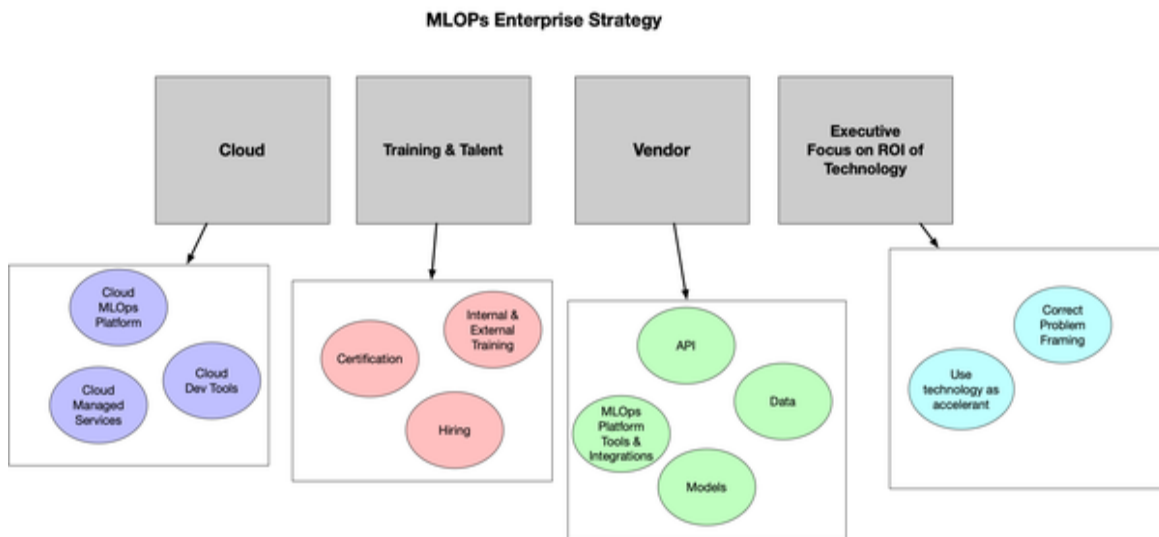


Figure 1-15. Enterprise MLOPs Strategy

Let's discuss each of these four categories:

Cloud

There is no perfect answer for which cloud platform to use. Any central platform will offer advantages of economies of scale. What is essential in an MLOps strategy is to be aware of how a cloud platform fits into the unique goals of each specific organization and how it aligns with other strategic components like hiring or 3rd party vendor integration.

Training & Talent

Often organizations only look at the power of new technology and don't consider the training and talent component of using the technology. In almost all cases, an organization should use a less powerful technology if hiring and training are better with a less powerful solution. This fact means the widespread use of technology is crucial when implementing new technology. Ultimately, the latest technology is dead on arrival if you cannot hire or train your staff.

Vendor

An often overlooked issue with using cloud computing is that it usually needs to be augmented by specialized vendors to help an organization reach its goals with the technology. These strategic choices can lead to better ROI for both the cloud and the business strategies. Examples include using vendor technology specializing in Hadoop, Kubernetes, or pre-trained models. The vendors will be unique to each organization and their business goals.

Executive Focus on ROI

Ultimately the preceding three categories don't mean anything if the executive focus isn't on ROI. The purpose of technology is to drive long-term business value, meaning problems need accurate scoping.

With the enterprise MLOps strategy complete, let's wrap up the entire chapter.

Conclusion

This chapter sets the stage for understanding the crisis in enterprises getting machine learning and AI in production. From a common sense approach, the idea of just "hiring more data scientists" to increase ROI is as sensible as "just hiring more software engineers" to make a traditional software project go faster. In the case of the conventional software company, if there is no product, no goal, and no oversight, then hiring more developers

increases the capital expenditure of the organization without any added value.

Instead of this scenario, MLOps aims to add a methodology that builds off the successful lessons of DevOps while handling the unique characteristics of machine learning. Finally, at the enterprise level, ultimately, it comes down to ROI with data science. Technology is an accelerant of value for most organizations, not the value. Organizations that create a hunger for ROI can quickly adopt the MLOps mindset.

Critical Thinking Discussion Questions

- There are many methods to deploy machine learning models to production, including pre-trained models, APIs, AutoML, and bespoke training. What are the pros and cons of each of these approaches?
- What strategies could an enterprise implement to attract new machine learning engineering talent and train and retrain current talent?
- If your organization doesn't currently do any DevOps, a foundational component necessary for MLOps, how could they start a first DevOps project to test concepts like CI/CD and IaC?
- If your organization doesn't have large quantities of proprietary data, how can it use machine learning to gain a competitive advantage anyway?
- What is your organization's cloud strategy: single cloud, multi-cloud, hybrid cloud, private cloud, or something else? How does this help your organization reach your MLOps goals?

Exercises

- Go to a popular model hosting site like [TensorFlow Hub](#) or [Hugging Face](#) and deploy one of their models to your favorite cloud platform.

- Pick a cloud-based development environment like [Github CodeSpaces](#), [Amazon SageMaker Studio Lab](#), or [Google Colab](#) and explore the interface with an eye for building a machine learning engineering project.
- Use a machine learning app framework like [Gradio](#) or [Streamlit](#) to build a simple machine learning application.
- Brainstorm several organizational problems that may benefit from using machine learning and build a simple prototype using an MLOps technology.
- Convert a Kaggle project to an MLOps project by downloading the dataset and coding an MLOps technology to serve predictions.

-
- 1 Dr. Luks summarizes the systematic evidence-based strategy: “Create a caloric deficit, then stay lean. Get sleep. Eat real food. Move often, throughout the day. Push and pull heavy things. Socialize. Have a sense of purpose.”, Luks, Howard. *Longevity...Simplified: Living A Longer, Healthier Life Shouldn't Be Complicated* (p. 9). KWE Publishing. Kindle Edition.
 - 2 In the book “*Principles of Macroeconomics* (McGraw Hill, 2009)”, the authors share the story of how a talented chef could extract all of the profit from restaurants in a scenario of perfect competition since they would continuously leave for a higher salary to a competing restaurant thus ultimately removing all profit for the owner.
 - 3 A white paper from [Google Practitioners Guide to MLOps](#) discusses this issue in detail.
 - 4 Sociologists Robert K. Merton and Harriet Zuckerman [first coined](#) this term.

Chapter 2. The Stages of MLOps

MLOps is not about tracking the local experiments and is not about placing an ML model behind an API endpoint. Instead, MLOps is about building an automated environment and processes for continuously delivering ML projects to production.

MLOps consists of four major components (and is not confined to model training):

- Data Collection & Preparation
- Model Development & Training
- ML Service Deployment
- Continuous Feedback and Monitoring

The following chapter will describe the different components in more detail.

What You Need to Get Started

Begin with the end in mind - The first step in any ML project is to articulate: 1) The problem that needs to be solved using ML 2) What you want to predict 3) How to extract business value from the answer. Examples of business value we might require include decreasing fraud, increasing revenue by attracting new customers, cutting operational costs by automating various manual processes, etc.

Once you define the goal, don't rush straight into the implementation phase. First, consider:

- Which historical and operational data can be gathered and used in both the training and serving pipelines
- How to incorporate the ML model results in a new or existing application in a way that can make an impact
- How to verify and reliably measure that the ML model meets the target and generates valuable business outcomes

Figure 2-1 illustrates the different stages in an ML project. Note the feedback loop where the observations are used to recalibrate the business goals, data collection and preparation logic.



Figure 2-1. ML Project Life Cycle

If you only focus on the ML model, you may encounter pitfalls such as:

- Using the wrong datasets, which can easily lead to inaccurate or biased results
- Lacking enough labeled data to build a model
- Finding out historical features used to train the model are unavailable in the production or real-time environment
- Discovering there is no practical way to integrate the model predictions into the current application
- Realizing the ML project costs are higher than the generated value, or, in a worst case scenario, cause losses in revenue or customer satisfaction

Choose Your Algorithm

The next phase is to determine the type of ML problem and algorithm.

Supervised Learning (labels are required and known):

Classification

The algorithm will answer binary yes-or-no questions (fraud or not, is it an apple, will the customer churn) or make a multiclass classification (type of tree, etc.). You also need enough labeled data for the algorithm to learn from.

Regression

The algorithm predicts continuous numeric values based on various independent variables. For example, regression algorithms can aid in estimating the right price for a stock, the expected lifetime of a component, temperature, etc.

See [Figure 2-2](#), which compares those two algorithms.

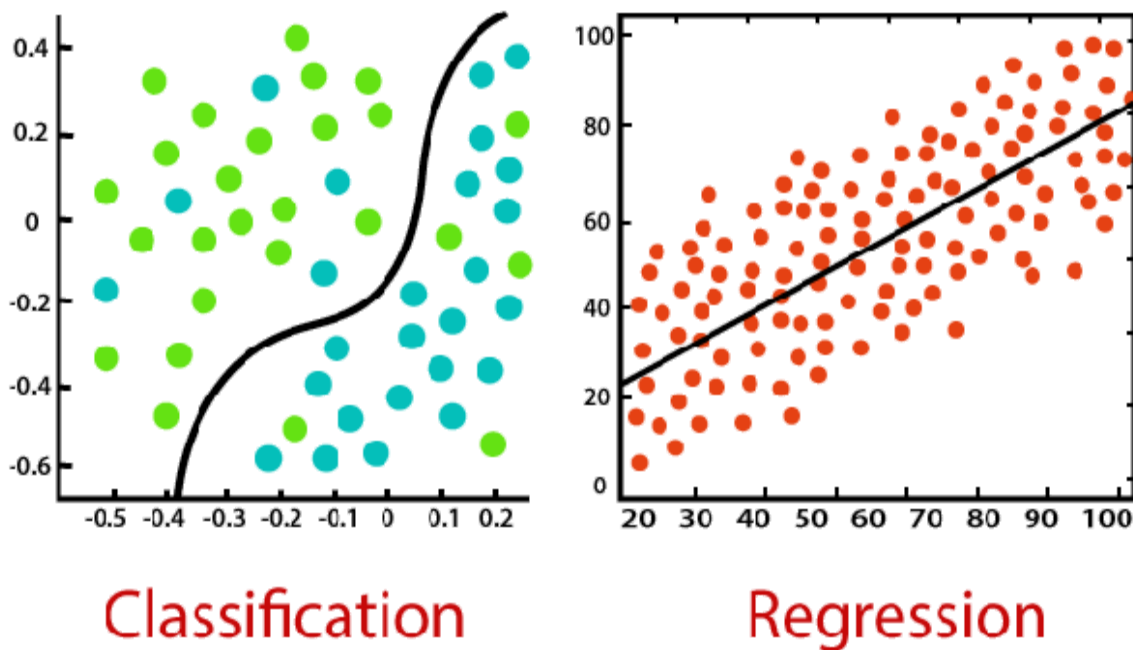


Figure 2-2. Regression vs Classification

Unsupervised Learning:

Clustering

The algorithm will look for meaningful groups or collections in the data (customers segmentation, medical imaging, music genre, anomaly detection, etc.) based on their similarity without the help of pre-labeled data.

Dimensionality Reduction

The algorithm will reduce the dimensionality (the number of input variables in a dataset) from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Dimensionality reduction allow us to avoid overfitting, reduce the model computation overhead, and handle less features than originally required.

Recommendation and Ranking

The algorithm recommends or ranks objects by considering their relevance, importance, and content score. Recommendation algorithms can be used to rank web pages, recommend movies or music in streaming services, or show the products that a customer might purchase with a high probability based on his or her previous search and purchase activities. Recommendation engines can be used either for supervised or unsupervised learning.

Note that Ranking algorithms rely on search queries provided by users, who know what they are looking for. Recommender systems, on the other hand, operate without any explicit inputs from users and aim to discover things they might not have found otherwise.

Some applications may incorporate multiple algorithms. For example, using an NLP algorithm to determine the sentiment in the text and using the sentiment as an input for making a purchase decision.

Design Your Pipelines

ML models have a limited lifetime since data patterns change (drift) over time, and models may have limited scope. For example, creating specific models per each user or device (trained on the relevant subset of the data). In many cases, we would like to train multiple models using different parameters or algorithms and compare or combine them.

For those reasons, the goal is NOT to build a model but rather to create an automated ML pipeline (factory) that can accept inputs (code, data, and parameters), produce high-quality model artifacts, and deploy them in the application pipeline.

The ML pipelines can be triggered every time the data, code, or parameters change or can be executed in a loop (each time with a different dataset or parameters) to produce multiple models. To understand, compare, or explain the model results, all the inputs (code, data, parameters), operational data (type of hardware, logs, etc.), and results must be recorded and versioned.

A model is usually deployed as part of a more extensive application pipeline, including API integration, real-time data enrichment and preparation, model serving, actions, and monitoring. The automated deployment cannot focus solely on the model but on deploying or updating the entire application pipeline.

The typical ML pipeline consists of data preparation, training, testing, registering, and deployment. In real life, the ML pipelines can incorporate additional steps for data validation, optimization, etc. In addition, some ML pipelines build and use multiple models.

Figure 2-3 demonstrates a recommendation engine application that uses two models in cascade. The first model is used to identify similar products. The second model will use the output from the first model and other user data to determine the buying probability (and filter the results).

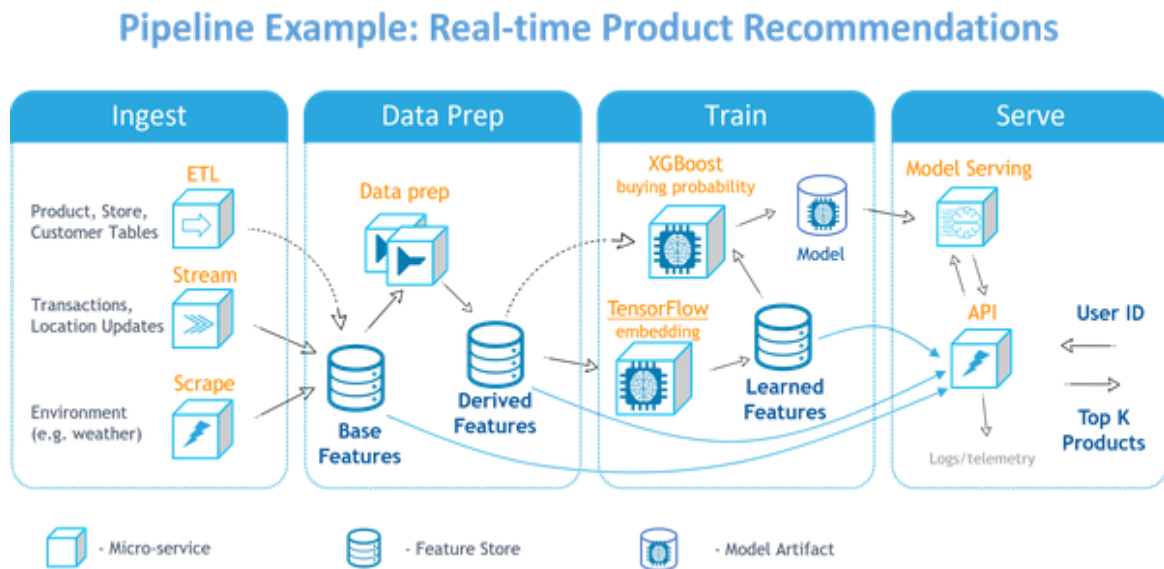


Figure 2-3. ML Pipeline Example: Real-time Product Recommendations

Data Collection & Preparation

There is no ML without data. Before everything else, ML teams need access to historical or online data from multiple sources. They must ingest, prepare and explore the data before building any model.

The first step is to define your goal, which problem or challenge you intend to solve, and which data sources or features can help you predict the outcome. Once you identify the target and raw datasets, you must gather enough data, prepare, label, and explore it for use in your model.

In most cases, the raw data cannot be used as-is for machine learning algorithms for various reasons such as:

- The data is low quality (missing fields, wrong spelling, null values, etc.) and requires cleaning and imputing.
- The data needs to be grouped or aggregated to make it meaningful.
- The data needs to be converted to numerical or categorical values, which algorithms can process.

- Feature values should be normalized and scaled to guarantee they have equal importance
- The data is encoded or requires joins with reference information.

According to IDC, by 2025, 80% of data will be **unstructured**, so an essential part of building operational data pipelines is to convert unstructured textual, audio, and visual data into machine learning- or deep learning-friendly data organization or vector formats.

The ML process starts with manual exploratory data analysis and feature engineering on small extractions from historical data. However, to bring accurate models into production, ML and data engineering teams must work on larger, more up-to-date datasets and automate the collection and preparation process. Furthermore, batch collection and preparation methodologies and batch analytics don't work well for operational or real-time pipelines. As a result, ML teams often build separate real-time data pipelines (pipelines that handle a very large number of events at scale in real-time) that use stream processing (the ingestion and processing of a continuous data stream).

Many algorithms require labeled data for training the model. Therefore, we must design and implement labeling solutions for the historical data as part of the data preparation process. In addition, many applications require constant re-training to maintain the model's accuracy and relevancy. Therefore, you should design a pipeline for automatically generating data labels in such cases.

Models are as good as the data they are trained on. To compare or explain model behavior and to address regulatory compliance, you must have access to the data used in training. Therefore, you must save information about the data origin with the model or save a unique copy of the dataset used for every training run. Data lineage and versioning solutions are a must in every MLOps solution.

A key solution in any modern MLOps solution is a **Feature Store** which automates the collection, transformation, cataloging, versioning, and

serving of offline and online data.

Data Storage And Ingestion

Data is the foundation for AI and ML. It can be persistent or in transit and can be broken into two main categories, structured and unstructured.

Unstructured data is usually stored in file systems, object storage (data lakes), logging, or messaging systems (such as email). Structured data has some schema and is stored in tables, documents, or graphs.

Since it is scalable and cost-effective, we usually use object storage for deep learning workloads that process images, video, and text (NLP). In some cases, we will use local or distributed file storage.

When the data is structured, we can use files (CSV, Excel, etc.) to do simple exploration and model training, but this cannot scale for production. In production, we store data in one of those two categories:

- Archival data systems (data warehouses or objects with structured file formats like CSV, Parquet, JSON, etc.) - record all the historical transactions and allow efficient analytics queries.
- Operational or real-time databases - frequently updated and enable fast data retrieval by index.

Use archival storage (data warehouses or data lakes with structured objects) for the training process since a model is an equation that learns how to predict results based on historical data patterns. Suppose the data source is a real-time or operational data system. In that case, you first need to copy and transform the data to the archival system, which is better at analytical workloads, for example, using an ETL process (Extract, Transform, Load). Structured object formats are usually the cheapest storage option, especially when using efficient compression techniques (like Parquet files). But data warehouses (like [Google BigQuery](#), [Snowflake](#), [Amazon Redshift](#), etc.) support faster and more flexible data queries and are easier to update.

When you collect data for training, it is essential to make sure there is no bias in the data since this can lead to poor model results and even a total failure of your project (see [Amazon scrapped sexist AI tool](#)).

MLOps solutions and the training flow should incorporate data version control. Every training job should point to a unique version of the data, which allows for reconstructing the exact content of the data. While this may be simple for static historical content, it is harder for continuous and dynamic data like user information or transactions, which can change frequently.

The solution is to snapshot and store the dataset in archival storage and add the appropriate link (data lineage) to the job and model objects, allowing viewing the data associated with each run easily. Some MLOps frameworks and feature stores (like [MLRun](#)) provide this as a built-in feature.

In the serving process, a request arrives with partial data, for example, a user ID; you enrich the data with additional features for that user (such as his age, gender, income, etc.) from an online database and pass it to the model. You cannot use archival storage for serving since it's too slow and cannot support a high number of concurrent requests. Instead, indexed NoSQL or SQL databases (like [Redis](#), [DynamoDB](#), or [MySQL](#)), also referred to as the online features store, are better since they are faster and you have the index key (user ID).

To use the online features, You must first copy them to the online database; this can be simple with static features (like age or gender) but challenging with transactional features (like the total amount of purchases in the last hour) that are frequently updated. Stream processing is usually used to calculate and update real-time features efficiently. This means the real-time data pipeline uses a different implementation than the offline feature calculation (implemented for training).

[Figure 2-4](#) demonstrates different components used in the data ingestion flow.

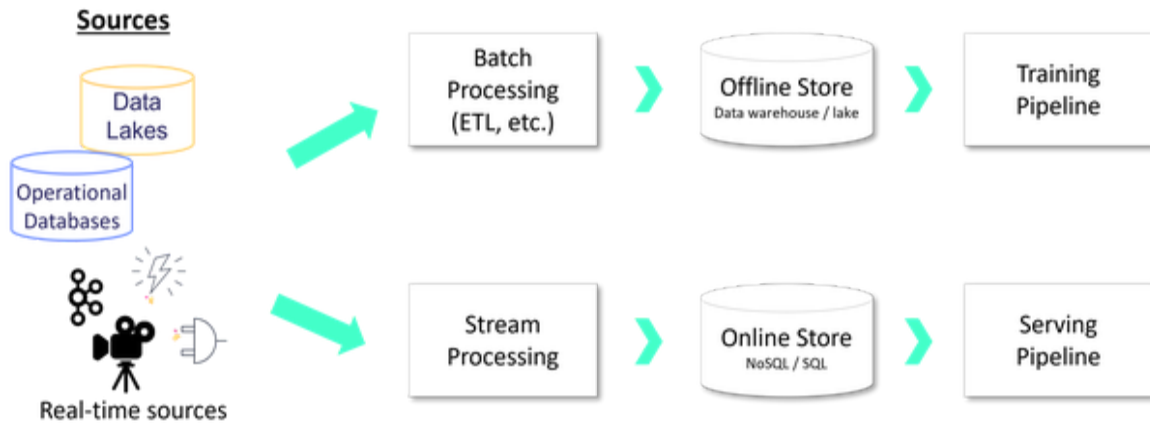


Figure 2-4. Offline and online data ingestion flow

Using different databases and data processing technologies in training and serving leads to higher complexity and data synchronization challenges. Feature Stores are used to abstract away much of that complexity and will be discussed in the following sections.

NOTE

Learn more about feature stores from [this online resource](#).

Data Exploration and Preparation

In most cases, you cannot use data in its raw format, so the first step is applying cleansing, transformations, or calculations to the data. Once we have a clean set of meaningful features, we can start evaluating the data and selecting the best features for our model.

Here are some examples of required data conversions:

- Data arrives in a JSON format, and we need to convert it to an array or vector
- Data contains a string (like a city name), and we need to convert it to a numeric value using some encoding strategy

- You have a transaction log, but you need the total value of transactions in the last month
- You have a person's zip code, but you need to translate it to a numeric value representing a social-economical score
- Dataset has missing values or misspelled names

It is easier to start data exploration with a subset of the data and use interactive visual tools or standard python packages like **Pandas**, **Matplotlib**, **Bokeh** and **Plotly**.

The first step is to visually inspect the data's nature and quality (inconsistencies, outliers, anomalies, missing data, etc.) and clean the data. Next, we transform and add derived features, examine the correlation between the data or its derivatives and the target feature (goal), to support or disprove your theory, and generate a training set (feature vector). Creating new derived features to improve a model's output, is the main craft of data scientists. Choosing relevant features to analyze and eliminate non relevant or redundant ones is also essential.

Note that in the production implementation, there is a need to process more significant amounts of data in an automated way. Therefore, you must re-implement the data cleansing and transformations steps as part of a scalable and automated data processing pipeline and may need to use scalable or real-time data processing engines (like **Spark**, **Flink**, **Nuclio**, etc.) instead of interactive tools.

Figure 2-5 illustrates the data preparation and feature engineering flow.

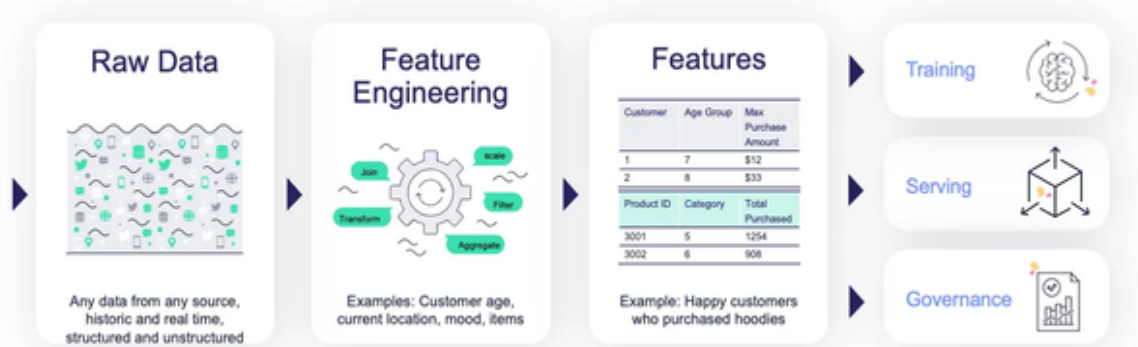


Figure 2-5. Feature Engineering Flow

These are some of the most common data transformations operations:

- Drop rows/columns with too much missing data
- Imputing - replace missing values with a constant or a statistical value (e.g. median of the column)
- Outlier detection - drop rows where the values don't fall under the expected range (for example, compare the row value with $\text{mean} \pm N * \text{stddev}$)
- Binning - group multiple values into a single category (for example, Chile and Brazil map to South America)
- Log Transform - convert a linear scale to a log scale
- One-hot encoding - map different categorical values to a binary (yes/no) feature.
- Grouping and aggregations - aggregate column values by time (hour, day, month, etc.) or by category (for example, number of units sold by product type)
- Scaling - re-scale column values (Normalization, Standardization)

- Date extractions - convert a date time to the hour, day of the week, month, season, is it a holiday, etc.
- Time recency - the time distance between two events (for example, time from the last login)

For unstructured data, there can be many more types of transformations (extract text elements, resize or rotate an image, etc.).

In training and during serving, you must use the same features; this requires you to implement two data pipelines, a batch pipeline for training and a real-time (streaming) pipeline for serving.

Some feature stores provide simple ways to define the data transformation logic and will automatically deploy and manage both offline and online data pipelines for you.

Data Labeling

Data labeling, or data annotation, is part of the preprocessing stage required for supervised learning. You add tags to raw data (numeric, text, images, etc.) to show a machine learning model the expected target attribute (prediction). Some prominent examples include [Amazon Sagemaker Ground Truth](#), [Label Studio](#), [DataTurks](#) and [CVAT](#).

For numeric values, labeling can be deducted from the raw data. So, for example, in a churn model that tries to predict which customers are about to churn, you can examine historical records and mark the customers who churned by looking to see if they stayed as a customer in the consecutive month. A simple analytics query will do the trick and shift the results back by one month.

Labeling is harder for unstructured data (text, images, video, audio, etc.) and usually involves a manual labeling process (by a human). However, many solutions in the market can simplify and automate parts of the process. Nevertheless, some challenges remain, like the need for domain expertise, the risk of inconsistency, and the error proneness of the process.

When the historical datasets are static, the labeling is done once. So, for example, the problem of classifying images as cats or dogs probably won't change anytime soon. But when the data is dynamic, for instance, in a face or finger recognition application, new people can be added any day. In such cases, the labeling solution must be part of the application. For example, new users can take their pictures and attach their ID (for the application to verify their identity). If an image is not classified, it should alert or fall into a manual identification flow. When new pictures are added, the model training process needs to be triggered, and the online models must be refreshed to take the new images into account.

Data can be associated with labels and tags during ingestion time. For example, images arrive from a car along with metadata (car ID, model, driver, etc.) and telemetry (geolocation, timestamp, speed, weather, sensor metrics, etc.). This information should be stored and linked to the image and can be used to generate labels.

When considering MLOps with automated (re)training flow, you should consider a mechanism for automated labeling. In some applications, the labels arrive in a delay (for example, if the user churned, if the stock price went up, or if the customer purchased the product). Therefore, the training dataset should be shifted to accommodate the delay (if we re-train the churn model based on the last three months, the data range should be between 4 and 1 month ago).

Feature Stores

As we've established, most of the complexities in any ML project arise from the data:

- Work is typically done in silos (data scientists and engineers).
- Labor-intensive data engineering to produce high-quality features.
- Duplicate efforts and resources in generating offline and online features which also lead to inaccurate results

- Hard to incorporate data versioning and governance
- Feature development work is duplicated for every new project.
- Lack of simple access to production-ready features at scale.
- Disjointed or nonexistent model and feature monitoring.

ML teams need a way to continuously deploy AI applications in a way that creates real, ongoing business value for the organization. Features are the fuel driving AI for the organization and feature stores are the architectural answer that can simplify processes, increase model accuracy and accelerate the path to production.

A feature store provides a single pane of glass for sharing all available features across the organization along with their metadata. When data scientists start a new project, they can access this catalog and easily find features. But a feature store is not only a data layer; it is also a data transformation service enabling users to manipulate raw data and store it as features ready to be used for offline (training) and online (serving) without duplicating the work. In addition, some feature stores support strong security, versioning, and data snapshots, enabling better data lineage, compliance, and manageability.

Some of the largest tech companies that deal extensively with AI have built their own feature stores (Uber, Twitter, Google, Netflix, Facebook, Airbnb, etc.). The open-source and commercial landscape for feature stores has exploded in the last few years. This is a good indication to the rest of the industry of how important it is to use a feature store as a part of an efficient ML pipeline.

Most feature stores are limited to structured data handling (ML), but some can support both structured data and unstructured (text, documents, images, audio, etc.).

Feature stores are described in detail later in the book. As illustrated in **Figure 2-6**, they provide a mechanism to read data from various online or offline sources, conduct a set of data transformations, and persist the data in

online and offline storage. Features are stored and cataloged along with all their metadata (schema, labels, statistics, etc.), allowing users to compose “Feature Vectors” (join multiple features from different feature sets) and use them for training or serving. The Feature Vectors are generated when needed, taking into account data versioning and time correctness (time traveling). Different engines are used for feature retrieval, real-time engine for serving, and batch one for training.

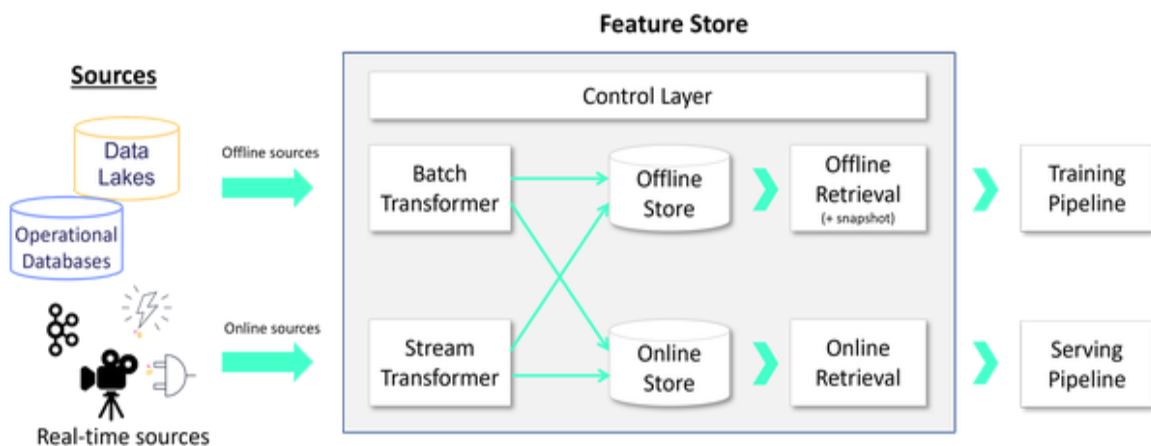


Figure 2-6. Common Feature Store Architecture

The major benefits of a feature store include:

- Faster development with far fewer engineering resources
- Smooth migration from development to production.
- Increased model accuracy (same pipeline for online and offline).
- Better collaboration and security across teams.
- The ability to track lineage and address regulatory compliance.

It is important to note that not all feature stores are born equal. Some are focused on cataloging and don't automate the process of ingestion and online or offline transformation, which are the most labor-intensive tasks. Therefore, make sure you conduct a proper evaluation before selecting a solution.

Model Development & Training

Data scientists generally go through the following process when developing models:

1. Extract data manually from external sources
2. Data labeling, exploration and enrichment to identify potential patterns and features
3. Model training and validation
4. Model evaluation and testing
5. Go back to step 1 and repeat until the desired outcomes have been achieved

The traditional way is to use notebooks, small-scale data, and manual processes, but this does not scale and is not reproducible. Furthermore, in order to achieve maximum accuracy, experiments often need to be run with different parameters or algorithms (AutoML).

With MLOps, ML teams build machine learning pipelines that automatically collect and prepare data, select optimal features, run training using different parameter sets or algorithms, evaluate models, and run various model and system tests. All the executions, along with their data, metadata, code, and results, must be versioned and logged, providing quick results visualization, comparing them with past results, and understanding which data was used to produce each model.

Pipelines can be more complex—for example, when ML teams need to develop a combination of models or use Deep Learning or NLP. You can see a basic model development flow example in [Figure 2-7](#).

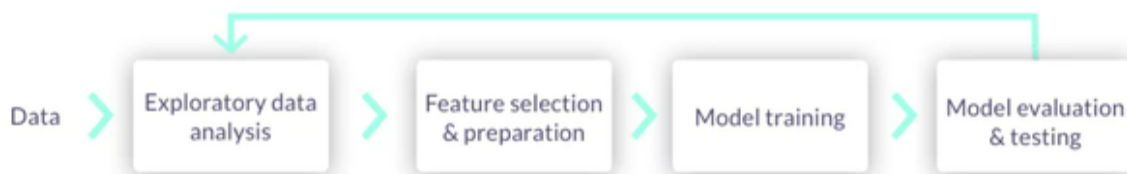


Figure 2-7. Model development flow

ML pipelines can be started manually or preferably triggered automatically when:

1. The code, packages, or parameters change
2. The input data or feature engineering logic change
3. Concept drift is detected, and the model needs to be retrained with fresh data

ML pipelines:

- Are built using microservices (containers or serverless functions), usually over Kubernetes.
- Have all their inputs (code, package dependencies, data, parameters) and the outputs (logs, metrics, data/features, artifacts, models) tracked for every step in the pipeline in order to reproduce or explain our experiment results.
- Use versioning for all the data and artifacts used throughout the pipeline.
- Store code and configuration in versioned Git repositories.
- Use Continuous Integration (CI) techniques to automate the pipeline initiation, test automation, review, and approval process.

Pipelines should be executed over scalable services or functions, which can span elastically over multiple servers or containers. This way, jobs complete faster, and computation resources are freed up once they are complete, saving high costs.

The resulting models are stored in a versioned model repository along with metadata, performance metrics, required parameters, statistical information, etc. Models can be loaded later into batch or real-time serving microservices or functions.

Writing and Maintaining Production ML Code

Many data scientists like the usability and interactivity of Jupyter notebooks when they develop and evaluate models. It is convenient indeed to manipulate some code and immediately see a visual table or a chart, and most ML tutorials, examples, and Kaggle projects are consumed as Notebooks.

You can find projects where the data preparation, training, evaluation, and even prediction are all made in one huge notebook, but this approach can lead to challenges when moving to production, for example:

- Very hard to track the code changes across versions (in Git).
- Almost impossible to implement test harnesses and unit testing.
- Functions cannot be reused in various projects.
- Moving to production requires code refactoring and removal of visualization or scratch code.
- Lack of proper documentation.

The best approach is to use functional programming for code segments and Notebooks for interactive and visualization parts. See the following **Example 2-1** example, which implements a data preparation function that accepts a dataset (dataframe) and some properties as inputs and returns the manipulated dataset. You can notice that the function is documented and allow users to understand the purpose and usage.

Example 2-1. Data Prep Function (data_prep.py)

```
import pandas as pd

def add_date_features(data, time_column: str = "timestamp",
drop_timestamp: bool = False):
    """Add numeric date features (day of week, hour, month) to a
    dataframe

    :param time_column:    The name of the column containing the
    timestamps to extract from
    :param drop_timestamp: set to True to drop the timestamp column
    from the original dataframe
    :return dataframe
```

```

"""
timestamp = pd.to_datetime(data[time_column])
data["day_of_week"] = timestamp.dt.day_of_week
data["hour"] = timestamp.dt.hour
data["month"] = timestamp.dt.month
if drop_timestamp:
    data.drop([time_column], axis=1, inplace=True)
return data

```

Place the function in a separate python file (data_prep.py), and now you can call it from the notebook, inject data and examine or visualize its output using the following code cell:

```

import pandas as pd
from data_prep import add_date_features

df = pd.read_csv("data.csv")
df = add_date_features(df, "timestamp", drop_timestamp=True)
df.head()

```

Once the code is well defined, we can use the python test framework (pytest) and implement unit testing for each of the functions in the following way:

Example 2-2. Data Prep Test Function (test_data_prep.py)

```

import pytest
import data_prep
import pandas as pd

# tell pytest to test both drop values (True/False)
@pytest.mark.parametrize("drop_timestamp", [True, False])
def test_add_date_features(drop_timestamp):
    df = pd.DataFrame({'times': ['2022-01-01 08:00', '2022-02-02 09:00', '2022-03-03 10:00'],
                        'vals': [1, 2, 3]})
    new_df = data_prep.add_date_features(df, "times",
                                         drop_timestamp=drop_timestamp)

    # verify the results are as expected
    assert new_df["day_of_week"].to_list() == [5, 2, 3]
    assert new_df["month"].to_list() == [1, 2, 3]
    assert new_df["hour"].to_list() == [8, 9, 10]
    assert ("times" in new_df.columns.values) != drop_timestamp

```

The code in **Example 2-2** will execute the `add_date_features()` function with different input options and verify that the outputs are correct.

Using this approach, you gain some immediate benefits like:

- Easily see changes to your data prep code in the version control
- The same code can be tested later with a test harness (e.g., using `pytest`)
- The function can be moved to production without the need to refactor the notebook.
- The function is documented, and you can easily understand how to use it and what to expect.
- The function can later be saved to a shared library and used across different projects.
- The code becomes more readable.

Another benefit of the functional approach will be demonstrated in the following chapters: an automated way to convert development code into production services and pipelines using tools such as **MLRun** (MLOps orchestration framework).

Tracking and Comparing Experiment Results

When running ML experiments, it is essential to track every run, this way, you can reproduce experiment results (for example, which parameters and inputs yield the best results), visualize the various metrics, and compare the results of different algorithms or parameters sets.

Each execution involves input and output datasets. It is crucial to track and version the datasets and not only the parameters. Any MLOps solution should provide a mechanism to version data and track the data propagation (lineage) together with the rest of the execution parameters, outputs, and metadata.

Today various open-source and commercial frameworks track the results of every experiment run, store it in a database, and visualize it. Some examples shown in [Figure 2-8](#) include [MLFlow](#), [Weights & Biases](#), [MLRun](#) and [ClearML](#).

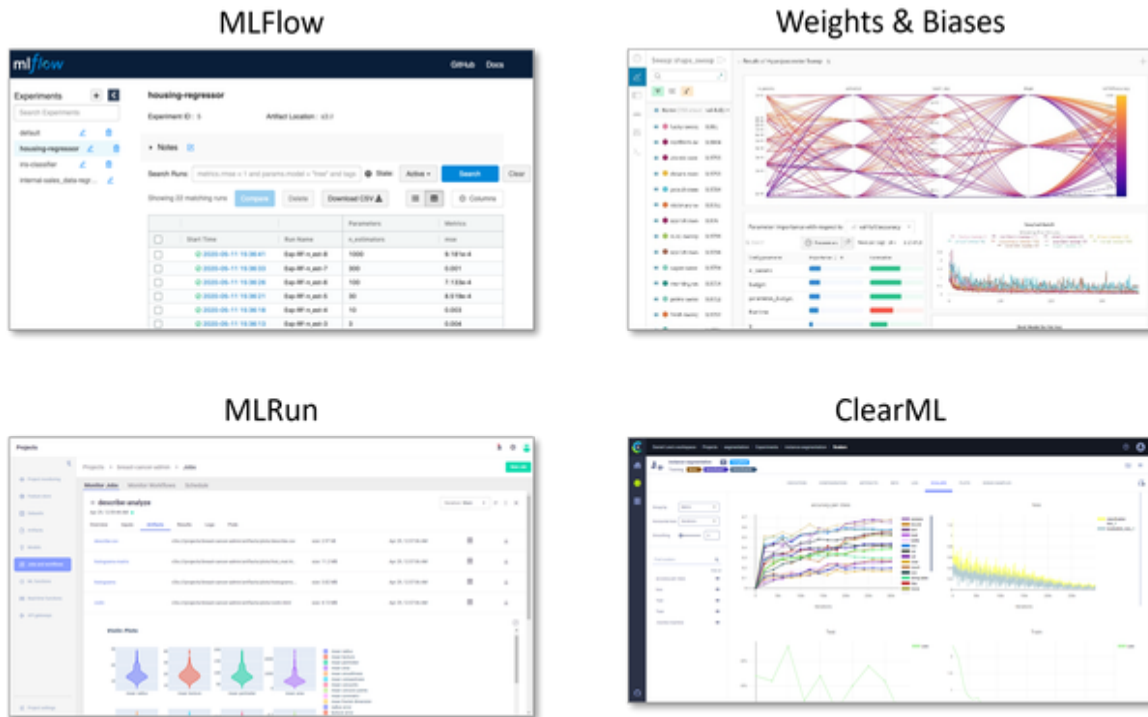


Figure 2-8. Different Tools for ML Execution Tracking

In the real world, experiments can run in an automated ML pipeline (see [Figure 2-9](#)), which comprises different steps (data prep, train, test, etc.). Each stage of the pipeline accepts parameters and inputs data and generates results such as output values, metrics, and data, to be used in subsequent pipeline steps. In addition, the tracking should be extended to operational data (which code was used, packages, allocated and used resources, systems, etc.).

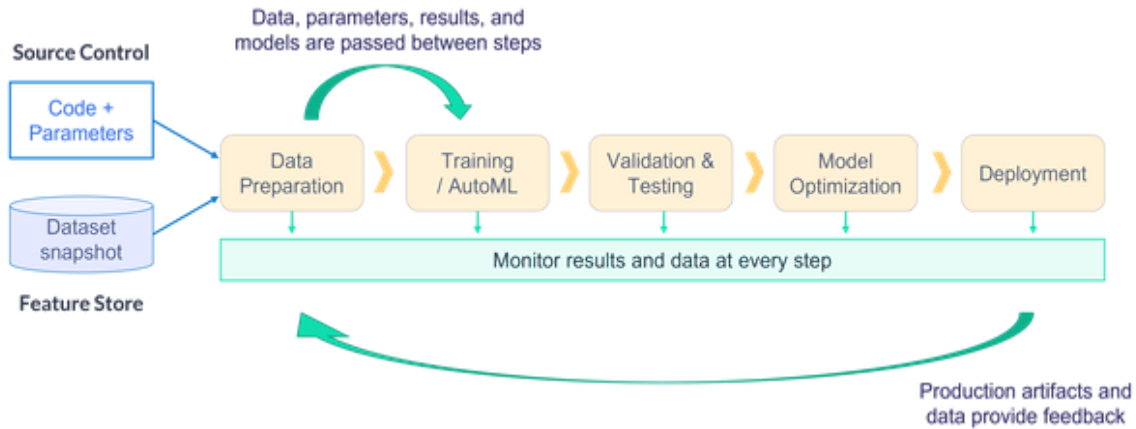


Figure 2-9. Multi-stage (Pipeline) Execution Tracking

In figure **Figure 2-10** you can see the general architecture of an execution tracking system. Inputs may include parameters, the user or system-defined tags (to allow filtering and comparisons), secrets (hidden credentials used by the execution), and data objects (files, tables, etc.). Outputs include the result metrics, logs, usage data, output data objects, and artifacts. A good tracking system also records the code version, used packages, runtime environment and parameters, resources, code profiling, etc.

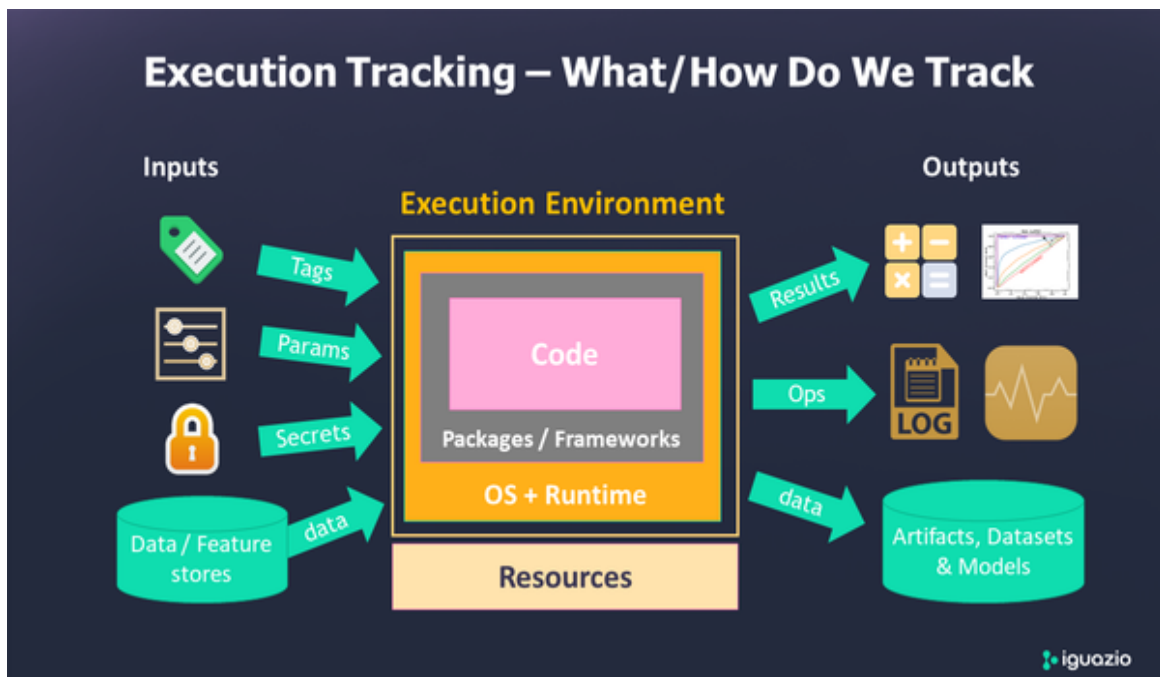


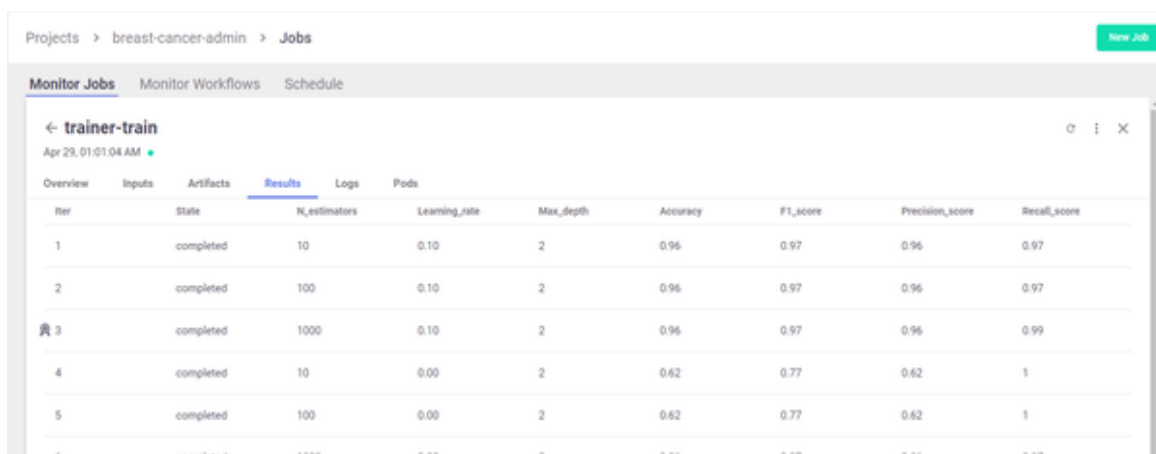
Figure 2-10. Execution Tracking: What and How do we Track

The downside of execution tracking is that it requires code instrumentation (adding code to explicitly log parameters, tags, results, and data). Some MLOps frameworks provide Auto-Logging for ML/DL workloads where you can import a library that automatically records all the ML framework-specific metrics.

A new technology, AutoMLOps, is pioneered in the MLRun framework, which not only records metrics along with the parameters, data lineage, code versioning, and operational data. It also automatically adds production features for auto-scaling, resource management, auto-documentation, parameter detection, code profiling, security, model registry, etc. Eliminating significant engineering efforts.

Distributed Training and Hyper-Parameter Optimization

To get to the best model results, there is a need to try out various algorithms or parameter combinations and choose the best one based on a target metric like best accuracy. This work can be automated using multiple hyper-parameter optimization and AutoML frameworks, which try out the different combinations, record all the metrics for each run and mark the best. To shorten training time, some frameworks support executing each individual run on a different compute resource. **Figure 2-11** shows the tracking of multiple children runs in a hyper-parameter job and the best-selected result.



Projects > breast-cancer-admin > Jobs

Monitor Jobs Monitor Workflows Schedule

← trainer-train Apr 29, 01:01:04 AM

Overview Inputs Artifacts Results Logs Pods

Iter	State	N_estimators	Learning_rate	Max_depth	Accuracy	F1_score	Precision_score	Recall_score
1	completed	10	0.10	2	0.96	0.97	0.96	0.97
2	completed	100	0.10	2	0.96	0.97	0.96	0.97
3	completed	1000	0.10	2	0.96	0.97	0.96	0.99
4	completed	10	0.00	2	0.62	0.77	0.62	1
5	completed	100	0.00	2	0.62	0.77	0.62	1
6	incomplete	1000	0.00	9	0.96	0.97	0.96	0.97

Figure 2-11. Execution Tracking of a Hyper Parameter Job (in the MLRun Framework)

Parallel Hyper-parameter jobs are not limited to model training. They can be used for parallel loading and preparation of many data objects, parallel testing of different test cases, etc.

There are several hyper-parameter execution strategies:

- **Grid Search** - running all the parameter combinations.
- **Random** - running a sampled set from all the parameter combinations.
- **Bayesian optimization** - build a probability model of the objective function and use it to select the most promising hyperparameters to evaluate in the true objective function.
- **List** - running the first parameter from each list followed by the second from each list and so on

You can specify selection criteria to select the best run among the different child runs (for example, the model's accuracy) and the stop condition to stop the execution of child runs when some criteria, based on the returned results, are met (for example: stop condition="accuracy>=0.9").

Some data engineering, ML or DL jobs cannot fit into a single container or virtual machine and must be distributed across multiple containers. Few open-source frameworks (like Spark, **Dask**, **Horovod**, and Nuclio) support workload distribution. When distributing the workload in combination with the parallel run of child (hyper-parameter) tasks, you need to control and limit the total amount of resources used.

Tracking a distributed workload may be more challenging. Make sure the MLOps framework you use supports that.

Building and Testing Models For Production

When models are used in real-world applications, it is critical to ensure they are robust and well-tested. Therefore, in addition to traditional software testing (unit tests, static tests, etc.), tests should cover the following categories:

Data quality tests

The dataset used for training is of high quality and does not carry bias

Model performance tests

The model produces accurate results

Serving application tests

The deployed model along with the data pre/post-processing steps are robust and provide adequate performance

Pipeline tests

Ensuring the automated development pipeline handles various exceptions and the desired scale.

When the training dataset is of low quality, you may presume that the model is accurate, but it can make harmful predictions. Therefore, it is essential to validate that the data is high quality. Here are some examples of data quality tests:

- There are no missing values
- Values are of the correct type and fall under an expected range (for example, user age is between 0-120, with anticipated average and standard deviation)
- Category values fall within the possible options (for example, city names match the options in a city name list)
- There is no bias in the data (for example, the gender feature has the anticipated percentage of men and women).

The data quality tests can be implemented in the data pipeline (and feature store) or in the ML pipeline before the training. Note that some feature stores automate the data quality validation using built-in functions.

Once we train the model, the next step is to make sure the model is accurate and resilient. Beyond the common practice of setting aside a test dataset and measuring the model accuracy using that dataset, here are several additional tests which can improve the model quality:

- Verify the performance is maintained across essential slices of the data (for example, devices by model, users by country or other categories, movies by genre) and that it does not drop significantly for a specific group
- Compare the model results with previous versions or a baseline version and verify the performance does not degrade
- Test different parameter combinations (hyper-parameter search) to verify we choose the best parameter combination
- Test for bias and fairness - verify that the performance is maintained per gender and specific populations.
- Check feature importances and if there are features with a marginal contribution that can be removed from the model.
- Test for immunity to fake, random, or malicious input vectors to increase robustness and defend against adversarial attacks.

Particular attention should be given to how we generate the test set independently that considers fairness and unbiased and minimizes the dependencies with the training set.

When the models are deployed into production serving applications, they contain additional data pre or post-processing logic (extraction, formatting, validation, transformations, API integration, etc.). In addition, the model code may depend on various software packages or infrastructure (memory, CPUs, GPUs, etc.). Therefore, models must be thoroughly tested in their target serving application environment and through the API before they are deployed into the production environment.

Here are some examples for serving application tests:

- API coverage - all serving APIs behave as expected
- Performance tests - verify the serving application can sustain the target number of requests per second and respond within the required latency.
- Package consistency - verify that the model training and serving are using the same framework version (for example, Sklearn).
- Test data validation logic - verify the model endpoint fails or logs the request if improper data is sent to the model.
- Test resiliency - test that the serving application can sustain malicious attacks and impersonation.
- Test correctness - verify that the model prediction results via the serving API are the same as those in the model validation step.
- Test the outcome - verify that prediction results translate to the proper action (writing to a database, generating an alert, updating the user interface, etc.)

The different tests should all be part of an automated CI/CD pipeline. Every time the dataset or code changes, the pipeline is executed and produces a new set of deployable objects (models, applications, features, etc.) and logs all the results to enable reproducibility and explainability.

Some attention should be given to testing the ML pipeline, ensuring it will run correctly every time it's triggered, will not fail due to missing parameters or inadequate resources, and can handle data at scale.

Once the model and other production artifacts are ready, they must be stored in a versioned artifacts repository along with all their metadata and the parameters required to generate the production deployments.

In many cases, the trained model can be further optimized for production and higher performance, for example, by performing feature selection and removing redundant features or by compressing the models and storing them in more machine-efficient formats like **ONNX**. Therefore, ML Pipelines may incorporate model optimization steps.

Figure 2-12 illustrates how different test and optimization steps can be used as part of an ML pipeline.

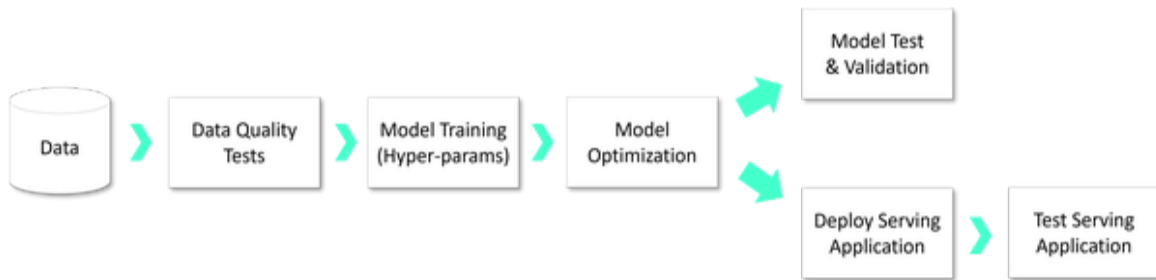


Figure 2-12. Example: Adding Tests & Optimizations to An ML Pipeline

Deployment (and Online ML Services)

Once an ML model has been built, it needs to be integrated with real-world data and the business application or front-end services. The whole application or parts thereof need to be deployed without disrupting the service. Deployment can be extremely challenging if the ML components aren't treated as an integral part of the application or production pipeline.

ML Application pipelines usually consist of:

- API services or application integration logic
- Real-time data collection, enrichment, validation, and feature engineering logic
- One or more model serving endpoints
- Data and model monitoring services
- Resource monitoring and alerting services
- Event, telemetry, and data/features logging services
- A set of actions following the prediction results

You can see a real-time pipeline example in Figure 2-13.

The different services are interdependent. For example, if the inputs to a model change, the feature engineering logic must be upgraded along with the model serving and model monitoring services. These dependencies require online production pipelines (graphs) to reflect these changes.

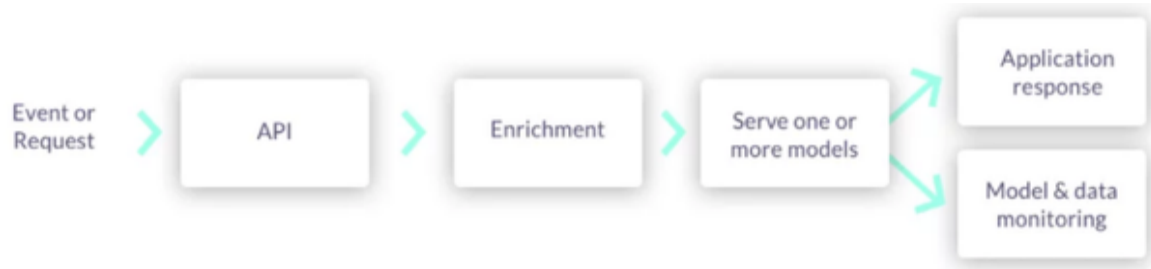


Figure 2-13. Building online ML services

Application pipelines can be more complex when using unstructured data, deep learning, NLP, or model ensembles, so having flexible mechanisms to build and wire up our pipeline graphs is critical.

Application pipelines are usually interconnected with fast streaming or messaging protocols, so they should be elastic to address traffic and demand fluctuations, and they should allow non-disruptive upgrades to one or more elements of the pipeline. These requirements are best addressed with fast serverless technologies.

Application pipeline development and deployment flow:

1. Develop production components:
 - API services and application integration logic
 - Feature collection, validation, and transformation
 - Model serving graphs
2. Test online pipelines with simulated data
3. Deploy online pipelines to production
4. Monitor models and data and detect drift
5. Retrain models and re-engineer data when needed

6. Upgrade pipeline components (non-disruptively) when needed

From Model Endpoints to Application Pipelines

Today's common practice is to build model serving endpoints that merely accept the numeric feature vector and respond with a prediction. The pre or post-processing logic which is usually tightly coupled with the model, is done in separate microservices. This complicates the delivery, scaling, and maintenance of the ML application.

In some cases, the prediction is made using a combination of models, for example, by implementing an ensemble of models which cover different time scopes (recent time and seasonal models) or other algorithms. Another example is cascading two models. The first extracts sentiments from text, and the second makes a prediction based on the sentiments and other features.

A preferred approach is to design online (or real-time) application pipelines where the model serving is just one step in it and be able to deploy, upgrade or roll back that pipeline as a whole. Unlike the data and model training pipelines which run slow batch tasks, the application pipeline should process thousands of requests per second and use streaming or serverless processing engines.

Figure 2-14 Demonstrates a simple application pipeline that accepts a user request (via HTTP or a stream message), processes it, predicts a result using a three-model ensemble, and does post-processing (response to the user, updated the result in a database, generates an alert, etc.)

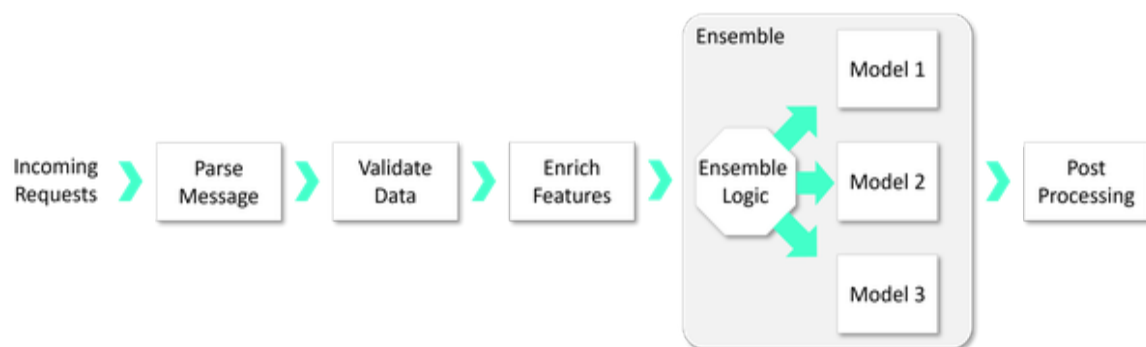


Figure 2-14. Online Application Pipeline Example

Machine learning (ML) or Deep Learning (DL) applications may work with unstructured data and complex processing stages such as image manipulations (detect objects, resize, sample, re-color, crop, etc.) or text manipulations (parse, format, tokenize, etc.). Application pipelines are not limited to structured data. As illustrated in [Figure 2-15](#) a pipeline can branch and process different parts of the data using various technologies and models. In the example, a document URL is sent to the pipeline (via a Kafka stream), the first step fetches the document from an object storage repository, following text and image processing steps, and finally combines the results and updates a search database which hosts the document information.

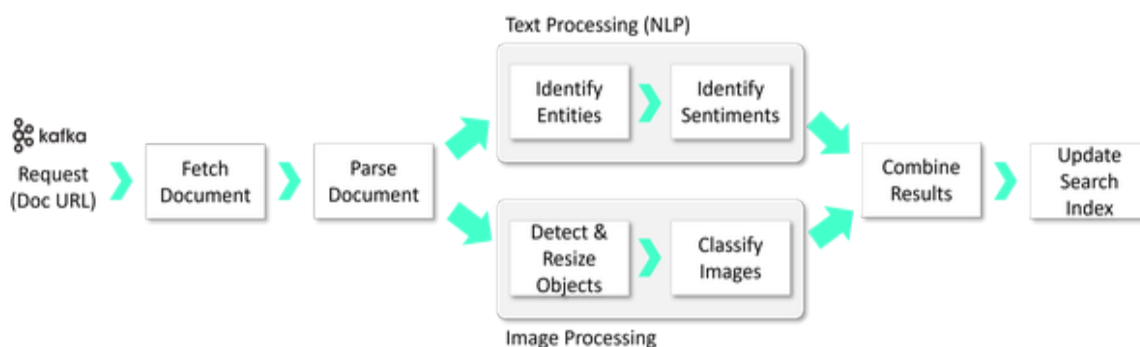


Figure 2-15. Advanced Online Application Pipeline

Some examples of open source and commercial frameworks for building multi-stage online pipelines:

- **AWS Step Functions** - AWS cloud service composing online pipelines from AWS Lambda serverless functions and other AWS cloud services.
- **MLRun Serving Graphs** - Open source and commercial MLOps framework, its serving layer enables the composition of online data and ML/DL pipelines (graphs), provisioned automatically into auto-scaling real-time serverless functions.
- **Apache Beam** - Open source stream processing framework, focused on online structured data processing. (**Google Dataflow** is a managed version of Apache Beam).

- **Seldon** - Open source and commercial model serving framework with basic online pipeline capabilities.

MLRun Serving Graphs provide flexible configuration of pipeline steps breaking into the underline auto-scaling real-time Nuclio serverless functions. As illustrated in **Figure 2-16** You can specify which steps run on the same microservice (eliminating network and serialization overhead) and which must spread across microservices (allowing the use of different OS/Software packages or resources like GPUs by the steps). This enables maximum performance and scalability at minimal costs.

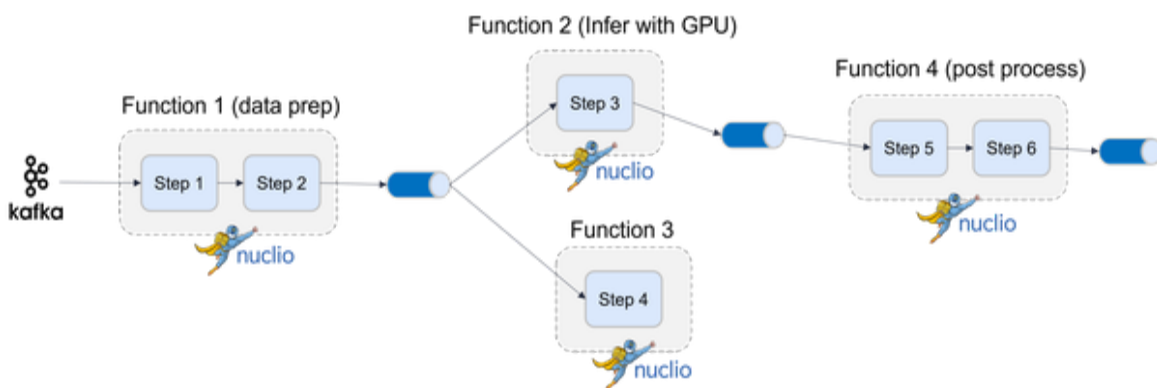


Figure 2-16. MLRun Serving Graph Mapped to Multiple Nuclio Serverless Functions

Online Data Preparation

A dominant part of the online application pipeline is data processing with tasks such as data parsing, formatting, validations, transformations, aggregations, logging, persisting, joining, etc.

Processing data in a batch is quite a common practice. For example, you can use data warehouse queries, ETL processes, Spark, etc. But the same technologies don't work for online pipelines where thousands of events or user requests arrive every second and may need to be answered within milliseconds.

In online data pipelines, the features are accumulated in memory or a fast SQL/NoSQL database, fetched per event to enrich the user request, and passed into the model for prediction. When the features are based on historical or static data (gender, age, annual income, etc.), you can use a

periodic batch process to copy such features to the online database. However, this won't work when the features are frequently updated (current geo-location, last transaction value, money spent last hour, time from the previous login, etc.).

Online data pipelines are implemented using stream processing (Spark Streaming, Flink, [Amazon Kinesis Data Analytics](#), Nuclio, etc.), where events are ingested, transformed, or aggregated on the fly to form the real-time feature vectors. In addition, a fast key/value database is used to persist and share the distributed state. See [Figure 2-17](#), which illustrates how stateful stream processing work. Events arrive and are distributed to stream workers (partitioned by the user key). Each worker processes the data and merges or aggregates it with the accumulated state.

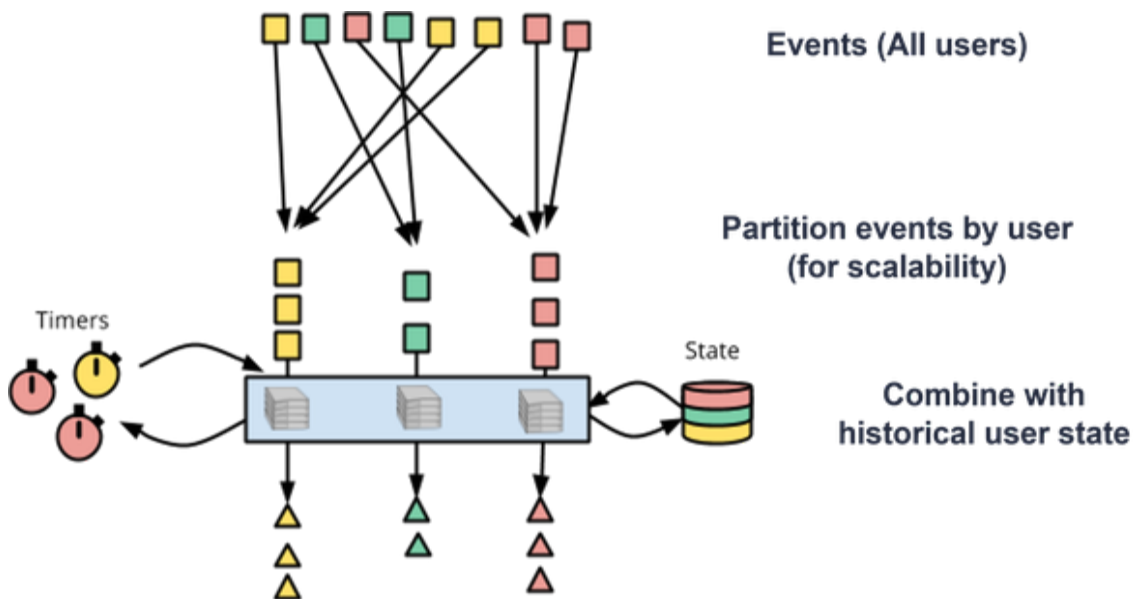


Figure 2-17. How Stateful Stream Processing Work (source: <https://beam.apache.org>)

The major challenge is that stream processing code and methodologies differ quite a bit from batch data analytics approaches and require re-implementing the batch pipeline used for the training into a real-time streaming pipeline. However, some feature stores allow you to define the data pipeline using high-level primitives and automatically generate the batch or streaming pipelines, ensuring the same logic is preserved and saving you a significant engineering effort.

Online data pipelines are not limited to structured data. Modern applications need to process unstructured visual and textual data with operations such as resizing or rotating images, parsing text, tokenizing statements, etc. Therefore, the technology and framework you select need to support such applications.

NOTE

The line between data processing and ML or DL can be blurred. For example, text can be converted to sentiment or a category feature using an NLP model, so is it model serving or a data transformation?

Deploy With Efficiency in Mind

TBD

Continuous Model and Data Monitoring

AI services and applications are becoming an essential part of any business. Poor model performance can lead to liabilities, revenue loss, damage to the brand, and unsatisfied customers. Therefore, it is critical to monitor the data, the models, and the entire online applications pipeline and guarantee that models continue to perform and that business KPIs are met. Thanks to well-implemented monitoring solutions, we can quickly react to the problems by notifying users, retraining models, or adjusting the application pipeline.

Monitoring systems track various infrastructure, data, model, and application metrics and can report or alert on different situations such as:

- Data or concept drift - the statistical attributes of the model inputs or outputs change (an indication that the model will underperform)
- Model performance problems - the results of the model are inaccurate

- Data quality problems - the data provided to the model is of low quality (missing values, NaNs, values are out of the expected range, anomalies, etc.)
- Model bias - Detect changes between the overall scoring and scoring for specific populations (like male and female, minorities, etc.).
- adversarial attacks - malicious attempts have been made to deceive the model.
- Business KPIs - verify that the model meets the target business goals (revenue increase, customer retention, etc.)
- Application performance - The application manages to properly serve requests and without delays
- Infrastructure usage - track the usage of computing resources
- model staleness - alert if it is too long since the last time a model version was deployed
- Anomaly detection - model data or results don't fall under the expected norm or classes (for example, using an encoder-decoder neural network model).

Figure 2-18 shows a typical model monitoring architecture. The data inputs, outputs, and application metrics are sent to a stream. A real-time stream processing application reads the data. It can detect or alert on immediate problems, aggregate the information, and write to various data targets (key/value, time-series, and files or data warehouse).

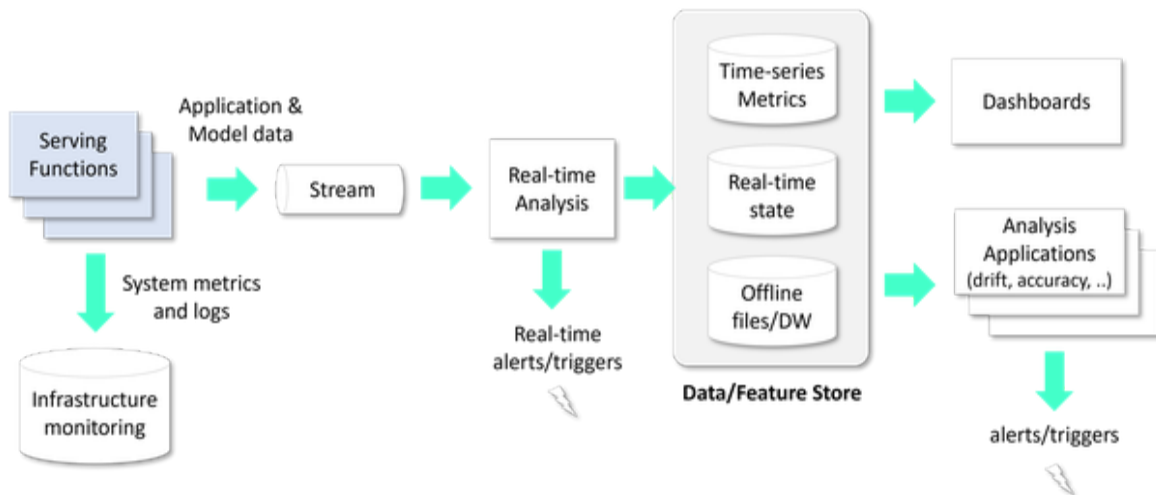


Figure 2-18. Online Model and Data Monitoring Architecture

Alerts generated by the monitoring system can notify users (via emails, slack, etc.) or trigger a corrective action such as retraining a model with newer data, changing model weights, etc.

Feature stores can play a significant part in monitoring data and models. Feature stores store the schema and statistics per feature, which can be used in the different validation and analysis tasks. If the production data is returned to the feature store, it's easier to analyze, join, and compare production datasets with other historical or offline datasets.

Monitoring Data and Concept Drift

Concept drift is a phenomenon where the statistical properties of the target variable (y - which the model is trying to predict) change over time. Data drift (virtual drift) happens when the statistical properties of the inputs changes. In drift, the model built on past data does not apply anymore, and assumptions made by the model on past data need to be revised based on current data.

Figure 2-19 illustrates the differences between concept drift and virtual (data) drift.

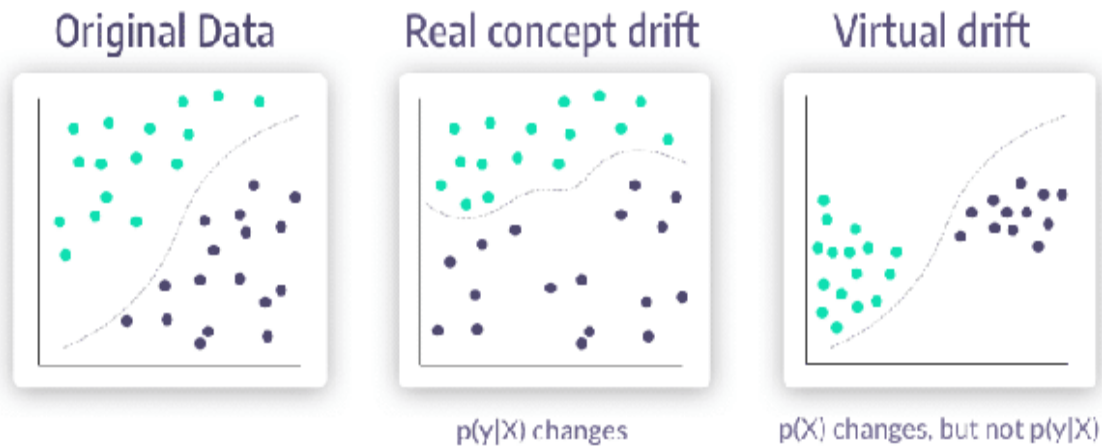


Figure 2-19. Concept Drift Vs. Virtual (data) Drift

Going back to the business level, we can see examples of drift in many use cases:

- Wind power prediction: when predicting the electric power generated from wind from an off-line dataset based model, we have concept drift vs. on-line training models due to the non-stationary properties of winds and weather.
- Spam detection: Email content and presentation change constantly (data drift). Since the collection of documents used for training changes frequently, the feature space representing the collection changes. Also, users themselves change their interests, causing them to start or stop considering some emails as spam (concept drift).

Concept drift changes can be:

Sudden

The move between an old concept and a new one happens simultaneously. The behavioral patterns associated with the COVID-19 pandemic have provided us with striking examples, like the lockdowns that abruptly changed population behaviors worldwide.

Incremental / Gradual

The change between concepts happens over time as the new concept emerges and starts to adapt. The move from summer to winter could be an example of gradual drift.

Recurring / Seasonal

The changes re-occur after the first observed occurrence. An example is a seasonal shift in weather, which dictates that consumers buy coats in colder months, cease these purchases when spring arrives, and then begin again in the fall.

In **Figure 2-20** we can see how model drift detection work. First, the model inputs and outputs are collected, and the system calculates the statistics over a time window and compares them with the sample set statistics (saved at training time) or with the data from an older time window.

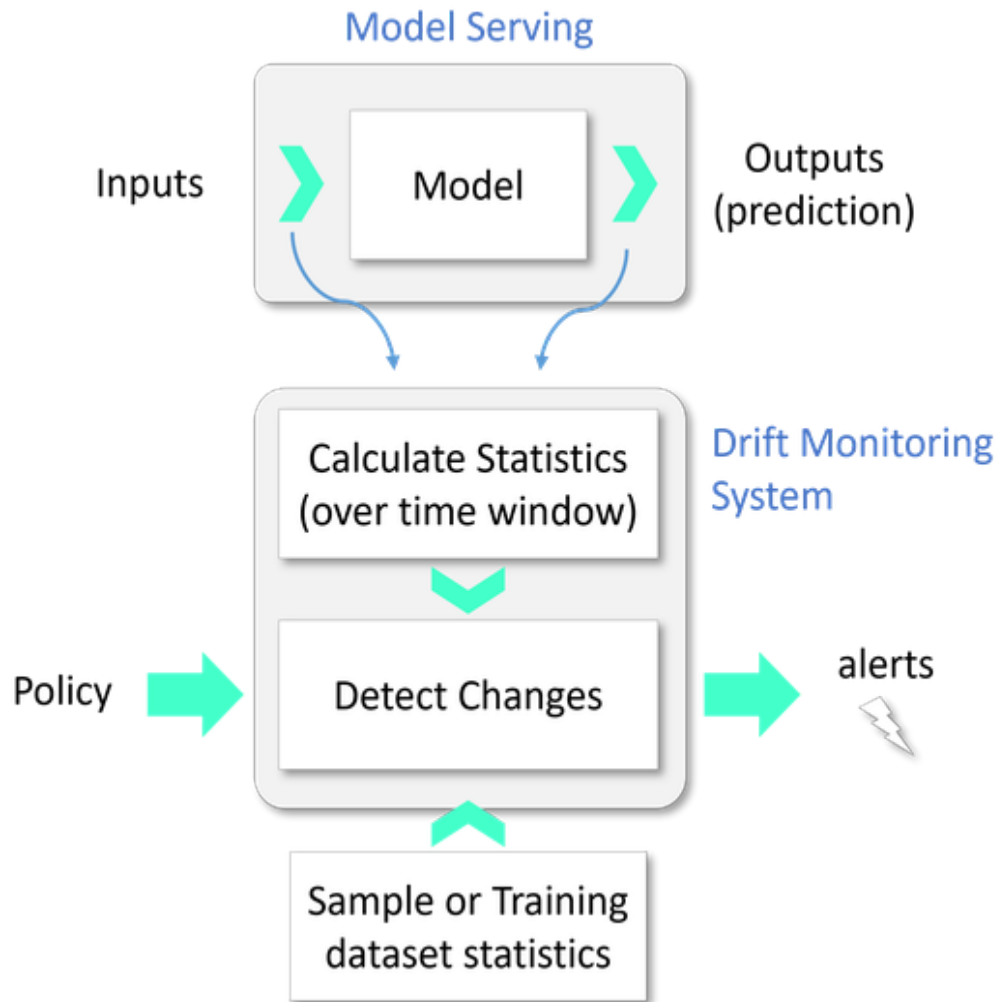


Figure 2-20. Drift Detection Logic

The monitoring system saves the various feature statistics (min, max, average, stddev, histogram, etc.), and the drift level is calculated using one or more of the following metrics:

- Kolmogorov–Smirnov test
- Kullback–Leibler divergence
- Jensen–Shannon divergence
- Hellinger distance
- Standard score (Z-score)
- Chi-squared test

- Total Variance Distance

Figure 2-21 Demonstrates How Drift Can Be Detected

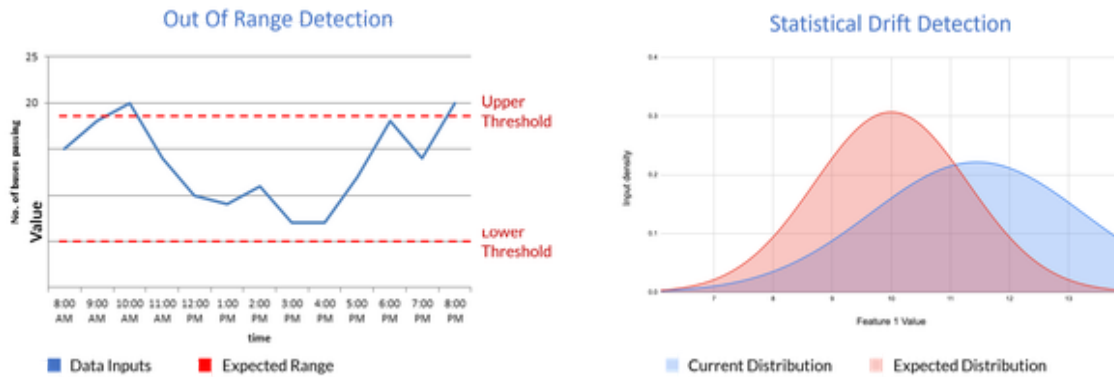


Figure 2-21. Drift Detection Types

Drift is easily detected using those methods when the data consists of simple numeric metrics, but how can it be detected when the data is unstructured, an image, or a piece of text?

One trick is to convert the input data to flat metrics that represent the data and monitor the Drift on those metrics. For example, let's say you classify images of fruits. Then, you can convert the images to their RGB color metrics and check that the color distribution in production is the same as in training.

Monitoring Model Performance and Accuracy

An important metric is to measure model accuracy in production. For that, we must have the Ground Truth (the actual result which matches the prediction). In some models obtaining the ground truth is relatively simple. For example, if we predict that a stock price will go up today, we can wait a few hours and know if the prediction was accurate. This is the same with other prediction applications like predicting customer churn or machine failure where the actual result arrives in some delay.

In some applications, a prediction is made for a specific transaction (for example, exposure or click on an advertisement). The transaction or prediction can be tagged with a UID (unique identifier) in such a case. Once

the actual result is known (the customer bought the product), you can update the transaction (identified by the UID) with the ground truth value. This requires that the model serving and monitoring frameworks will have the ability to store or generate a UID per prediction and add the ground truth values to specific transactions/predictions.

The accuracy monitoring is done periodically (for example, every hour or day). First, a dataset is generated with the predicted Y values (calculated by the model) and the ground truth values (the actual result with the proper time shifting or obtained using the UID). Then, it is used to calculate the accuracy metrics and compare them with the accuracy during training.

This is illustrated in [Figure 2-22](#).



Figure 2-22. Monitoring Model Accuracy In Production

NOTE

The ground truth values calculated for the accuracy monitoring are the same Y labels required for retraining the model. Therefore, the best approach is to generate them once, store them in the feature store, and use them for both retraining and accuracy monitoring.

Just like in training, it is recommended to use several metrics to determine the prediction accuracy, especially in the case when classes are not balanced:

- **Accuracy** - General overall accuracy.
- **Recall** - what fraction of overall positives were correct.

- **Precision** - determine when the costs of False Positive are high.
- **F1 Score** - analysis of the trade-off between Recall and Precision.

It is important to be aware that the ground truth may contain bias. For example, in an application that predicts fraud to approve or reject transactions, the ground truth only includes information on the approved transactions. There is no data about declined transactions that may not have been fraudulent, which can lead to bias.

The Strategy of Pre-Trained Models

One of the most prolific authors on business strategy is the **Harvard Business School professor Michael Porter**, who often said, “The essence of strategy is choosing what not to do.” With most organizations struggling to implement machine learning projects that provide ROI, there is a need for a better strategy. In particular, organizations should ask what they should not be doing while doing machine learning projects. In many cases, they shouldn’t be building a specific type of model and should instead use pre-trained models.

In the book “Understanding Michael Porter (Harvard Business Review Press, 2011)”, the author Joan Magretta summarizes the essence of competitive advantage as outlined by Michael Porter in the following **Figure 2-23**. Companies that compete on execution become part of a prisoner’s dilemma game theory problem where both they and their competitors increasingly lower prices and costs while lowering the company’s profit. And this is the best-case scenario; in many cases, it is impossible for a company to out-execute a bigger rival, say training a better NLP or Computer Vision model.

Competitive Advantage in Companies Value Chain (Understanding Michael Porter-Joan Magretta)

ACTIVITIES	Perform SAME activities as rivals, execute better.	Perform DIFFERENT activities from rivals.
VALUE CREATED	Meet the SAME needs at lower cost.	Meet DIFFERENT and/or same needs at lower cost.
ADVANTAGE	Cost advantage but hard to sustain	Sustainably higher prices and/or lower costs
ADVANTAGE	Be the BEST compete on EXECUTION	Be UNIQUE compete on STRATEGY

Figure 2-23. Competing on Strategy, Not Execution

This conceptual understanding of strategy is the essence of why pre-trained models are an essential component of a holistic strategy to create unique competitive advantages while implementing machine learning projects.

There are several vendors of pre-trained models. The most popular platform is **Hugging Face**, which has over 60 thousand models. Google's **TensorFlow Hub** has a unique collection of pre-trained models in various formats, including formats targeting runtimes like **Javascript** or **embedded hardware**

or [mobile](#). One more format and repository is [ONNX](#) which contains many examples of pre-trained computer vision and language models.

With strategy out of the way, let's explore the concept of pre-trained models by building an end-to-end Hugging Face application.

Building an End-to-End Hugging Face Application

The best way to understand pre-trained models is to build an end-to-end solution with one. Fortunately, HuggingFace makes it simple to do this. First, you need to [sign up for a free account](#).

NOTE

You can view a walkthrough of this Hugging Face application on [YouTube](#) or the [O'Reilly platform](#). The application source code is in [Github](#).

Next, let's look at the application architectures in [Figure 2-24](#). A user account creates an authentication token that later becomes part of a continuous delivery pipeline in a cloud-based build system in [Github Actions](#). The code itself develops in Github CodeSpaces. A Hugging Face model then lives inside a Gradio application, allowing for quick prototyping of an MLOps workflow by providing a user interface. Finally, the Hugging Face Spaces functionality allows users to create applications hosted on the platform using [Gradio](#) a technology for building machine learning apps.

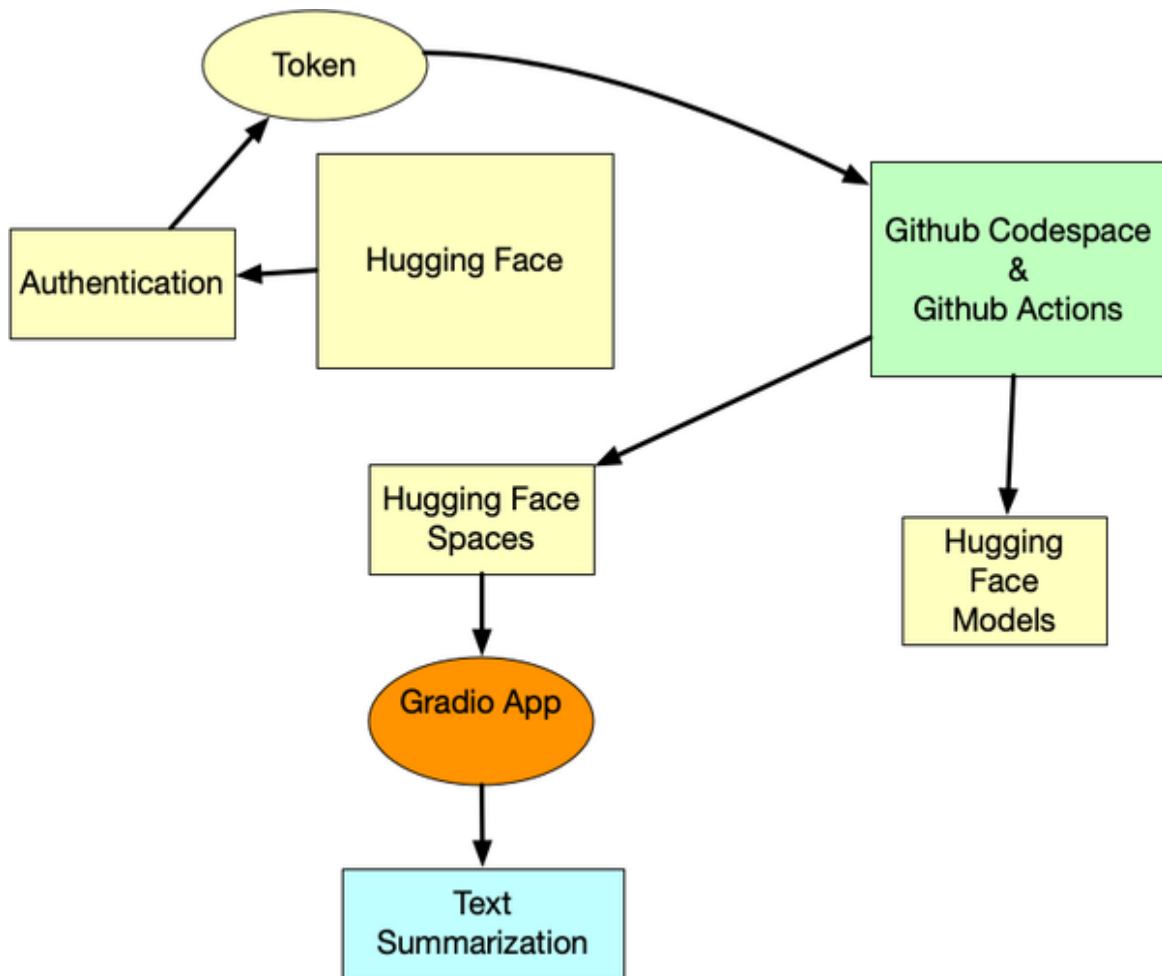


Figure 2-24. MLOps Prototyping with Hugging Face Pre-Trained Models

Let's break each core application file; first there is the `app.py`. <1> Use hugging face transformers (pre-trained model) <2> Create the predict function <3> Build the gradio UI

```

from transformers import pipeline
import gradio as gr

model = pipeline("summarization")❶

def predict(prompt): ❷
    summary = model(prompt)[0]["summary_text"]
    return summary

with gr.Blocks() as demo: ❸
    textbox = gr.Textbox(placeholder="Enter text block to summarize", lines=4)
    gr.Interface(fn=predict, inputs=textbox, outputs="text")
  
```

```
demo.launch()
```

The other key file is `main.yml`, which controls the continuous delivery to Hugging Face. The actions are as follows:

- ❶ On push to Github build the project
- ❷ Use the Hugging Face authentication token

```
name: Sync to Hugging Face hub
on:
  push: ❶
    branches: [main]

# to run this workflow manually from the Actions tab
workflow_dispatch:

jobs:
  sync-to-hub:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0
      - name: Add remote
        env: ❷
          HF: ${{ secrets.HF }} Use the token from Hugging Face
        run: git remote add space <your account>
      - name: Push to hub
        env:
          HF: ${{ secrets.HF }}
        run: git push --force <your account>
```

Finally, with the build process set up, you can see the working application in [Figure 2-25](#). Any text passed into the submit box is then summarized using the Hugging Face pre-trained model. Later different models could be swapped out with just a line of code changed, and the entire application and the model would go live. A key takeaway is pre-trained models deployed in this MLOps fashion allow for rapid prototyping of what could later become a more sophisticated MLOps system.

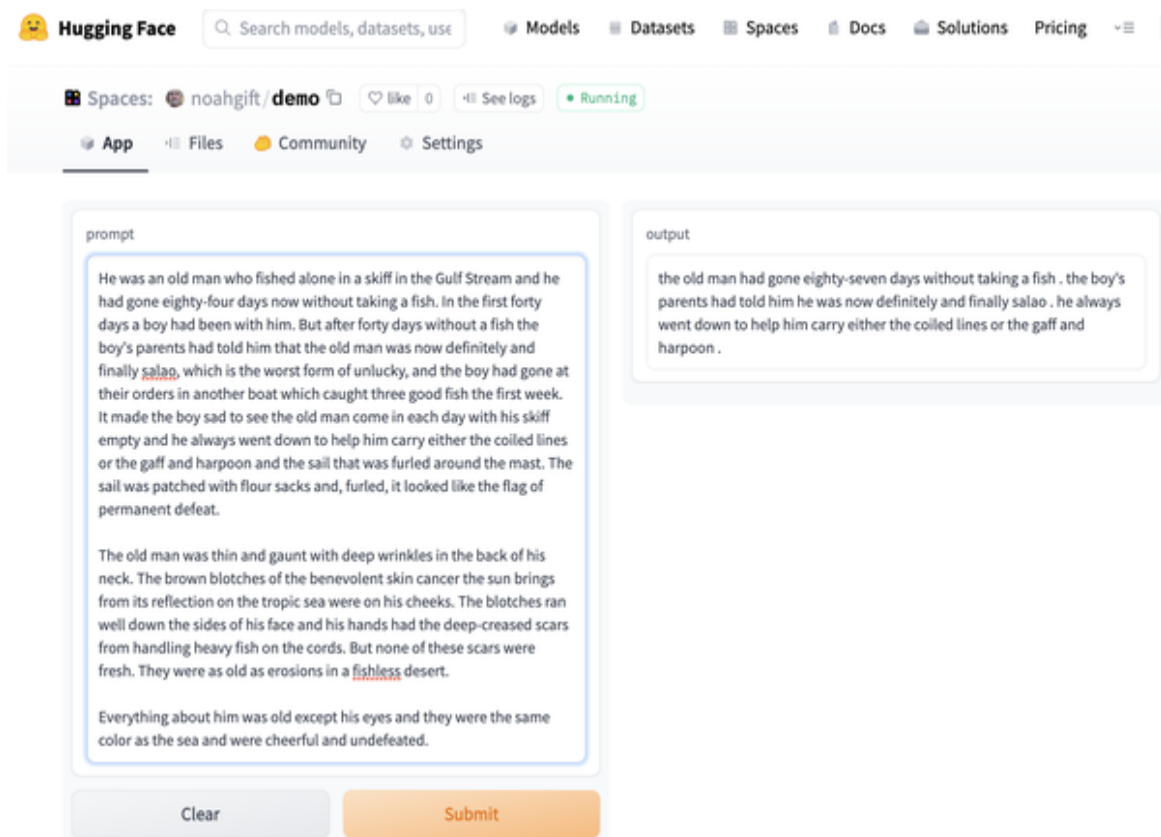


Figure 2-25. Gradio Application Summarizing “Old Man and the Sea” Text

Flow Automation (CI/CD for ML)

Continuous integration and continuous delivery (CI/CD) is an agile development approach for managing the life cycle of software and continuously deploying robust code updates to production. Using CI/CD, multiple developers can contribute code updates to a shared project repository, conduct automated testing, and have a controlled and continuous deployment process. The outcome is faster time to market using fewer resources and lower software failure rates.

The development of ML models and applications brings additional challenges which are not present in traditional software development:

- Multiple personas participate in the development (data scientists, data engineers, software developers, ML engineers, etc.), each with different development skills, tools, and practices.

- A "Version" definition extends beyond code and incorporates data source objects, parameters, and multiple artifacts.
- The different data and ML workloads (data preparation, model training, model, data and application testing, etc.) require high scalability and distributed processing using CPUs and GPUs.
- Deployment of new versions to production involves merging different data assets and states (tables may change the schema, streams may be partially processed, new features are added and require historical values or imputing missing values, etc.).
- Monitoring and observability are far more complex and less deterministic (as discussed in the previous section about the model and data monitoring)

To address the data and ML-specific challenges, organizations must extend their **CI/CD practices with MLOps automation** practices and ensure the engineering and data science teams are aligned on the same development methodologies and tools, some practices to follow:

- Data scientists' code can no longer be maintained in giant notebooks but rather broken into smaller functional code components (see: **"Writing and Maintaining Production ML Code"**).
- All data, code, parameters, artifacts, and results must be automatically collected, versioned, and correlated (see: **"Tracking and Comparing Experiment Results"**).
- Tests should be extensive and cover all data, model, application aspects (see: **"Building and Testing Models For Production"**)
- Pipelines must support high-performance and distributed processing and efficient movement and versioning of data assets across the pipeline.
- Model and data monitoring solutions should provide a feedback loop and be incorporated into the automation flow (see: **"Continuous Model**

and Data Monitoring”).

The following **Figure 2-26** demonstrate a typical CI/CD flow for ML applications. It consists of three main parts:

Development - a user (data scientists, data engineers, software developers, etc.) creates a development branch from the latest code, adds his features, and conducts local tests using some sample data.

Staging (or integration) - the user requests to merge the new feature into the development branch. At this point, automated test procedures run over the new code, user larger datasets, and distributed or more scalable computation resources. Once the new code passes the tests and is approved, it merges into the development release and may undergo additional stress testing.

Deployment to production - the development release is partially promoted to production (use canary or A/B testing deployment method to process small parts of the actual transactions). Once the new version is verified to work correctly and is compared to the prior release, it is approved and released to production. In case of failures or lower model performance, the system can be rolled back to the previous release.

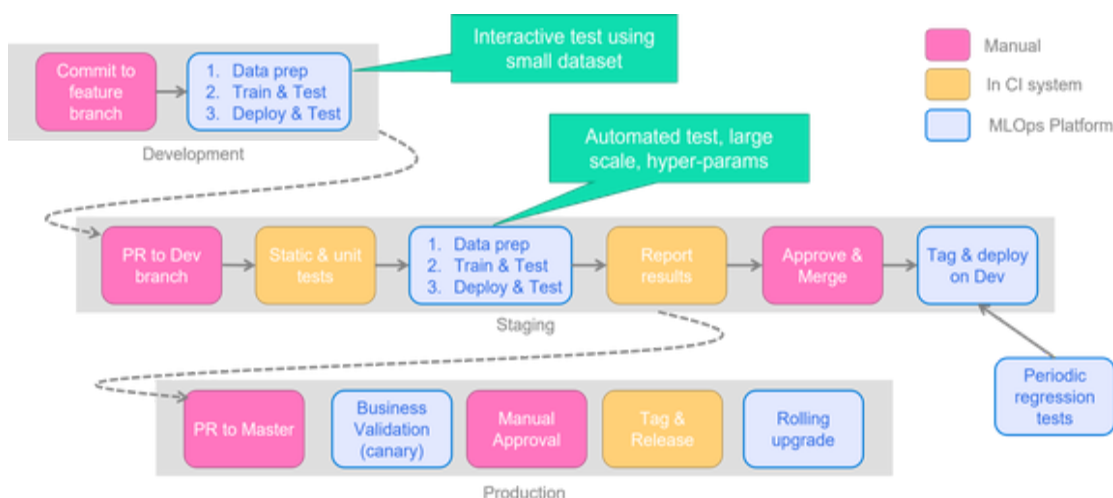
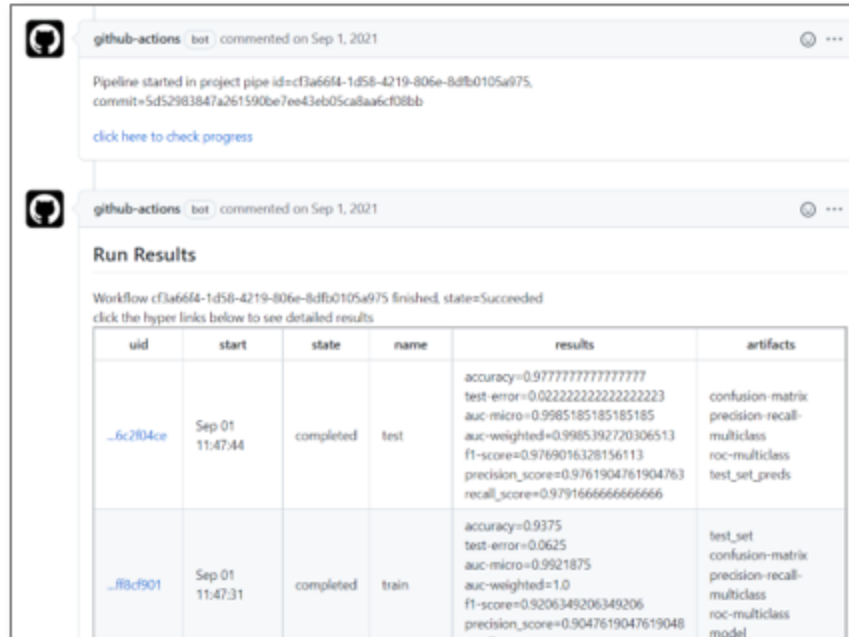


Figure 2-26. Automating The Flow From Development To Production

The MLOps framework must have tight integration with the source control (GIT) and CI/CD framework you choose (**Jenkins**, **GitHub Actions**, **Gitlab**

CI/CD, etc.). Various metadata and configuration objects must be stored in the source repository along with the code, data referencing should be abstract and versioned, and reporting and APIs should be integrated to provide a holistic and avoid manual or complex integrations. You can see an example for integration in the following [Figure 2-27](#).



github-actions bot commented on Sep 1, 2021

Pipeline started in project pipe id=c3a66f4-1d58-4219-806e-8dfb0105a975, commit=5d52983847a261590be7ee43eb05ca8a6c08bb

[click here to check progress](#)

github-actions bot commented on Sep 1, 2021

Run Results

Workflow c3a66f4-1d58-4219-806e-8dfb0105a975 finished, state=Succeeded
click the hyper links below to see detailed results

uid	start	state	name	results	artifacts
...fc2804ce	Sep 01 11:47:44	completed	test	accuracy=0.9777777777777777 test-error=0.02222222222222223 auc-micro=0.9985185185185185 auc-weighted=0.9985392720306513 f1-score=0.9769016328156113 precision_score=0.9761904761904763 recall_score=0.9791666666666666	confusion-matrix precision-recall-multiclass roc-multiclass test_set_preds
...f8bc901	Sep 01 11:47:31	completed	train	accuracy=0.9375 test-error=0.0625 auc-micro=0.9921875 auc-weighted=1.0 f1-score=0.9206349206349206 precision_score=0.9047619047619048	test_set confusion-matrix precision-recall-multiclass roc-multiclass model

Figure 2-27. View Automated Data & ML Test Reports Inside The Version Control (GIT) System

Critical Thinking Discussion Questions

- Why is problem framing the initial suggested step for implementing a project following the MLOPs philosophy?
- Name two or three examples of problems an organization could solve better with a heuristic than with machine learning?
- How could your organization design an effective data governance strategy that proactively prevents PII, bias, or regulatory risk problems?
- How could you use a Feature Store to decrease the computational training time of a model?

- Consider a situation when your organization could face issues with data drift and when it faces problems with concept drift. Which would be the most significant impact if not resolved?

Exercises

- Use `mlrun serving` to serve out a **Hugging Face model**.
- Use a Feature Store on an MLOPs platform to train a model that requires data transformation before training.
- Use an experiment tracking technology like MLFlow, MLRun, ClearML, Sagemaker, or another MLOps platform to train multiple versions of a model and compare the accuracy of numerous runs.
- Use an open-source framework like Spark, Dask, Horovod, or Nuclio for workload distribution to perform distributed hyperparameter tuning.
- Write a serving application test using one of the examples covered earlier in the book.

Table of Contents

[Front Matter](#)

[Implementing MLOps in the Enterprise](#)

[Copyright](#)

[Revision History for the First Edition](#)

[Preface](#)

[Conventions Used in This Book](#)

[Tip](#)

[Note](#)

[Warning](#)

[Using Code Examples](#)

[O'Reilly Online Learning](#)

[Note](#)

[How to Contact Us](#)

[Acknowledgments](#)

[Chapter 1. MLOps - What is it, and why do we need it?](#)

[What is MLOps?](#)

[MLOps in the Enterprise](#)

[Note](#)

[Understanding ROI in Enterprise Solutions](#)

[Figure 1-1. Evaluating Technology Platform Solutions](#)

[ROI](#)

[Figure 1-2. Bespoke System Dilemma](#)

[Understanding Risk and Uncertainty in the Enterprise](#)

[Note](#)

[MLOps vs. DevOps](#)

[Note](#)

[Figure 1-3. Naught Child Problem by Taleb-Data Drift](#)

[Note](#)

[What isn't MLOps?](#)

[Mainstream definitions of MLOps](#)

[What is ML Engineering](#)

[Note](#)

[MLOps and Business Incentives](#)

[Technical & Organizational Challenges - Why Most Models
Never Make it to production](#)

[MLOps in the Cloud](#)

[Figure 1-4. Cloud MLOPs Landscape](#)

[Key Cloud Development Environments](#)

[Note](#)

[Figure 1-5. Google Cloud Shell Editor](#)

[Figure 1-6. Google Cloud Shell Editor API](#)

[Figure 1-7. Google Cloud Shell Terminal](#)

[Note](#)

[Figure 1-8. AWS Cloud Shell Terminal](#)

[Note](#)

[Figure 1-9. Cloud Developer Workspace Advantages](#)

[Note](#)

[The Key Players in Cloud Computing](#)

[Figure 1-10. Cloud Computing Market](#)

[Note](#)

[AWS View of Cloud Computing as it relates to MLOps](#)

[Note](#)

[Figure 1-11. AWS Sagemaker MLOPs workflow](#)

[Note](#)

[Azure View of Cloud Computing as it relates to MLOps](#)

[Figure 1-12. Azure MLOps](#)

[GCP View of Cloud Computing as it relates to MLOps](#)

[Figure 1-13. Google's view of MLOps](#)

[Figure 1-14. Google's view of MLOps](#)

[MLOps On-Premises](#)

[MLOps in Hybrid Environments](#)

[Enterprise MLOps Strategy.](#)

[Figure 1-15. Enterprise MLOps Strategy.](#)

[Conclusion](#)

[Critical Thinking Discussion Questions](#)

[Exercises](#)

[Chapter 2. The Stages of MLOps](#)

[What You Need to Get Started](#)

[Figure 2-1. ML Project Life Cycle](#)

[Choose Your Algorithm](#)

[Figure 2-2. Regression vs Classification](#)
[Design Your Pipelines](#)
[Figure 2-3. ML Pipeline Example: Real-time Product Recommendations](#)
[Data Collection & Preparation](#)
[Data Storage And Ingestion](#)
[Figure 2-4. Offline and online data ingestion flow](#)
[Note](#)
[Data Exploration and Preparation](#)
[Figure 2-5. Feature Engineering Flow](#)
[Data Labeling](#)
[Feature Stores](#)
[Figure 2-6. Common Feature Store Architecture](#)
[Model Development & Training](#)
[Figure 2-7. Model development flow](#)
[Writing and Maintaining Production ML Code](#)
[Example 2-1. Data Prep Function \(data_prep.py\)](#)
[Example 2-2. Data Prep Test Function \(test_data_prep.py\)](#)
[Tracking and Comparing Experiment Results](#)
[Figure 2-8. Different Tools for ML Execution Tracking](#)
[Figure 2-9. Multi-stage \(Pipeline\) Execution Tracking](#)
[Figure 2-10. Execution Tracking: What and How do we Track](#)
[Distributed Training and Hyper-Parameter Optimization](#)
[Figure 2-11. Execution Tracking of a Hyper Parameter Job \(in the MLRun Framework\)](#)
[Building and Testing Models For Production](#)
[Figure 2-12. Example: Adding Tests & Optimizations to An ML Pipeline](#)
[Deployment \(and Online ML Services\)](#)
[Figure 2-13. Building online ML services](#)
[From Model Endpoints to Application Pipelines](#)
[Figure 2-14. Online Application Pipeline Example](#)
[Figure 2-15. Advanced Online Application Pipeline](#)
[Figure 2-16. MLRun Serving Graph Mapped to Multiple Nuclio Serverless Functions](#)

Online Data Preparation

Figure 2-17. How Stateful Stream Processing Work
(source: <https://beam.apache.org>)

Note

Deploy With Efficiency in Mind

Continuous Model and Data Monitoring

Figure 2-18. Online Model and Data Monitoring
Architecture

Monitoring Data and Concept Drift

Figure 2-19. Concept Drift Vs. Virtual (data) Drift

Figure 2-20. Drift Detection Logic

Figure 2-21. Drift Detection Types

Monitoring Model Performance and Accuracy

Figure 2-22. Monitoring Model Accuracy In Production

Note

The Strategy of Pre-Trained Models

Figure 2-23. Competing on Strategy, Not Execution
Building an End-to-End Hugging Face Application

Note

Figure 2-24. MLOps Prototyping with Hugging Face
Pre-Trained Models

Figure 2-25. Gradio Application Summarizing “Old
Man and the Sea” Text

Flow Automation (CI/CD for ML)

Figure 2-26. Automating The Flow From Development To
Production

Figure 2-27. View Automated Data & ML Test Reports
Inside The Version Control (GIT) System

Critical Thinking Discussion Questions

Exercises