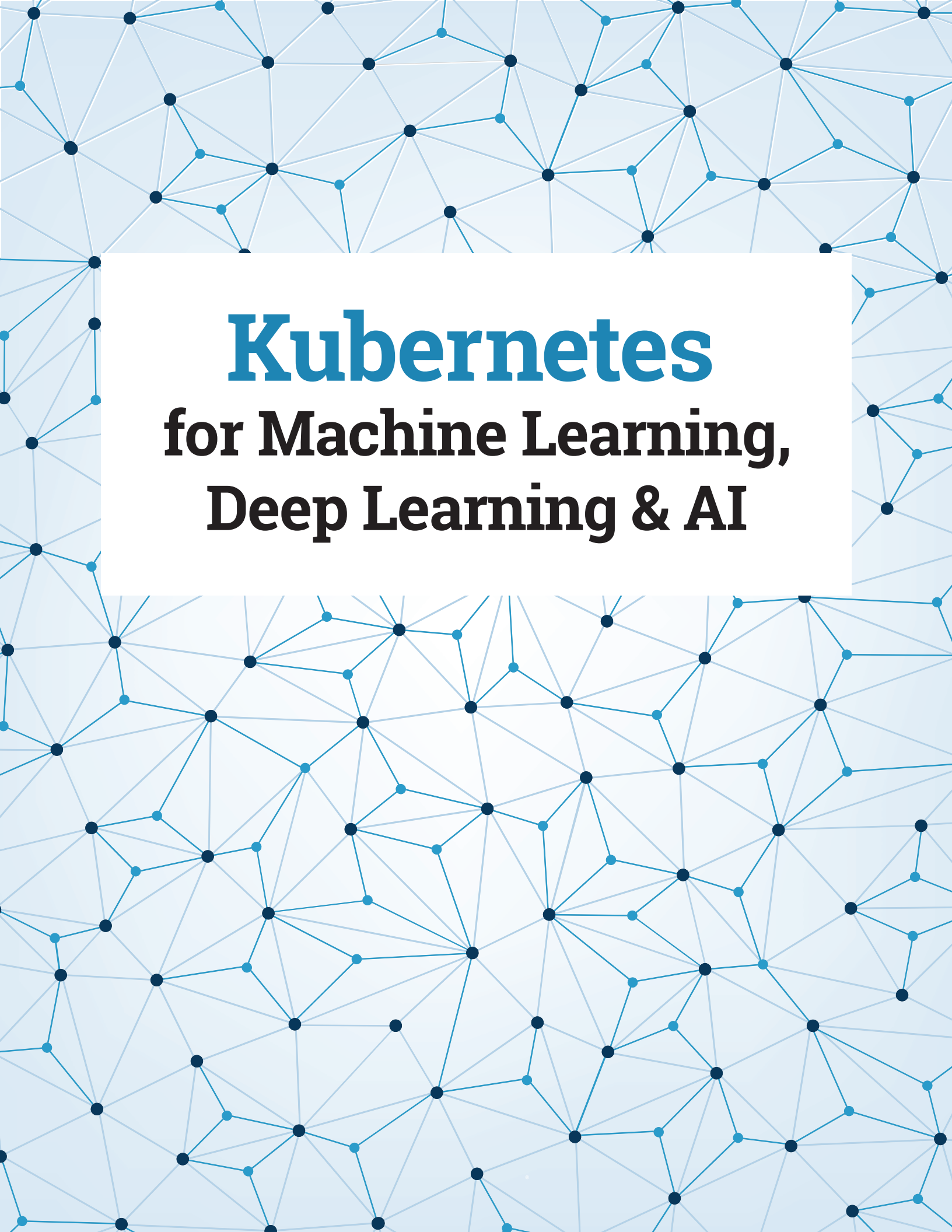

Kubernetes
for Machine Learning,
Deep Learning & AI

Kubernetes
for Machine Learning, Deep Learning & AI

By Sam Charrington
Founder & Host - This Week in Machine Learning & AI
Principal Analyst - CloudPulse Strategies

with

Sriram Subramanian, Contributor

Mia Figueroa, Copy Editor

Table of Contents

Introduction

Machine learning is a subfield of artificial intelligence (AI) that enables computer systems to identify, or **learn**, patterns in data using statistical methods. These learned patterns are captured in a **model** that can be used to make predictions or perform tasks without explicit programming.

The **training** process is central to machine learning (ML). In order to produce a model, we train the system by applying a learning algorithm to a training dataset. If this process is successful, the model will be able to identify the relationship between **features** and **labels** in the training data.* Features refer to specific attributes or characteristics of the training data likely to help the model make accurate predictions. Labels refer to the answers that we want our model to predict given a set of input features. Once a model is trained, it can then be used for inference or scoring; that is, querying the model with new, unlabeled data to predict how it should be labeled.

* **There are several types of machine learning.** While the specific use of labeled training datasets is characteristic of *supervised learning*, as opposed to *unsupervised or reinforcement learning*, the first is the most popular type of machine learning in use today. While this book occasionally refers to labels and other supervised learning concepts, its key messages apply broadly.

For a simple example, consider the problem of predicting home values from available real estate data. We know intuitively that a relationship exists between the characteristics of a home such as its size, number of bedrooms and baths, and location (i.e., our features), and the prices the properties ultimately sell for (i.e., our labels). Yet without using machine learning, it would be very difficult to accurately capture this relationship as a rule or set of rules, especially as we increase the number of features.

This example helps illustrate the importance of machine learning not just as a statistical tool, but as a new way of developing software. This viewpoint has been captured succinctly as "Software 2.0"[1] by Andrej Karpathy, Director of AI at Tesla. Software 2.0 refers to the idea that in traditional programming (i.e., "Software 1.0"), software developers start with some data they want to operate on and a set of explicit rules for processing that data. Those rules are manually encoded as software programs via the software development process. These programs are then used to solve the task at hand, whatever that might be.

In contrast, machine learning (i.e., Software 2.0) allows us to extract the rules for solving some task automatically from a set of features and labels in a training dataset. This is particularly useful in situations for which it would be very difficult for humans to identify, much less program, the complete set of governing rules.

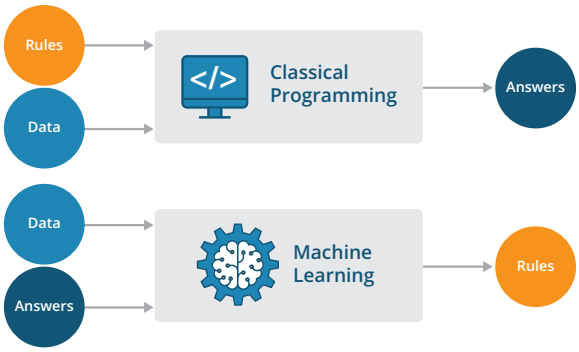


Figure 1. Classical programming vs machine learning

For such tasks, like identifying faces or objects in photos, distinguishing spam emails, or identifying fraudulent transactions, it has proven much more effective to use machine learning models than manually coded programs. With machine learning, we can essentially teach computers how to program themselves using data!

Modern ML is powered by algorithms that, for the most part, have been available for decades. Nonetheless, the field is experiencing dramatic growth due to a number of shifts in the industry. Recent changes include an explosion in the amount of available training data and affordable computing power, advances in how models are trained, and a boom in the amount and quality of tools for developing machine learning solutions.

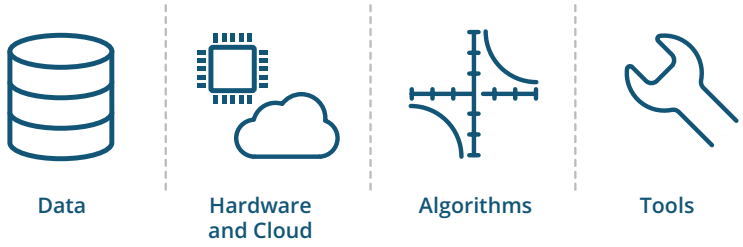


Figure 2. Advances in data, hardware, algorithms, and tools have contributed to the modern ML revolution

A notable subfield of machine learning is deep learning. Deep learning (DL) today largely refers to the use of a specific type of machine learning model—one made up of neural networks with many layers. Deep neural networks (DNNs) can identify very complex patterns in their input data, making deep learning a powerful technique. Today, DNNs represent the state of the art in many applications involving voice, video, and pictures, and are increasingly finding use in natural language processing and tabular data. The unfortunate drawback of deep learning is that training these models can require substantially more labeled data, time, and computational horsepower than traditional ML models.

Over the past several years, enterprises have begun exploring the benefits of applying machine and deep learning to deliver business results in a variety of areas. For example, Home Depot uses a machine learning model to reduce what it calls "shelf outs,"[2] when items exist in in-store inventory but not on store shelves. Satellite imagery startup Descartes Labs applies

machine and deep learning to enable analytics and search over geospatial data.[3] And Booking.com, a travel e-commerce company, uses machine learning to provide travel recommendations to their customers.[4]

Industries spanning healthcare, publishing, finance, manufacturing, and more are increasingly turning to ML and DL to improve facets of their business. Potential applications include everything from decision-making, customer service, recommendations, and even building entirely new offerings like self-driving vehicles.

Many early ML adopters have found that as they grow their internal machine learning capacity, more (and more profitable) ways to apply the technology become apparent. For enterprises seeking to take full advantage of the transformational opportunities offered by machine learning, improving their ability to bring ML-based solutions to more areas of the business can be a worthwhile investment.

The Machine Learning Process

Before we can discuss scaling the delivery of machine learning in the enterprise, we must understand the full scope of the machine learning process. Much of the industry dialogue around machine learning is focused on modeling, but to achieve scale models must be developed within a repeatable process that accounts for the critical activities that precede and follow model development.

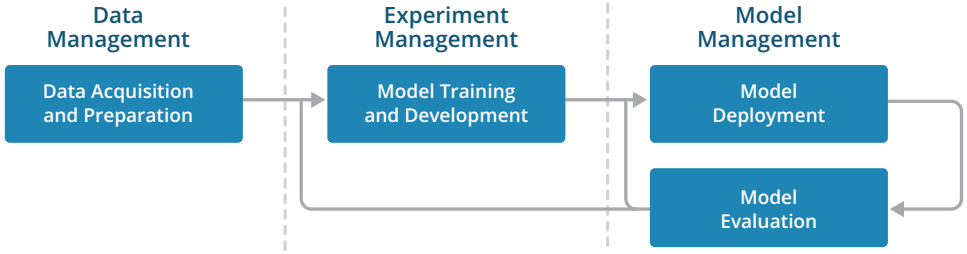


Figure 3. The machine learning process

Data acquisition and preparation

To build models, data scientists and machine learning engineers* must have access to large quantities of high-quality labeled training data. This data very rarely exists in a single place in a form directly usable by data scientists. Rather, in the vast majority of cases, the training dataset must be built by data scientists, data engineers, machine learning engineers, and business domain experts working together.

* **Machine learning engineer (MLE)** is an emerging job title for software engineers specializing in building machine learning systems. MLEs have an understanding of data science and machine learning, though they are perhaps not as strong on the fundamentals as a data scientist. They are much stronger, though, in their understanding of how to build, deploy and support production-quality software.

The creation of training datasets involves combining data from one or more sources. For example, a data scientist building a product recommendation model might build the training dataset by joining data from web activity logs, search history, mobile interactions, product catalogs, and transactional systems.

In a large enterprise, each of these point data sources might come from a distinct silo, with its own owner, access controls and protocols, data format, and compliance requirements.

Once acquired, training data must then be prepared for model development. To do this, data scientists will apply a series of transformations to the raw data to cleanse and normalize it. Examples include removing corrupt records, filling in missing values, and correcting inconsistencies like differing representations for states or countries.

Transformations may also be required to extract labels. For example, developing a model that predicts the likelihood that a customer will be lost, or churn, will require a label indicating which of the customers in our transactional database are examples of churn. This can, in turn, require a complex query against the data warehouse that considers factors such as the products or services that we're basing the prediction on, the number of days without a transaction that we consider churn, the window in which we want to make predictions, and more.[5]

As a result of the organizational and technical complexity involved, the process of acquiring and preparing enterprise data can consume the vast majority of a data scientist's effort on a given project—50 to 80 percent, according to some reports.[6]

Model training and development

A fine line separates the preliminary data cleansing and normalization steps associated with data preparation from **feature engineering**, which is inextricably tied to the modeling process.

Feature engineering is the iterative process of creating the features and labels needed to train the model through a series of data transformations. Feature engineering is often performed in lock-step with model training, because the ability to identify helpful features can have a significant impact on the overall success of the modeling effort. Simple examples of feature engineering include generating derived features (such as calculating an age from a birthdate) or converting categorical variables (such as transaction types) into one-hot encoded, or binary, vectors.

With deep learning, features are usually straightforward because DNNs generate their own internal transformations. With traditional machine learning, feature engineering can be quite challenging and relies heavily on the creativity and experience of the data scientist or MLE and their understanding of the business domain or ability to effectively collaborate with domain experts.

During modeling, data scientists and MLEs run a series of experiments against the available data to identify a robust predictive model. Typically many models—possibly hundreds or even thousands—will be trained and evaluated during modeling in order to identify the architectures, learning algorithms, and parameters that work best for a particular problem.

As a result, training can be quite computationally intensive. Having access to the right infrastructure for each experiment, such as GPUs, can significantly impact the speed and agility of the data science team during training.

Model deployment

Once a model has been developed, it must be deployed for use in applications. This typically involves embedding the model directly into application code or putting it behind an API. REST APIs are increasingly used so that developers can access model predictions as microservices.

Model inference is even more computationally expensive than model training, though they use computing resources very differently.* While training might require large bursts of CPU or GPU over the course of several hours, days or weeks, each inference against a deployed model requires a small but significant amount of computing power. Unlike the demands of training, the computational burden of inference scales with the number of inferences made and continues for as long as the model is in production.

* **Google's efforts to build its Tensor Processing Unit (TPU),** a specialized chip for inference, began in 2013 when engineer Jeff Dean projected that if people were to use voice search for three minutes a day, meeting the inference demands of speech recognition would require Google data centers to double in capacity.[7]

Just like other application components, it is important to manage the lifecycle of deployed models. Managing model versions, comparing the performance of competing alternatives, and rolling back models that don't work or underperform once deployed, are all examples of the kinds of activities needed to effectively manage production models.

Model evaluation

Putting a model in production is the beginning of the model's journey, not the end. Machine learning models are perishable and their performance must be continuously evaluated to identify degradation before it negatively impacts the business.

Models should be instrumented so that the inputs to and results of each inference are logged, allowing usage to be reviewed and performance to be monitored on an ongoing basis. Owners of models experiencing degraded performance should be notified so that corrective action, such as retraining or re-tuning, may be taken.

Supporting Machine Learning at Scale

When an enterprise is just getting started with machine learning, it has few established ML practices or processes. During this period, its data scientists and MLEs are typically working in an ad hoc manner to meet the immediate needs of their projects. Data acquisition and preparation, as well as model training, deployment, and evaluation, are all done manually with little automation of or integration between these steps.

Once a team has operated in this way for more than a handful of projects, it becomes clear that a great deal of effort is spent on repetitive tasks that can be automated, or worse, on reinventing the wheel. For example, they may find themselves repeatedly copying the same data, performing the same data transformations, engineering the same features, or following the same deployment steps.

Left to their own devices, individual data scientists or MLEs will build scripts or tools to help automate some of the more tedious aspects of the ML process so as to work more quickly. This can be an effective stopgap, but left unplanned and uncoordinated, these efforts can be a source of distraction and technical debt.

For organizations at a certain level of scale—typically when multiple machine learning teams and their projects must be supported simultaneously—ML infrastructure teams are established to drive efficiency and ensure that data scientists and MLEs have access to the tools and resources they need to work efficiently.*

Common challenges encountered by ML infrastructure teams include:

Data management and automation

During exploratory machine learning efforts, including building proofs-of-concepts, it is quite normal for data scientists or MLEs to spend considerable time manually procuring the data required to build new models. Ad hoc data exploration and transformation are typical at this stage.

As the organization's reliance on machine learning matures, automating its various data transformation, feature engineering, and ETL pipelines is necessary to increase modeling efficiency and ensure reproducibility. Ideally, feature transformations are also stored for later

* **At Airbnb,** for example, after gaining experience with applying machine learning to applications like search ranking, smart pricing, and fraud prevention, the company realized that it would need to dramatically increase the number of models it was putting into production in order to meet its business goals. To enable this, an ML infrastructure team was established. The team's mission is to eliminate what it calls the *incidental* complexity of machine learning—that is, getting access to data, setting up servers, and scaling model training and inference—as opposed to its *intrinsic* complexity—such as identifying the right model, selecting the right features, and tuning the model's performance to meet business goals.

use and cataloged for easy sharing across projects and teams. Beyond the modeling phase, automated pipelines are critical to delivering ready-to-use feature data to production models at inference time.

Data and feature transformations create new data which is often retained not just for training but for future inference. The data produced by these transformations is not typically be saved back in the systems of origin, such as transactional databases or log storage. Providing scalable data storage and management is thus a significant challenge faced by teams supporting the machine learning process.

Data management is greatly facilitated with a flexible, scalable, and secure data repository such as a data lake, fabric, or warehouse in place. These technologies can also support the low-latency and high-throughput access required by training and inference workloads without requiring additional data replication.

Driving efficient resource use

Today, we have more raw computing power at our disposal than ever before. In addition, hardware innovations such as high-density cores, GPUs, and TPUs are increasingly targeting machine and deep learning workloads, promising a continued proliferation of computing resources for these applications.

Despite declining computing costs, the machine learning process is so bursty and resource-intensive that efficient use of available computing capacity is critical to supporting ML at scale.

The following are key requirements for efficiently delivering compute to machine learning teams:

- **Elasticity.** Data preparation, model training, and model inference differ in the amount, type, and timing of the resources they require. The machine learning process works best when the individual tasks or workloads in the process can be scaled up when needed, and scaled back down when done.
- **Multitenancy.** Given the computationally intensive and bursty nature of machine learning training and inference, establishing dedicated hardware environments for each machine learning team or workload is inefficient. Rather, the focus should be on creating shared environments that can support the training and deployment needs of multiple concurrent projects.
- **Immediacy.** Data scientists and MLEs should have direct access to a computing environment where they can easily specify the number and type of resources needed without waiting for manual provisioning.
- **Programmability.** The creation of new environments, and the ability to scale existing environments up and down, must be available via APIs to enable the automated provisioning of infrastructure and maximize resource utilization.

These are, of course, the characteristics of modern, cloud-based environments. This does not, however, mean that we're required to use the public cloud to do machine learning at scale.

While the public cloud's operating characteristics make it a strong choice for running machine learning workloads, there are often other considerations at play. For example, cloud-based GPU instances can be pricey, leading many organizations to choose local GPU servers instead. In addition, it is often best to colocate machine learning training and inference workloads with production applications and data in order to minimize network latency and reduce bandwidth requirements. As a result, private clouds are a worthy consideration for many organizations.

Ultimately, in a world of declining hardware prices, decreasing cloud costs, and shifting workloads, it is prudent to build flexibility into new tools and platforms, so that they can be efficiently operated in both public and private cloud environments.

Today we've got more—and more affordable—raw compute power at our disposable than ever before. While compute costs continue to decline rapidly, the machine learning process is so bursty and resource intensive that efficient use of compute is critical to supporting machine learning at scale.

Hiding complexity

With the rise of Platform as a Service (PaaS) offerings and DevOps automation tools, software developers can now operate at a higher level of abstraction than ever before, allowing them to focus on the applications they are building and not worry about the underlying infrastructure on which their software runs.

Similarly, in order for the machine learning process to operate at full scale and efficiency, data scientists and MLEs must be able to focus on their models and data products rather than infrastructure.

This is especially important because data products are built on a complex stack of rapidly evolving technologies. These include deep learning frameworks like TensorFlow and PyTorch, language-specific libraries like SciPy, NumPy and Pandas, and data processing engines like Spark and MapReduce. These tools are supported by a variety of low-level drivers and libraries like NVIDIA's CUDA, which allows jobs to take advantage of the GPU and is notoriously difficult to correctly install and configure.

Managing these dependencies manually can be a constant drain on the machine learning process, and can be the source of hard-to-debug discrepancies between results seen in training and production.

For all these reasons and more, many machine learning and deep learning teams have turned to containers and Kubernetes to overcome the infrastructure challenges of machine and deep learning.

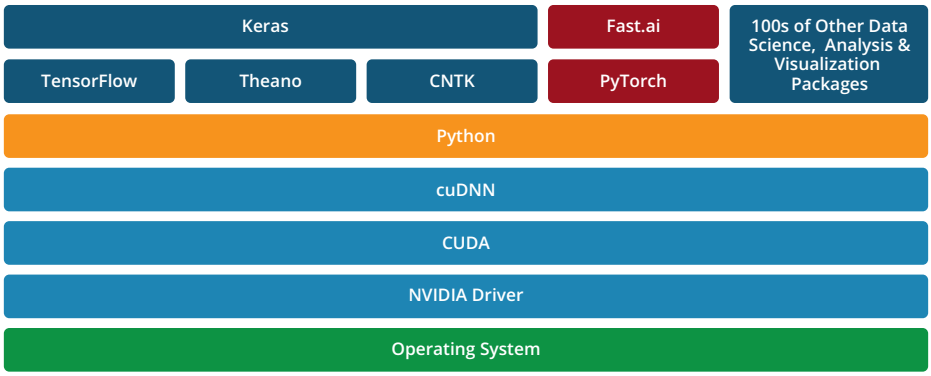


Figure 4. Data products are built on a complex stack of technologies

Enter Containers and Kubernetes

The introduction of Docker containers in 2013 initiated a dramatic shift in the way software applications are developed and deployed.

Container images provide a standardized, executable package that provides everything needed to run an application, including its code, dependencies, tools, libraries and configuration files. A running Docker container is an instantiation of a container image.

Compared with virtual machines, container images are lighter weight, easier to move between environments, and faster to spin up. This is in large part because they can share the host operating system's (OS) kernel, as opposed to containing their own copy of a complete OS.

The fact that lightweight containers could be reliably run across disparate computing environments helped address many of the challenges faced by software development and operations organizations as they sought to modernize their software delivery processes, applications, and infrastructure, leading to their ultimate popularity.

As we've seen, data science and ML engineering teams face many of the same challenges as application developers, so it should come as no surprise that containers can play a role in solving their problems too.

But containers alone don't provide the complete solution. In real-world systems, it is often necessary to use multiple containers, make the services they offer easily accessible, and connect them to a variety of external data sources. An orchestration platform is required to efficiently manage the containers and their interactions with one another and the outside world.

That's where Kubernetes comes in. Kubernetes is an open source container orchestration platform developed and open-sourced by Google in 2014. Kubernetes has since become the de facto standard for container orchestration with support from all major vendors due to its early success, user adoption, and the workload portability it creates between on-premises and public cloud environments.

Kubernetes provides the necessary features required for complete lifecycle management of containerized applications and services in a manner that has proved to be highly scalable and reliable.

Before exploring what this means for machine learning workloads, let's learn a bit about Kubernetes itself.

Getting to Know Kubernetes

Kubernetes takes a hierarchical approach to managing the various resources that it is responsible for, with each layer hiding the complexity beneath it.

The highest-level concept in Kubernetes is the **cluster**. A Kubernetes cluster consists of at least one **master** which controls multiple worker machines called **nodes**. Clusters abstract their underlying computing resources, allowing users to deploy workloads to the entire cluster as opposed to on particular nodes.

A **pod** is a collection of one or more containers and their configuration. It is the basic workload unit in Kubernetes. All containers within a pod share the same context, resources (such as storage and networking) and lifecycle.

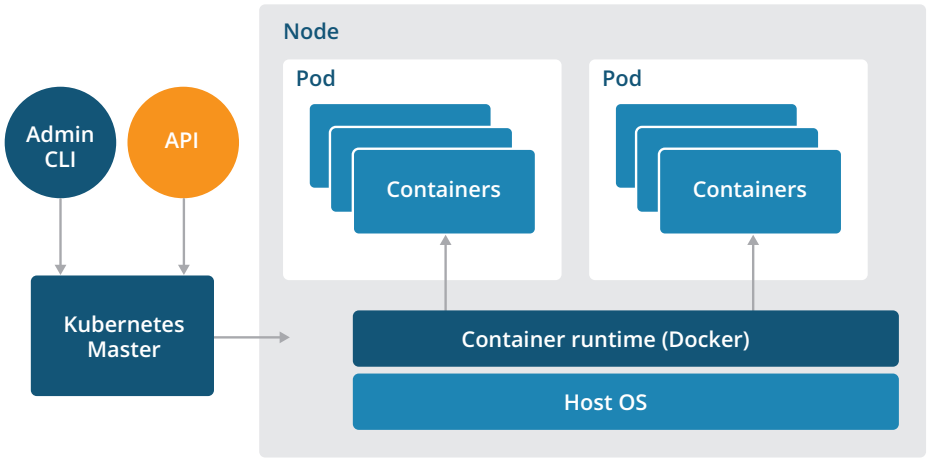


Figure 5. Basic Kubernetes architecture

The Kubernetes master can be thought of as the "brain" of the cluster. It responds to cluster configuration and management requests submitted via the Kubernetes client or API. It is responsible for determining which pods are deployed to which nodes based on their respective requirements and capabilities, a process called **scheduling**. The master maintains the overall health of the cluster by re-scheduling pods in reaction to faults such as server failures.

Nodes in a Kubernetes cluster listen for instructions from the master and create, run, or destroy containers accordingly. This necessitates the presence of a container runtime, such as Docker, running on each node.

To run a workload on a Kubernetes cluster, the user provides a plan that defines which pods to create and how to manage them. This plan can be specified via a configuration document which is sent to the cluster via Kubernetes' APIs or client libraries.

When the master receives a new plan, it examines its requirements and compares them to the current state of the system. The master then takes the actions required to converge the observed and desired states. When pods are scheduled to a node, the node pulls the appropriate container images from an image registry and coordinates with the local container runtime to launch the container.

Files created within a container are ephemeral, meaning they don't persist beyond the lifetime of the container. This presents critical issues when building real-world applications in general and with machine learning systems in particular.

First, most applications are stateful in some way. This means they store data about the requests they receive and the work they perform as those requests are processed. An example might be a long training job that stores intermediate checkpoints so that the job doesn't need to start over from the beginning should a failure occur. If this intermediate data were stored in a container, it would not be available if a container fails or needs to be moved to another machine.

Next, many applications must access and process existing data. This is certainly the case when building machine learning models, which requires that training data be accessible during training. If our only option for data access was files created within a container, we would need to copy data into each container that needed it, which wouldn't be very practical.

To address these issues, container runtimes like Docker provide mechanisms to attach persistent storage to the containers they manage. However, these mechanisms are tied to the scope of a single container, are limited in capability, and lack flexibility.

Kubernetes supports several abstractions designed to address the shortcomings of container-based storage mechanisms. **Volumes** allow data to be shared by all containers within a pod and remain available until the pod is terminated.

Persistent volumes are volumes whose lifecycle is managed at the cluster level as opposed to the pod level. Persistent volumes provide a way for cluster administrators to centrally configure and maintain connections to external data sources and provide a mechanism for granting pods access to them.

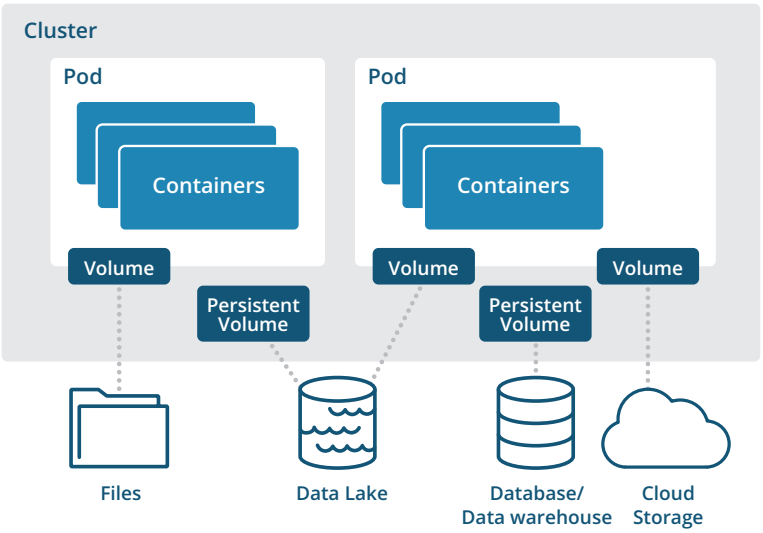


Figure 6. Kubernetes volumes provide a flexible abstraction for accessing a wide variety of storage systems from containerized workloads

First and third-party volume drivers allow Kubernetes users to access a wide variety of volume types including local disk, iSCSI, NFS, vSphere, major cloud providers' block and object storage systems, data lakes and data warehouses, distributed storage systems, and more.

"Deep learning is an empirical science, and the quality of a group's infrastructure is a multiplier on progress. Fortunately, today's open-source ecosystem makes it possible for anyone to build great deep learning infrastructure."

OpenAI

Kubernetes for Machine and Deep Learning

Containers and Kubernetes help organizations address each of the aforementioned machine learning process challenges with a broadly-supported open-source platform.

Data management and automation

By providing the fundamental mechanism with which storage is connected to containerized workloads, volumes and persistent volumes provide the raw materials that enable Kubernetes' support for stateful applications, including machine and deep learning.

Upon these primitives, a variety of tightly integrated third-party solutions are available that enable highly automated data processing pipelines to be constructed, ensuring the ability to reliably deliver properly transformed data to models without manual intervention. Several of these options are discussed in a later section of this document.

Third-party data fabric products allow Kubernetes workloads to gain unified access to enterprise data in distributed storage systems or data warehouses. This eliminates the need for teams to navigate multiple systems and data access methods to obtain data, and facilitates the reuse of transformed data and processed features across projects.

Driving efficient resource use

Users can scale out a Kubernetes cluster as needed, simply by adding more physical or virtual servers to it.

Kubernetes has the ability to track different node attributes, such as the type and number of CPUs or GPUs present, or the amount of RAM available. These attributes are considered when scheduling jobs to nodes, ensuring that sparse resources are allocated efficiently.

For resource-intensive workloads like machine learning, users face the conundrum of over-allocation or under-utilization. That is, they must manage the fine balance between either employing more resources than required, or running out of resources. Kubernetes' solution to this is auto-scaling, which is the ability to scale up or down the number of nodes a given workload is running on at any time.

A single physical Kubernetes cluster can be partitioned into multiple virtual clusters using Kubernetes' **namespaces** feature. This allows a single cluster to more easily support different teams, projects or functions.

Each namespace can be configured with its own resource quotas and access control policies, making it possible to specify the actions a user can perform and the resources they have access to, allowing sophisticated multitenancy needs to be accommodated. Namespace-level configuration allows resources like storage to be configured on a per-team, -project, or -function basis.

A Kubernetes cluster can consist of either physical or virtual servers. These servers can use commodity or specialized hardware and can be hosted anywhere. As a result, the same containerized workload can be run on any platform or in any location without any changes to the application's code.

Hiding complexity

Containers provide an efficient way of packaging machine learning workloads that is independent of language or framework. Kubernetes provides a reliable abstraction layer for managing containerized workloads and provides the necessary configuration options, APIs, and tools to manipulate these workloads declaratively.

While working with containers and containerized workloads may be a new skill for many data scientists, they should be a familiar tool for machine learning engineers exposed to modern software development practices. User interfaces, command-line tools, or integration with source code repositories can simplify creating and managing containers for end users.

While introducing its own set of tools, the use of containers to encapsulate data science jobs provides the valuable benefit of shielding those workloads from the complexity of the underlying technology stack. This ensures that the correct and consistent dependencies are in place wherever jobs are run, whether on the developer laptop, training environment, or production cluster.

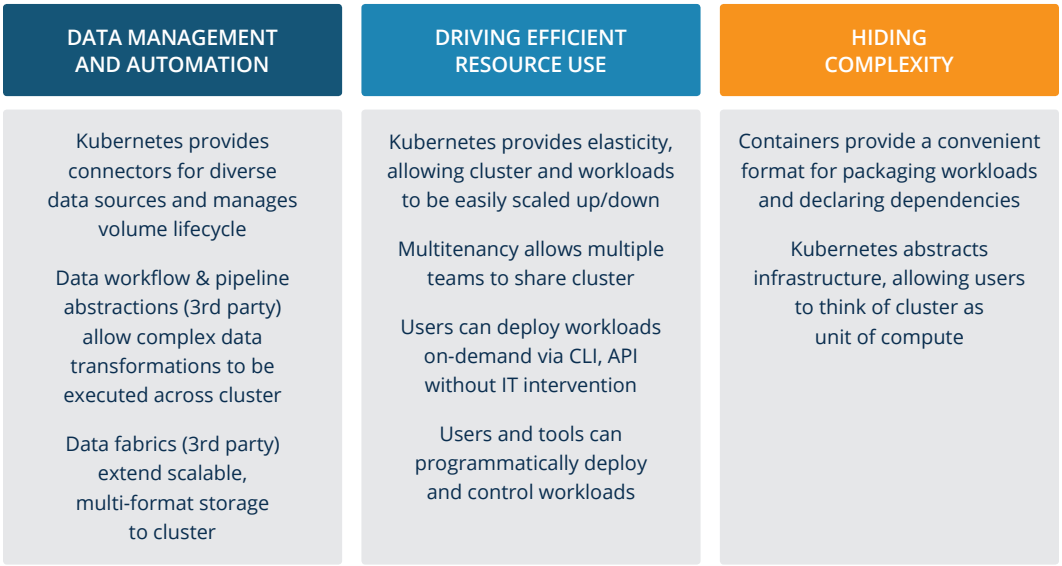
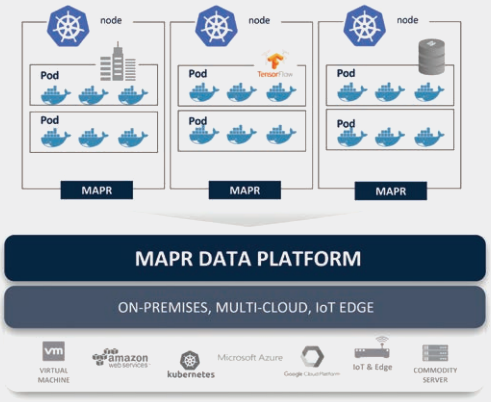


Figure 7. Containers and kubernetes address major ML/DL challenges

MapR Data Platform

MapR Data Platform is an all-software data platform designed to deliver exabyte scale, mission-critical reliability, and high performance. MapR provides unmatched data protection, disaster recovery, security, and management services for disparate data types, including files, objects, tables, events, and more.



MapR Data Fabric for Kubernetes

MapR Data Fabric for Kubernetes builds on this platform to provide persistent storage for containers. It enables the deployment of stateful containers for data pipelines and machine learning training and inference workloads, as well as a variety of production use cases.

MapR features supporting machine and deep learning include:

- **Vast scalability.** Store exabytes of structured and unstructured data in a distributed file and object store.
- **Global namespace.** Access globally distributed data, within and across clouds and on-premises deployments.
- **Diverse data.** Unify files, objects, containers, tables, and publish/subscribe events with one comprehensive solution.
- **Multi-tenant security.** Secure distributed data with built-in security, encryption, and access control.
- **Seamless AI readiness.** Provide fast data access via the latest ML, DL, and analytical toolkits without data movement.

MapR and NVIDIA recently published a reference architecture illustrating how customers running containerized data science workloads on NVIDIA's DGX-1 deep learning supercomputer and the MapR Data Fabric for Kubernetes can benefit from this combination of technologies. The following benefits were noted and apply equally well to non-DGX-1 scenarios:

- **Ease-of-Use.** Developers and data scientists can transparently access data from any container.
- **Speed.** The joint solution delivered 10x faster performance than traditional GPU-based deep learning training.
- **Flexibility.** Customers can choose from a variety of multi-tenant data access combinations, accessing data either directly in the cluster or across environments.
- **Future-proof architecture.** Customers can leverage this reference architecture for additional deep learning workloads as GPU technologies evolve.

The Kubernetes ML/DL Ecosystem

Kubernetes offers, out of the box, several important features that facilitate the machine learning process. However, your use of Kubernetes for machine learning can be enhanced by taking advantage of one or more of the many specialized tools or platforms available.

Note that the Kubernetes ecosystem has exploded since the project's launch and continues to grow at a rapid pace—a testament to the project's success. The ecosystem map below, and the descriptions that follow, present just a few of the many projects worth exploring for machine learning and deep learning workloads.

Kubernetes Machine Learning & Deep Learning Landscape

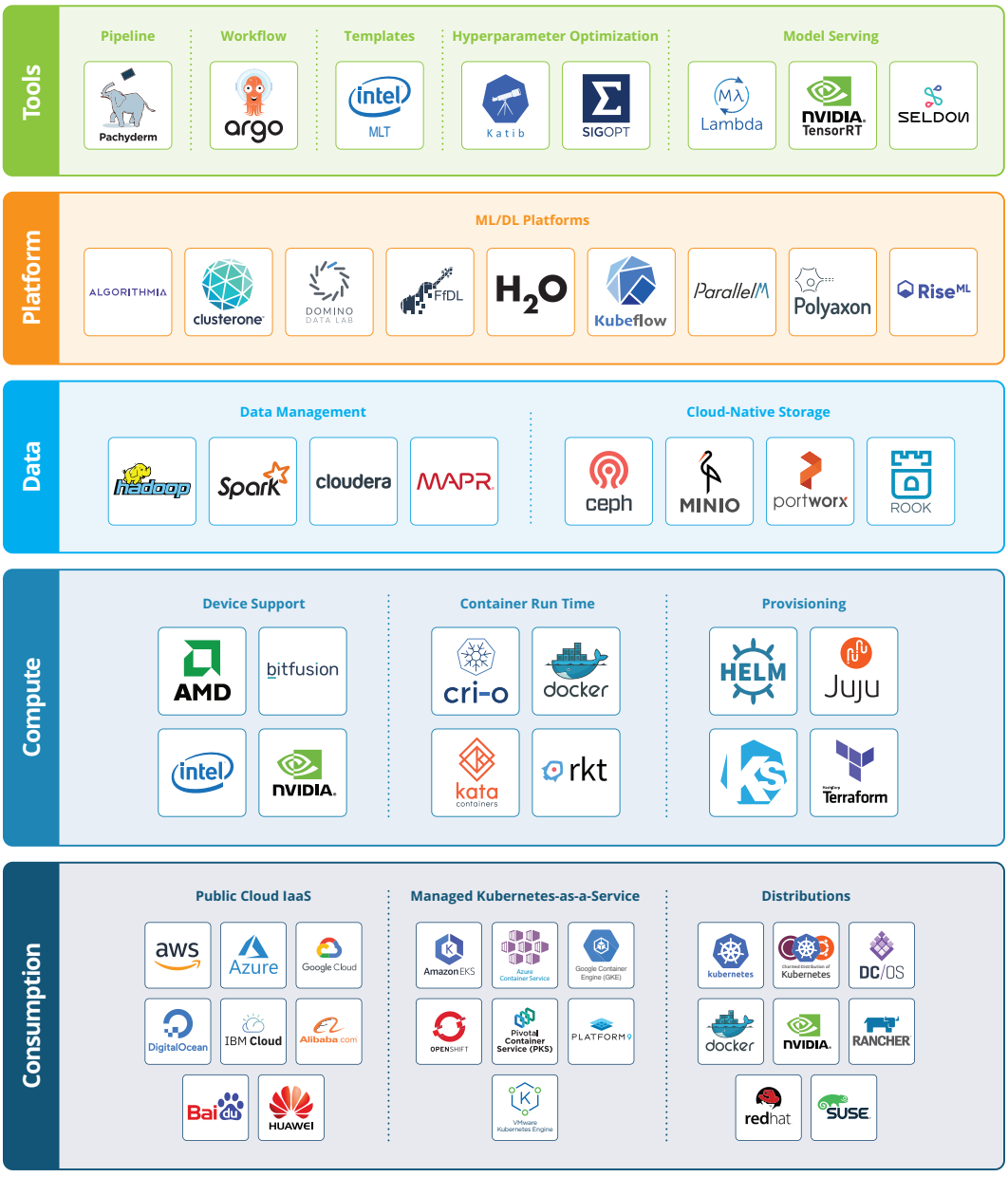


Figure 8. The Kubernetes ML/DL ecosystem.

Kubernetes Distributions and GPU Support

GPU support is critical for deep learning applications. Kubernetes has supported managing NVIDIA GPUs since version 1.6, though this support was experimental and unsupported by NVIDIA.

Starting with version 1.8 of Kubernetes, the recommended and supported way to consume GPUs is through the device plugin framework. A number of device plugins for ML/DL users exist but most notable is the NVIDIA GPU device plugin. Plugins for AMD GPUs, Intel GPUs and FPGAs are also available.

Upstream Kubernetes refers to the official community-supported Kubernetes project, but several other vendor-supported Kubernetes distributions have been developed. In fact, the Pick the Right Solution page in the Kubernetes documentation lists nearly 50 offerings.

Recently announced, and of particular interest to deep learning users is "Kubernetes on NVIDIA GPUs" by NVIDIA. This distribution was created to offer better, self-contained support for NVIDIA hardware on Kubernetes, and is the preferred Kubernetes distribution for NVIDIA's servers and the NVIDIA GPU Cloud.

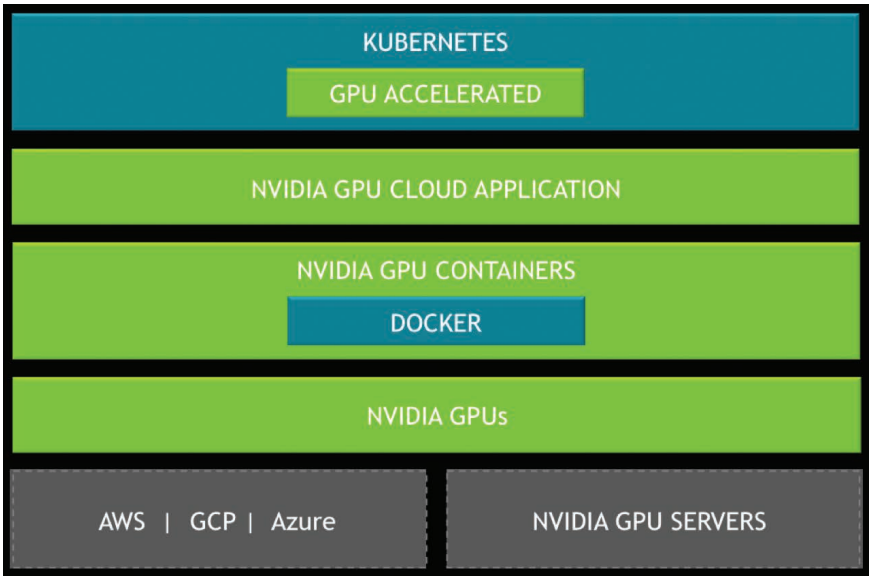


Figure 9. Kubernetes on NVIDIA GPUs

Kubernetes on NVIDIA GPUs exposes fine-grained GPU attributes such as memory and compute capability to the Kubernetes scheduler, and provides an integrated GPU monitoring stack.

As is often the case with vendor distributions, the disadvantage of the Kubernetes on NVIDIA GPUs distribution is that it lags behind upstream Kubernetes and thus may not offer the latest Kubernetes features.

Useful Tools

Argo (argoproj.io) is a Kubernetes-native workflow engine. Originally designed to support CI/CD pipelines, Argo workflows are well-suited to manage different aspects of the machine learning process.

Each step in an Argo workflow is a container. Workflows can be defined as a simple sequence of steps or a directed acyclic graph (DAG) that captures the dependencies between tasks. Argo workflows support advanced control structures such as loops, conditionals, and recursion. Timeouts, retries, and resubmits are supported for running workflows, as are workflow suspension, resumption and cancellation.

While Argo workflows can automate various steps in the machine learning process, it is particularly well suited to pull the different pieces of the puzzle together to support end-to-end machine learning.

Consider a retailer updating its content-based recommendation system whenever new catalog and pricing data are available. Argo sensors can determine when new data is available and dependencies can ensure that both sets of data are ready before triggering the workflow. The workflow includes a job (container) that performs batch scoring, another that uploads the new scores to the production database, and a third that archives the input data.

The Argo project is open source, supported by Intuit, and is used by major companies including Adobe, BlackRock, Datadog, Google, and Nvidia.

Pachyderm (pachyderm.io) is an open-source project that lets users deploy and manage multi-stage, language- and framework-agnostic data pipelines. In this regard, it can be thought of as a workflow system (i.e. like Argo) but with specialized features for data pipelines. It can also act as a data repository or data lake where users can collect and process large datasets.

Pachyderm, like Argo, defines workflows as DAGs. Also like Argo, each step in a Pachyderm pipeline is a container, allowing users to use and combine their tools of choice for the problem at hand. Pachyderm can run data processing jobs in parallel, automatically sharding, or partitioning, data across running jobs as needed. Pachyderm makes use of Kubernetes to ensure that data is processed by nodes with the right capabilities (e.g. RAM or GPUs) and to auto-scale the number of workers running at any given time.

Pachyderm is marketed as a "git for data science." All data that flows into and out of each step in a Pachyderm pipeline is versioned.

Data transformation, model training, model storage, and model scoring can be integrated into a single Pachyderm pipeline. This provides an historical record that tracks what models were invoked, what data is passed to them, and what results they produce. When new training data is committed to Pachyderm, a model update can be automatically triggered. If the new model degrades system performance once deployed, it may be rolled back to a previous version.

A web-based user interface is provided for users of Pachyderm's Enterprise product.

Katib (https://github.com/kubeflow/katib) is an open-source hyperparameter tuning project inspired by Google's internal black-box optimization service, Vizier. Katib is framework agnostic and tightly integrated with Kubernetes, to which it deploys optimization tasks as pods.

Katib, as in Google Vizier, organizes optimization tasks around studies, trials, and suggestions. A **study** is a single optimization run described by a fixed objective function, the parameter space over which optimization will take place, as well as a set of trials. A **trial** is a list of parameter values that will lead to a single evaluation of the objective function. Trials are automatically generated by Katib as **suggestions**, based on supplied algorithms, including random search, grid search, Hyperband, and Bayesian optimization.

Each trial in Katib is run as a Kubernetes pod, as are the services that generate suggestions. Trials can be run in parallel up to a number of concurrent jobs specified by the user.

A web-based dashboard, integrated with Google's TensorBoard visualization package, is provided.

Katib is included in Kubeflow (see below) but maintained separately by developers from NTT and Shanghai Jiao Tong University.

Seldon Core (http://seldon.io) is an open source platform for deploying machine learning models on Kubernetes. It is exclusively focused on model serving and inference; it does not provide any support for the training phase of the machine learning process.

Seldon Core supports serving models built with a wide variety of machine and deep learning tools and technologies such as Python's scikit-learn, TensorFlow, PyTorch, Spark, H2O, R, ONNX, and PMML.

To deploy a model to Seldon Core, developers wrap it using provided Python, R, or Java wrappers. A provided tool is then run to expose REST or gRPC interfaces and encapsulate the model in a Docker container. A Seldon deployment manifest (itself a Kubernetes custom resource written in YAML) is then created to describe the resources required to serve up a prediction service using one or more models. The deployment manifest can then be deployed using standard Kubernetes tools.

Decision services in Seldon Core can be based on simple model calls or complex runtime inference graphs. This allows real-time routing of requests (e.g. for running model A/B tests), combining responses from sub-graphs (e.g. for ensembles of models), and transformation of model request or response data.

Full-lifecycle management of deployed models is provided, including zero downtime updates, auto-scaling, logging and monitoring.

Higher-Level Platforms

A number of projects offer high-level machine and deep learning platforms running on top of Kubernetes. A few interesting examples include:

Kubeflow (kubeflow.org) is an open source project that runs on top of Kubernetes. It is sponsored by Google and inspired by TensorFlow Extended, or TFX, the company's internal machine learning platform.

Kubeflow integrates services already used by ML and DL developers to make them easier to deploy and use anywhere that Kubernetes runs, from a developer laptop to a production cluster. The project includes support for Jupyter Notebooks, distributed training (for Tensorflow, PyTorch, MXNet and Chainer models, hyperparameter tuning (using Katib, model serving (using Seldon Core), workflows (using Argo), and more.

Kubeflow exposes Kubernetes concepts like namespaces to users and requires that they use standard Kubernetes tools to perform tasks like configuring and submitting jobs. As a result, it is more geared towards users with engineering skill sets, rather than those of the typical data scientist.

Using Kubeflow allows sophisticated Kubernetes shops to save time integrating support for common ML and DL components. However, the project is immature and moving quickly, which may cause problems for some organizations. Enterprise IT organizations might find that the project lacks an adequate security and governance framework that spans workloads and data.

RiseML (riseml.com) provides a higher level of abstraction for training and running ML workloads on Kubernetes clusters.

With RiseML, users can think in terms of experiments that train a model. A RiseML 'experiment' refers to a single instance of your model with its associated architecture, training data, and parameters. RiseML takes care of executing these experiments on the infrastructure in a robust manner, including deciding which nodes specific parts of an experiment are run on. Launching an experiment is as simple as executing the riseml train command.

RiseML supports all major deep learning frameworks such as TensorFlow, Keras and PyTorch. Arbitrary frameworks can be used by specifying custom Docker images for experiments. Enhanced integration is provided for TensorFlow, allowing users to specify the version used, add a Tensorboard to experiments, and use distributed TensorFlow.

Simple automated hyperparameter tuning is offered based on grid search, and support for distributed TensorFlow training is provided. Additionally, auto-scaling support is provided on AWS.

Basic user management is provided allowing administrators to secure the system at the cluster level.

RiseML does a nice job of hiding the complexity of Kubernetes from data scientists. However, the fact that it is closed source and currently offers limited capabilities are drawbacks.

Polyaxon (polyaxon.com) is marketed as an open source platform for building, training, and monitoring large scale deep learning applications. It relies on Kubernetes under the covers to manage cluster resources, launch and manage containerized workloads, and scale up and down as needed.

Like RiseML, Polyaxon is generally organized around the concept of experiments. A nicely organized conceptual hierarchy is provided consisting of:

- **Experiments.** A single execution of your model with a provided set of data and parameters.
- **Experiment groups.** A versioned collection of experiments run using different hyperparameters.
- **Projects.** The top-level organization concept for your efforts around a specific problem.

Experiments are deployed to the cluster as pods called Experiment Jobs, and Distributed Experiments are supported for TensorFlow, PyTorch and MXNet. Generic Jobs, which are run on the cluster but not tracked and managed as experiments, can be used for data processing or other operations.

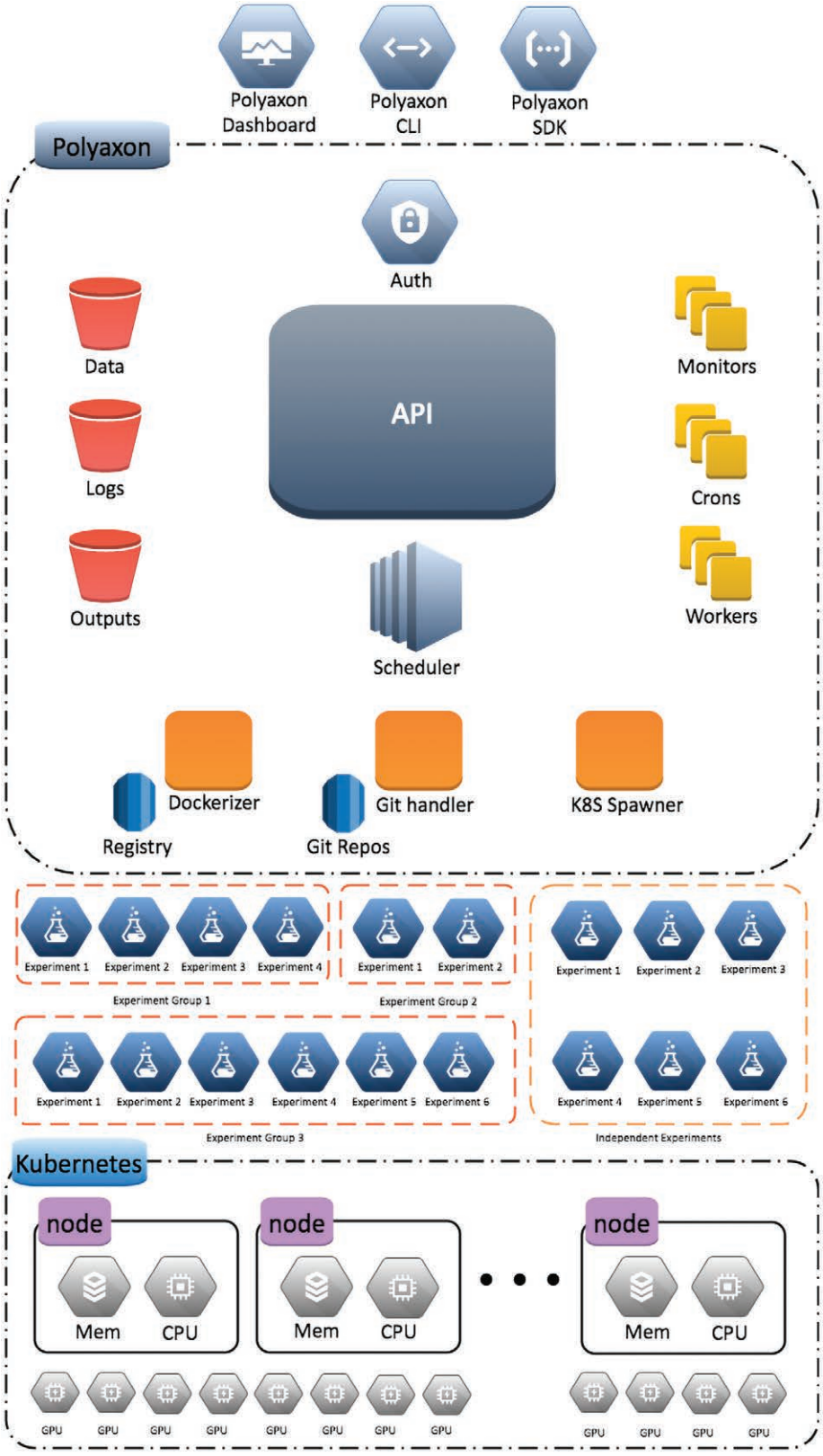


Figure 10. Polyaxon is organized around the concept of experiments, which are deployed to Kubernetes as jobs.

Polyaxon offers the same algorithms as Katib for hyperparameter tuning: grid search, random search, Hyperband, and Bayesian optimization. Unlike Katib, support for early stopping (a technique included in the Vizier paper) is included. High-level experiment and experiment group tracking APIs allows data scientists to log experiment parameters, code versions, metrics, and outputs from running machine learning code–both within and outside of a Polyaxon deployment.

First-class support is provided for running Jupyter Notebooks at the project level, and TensorBoards at the project, experiment group, or experiment level.

Two-tiered user management is provided and access controls can be attached at the project and experiment levels, with more robust team support on the roadmap.

While the project's functionality and documentation are quite impressive, Polyaxon is a small player in this space and appears to be the personal project of a single developer.

IBM Fabric for Deep Learning (https://github.com/IBM/FfDL) recently announced its Fabric for Deep Learning (FfDL, pronounced "fiddle"). FfDL is open sourced from the core code that powers the deep learning service within IBM's commercial Watson Studio offering.

FfDL uses a microservices architecture deployed as pods on Kubernetes. Support is provided for TensorFlow, PyTorch, Caffe/Caffe2, and Keras; arbitrary containerized deep learning frameworks are supported without modification. Existing modes are typically trainable without code changes, though some may be required for distributed training. A REST API is provided for programmatically managing the environment.

In terms of level of abstraction, FfDL sits somewhere between Kubeflow and something like RiseML or Polyaxon. FfDL offers distributed training for deep learning models across the Kubernetes cluster. A component called the Lifecycle Manager constructs and deploys containerized training workloads and supports pausing and terminating training jobs. Auto allocation and deallocation of compute resources is supported on IBM Cloud.

A Training Data Service automatically manages, tracks, and stores training history and artifacts. Experiment results are loaded into a managed Elasticsearch instance for search and analysis, and may be used for billing/chargeback purposes by organizations using FfDL as the basis for internal or external service platforms.

Some notable gaps exist in the FfDL offering. For instance, support for launching Jupyter Notebooks in the environment is not well documented. Support for model serving via Seldon Core is documented, but is a manual process.

In addition to traditional documentation, FfDL is also the subject of several IBM Research papers exploring its operating characteristics.[8, 9, 10]

Case Studies

Booking.com

Booking.com is one of the largest travel e-commerce sites, offering travelers over 1.4 million properties in over 220 countries with accommodation options ranging from luxury resorts to igloos. The company uses machine and deep learning models to solve a variety of business problems, including recommendations, image understanding and tagging, translation, and ad optimization.Booking.com chose Kubernetes upon which to build its internal machine learning processes for the elasticity, flexibility, and resource isolation it provides.[11]

Teams at Booking.com deploy training jobs running standardized base container images that provide support for their frameworks of choice, such as TensorFlow, Torch, and VowpalWabbit. To avoid the need for data scientists to containerize their model training routines, these base images fetch the training code from git and initiate training on startup.[12]

Persistent volumes managed by Kubernetes allow training pods to access data in Booking.com's Hadoop cluster. During training, the pods stream log entries back to Hadoop. Upon completion, the trained model is exported out to Hadoop as well. Kubernetes allows training pods to target CPU or GPU nodes as required.

Booking.com also relies on Kubernetes to support serving predictions for model inference. Prediction microservices are deployed as stateless microservices running a common containerized codebase.

Prediction pods pull trained models from Hadoop, loading them into the running container's memory, and exposing them via a REST API which clients can access for predictions. Kubernetes' built-in ability to probe the container for readiness, and, once ready, add the container to the appropriate load balancer pool, significantly simplifies operations.

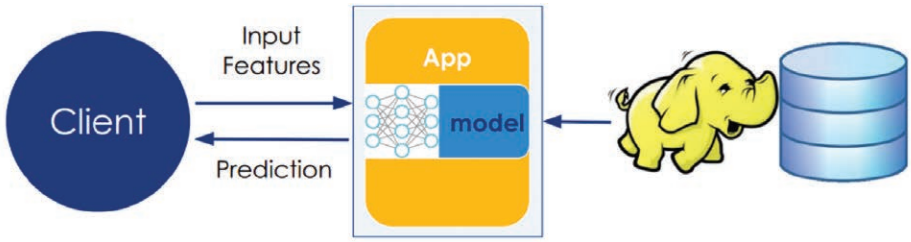


Figure 11. Serving model predictions at Booking.com

Allowing containers to fetch training code and trained models upon startup allows Booking.com to keep images small and fast, and avoid image sprawl.

The company credits Kubernetes' elasticity, self-service, GPU support, and ability to intelligently schedule workloads with helping them run their compute- and data-intensive, hard to parallelize, machine learning models at scale.

Using Kubernetes, Booking.com has been able to grow their use of machine learning models to serve predictions and recommendations to their customers on a mass scale, serving more than 400 million visitors a month.

OpenAI

OpenAI is a non-profit research company whose long-term goal is to ensure that the benefits of AI are as safely and widely distributed as possible. OpenAI is at the forefront of AI research, employing a full-time staff of 70 researchers and engineers who apply a variety of machine and deep learning techniques to develop flexible algorithms that can solve more than one type of problem.

OpenAI uses Kubernetes to provide the agility, flexibility and scale its teams require. To support its large scale experiments, OpenAI runs several Kubernetes clusters, with some in the cloud and some on bare-metal local hardware.[13]

Its largest reported cluster consists of 2,500+ nodes running in the Microsoft Azure cloud. This cluster runs a combination of CPU-only nodes (20-core D15v2 instances with 140 GB of RAM) and GPU nodes (NC24 instances with 24 CPU cores, 4 NVIDIA K80 GPUs and 224 GB of RAM). The company also runs similarly large clusters on AWS and Google Cloud Platform.

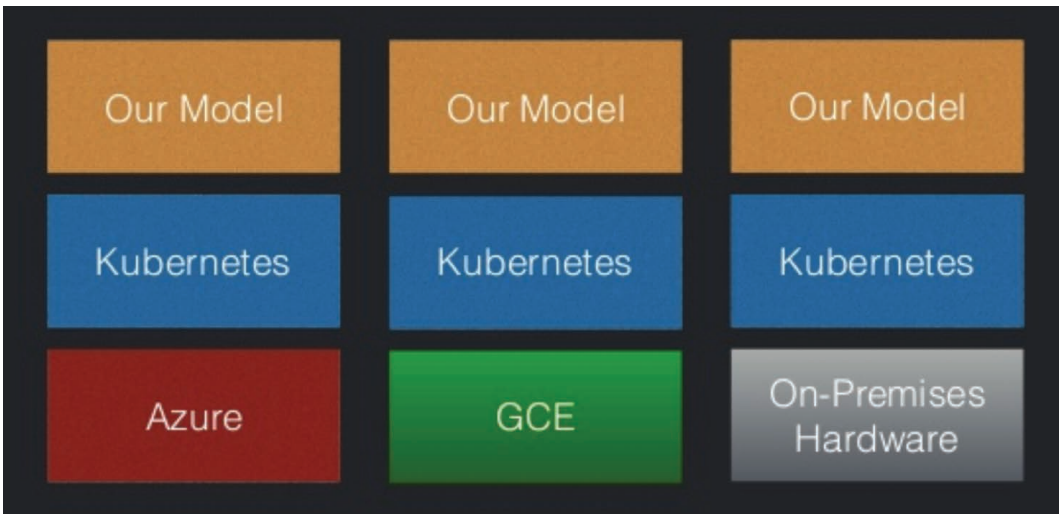


Figure 12. OpenAI uses Kubernetes to abstract infrastructure and enable workload portability across environments

The workloads deployed to OpenAI's Kubernetes clusters are primarily focused on batch training for its Python-based TensorFlow deep learning models. OpenAI takes advantage of pre-packaged Docker containers provided by Anaconda. These incorporate all of the dependencies required for their workloads, including difficult to install packages like OpenCV, and provide performance-optimized versions of select scientific libraries.

OpenAI infrastructure teams provide researchers with custom tooling that allows them to transparently deploy code from their laptop development environments to a standardized Docker container image.

According to OpenAI, its batch jobs are bursty and unpredictable. A research project can quickly scale from an exploratory single-machine effort, to a large-scale training job running on thousands of cores. For this reason, the company relies heavily on autoscaling to dynamically scale up and down cloud-based clusters as needed. This helps them keep the costs associated with idle nodes low while still supporting the ability to iterate quickly without having to be concerned with infrastructure configuration.

To overcome some of the deficiencies of Kubernetes' built-in autoscaling when working with large batch jobs, OpenAI developed and open-sourced the kubernetes-ec2-autoscaler project. The custom autoscaler runs as a normal pod and takes advantage of the ability to query the master for the entire state of the cluster. It then calculates the required cluster resources in a way that works best for their jobs and issues the appropriate requests to EC2.

OpenAI takes advantage of a variety of built-in and open-source tooling for supporting Kubernetes and its workloads, and credits the system for helping maximize the productivity of its deep learning researchers, allowing them to focus on their science and not on their infrastructure.[14]

"Research teams can now take advantage of the frameworks we've built on top of Kubernetes, which make it easy to launch experiments, scale them by 10x or 50x, and take little effort to manage."

CHRISTOPHER BERNER,
HEAD OF INFRASTRUCTURE FOR OPENAI

Getting Started

Meeting the infrastructure needs of machine and deep learning development represents a significant challenge for organizations moving beyond one-off ML projects to scaling the use of AI more broadly in the enterprise.

Fortunately, enterprises starting down this path need not start from scratch. The pairing of container technologies and Kubernetes addresses many of the key infrastructure challenges faced by organizations seeking to industrialize their use of machine learning. Together, they provide a means for delivering scalable data management and automation, the efficient utilization of resources, and an effective abstraction between the concerns of data scientists and MLEs and the needs of those providing the infrastructure upon which they depend.

In addition to supplying many of the features required of a robust machine learning infrastructure platform right out of the box, the open source Kubernetes project—with its broad adoption, active community, plentiful third-party tools ecosystem, and multiple commercial support options—also checks the most important boxes for executives and business decision-makers investing in a platform for future growth.

As with any new technology, the best course of action is to research and understand your requirements and options and start small, with an eye towards your anticipated needs. Your exploration of Kubernetes can start with a "cluster" running on your laptop or a handful of machines in the cloud, allowing you to experience and appreciate firsthand what it brings to the table for your machine learning workloads.

Godspeed.

References

1 Andreij Karpathy, “Software 2.0,” Medium, November 11, 2017, https://medium.com/@karpathy/software-2-0-a64152b37c35.

2 Sam Charrington, “How ML Keeps Shelves Stocked at Home Depot with Pat Woowong,” This Week in Machine Learning & AI, August 23, 2018, https://twimlai.com/talk/175.

3 Sam Charrington, “ML for Understanding Satellite Imagery at Scale with Kyle Story,” This Week in Machine Learning & AI, August 16, 2018, https://twimlai.com/talk/173.

4 Themis Mavridis, Pablo Estevez, and Lucas Bernardi, “Learning to Match,” arXiv [cs.IR], https://arxiv.org/abs/1802.03102v1

5 William Koehrsen, “Prediction Engineering: How to Set Up Your Machine Learning Problem,” Towards Data Science, November 7, 2018, https://towardsdatascience.com/prediction-engineering-how-to-set-up-your-machine-learning-problem-b3b8f622683b

6 Steve Lohr, “For Big-Data Scientists, ‘Janitor Work’ Is Key Hurdle to Insights,” The New York Times, August 17, 2014, https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

7 Norman Jouppi, et al., “In-Datacenter Performance Analysis of a Tensor Processing Unit,” arXiv [cs.AR], https://arxiv.org/abs/1704.04760v1

8 Bishwaranjan Bhattacharjee et al., “IBM Deep Learning Service,” arXiv [cs.DC], https://arxiv.org/abs/1709.05871

9 Scott Boag et al., “Scalable Multi-Framework Multi-Tenant Lifecycle Management of Deep Learning Training Jobs,” Workshop on ML Systems at NIPS’17, http://learningsys.org/nips17/assets/papers/paper_29.pdf

10 Scott Boag et al.,“Dependability in a Multi-tenant Multi-framework Deep Learning as-a-Service Platform,” arXiv [cs.DC], https://arxiv.org/abs/1805.06801

11 Manuel Pais, “How Booking.com Uses Kubernetes for Machine Learning,” InfoQ, April 1, 2018, https://www.infoq.com/news/2018/04/booking-kubernetes-machine-learn

12 Sahil Dua, “Putting Deep Learning Models in Production,” QCON London Presentation, https://qconlondon.com/system/files/presentation-slides/qconlondon2018_ai_track_-_session_1_-_qcon_london_2018.pdf

13 Vicki Cheung et al., “Infrastructure for Deep Learning,” OpenAI Blog, August 29, 2016, https://blog.openai.com/infrastructure-for-deep-learning/

14 Sam Charrington, “Scaling Deep Learning on Kubernetes at OpenAI with Christopher Berner,” This Week in Machine Learning & AI, November 12, 2018, https://twimlai.com/talk/199.

Image Credits

Figure 1 adapted from Francois Chollet’s *Deep Learning with Python* (Manning, 2017.

Images depicting Booking.com, MapR, NVIDIA, OpenAI systems, and Polyaxon provided by those respective companies.