Scaling Up AI/ML with Kubernetes

Kubernetes Architecture for Data Science Workloads

The first section of this whitepaper explains how Kubernetes Architecture as a platform for containerized AI workloads came to be used inside many companies. The guide explains some of the things to consider when implementing Kubernetes architecture to orchestrate AI workloads.

The second section of this guide explains the basics of Kubernetes scheduling. The guide explains how Kubernetes, the de-facto choice for container orchestration, is not suited for scheduling and orchestration of Deep Learning workloads. We will address the specific areas where Kubernetes falls short for AI and how you can address those shortfalls.

Kubernetes Overview

Originally developed inside Google, Kubernetes has been an open-source project since June 2014 and managed by the Cloud Native Computing Foundation (CNCF) since Google and Linux partnered to found the CNCF in July 2015. Kubernetes is an orchestration system that automates the processes involved in running thousands of containers in production. It eliminates the infrastructure complexity associated with deploying, scaling, and managing containerized applications.

There is a strong correlation between the growth in containers and microservice architectures and the adoption of Kubernetes. According to a recent Gartner report, "By 2023, more than 70% of global organizations will be running more than two containerized applications in production, up from less than 20% in 2019." And Kubernetes usage will continue to grow as companies deepen their commitment to containerization. According to a recent survey of 250 IT professionals conducted by Dimensional Insight, "Well over half (59%) are running Kubernetes in a production environment, with one-third (33%) operating 26 clusters or more and one-fifth (20%) running more than 50 clusters." The Kubernetes website is full of case studies of companies from a wide range of verticals that have embraced Kubernetes to address business-critical use cases—from Booking.com, which leveraged Kubernetes to dramatically accelerate the development and deployment of new services; to CapitalOne, which uses Kubernetes as an "operating system" to multiply productivity while reducing costs; and the New York Times, which maximizes its cloud-native capabilities with Kubernetes-as-a-service on the Google Cloud Platform.
This guide looks specifically at how Kubernetes can be used to support data science workloads in general and machine/deep learning in particular. As data science workloads require some specific tooling for their needs, utilizing Kubernetes for deep learning has some challenges that we will identify in this post.

Kubernetes Architecture

Containers generally require automated orchestration that, for example, starts a particular container on demand, allows containers to talk to each other, dynamically spins up and terminates compute resources, recovers from failures and manages the lifecycle of containers, and generally ensures optimal performance and high availability. In this section, we review briefly how Kubernetes works.

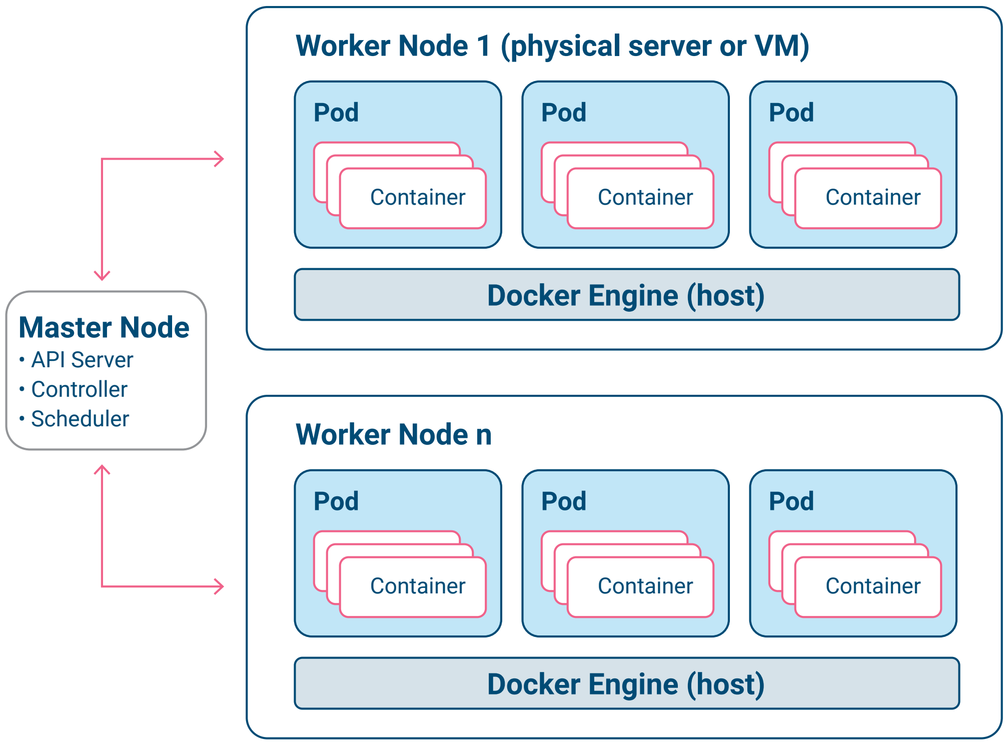

Figure 1: Schematic view of a Kubernetes cluster

As shown in Figure 1, each Kubernetes cluster contains at least one master node, which controls and schedules the cluster, and a number of worker nodes, each running one or more pods deployed to the same host (in our example, a Docker engine). A pod represents a unit of work and runs either a single container as an encapsulated service, or several tightly coupled containers that share network and storage resources. Kubernetes takes care of connecting pods to the infrastructure and managing them during runtime (monitoring, scaling, rolling deployments, etc.).

Every pod has its own IP address, which makes it easily discoverable to applications through Kubernetes service discovery. Multiple containers within a pod share the same IP address and network ports, while communicating among themselves using localhost.

Other Kubernetes concepts that are important to understand include:

- **Service:** A logical collection of pods presented as a single entity, with a single point of access and easy communications among pods in the service.
- **Volume:** A resource where containers can store and access data, including persistent volumes for stateful applications.
- **Label:** A user-defined metadata tag that makes Kubernetes resources easily searchable.
- **Job:** Jobs run containers to completion – that is, the containers start and end automatically. A job creates one or more pods and ensures that a specified number of them successfully run to completion. Jobs are particularly useful for running machine learning workloads, which will be addressed later in this guide.
- **Replica:** Pods do not self-heal. If a pod fails or is evicted for some reason, a replication controller immediately uses a template to start up another replica pod so that there are always the correct number of pods available.
- **Namespace:** A grouping mechanism for Kubernetes resources (pods, services, replication controllers, volumes, etc.) that isolates those resources within the cluster.

How Kubernetes Addresses Data Science Challenges

Containers and the Kubernetes ecosystem have been embraced by developers for their ability to abstract modern distributed applications from the infrastructure layer. Declarative deployments, real-time continuous monitoring, and dynamic service routing deliver repeatability, reproducibility, portability, and flexibility across diverse environments and libraries.

These same Kubernetes features address many of the most fundamental requirements of data science workloads:

- **Reproducibility across a complex pipeline:** Machine/deep learning pipelines consist of multiple stages, from data processing through feature extraction to training, testing, and deploying models. With Kubernetes, research and operations teams can confidently share a combined infrastructure-agnostic pipeline.
- **Repeatability:** Machine/deep learning is a highly iterative process. With Kubernetes data scientists can repeat experiments with full control over all environmental variables including data sets, ML libraries, and infrastructure resources.
- **Portability across development, staging, and production environments:** When run with Kubernetes, ML-based containerized applications can be seamlessly and dynamically ported across diverse environments.
- **Flexibility:** Kubernetes provides the messaging, deployment, and orchestration fabric that is essential for packaging ML-based applications as highly modular microservices capable of mixing and matching different languages, libraries, databases, and infrastructures.

Considerations for Successful Kubernetes Architecture for AI Workloads

With all of the advantages described above, it is not surprising that Kubernetes has become the de facto container orchestration standard for data science teams. This section provides best practices for optimizing how data science workloads are run on Kubernetes.

KUBERNETES MONITORING

Monitoring Kubernetes clusters is essential for right-scaling Kubernetes applications in production and for maintaining system availability and health. However, legacy tools for monitoring monolithic applications cannot provide actionable observability into distributed, event-driven, and dynamic Kubernetes applications. The new monitoring challenges raised by Kubernetes deployments include:

- With seamless deployment across complex infrastructures, diverse streams of compute, store, and network data must be normalized, analyzed, and visualized to achieve real-time actionable insight into environment topology and performance.
- Highly ephemeral containers make it tricky to capture and track important metrics such as the number of containers currently running, container restart activity, and each container's CPU, storage, memory usage, and network health.
- Effectively harnessing Kubernetes' rich array of internal logs for quick detection and remediation of cluster performance issues, including node and control plane component metrics.

The current gold standard for monitoring Kubernetes ecosystems is Prometheus, an open-source monitoring system with its own declarative query language, PromQL. A Prometheus server deployed in the Kubernetes ecosystem can discover Kubernetes services and pull their metrics into a scalable time-series database. Prometheus' multidimensional data model based on key-value pairs aligns well with how Kubernetes structures infrastructure metadata using labels.

The Prometheus metrics, which are published using the standard HTTP protocol, are human-readable and easily accessed via API calls by, for example, visualization and dashboard-building tools such as Grafana. Prometheus itself provides basic visualization capabilities by displaying the results of PromQL queries run on the aggregated time-series data as tables or graphs. Prometheus can also issue real-time alerts to the relevant teams when predefined performance thresholds are breached.

- Run batch AI workloads as jobs and interactive sessions as replicas
- Use CronJobs for better scheduling

Traditionally, when used for applications and services, K8s containers are run as replicas, not as jobs. But for ML and DL workloads, running as jobs is a better fit. This is because jobs run to completion and can support parallel processing. Jobs can run at the same time multiple pods, enabling set up of a parallel processing workflow while making sure those pods terminate and free their resources when the job runs to completion. Replicas are not set up to enable this functionality, which is critical for batch experimentation and for increasing resource utilization and reducing cloud spending. Replicas are a better fit for interactive sessions where users build and debug their models or experiment with data.

Kubernetes architecture includes CronJob, which is the native way to trigger jobs in a schedule. CronJobs are used when creating periodic and recurring tasks. CronJobs can also schedule specific tasks at determined times, such as scheduling a Job for when your cluster is likely to be idle.

The Challenges of Scheduling AI Workloads on Kubernetes

Now we will address the specific areas where Kubernetes falls short for AI and how you can address those shortfalls.

Kubernetes Scheduling Basics

In Kubernetes, scheduling means making sure that pods are attached to worker nodes. The default Kubernetes scheduler is kube-scheduler, which runs in the cluster's master node and "watches" for newly created pods that have no node assigned. The scheduler first filters the existing cluster nodes according to the container/pod's resource configurations and identifies "feasible" nodes that meet the scheduling requirements. It then scores the feasible nodes and picks the node with the highest score to run the pod. The scheduler notifies the master node's API server about the decision in a binding process.
If no suitable node is found, the pod is unscheduled until the scheduler succeeds in finding a match.

WHAT'S MISSING?

Kubernetes was built for running microservices with scale-out architecture in mind. The default Kubernetes scheduler is therefore not ideal for AI workloads, lacking critical high-performance scheduling components like batch scheduling, preemption, and multiple queues for efficiently orchestrating long running jobs. In addition, K8s is missing gang scheduling for scaling up parallel processing AI workloads to multiple distributed nodes, and topology awareness for optimizing performance.

Scale-out vs. Scale-up Architecture

Kubernetes was built as a Hyperscale System with Scale-out architecture for running services. AI/ML workloads require a different approach. They should run on high-performance systems that can efficiently scale-up workloads.

WHAT IS A HYPERSCALE SYSTEM?

Hyperscale systems were designed and built to run microservices that can serve millions of requests. Such services are always up, waiting for triggers to take action and serve incoming calls, needing to support peak demands that can grow notably with respect to average demand.

Hyperscale systems are typically based on cost-efficient hardware that allows each application to support millions of service requests at a sufficiently low price.

SCHEDULING FOR HYPERSCALE SYSTEMS

Hyperscale systems require a scheduling approach that spreads a large number of service instances on multiple servers to be resilient to server failures, and even to multiple zones and regions to be resilient to data center outages. They are based on auto-scaling mechanisms that quickly scale out infrastructure, spinning machines up and down to dynamically support demand in a cost-efficient way. Kubernetes was built to satisfy such requirements.

WHAT IS A HIGH-PERFORMANCE SYSTEM?

A high-performance system with scale-up architecture is one in which workloads are running across multiple machines, requiring high-speed, low-latency networking and software programs that can run distributed processes for parallel computing.

High-performance systems support workloads for data science, big data analytics, AI, and HPC. In these scenarios the infrastructure should support tens to thousands of long-running workloads concurrently, not millions of short, concurrent service requests as is the case with microservices. AI workloads run to completion, starting and ending by themselves without user intervention (called 'batch jobs', which we will address in more detail later), typically for long durations ranging from hours, days and in some cases even for weeks.

Infrastructure for data science and HPC needs to have the capability to host compute-intensive workloads and process them fast enough. It is therefore based on high end, expensive hardware, including in some cases specialized accelerators like GPUs which typically results in high cost per workload/user.

SCHEDULING FOR HIGH-PERFORMANCE SYSTEMS

For high-performance systems to work efficiently, they need to enable large workloads that require considerable resources to coexist efficiently with small workloads requiring fewer resources. These processes are very different than the spread scheduling and scale-out mechanism required for microservices. They require scheduling methods like bin packing and consolidation to put as many workloads as possible on a single machine to gain efficiency of hardware utilization and reduce machine fragmentation. Reserved instances and backfill scheduling are needed to prevent cases where large workloads requiring multiple resources need to wait in queue for a long time and batch scheduling and preemption mechanisms are needed to orchestrate long running jobs dynamically according to priorities and fairness policies. In addition, elasticity is required to scale up a single workload to use more resources according to availability.

Batch Scheduling Explained

Batch workloads are jobs that run to completion unattended (i.e., without user intervention). Batch processing and scheduling is commonly used in High Performance Computing (HPC) but the concept can easily be applied to data science and AI. With batch processing, training models can start, end, and then shut down, all without any manual intervention. Plus, when the container terminates, the resources are released and can be allocated to other workloads.

The scheduler that is native to **Kubernetes does not use batch scheduling methods like multi-queue scheduling, fairness, advanced preemption mechanisms**, and more, all of which are needed to efficiently manage the lifecycle of batch workloads. With such capabilities jobs can be paused and resumed automatically according to predefined priorities and policies, taking into account the fluctuating demands and the load of the cluster. Batch scheduling also prevents jobs from being starved by heavy users and ensures fairness between multiple users sharing a cluster.

WHAT IS TOPOLOGY AWARENESS?

Another challenge of running AI workloads on Kubernetes relates to a concept called 'topology awareness'. This refers to:

1. inter-node communication and
2. how resources within a node inter-connect

These two topological factors that have major impact on the runtime performance of workloads. In clusters managed by a centralized orchestration system, the responsibility of provisioning resources and optimizing allocations according to these topological factors is at the hands of the cluster manager. **Kubernetes has not yet addressed topology awareness efficiently**, resulting in lower performance when sub-optimal resources are provisioned. Performance inconsistency is another issue -workloads may run at maximum speed, but often poor hardware setup leads to lower performance.
Scheduler awareness to the topology of interconnect links between nodes is important for distributed workloads with parallel workers communicating across machines. In these cases, it is critical that the scheduler binds pods to nodes with fast interconnect communication links. For example, nodes located in the same rack would typically communicate faster and with lower latency than nodes located in different racks. **The default K8s scheduler today does not account for inter-node communication.**

Another important aspect of topology awareness relates to how different resources within a node are communicating. Typically, multiple CPU sockets, memory units, network interface cards (NICs), and multiple peripheral devices like GPUs, are all set up in a node in a topology that is not always symmetric. For example, different memory units can be connected to different CPU sockets and a workload running on a specific CPU socket would gain the fastest read/write data access when using the memory unit closest to the CPU socket. Another example would be a workload running on multiple GPUs in a node with non-uniform topology of inter-GPU connectors. Provisioning the optimal mix of CPUs, memory units, NICs, GPUs, etc., is often called **NUMA (non-uniform memory access) alignment.**

Topology awareness relating to NUMA alignment has been addressed by Kubernetes but the current implementation is limited and highly inefficient – the Kubernetes scheduler allocates a node for a workload without knowing if CPU/memory/GPU/NIC alignment can be applied. If such alignment is not feasible on the chosen node, best-effort configuration would run the workload using a sub-optimal alignment while restricted configuration would fail the workload. Importantly, sub-optimal alignment and a failure to run a workload can occur even in cases where other nodes that can satisfy NUMA alignment are available in the cluster.

The limitations of topology-awareness relate to a basic flaw in Kubernetes architecture. The scheduling mechanism of Kubernetes is based on splitting responsibilities between the scheduler which operates at the cluster level and Kubelet which operates at the node level. The scheduler allocates nodes for containers based on information about the number of resources available in each node, without being aware of the topology of the nodes, the topology of the resources within a node, and which exact resources are actually available at a given moment. Kubelet, together with components of Linux OS and device plugins, is responsible for scheduling the containers and for allocating their resources within the node. This architecture is perfect for orchestrating microservices running within a node, but fails to provide high, consistent performance when orchestrating compute-intensive jobs and distributed workloads.

Gang Scheduling

The third AI-focused component missing from Kubernetes is gang scheduling. Gang scheduling is used when containers need to be launched together, start together, and end together. For example, this capability is required for distributed workloads to ensure that different containers are launched on different nodes only when enough resources are available, preventing inefficiencies and dead-lock situations where one group of containers are launched while others are waiting for resources to become available. Gang scheduling can also help with recovery when some of the containers fail, without requiring a restart of the entire workload.

Automate Job Scheduling with Run:AI

If the key scheduling features discussed above, like batch system capabilities, are necessary for your AI workloads, Run:AI's Scheduler is a simple plug-in to Kubernetes that enables optimized orchestration of high-performance containerized workloads. The Run:AI platform includes:

- **High-performance for scale-up infrastructures** – pool resources and enable large workloads that require considerable resources to coexist efficiently with small workloads requiring fewer resources.
- **Batch scheduling** – workloads can start, pause, restart, end, and then shut down, all without any manual intervention. Plus, when a container terminates, the resources are released and can be allocated to other workloads for greater system efficiency.
- **Topology awareness**— inter-resource and inter-node communication enable consistent high performance of containerized workloads.
- **Gang scheduling** – containers can be launched together, start together, and end together for distributed workloads that need considerable resources.

Run:AI simplifies Kubernetes scheduling for AI and HPC workloads, helping researchers accelerate their productivity and the quality of their work. Learn more about the Run.ai Kubernetes Scheduler.

Book a demo by contacting info@run.ai.