

Kubernetes for MLOps:

Scaling Enterprise Machine Learning,
Deep Learning, and AI



By Sam Charrington

Founder, TWIML

Host, TWIML AI Podcast

SPONSORED BY


Hewlett Packard
Enterprise



Table of Contents

Preface to the Second Edition	4
Introduction	5
The Machine Learning Process	6
Machine Learning at Scale	10
Enter Containers and Kubernetes	14
Getting to Know Kubernetes	15
Kubernetes for Machine and Deep Learning.....	18
MLOps on Kubernetes with Kubeflow.....	20
The Kubernetes MLOps Ecosystem	23
Case Study: Volvo Cars	23
Getting Started	25
About TWIML	29

Sponsors

This copy of *Kubernetes for MLOps: Scaling Enterprise Machine Learning, Deep Learning, and AI* is brought to you by HPE and Intel. We thank them for their support.



Hewlett Packard Enterprise

Hewlett Packard Enterprise is the global edge-to-cloud platform-as-a-service company that helps organizations accelerate outcomes by unlocking value from all of their data, everywhere. Built on decades of reimagining the future and innovating to advance the way we live and work, HPE delivers unique, open and intelligent technology solutions, with a consistent experience across all clouds and edges, to help customers develop new business models, engage in new ways, and increase operational performance. For more information, visit: www.hpe.com.



Intel is an industry leader, creating world-changing technology that enables global progress and enriches lives. Inspired by Moore's Law, we continuously work to advance the design and manufacturing of semiconductors to help address our customers' greatest challenges. By embedding intelligence in the cloud, network, edge and every kind of computing device, we unleash the potential of data to transform business and society for the better. To learn more about Intel's innovations, visit www.intel.com.

Preface to the Second Edition

In December 2018, I raced to complete the first edition of this ebook in time for Kubecon Seattle. After weeks of research, interviews, and writing, I was excited to unveil a timely and needed resource for a small but growing community.

I made the deadline, but as I sat through the conference's keynotes and sessions, listening to the many new announcements relating to machine learning (ML) and artificial intelligence (AI) workloads on Kubernetes, my heart sank. While I knew the ebook would prove a valuable resource to readers, I knew too that it would soon need an update.

Since that time, the use of Kubernetes to support an organization's ML/AI projects has evolved from an obscure idea to one being explored and pursued by many organizations facing the challenge of getting ML models into production and scaling them.

As interest in running ML/AI workloads atop Kubernetes has grown, so has the surrounding ecosystem. When the first edition was written, KubeFlow was yet in its infancy, having just seen its 0.3 release.¹ At the time, the Google-founded project was far from production ready, and still fairly narrowly focused on running TensorFlow jobs on Kubernetes.

KubeFlow has since published a general availability (GA) release of the project, and now aspires to support a wide variety of end-to-end machine learning workflows on Kubernetes. The announcement of version 1.0 of the project in March of 2020 was accompanied by testimonials from companies such as US Bank, Chase Commercial Bank, and Volvo Cars. (I'm excited to include the first public profile of Volvo's journey in this book.)

The broader Kubernetes ML/AI ecosystem has exploded as well, with a wide variety of open source and commercial offerings building upon the foundation offered by Kubernetes. We'll mention a few of these in this book, but with so many services and offerings to keep track of, and with offerings evolving so quickly, we've decided to move our directory of open source and commercial tools to a new home: the online [**TWIML AI Solutions Guide**](#).

Following the evolution of this space has been a fascinating journey, and I'm pleased to bring you the second edition of this ebook. My hope is that the information shared here will help you better understand this promising direction for building, deploying, and scaling machine learning models in your organization, and ultimately contribute to your organization's own innovative use of machine learning and AI.

Thanks for reading!

Sam

Introduction

Enterprise interest in machine learning and artificial intelligence continues to grow, with organizations dedicating increasingly large teams and resources to ML/AI projects. As businesses scale their investments, it becomes critical to build repeatable, efficient, and sustainable processes for model development and deployment.

The move to drive more consistent and efficient processes in machine learning parallels efforts towards the same goals in software development. Whereas the latter has come to be called DevOps, the former is increasingly referred to as MLOps.

While DevOps, and likewise MLOps, are principally about practices rather than technology, to the extent that those practices are focused on automation and repeatability, tools have been an important contributor to their rise. In particular, the advent of container technologies like Docker was a significant enabler of DevOps, allowing users to drive increased agility, efficiency, manageability, and scalability in their software development efforts.

Containers remain a foundational technology for both DevOps and MLOps. Containers provide a core piece of functionality that allow us to run a given piece of code—whether a notebook, an experiment, or a deployed model—anywhere, without the “dependency hell” that plagues other methods of sharing software. But, additional technology is required to scale containers to support large teams, workloads, or applications. This technology is known as a container orchestration system, the most popular of which is Kubernetes.

In this book we explore the role that Kubernetes plays in supporting MLOps:

- **Chapter 1** presents a high-level overview of the machine learning process and the challenges organizations tend to encounter when it comes time to scale it.
- **Chapter 2** elaborates on why supporting machine learning at scale is so difficult, and covers a few of the many challenges faced by data scientists and machine learning engineers in getting their models into production, at scale.
- **Chapter 3** provides an introduction to container technology and Kubernetes, and shows how they help address the challenges referenced earlier.
- **In chapter 4**, we zoom in to discuss Kubernetes in more detail.
- **In chapter 5**, we return to the ML workflow, and show how Kubernetes can help organizations overcome the various challenges it presents.
- **Chapters 6 and 7** look at the role of Kubeflow and other higher-level tools in delivering MLOps.
- **In chapter 8**, we get an opportunity to sit in the passenger seat and ride along with Volvo Cars, an early Kubernetes and Kubeflow user, to learn from their experiences building a ML platform atop these technologies.
- **Chapter 8** offers some brief advice on how you can get started with your own project.

If you're just getting started with MLOps, know that there's no better time than the present to begin this journey. The user and practitioner communities are exploding, the practices and technology are maturing, and the support and educational resources are more plentiful and of higher quality with each passing day.

The Machine Learning Process

Before we can discuss enabling and scaling the delivery of machine learning, we must understand the full scope of the machine learning process. While much of the industry dialogue around machine learning continues to be focused on modeling, the reality is that modeling is a small part of the process. This was poignantly illustrated in a 2015 paper² by authors from Google, and remains true today.

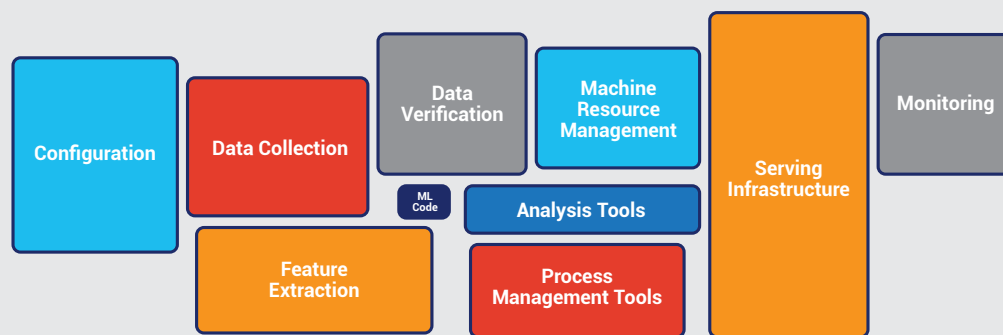


Figure 1. Illustration from Hidden Technical Debt in Machine Learning Systems paper. Machine learning models are represented by the small box labeled “ML Code” in the center of the diagram. The paper refers to the supporting infrastructure as “vast and complex.”

In order for organizations to reap the rewards of their machine learning efforts and achieve any level of scale and efficiency, models must be developed within a repeatable process that accounts for the critical activities that precede and follow model development.

When presented with a new problem to solve, data scientists will develop and run a series of experiments that take available data as input and aim to produce an acceptable model as output. Figure 2 presents a high-level view of the end-to-end ML workflow, representing the three major “steps” in the process—data acquisition and preparation, experiment management and model development, and model deployment and performance monitoring—as well as the iterative loops that connect them.

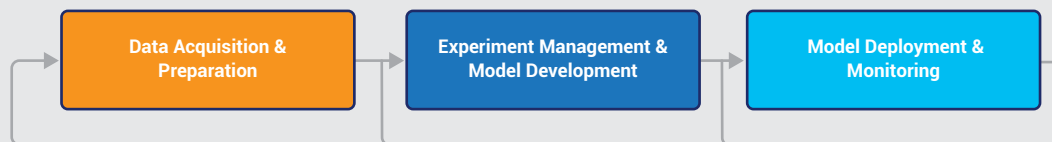


Figure 2. The machine learning process

Data acquisition & preparation

To build high quality 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 or 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.

In the real world, as opposed to in academia or Kaggle competitions, the creation of training datasets usually involves combining data from multiple 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.

While the specific use of *labeled* training datasets is characteristic of supervised learning, as opposed to unsupervised, self-supervised or reinforcement learning, the former remains 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.



Once compiled, 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 of churn among customers 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 are basing the prediction on, the number of days without a transaction, the window in which we want to make predictions, and more.³

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.⁴

If not accounted for during planning or supported by appropriate processes and tools, data acquisition can consume all of the time and resources allocated for a project, leaving little available for the remaining steps in the machine learning process.

Experiment management and model development

Experimentation is central to the machine learning process. During modeling, data scientists and machine learning engineers (MLEs) run a series of experiments to identify a robust predictive model. Typically, many models—possibly hundreds or even thousands—will be trained and evaluated in order to identify the techniques, architectures, learning algorithms, and parameters that work best for a particular problem.

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 lockstep with model training, because the ability to identify helpful features can have a significant impact on the overall success of the modeling effort. That said, a fine line separates the preliminary data cleansing and normalization steps associated with data preparation from feature engineering.

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 deep neural networks (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 and their understanding of the business domain or ability to effectively collaborate with domain experts.


In addition, a variety of tools under the broad banner of “AutoML” are finding popular use to automate aspects of experimentation and model development. Offerings exist to automate feature engineering by creating and testing a wide variety of candidate features, identify the best machine learning or deep learning models (the latter referred to as neural architecture search), automatically tune or optimize model hyperparameters, and automate the compression of large DNNs so that they're more amenable to mobile or edge deployment.

Whether manually performed or automated, the various aspects of experimentation and model training can be quite computationally intensive, especially in the case of deep learning. Having ready access to the right infrastructure for each experiment, such as CPUs, GPUs, memory, etc.—whether locally or in the cloud—can significantly impact the speed and agility of the data science team during the model development phase of the process.

Model deployment

Once a model has been developed, it must be deployed in order to be used. While deployed models can take many forms, typically the model is embedded directly into application code or put behind an API of some sort. HTTP-based (i.e. REST or gRPC) APIs are increasingly used so that developers can access model predictions as microservices.

While model training might require large bursts of computing hardware over the course of several hours, days or weeks, model inference—making queries against models—can be even more computationally expensive than training over time. 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.



Or not so small in the case of some deep learning models. 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 3 minutes a day, meeting the inference demands of speech recognition would require Google data centers to double in capacity.⁵

Meeting the requirements of inference at scale is a classic systems engineering problem. Addressing issues like scalability, availability, latency, and cost are typically primary concerns. When mobile or edge deployment is required, additional considerations like processing cycles, memory, size, weight, and power consumption come into play.

At a certain point, we often end up needing to turn to distributed systems to meet our scalability or performance goals. This of course brings along its own set of challenges and operational considerations. How do we get our model onto multiple machines and ensure consistency over time. How do we update to new models without taking our applications out of service? What happens when problems arise during an upgrade? How can we test new models on live traffic?

Fortunately for those deploying models, much progress has been made addressing these questions for microservices and other software systems.

Model monitoring

Once a model is put into production, it is important to monitor its ongoing performance. Machine learning models are perishable, meaning that the statistical distribution of the data a model sees in production will inevitably start to drift away from that seen during training. This will cause model performance to degrade, which can result in negative business outcomes such as lost sales or undetected fraudulent activity if not detected and addressed.

Production 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 can use this information to take corrective action, such as retraining or re-tuning the model.

Business or regulatory requirements may impose additional requirements dictating the form or function of model monitoring for audit or compliance purposes, including in some cases the ability to explain model decisions.

Model monitoring is generally a concern shared by all of an organization's production models, and thus ideally supported by a common framework or platform. This has the advantage of making monitoring "free" for data scientists, that is, something they get the benefit of but don't need to worry about building. While some models or applications may have unique monitoring or reporting requirements requiring the involvement of data scientists or ML engineering staff, they should ideally be able to take advantage of low level "plumbing" that is already in place.

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 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 hand crafted with little automation or integration between steps.

Once a team has operated this way for more than a handful of projects, it becomes clear that a great deal of effort is spent on repetitive tasks, 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. This can be an effective stopgap, but left unplanned and uncoordinated, these efforts can be a source of distraction and lead to technical debt.

For organizations at a certain level of scale—typically when multiple machine learning teams and their projects must be supported simultaneously—"data science platform" or "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.🔗



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.

Common challenges encountered by ML infrastructure teams include:

Simplifying and automating data access & management

Because so much of model building involves acquiring and manipulating data, providing a way to simplify and automate data acquisition and data transformation, feature engineering, and ETL pipelines is necessary to increase modeling efficiency and ensure reproducibility.

Data acquisition is greatly facilitated when a centralized data repository or directory is available, such as a data lake, fabric, warehouse, or catalog. These enable efficient data storage, management, and discovery, allowing data that is generated from a variety of disparate systems to be more easily worked with by minimizing the time data scientists spend looking for data or trying to figure out how to access new systems.

Data and feature transformations create new data needed for training and often inference. The data produced by these transformations is not typically saved back to the systems of origin, such as transactional databases or log storage. Rather, it is persisted back to facilities such as those mentioned above. Ideally, transformation and feature engineering pipelines are cataloged for easy sharing across projects and teams.

In addition to the efficiency benefits they offer, feature data repositories can also help eliminate time consuming and wasteful data replication when architected in a way that meets the latency and throughput requirements of a wide range of machine learning training and inference workloads. Because of the scalability, security, and other technical requirements of feature data repositories, ML infrastructure teams typically work with data engineers and corporate IT to establish them.

Driving efficient resource use

Today, we have more raw computing power at our disposal than ever before. In addition, innovations such as high-density CPU cores, GPUs, TPUs, FPGAs, and other hardware accelerators are increasingly targeting ML and DL 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:

- **Multitenancy.** 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.

- **Elasticity.** Data preparation, model training, and model inference are all highly variable workloads, with the amount and type of resources they require often varying widely in time. To efficiently meet the needs of a portfolio of machine learning workloads it is best when the resources dedicated to individual tasks can be scaled up when needed, and scaled back down when done.

- **Immediacy.** Data scientists and MLEs should have direct, self-service access to the number and type of computing resources they need for training and testing models without waiting for manual provisioning.

- **Programmability.** The creation, configuration, deployment and scaling of new environments and workloads should be available via APIs to enable automated infrastructure provisioning and to maximize resource utilization.

These are, of course, the characteristics of modern, cloud-based environments. However, this does not mean that we're required to use the third-party "public" cloud services to do machine learning at scale.

While the public cloud's operating characteristics make it a strong choice for running some machine learning workloads, there are often other considerations at play. Performance requirements often demand co-locating training and inference workloads with production applications and data in order to minimize latency.

Economics is an important consideration as well. The cost of renting computing infrastructure in the cloud can be high, as can the cost of inbound and outbound data transfers, leading many organizations to choose local servers instead.

As both cloud and on-premises resources have their place, hybrid cloud deployments that harness resources from both are a worthy consideration for many organizations. Hybrid cloud deployment allows organizations to distribute workloads across cloud and on-premises resources in ways that allow them to quickly and cost effectively meet fluctuating workload demands and provide increased computational power when needed.

Ultimately, in a world of rapid hardware innovation, dynamic infrastructure economics, and shifting workloads, it is prudent to build flexibility into new tools and platforms, so that they can be efficiently operated in any of these environments.

Hiding complexity

With the rise of Platform-as-a-Service (PaaS) offerings and DevOps automation tools, software developers gained the ability to operate at a higher level of abstraction, 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 more established tools like the TensorFlow and PyTorch deep learning frameworks; language-specific libraries like SciPy, NumPy and Pandas; and data processing engines like Spark and MapReduce. In addition, there has been an explosion of specialty tools aiming to address specific pain-points in the machine learning process, many of which we discuss in our accompanying book, *The Definitive Guide to Machine Learning Platforms*, and cover in our online [ML/AI Solutions Guide](#).

These high-level tools are supported by a variety of low-level, vendor-provided drivers and libraries, allowing training and inference workloads to take advantage of hardware acceleration capabilities. Some of these driver stacks are notoriously difficult to correctly install and configure, requiring that complex sets of interdependencies be satisfied.

Manually managing rapidly churning software tools and the resulting web of dependencies can be a constant drain on data scientist productivity, and the source of hard-to-debug discrepancies between results seen in training and production.

Unifying the ML workflow

Ultimately, as an organization's use of data science and machine learning matures, both business and technical stakeholders alike benefit from a unified ML workflow, with a common framework for working with the organization's data, experiments, models, and tools.

The benefits of a common platform apply across the ML workflow. A unified view of data, as we've previously discussed, helps data scientists find the data they need to build models more quickly. A unified view of experiments helps individual users and teams identify what's working faster, and helps managers understand how resources are being allocated. A unified view of deployed models helps operations and DevOps teams monitor performance across a wide fleet of services. And a unified view of infrastructure helps data scientists more readily access the resources they need for training.

With a unified approach to the machine learning workflow, it becomes much easier to facilitate and manage cross-team collaboration, promote the reuse of existing resources, and take advantage of shared skills. It also enables individual team members to more quickly become productive when transitioning to new projects, driving increased efficiency and better overall outcomes for data science and ML/AI projects.

Enter Containers and Kubernetes

Organizations are increasingly turning to containers and Kubernetes to overcome the challenges of scaling their machine and deep learning efforts.

It turns out we've seen this before. The introduction of Docker containers in 2013 initiated a dramatic shift in the way software applications are developed and deployed by allowing engineering teams to overcome a similar set of challenges to those faced by data scientists and ML engineers.

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 difficulties faced by software development and operations organizations as they sought to modernize their software delivery processes, applications, and infrastructure, leading to their ultimate popularity.

But containers alone offer a partial solution. For organizations operating at scale, supporting multiple concurrent machine learning projects in various stages of development, it quickly becomes necessary to use multiple containers, make the services they offer easily accessible, and connect them to a variety of external data sources. In this case, a container orchestration platform is required—in practice if not in theory—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, due to its popularity among users, the flexibility it confers to them, the support it has gained from cloud computing and software vendors, and the degree of portability it offers 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 Kubernetes **Master** which controls multiple worker machines called nodes. Clusters abstract their underlying computing resources, allowing users to deploy workloads to the cluster, as opposed to on particular **nodes**.

A **pod** is a collection of one or more containers that share common configuration and are run on the same machine. It is the basic workload unit in Kubernetes. All containers within a pod share the same context, resources, and lifecycle. Resources include local and remote storage volumes and networking. All containers in a pod share an IP address and port space. To run containers in pods, Kubernetes uses a container runtime, such as Docker, running on each node.

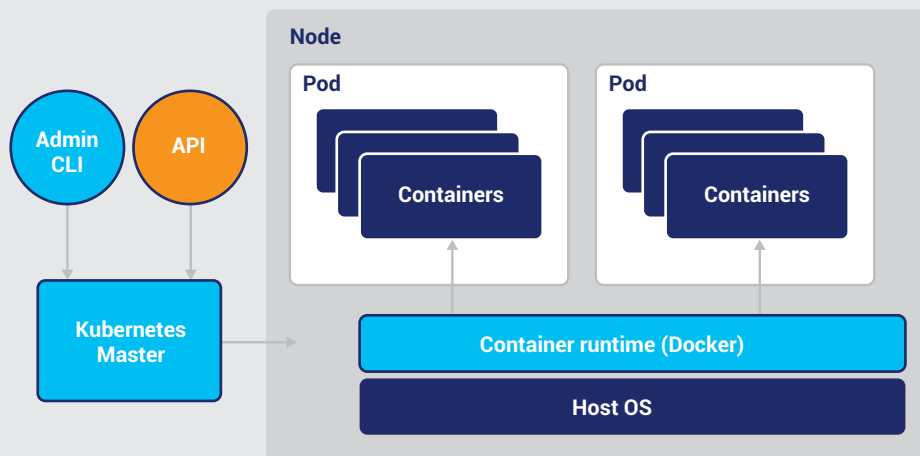


Figure 3. Basic Kubernetes architecture

The 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 run an agent called the **kubelet** that listens for instructions from the master and creates, runs, and destroys containers accordingly. Collectively, the master and the kubelet processes govern the operation of the cluster and are referred to as the Kubernetes Control Plane.

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 configuration documents which are sent to the cluster via Kubernetes' APIs or client libraries. In this way, Kubernetes is said to be a "declarative" system; users declare the state of the system they want, and Kubernetes tries to affect that state given the resources at its disposal.

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 configuration documents which are sent to the cluster via Kubernetes' APIs or client libraries. In this way, Kubernetes is said to be a "declarative" system; users declare the state of the system they want, and Kubernetes tries to affect that state given the resources at its disposal.

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.

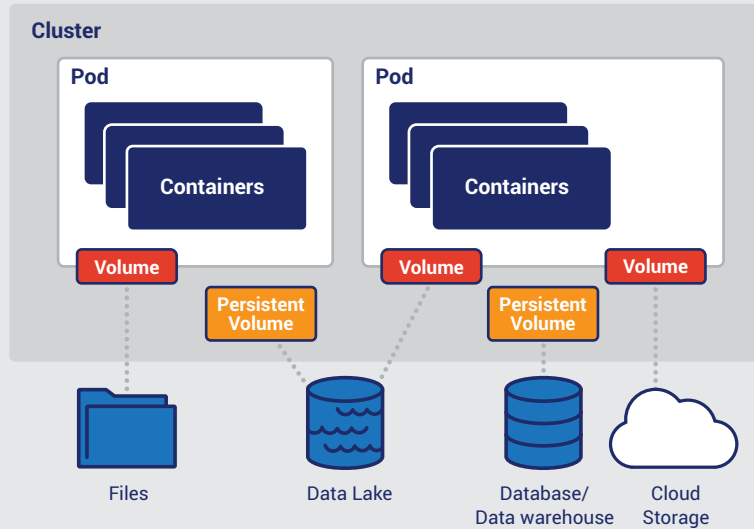


Figure 4. Kubernetes volumes provide a flexible abstraction for accessing a wide variety of storage systems from containerized workloads

Kubernetes has long supported first- and third-party volume plugins allowing users to access different types of storage resources. More recently, the general availability of support for the Container Storage Interface (CSI) specification has significantly expanded the ability to access a wide variety of software and hardware-based storage systems via community- and vendor-provided drivers.

Beyond CSI, Kubernetes offers additional capabilities that allow users to customize their deployments and support complex application scenarios, including:

- **Custom resources**, which extend the Kubernetes API to manage new types of objects
- **Operators**, which combine custom resources and controllers to manage stateful applications and systems
- **Scheduler extensions**, which can be used to change the way Kubernetes manages pods and assigns them to nodes
- **The Container Network Interface (CNI)** which provides a common interface between containers and the networking layer
- **Device plugins**, which allow Kubernetes nodes to discover new types of hardware resources and perform vendor specific initialization and setup. Once made discoverable, the availability of these resources can then be considered by the Kubernetes scheduler when making pod placement decisions.

"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

While the challenges faced by developers building traditional software are similar to those faced by data scientists and ML engineers, they aren't the same. In many ways, those faced by the latter are greater, significantly exacerbated by the highly iterative nature of the machine learning workflow.

Still, the capabilities provided by containers and Kubernetes can help organizations address the challenges presented by the machine learning process.

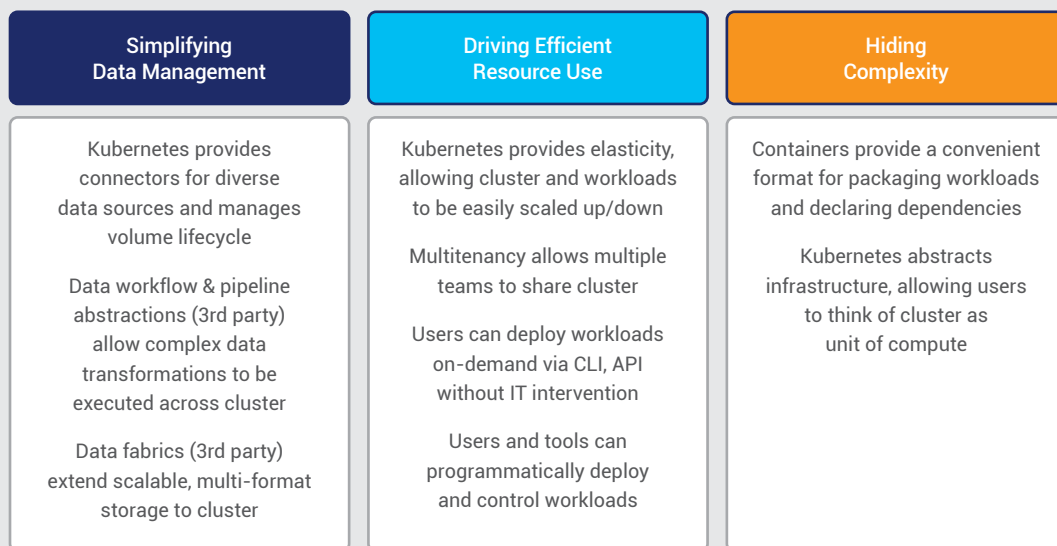


Figure 5. Containers and Kubernetes address major ML/DL challenges

Data acquisition and preparation

By providing a mechanism for connecting storage to containerized workloads, the Kubernetes CSI enables support for a wide variety of stateful applications, including machine and deep learning. CSI drivers provide direct access to many different file, object, and block storage systems, as well as data fabric products which provide unified access to data stored in a variety of enterprise systems.

Ready access to scalable storage is the foundation upon which users can construct automated and repeatable data processing pipelines. This ensures the ability to reliably deliver properly transformed data to models, without manual intervention, and also facilitates the reuse of transformed data and processed features across projects.

Driving efficient resource use

For resource-intensive workloads like machine learning, users face the constant conundrum of over-allocation or under-utilization. That is, they must manage the fine balance between either having more resources than is needed at any particular point in time, or not enough.

Kubernetes offers two solutions to this problem. The first is multi-tenancy. A single physical Kubernetes cluster can be partitioned into multiple virtual clusters using Kubernetes' **namespaces** feature. This allows a single cluster to easily support multiple 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. Namespace-level configuration allows resources like storage to be configured on a per-team, -project, or -function basis.

Multi-tenancy features allow many users to share the same physical resources, with the aggregate usage having less variance and requiring less over-provisioning. Another solution Kubernetes offers to the resource efficiency problem is auto-scaling, of which there are two types.

Workload (or pod) autoscaling which is the ability to scale up the number of pods running a given workload as needed. This works with multi-tenancy to take full advantage of the resources available in a given Kubernetes cluster

Cluster autoscaling is the ability to scale up or down the number of nodes in the entire cluster without disrupting the cluster or any of its applications or users. This is helpful, but of limited utility, when you're running the cluster on physical nodes in your own datacenter. When you're running Kubernetes on virtual servers in the cloud, though, this allows you to continually right-size your cluster so you're only paying for the resources you need at any given time.

Because Kubernetes-enabled workloads can be run on any Kubernetes cluster in any location without any changes to the application's code, users can move applications between deployment environments with relative ease. In theory a single Kubernetes cluster can span clouds and/or on-premises data centers, but this is difficult to implement and not recommended.

Hiding complexity

Containers provide an efficient way of packaging machine learning workloads that is independent of language or framework. Kubernetes builds on this to provide 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, these tools will be familiar to machine learning engineers exposed to modern software development practices.

Custom and third-party user interfaces, command-line tools, or integration with source code repositories can simplify creating and managing containers for end users. Because Kubernetes is workload independent, once users are familiar with the tools required to work with their Kubernetes environment, they can apply these tools across a wide range of projects.

While introducing its own set of tools, the use of containers to facilitate data acquisition, experiment management and model deployment ensures that the correct and consistent dependencies are in place wherever jobs are run—whether on the developer laptop, training environment, or production cluster. This provides the valuable benefit of shielding those workloads from the complexity of the underlying technology stack, while supporting a wide variety of runtime configurations.

Kubernetes also helps support and enforce a separation of concerns, giving end-users control over defining their workloads and administrators control over the properties and configuration of the infrastructure.

Unifying the ML workflow

For all its benefits, Kubernetes alone won't unify the end-to-end machine learning process. It will go a long way, however, in doing so for much of the software and hardware infrastructure that supports machine learning, and, in achieving that, it provides a strong platform for tools that go even further.

MLOps on Kubernetes with Kubeflow

Kubeflow is an open source project designed to make machine learning workflows on Kubernetes simple, portable and scalable. Kubeflow is sponsored by Google and inspired by TensorFlow Extended, or TFX, the company's internal machine learning platform.

Originally intended to simply allow TensorFlow users to run training jobs on their Kubernetes clusters, the project now integrates a broad set of tools in support of many steps in an end-to-end machine learning process.

Kubeflow includes components for:

- **Launching Jupyter Notebooks.** Kubeflow includes support for setting up Jupyter notebook servers and creating and managing notebooks. Notebooks launched via Kubeflow are integrated with the authentication and access control policies an organization has established with its Kubernetes deployment, are easier to share across the organization, can help organizations standardize notebook images, and can readily take advantage of the compute capacity and other features available in the Kubernetes environment.

- **Building ML Pipelines.** Kubeflow Pipelines is a platform for building and deploying complex, repeatable ML workflows on Kubernetes. Pipelines themselves consist of a graph detailing the relationships between steps in a workflow. The workflow steps themselves are called components, and are provided by the user as Docker containers. Pipelines allow users to package up the various data access and transformation, model training, and model evaluation steps that make up an experiment, simplifying experimentation and re-use.
- **Training Models.** Kubeflow provides a variety of Kubernetes Operators allowing users to easily create and manage training jobs for ML frameworks like TensorFlow, PyTorch, MXNet, XGBoost. In addition to the distributed training support provided by these frameworks, it also supports MPI-based distributed training through the MPI Operator.
- **Tracking Experiment Metadata.** The Kubeflow Metadata project provides a Python SDK for tracking and managing information about the executions (runs), models, datasets, and other artifacts generated during machine learning. It also provides a UI for viewing a list of logged artifacts and their details.
- **Hyperparameter Tuning.** Tuning model hyperparameters is an important element of optimizing model performance and accuracy, yet manually adjusting hyperparameters during experimentation is tedious. Kubeflow includes Katib, an open-source hyperparameter tuning project inspired by Google's internal black-box optimization service, Vizieer. Katib is framework agnostic and deploys optimization tasks as Kubernetes pods.
- **Serving Models.** Users may choose from two model serving systems within Kubeflow—KFServing and Seldon Core—as well as any of several standalone model serving systems. These options support a wide variety of ML frameworks, and offer features supporting various strategies for model rollout, scaling, logging, and analytics.

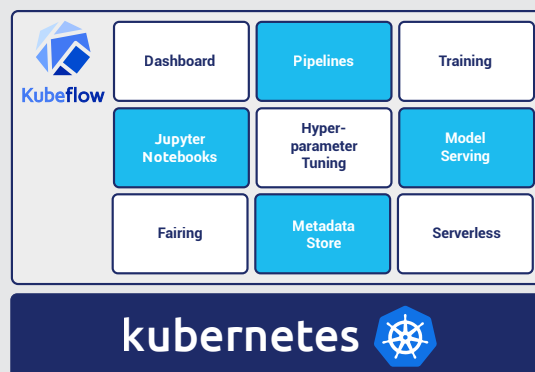


Figure 6. Kubeflow builds on top of Kubernetes to address key challenges in the ML workflow

In addition, Kubeflow provides a central dashboard through which users can access and manage the various components of a Kubeflow deployment, support for multi-tenancy and multi-user isolation, and through Kubeflow Fairing, the ability to more easily build, train, and deploy training jobs across a hybrid cloud environment directly from a Jupyter notebook.

Kubeflow exposes Kubernetes concepts like namespaces and pods to users and requires that they use command-line 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 traditional data scientist. The requisite job configuration files are not altogether unfamiliar, however, with their configuration parameters resembling those of other distributed tools they may be familiar with, like Apache Spark.

Furthermore, the project is making progress in this regard, with the dashboard and pipeline visualizations representing solid progress. Some users choose to build custom user interfaces for their data scientists that aim to further abstract the underlying Kubeflow and Kubernetes tooling.

Using Kubeflow allows users to save time integrating support for common ML and DL components. In the past, Kubeflow's integrations felt rather shallow, and the project felt a bit like a hodgepodge of third-party open source offerings. To its credit, the project is now a much more highly curated and unified offering.

That said, Kubeflow is still a collection of components under the covers. Though the Kubeflow core has attained GA status, the overall project remains immature, with many of the aforementioned components still in beta. This state of affairs may prove problematic for some organizations, who value the stability and predictability of more mature and tightly integrated offerings. Further, some enterprise IT organizations might find that Kubeflow lacks an adequate security and governance framework that spans workloads, data and infrastructure, though progress is being made in this regard.

On the topic of governance, though a different kind, there are concerns in some circles with Kubeflow's lack of independent governance. While the Kubernetes project is overseen by the CNCF, which provides for a certain level of transparency and community-driven governance, Kubeflow, remains under Google's control. This creates some uncertainty about the future direction of the project, though the project does have a broad base of contributors and advocates.

The Kubernetes MLOps Ecosystem

Kubeflow is one of the many available offerings that take advantage of Kubernetes to deliver MLOps capabilities. These offerings range in scope from narrow tools that aim to address particular pain-points experienced by data scientists, to end-to-end platforms that seek to provide comprehensive support for the ML workflow.🐦 Licensing and delivery models vary as well, including open source projects, commercially supported software products, and cloud-based services.



One of the most important distinctions among the various tools available to help you build out your organization's machine learning platform is whether the tool aims to be wide or deep. This distinction, and the interesting paradox that arises as a result, is explored in detail in TWIML's *The Definitive Guide to Machine Learning Platforms*.



The ML/AI tools, infrastructure, and platforms landscape continues to evolve rapidly. For this reason we have developed the *TWIML ML/AI Solutions Guide*, an online resource available at twimlai.com/solutions.

Visit the Solutions Guide for profiles of ML/AI tools and platforms supporting Kubernetes and Kubeflow, and check back frequently for updates as we expand its coverage and features.

Driving Data Science Innovation with Kubernetes: Volvo Cars' Journey

Sweden-based Volvo Cars is a luxury automobile manufacturer whose brand has long been associated with safety and innovation. The company's **Consumer Digital group** is responsible for a variety of products that aim to enhance the owner experience, including the Volvo On Call app, which offers features like roadside assistance, remote climate control, and the ability to lock and unlock owners' cars from a distance.

Data scientists in the Consumer Digital group use machine learning to both deliver end-user features, like search and recommendations, as well as to better understand how their products are being used.

The group's data scientists began using Kubernetes to support their workloads in 2018, when they needed a way to automate data acquisition, transformation, and reporting pipelines in support of a high-profile collaboration with Amazon to allow Volvo owners to receive in-trunk deliveries. The team wanted to build these pipelines using Jupyter Notebooks, and found no off-the-shelf tools at the company that let them easily do this.

Inspired by work done at Netflix⁶, Leonard Aukea, senior data scientist and tech lead for the company's data science platform, set out to build a notebook-centric data science workflow for his team.

The Netflix system, though container-based, used that company's homegrown container orchestration system. At Volvo, however, Kubernetes was already gaining traction to support containerized workloads. With the help of a DevOps colleague and considerable research, Leonard was able to get a Kubernetes cluster up and running for data science workloads.

The team's first step was to set up the Apache Airflow workflow management system on Kubernetes, and use this in conjunction with the Netflix Papermill tool to run notebooks as steps of a pipeline. Pipeline outputs were fed into Nteract, another Netflix tool, providing a convenient, notebook-based dashboard.

Automated pipelines were a big win for Leonard's team, and other teams soon reached out for help with their own workloads. It was then that the limitations of the team's initial approach became apparent.

One challenge was that the elements of the system weren't tightly integrated. Users had to visit several different internal web sites to get anything done.

"Here, you visualized the notebook; here, you specify the DAG in this GitHub repository; here, you go to the Airflow UI; and then here, you go to JupyterHub. It became a mess. It was hard to explain... and it didn't scale in the way that we wanted."

After obtaining the go-ahead to put a more enterprise-ready solution in place, Leonard eventually found the Kubeflow project.

The team at Volvo appreciated that Kubeflow is Kubernetes-native, so everything fit together under a unified user interface and set of APIs.

Kubeflow "glues all of these services together and gives a unified interface that a user could go in and enter the platform and do their job in an easier fashion."

"Some things were working quite nicely... in terms of setup, and then moving from one version to another, and just reliability of [the] service."

That didn't mean that everything was perfect, however.

The team found Kubeflow "hard to configure because you need to configure all of these individual services, dig into the manifests that you're actually deploying, and [deploy them using the] kubectl tool. I'm not a hundred percent a fan of it... You have to spend significant [effort] in order to make it work."

The integrated nature of the tool put more responsibility on Leonard and his team when it came to support.

They often found themselves needing to “dig deeper into the individual applications and look at source code, for example, where documentation is not centralized. [...] It’s also not what you would traditionally expect from a service in that sense.”

As of this writing, Leonard’s data science platforms team has Kubeflow running in beta with several teams on board. Adoption has been relatively smooth, though the team acknowledges a steep learning curve. The use of Docker containers and Kubeflow’s tooling is more natural for the Python data scientists they support than for the company’s R-based data scientists.

Ultimately, the team sees in Kubeflow a way to help the company’s data scientists move more quickly and be more self-sufficient. They want data science teams that deploy models to be responsible for them throughout their lifecycle, but also to have access to the tools and training and best practices that are needed for them to be successful.

The platform team is currently gathering statistics to compare user productivity on the new platform with previous results.

“I don’t have actual numbers on it, but what I hear from other teams and also [my own use], it’s way more effective when you get the hang of it.”

“I’m hoping that we have a lot more models in production that would otherwise go to waste or have been laying stagnant for a while in the organization because [a team] has not had the available tooling or the support in order to get their model to production. This could be very valuable for Volvo Cars as a company.”

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.

This is a great time to be getting started with Kubernetes and MLOps. The community of ML/AI-focused Kubernetes users is large, growing, and welcoming; the platform and tools are viable for production workloads and maturing quickly; and there is a growing pool of talent with experience in the requisite tech stack. (And, conversely, the skills you learn helping your organization scale its MLOps platform are portable and valued in the marketplace.)

While powerful, Kubernetes brings with it a degree of complexity that can be challenging for small data science teams. It is most appropriate for those organizations whose scale, resources, and requirements justify the investment not only in a centralized platform for data science but also a dedicated team to support it.

Today, organizations adopting Kubernetes for machine learning and AI have the benefit of many options to help mitigate its complexity. Each of the layers of the MLOps stack—from the physical infrastructure such as servers, network and storage; to the Kubernetes control plane; to MLOps software providing abstractions such as pipelines and experiments—can be implemented on-premises using off-the-shelf components or can be obtained as a fully managed service by various vendors.

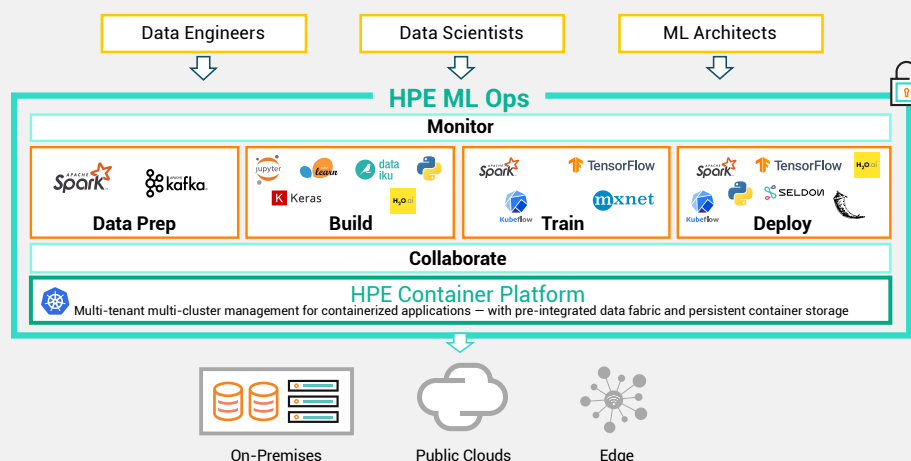
As with any new technology, the best course of action is to research and understand your requirements and options and start small, with your feet grounded firmly in your current needs but with an eye towards the future. Your exploration of Kubernetes can start with a “cluster” running on your laptop 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. From there you can grow as your experience, needs, and resources allow.

Godspeed.

HPE ML OPS AND INTEL:

PARTNERING TO DELIVER ENTERPRISE MACHINE LEARNING AT SCALE

Hewlett Packard Enterprise empowers large enterprises to overcome the barriers in deploying and operationalizing AI/ML across the organization. HPE ML Ops brings DevOps-like speed and agility to the ML lifecycle. The HPE ML Ops solution supports every stage of ML lifecycle—data preparation, model build, model training, model deployment, collaboration, and monitoring. HPE ML Ops is an end-to-end data science solution with the flexibility to run on-premises, in multiple public clouds, or in a hybrid model, allowing customers to respond to dynamic business requirements for a variety of use cases.



HPE ML Ops offers:

Model building	Prepackaged, self-service sandbox environments
Model training	Scalable training environments with secure access to big data
Model deployment	Flexible, scalable, endpoint deployment
Model monitoring	End-to-end visibility to track, measure, report model performance across the ML pipeline
Collaboration	Enable CI/CD workflows with code, model, and project repositories
Security and control	Secure multitenancy with integration to enterprise authentication mechanisms
Hybrid deployment	Deploy on-premises, public cloud, or hybrid

Learn more at hpe.com/info/MLOps.

HPE solutions take advantage of the latest Intel® technology to deliver the performance required to scale machine learning workloads across the enterprise on known and trusted infrastructure.

Intel innovations include:

- **Proven, world-class 2nd generation Intel® Xeon® Scalable processors** deliver scalable performance for a wide variety of analytics and other enterprise applications. Featuring Intel® Optane™ persistent memory (PMem), this powerful combination offers increased memory capacity close to the CPU to support in-memory workloads, along with native persistence enabling greater resilience and productivity.
- **Intel® Deep Learning Boost** brings new embedded performance acceleration for advanced analytics and AI workloads, with up to 30x performance improvement for Inference workloads compared to the previous generation – laying the foundations for future evolution of your analytics and AI capabilities.

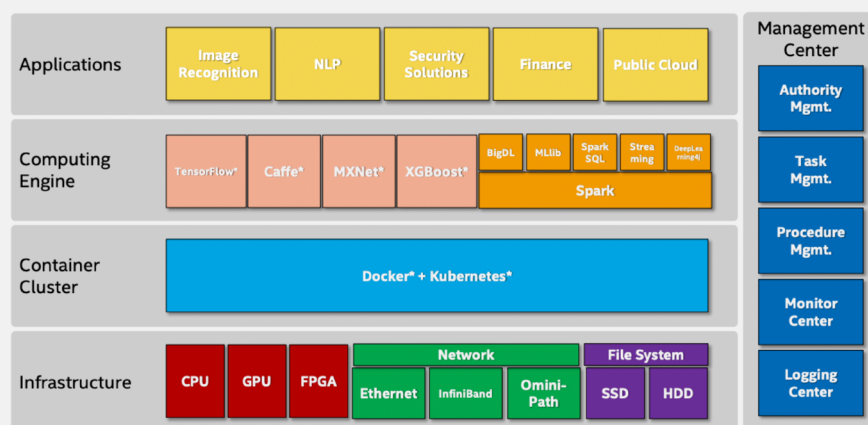
- **A track record of contributions to Kubernetes and Kubeflow**, including support for AVX instruction set feature discovery, enhancements to the Kubernetes scheduling framework, contribution of the CRI Resource Manager, and development of the Kubernetes Topology Manager which enhances support for high-performance Non-Uniform Memory Access (NUMA) Nodes.

JD offers an example of what's possible when Intel® innovations are delivered at scale. JD.com is China's largest retailer, with hyperscale operations driven by its own internal needs as well as those of its cloud computing and AI customers. The company uses machine learning and AI in support of a number of retail use cases, including advertising, warehouse automation and robotics, drone delivery, facial recognition based payments, and personalized marketing.

In 2016, JD.com began development of JDOS, its customized and optimized version Kubernetes which today supports a wide range of workloads and applications. The company runs the world's largest production Kubernetes cluster with over 20,000 nodes.

Intel worked closely with JD to optimize their big data and machine learning frameworks and toolkits, such as Apache Spark on Kubernetes, TensorFlow, and others, which are exposed to users via a single Kubernetes-based architecture, which it calls Moonshot.

MOONSHOT ARCHITECTURE



Intel and JD collaborated on a custom version of the next-generation Intel® Xeon® Scalable processor to support JD's unique JD Cloud and eCommerce workloads. Beyond the CPU, JD also incorporated other Intel innovations such as Intel® Optane™ DC SSDs, and plans to adopt our upcoming Intel® Optane™ DC Persistent Memory, a new class of storage technology offering the unprecedented combination of high-capacity, affordability, and persistence.

These customizations provide significant performance gains on analytics and AI workloads, and as a result help JD differentiate their cloud service offerings, personalize the online shopping experience and improve supply chain efficiency.

About TWIML




Machine learning and artificial intelligence are dramatically changing how businesses operate and people live. Through our publications, podcast, and community, TWIML brings leading ideas and minds from the world of ML and AI to a broad and influential community of data scientists, engineers and tech-savvy business and IT leaders.

TWIML has its origins in This Week in Machine Learning & AI, a podcast we launched in 2016 to a small but enthusiastic reception. Fast forward to 2020 and the TWIML AI Podcast is now a leading voice in the field, with over eight million downloads and a large and engaged community following. Our offerings now include research reports like this one, as well as [online meetups](#) and [study groups](#), [conferences](#), and a variety of [educational content](#).

The TWIML AI Podcast remains at the heart of our mission. By sharing and amplifying the voices of a broad and diverse spectrum of machine learning and AI researchers, practitioners, and innovators, our programs help make ML and AI more accessible, and enhance the lives of our audience and their communities.

TWIML was founded by Sam Charrington, a sought after industry analyst, speaker, commentator and thought leader. Sam's research is focused on the business application of machine learning and AI, bringing AI-powered products to market, and AI-enabled and -enabling technology platforms.

Connect with us:

-  sam@twimlai.com
-  [@samcharrington](https://twitter.com/samcharrington)
-  [@twimlai](https://twitter.com/twimlai)
-  www.linkedin.com/in/samcharrington
-  twimlai.com

References

1. Jeremy Lewi, et al, "Kubeflow 0.3 Simplifies Setup & Improves ML Development," Kubeflow Blog, October 29, 2018, <https://medium.com/kubeflow/kubeflow-0-3-simplifies-setup-improves-ml-development-98b8ca10bd69>
2. D. Sculley, et al, "Hidden Technical Debt in Machine Learning Systems," Advances in Neural Information Processing Systems, <https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>
3. 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>
4. 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>
5. Norman Jouppi, et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," arXiv [cs.AR], <https://arxiv.org/abs/1704.04760v1>
6. Michelle Ufford, et al, "Beyond Interactive: Notebook Innovation at Netflix," The Netflix Tech Blog, August 16, 2018, <https://netflixtechblog.com/notebook-innovation-591ee3221233>



Copyright © 2020 CloudPulse Strategies. CloudPulse, TWiML, and the CloudPulse Strategies and TWiML logos are trademarks of CloudPulse Strategies, LLC.

This document makes descriptive reference to trademarks that may be owned by others. The use of such trademarks herein is not an assertion of ownership of such trademarks by CloudPulse and is not intended to represent or imply the existence of an association between CloudPulse and the lawful owners of such trademarks. Information regarding third-party products, services and organizations was obtained from publicly available sources, and CloudPulse cannot confirm the accuracy or reliability of such sources or information. Its inclusion does not imply an endorsement by or of any third party.