

PROBLEM AWARE, MLOPS

# Kubernetes Scheduling for AI

–  
Challenges of  
Using K8s for AI



## Table of contents

<b>The Challenges of Scheduling AI Workloads on Kubernetes</b> .....	2
<b>Kubernetes Scheduling Basics</b> .....	3
What's Missing? .....	3
<b>Scale-out vs. Scale-up Architecture</b> .....	3-4
What is a Hyperscale System? .....	3
Scheduling for Hyperscale Systems .....	3
What is a High-Performance System? .....	4
Scheduling for High-Performance Systems .....	4
<b>Batch Scheduling Explained</b> .....	4
<b>What is Topology Awareness?</b> .....	5
<b>Gang Scheduling</b> .....	6
<b>Kubernetes Scheduling with Run:AI</b> .....	6
<b>See Our Additional Guides on Key Artificial Intelligence Infrastructure Topics</b> .....	6
<b>GPUs for Deep Learning</b> .....	7
<b>MLOps</b> .....	7



## The Challenges of Scheduling AI Workloads on Kubernetes

This article 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 Scheduling Basics**
- **Scale-out vs. Scale-up Systems**
- **Batch Scheduling**
- **Topology Awareness**
- **Gang Scheduling**

This is part of an extensive series of guides about [Kubernetes Architecture](#)



## 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. For more information on Kubernetes architecture [read the article here](#). 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: inter-node communication and

1. 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.

Compare [Kubernetes vs Slurm schedulers](#) in another guide from this series.

## 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.

Compare [Kubernetes vs Slurm schedulers](#) in another guide from this series.

---

## Kubernetes Scheduling with Run:AI


Run:AI's Scheduler plugs into Kubernetes clusters to enable optimized orchestration of high-performance AI workloads.

- **High-performance system** - for scale-up infrastructures that pool resources and enable large workloads that require considerable resources to coexist efficiently with small workloads requiring fewer resources.
- **Batch Scheduling** - training models can start, pause, restart, 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 for greater system efficiency.
- **Topology awareness**— inter-resource and inter-node communication enable consistent high performance of AI 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 workloads, helping data scientists accelerate their productivity and the quality of their models. [Learn more about the Run:AI platform.](#)

## See Our Additional Guides on Key Artificial Intelligence Infrastructure Topics

We have authored in-depth guides on several other artificial intelligence infrastructure topics that can also be useful as you explore the world of deep learning GPUs.



## GPUs for Deep Learning

Learn how to assess GPUs to determine which is the best GPU for your deep learning model. Discover types of consumer and data center deep learning GPUs. Get started with PyTorch for GPUs – learn how PyTorch supports NVIDIA's CUDA standard, and get quick technical instructions for using PyTorch with CUDA. Finally, learn about the NVIDIA deep learning SDK, what are the top NVIDIA GPUs for deep learning, and what best practices you should adopt when using NVIDIA GPUs.

- [Best GPU for Deep Learning: Critical Considerations for Large-Scale AI](#)
- [PyTorch GPU: Working with CUDA in PyTorch](#)
- [NVIDIA Deep Learning GPU: Choosing the Right GPU for Your Project](#)

---

## MLOps

In today's highly competitive economy, enterprises are looking to Artificial Intelligence in general and Machine and Deep Learning in particular to transform big data into actionable insights that can help them better address their target audiences, improve their decision-making processes, and streamline their supply chains and production processes, to mention just a few of the many use cases out there. In order to stay ahead of the curve and capture the full value of ML, however, companies must strategically embrace MLOps.

➤ **See top articles in our MLOps guide**

- [Machine Learning Ops: What is it and Why We Need It](#)
- [Machine Learning Automation: Speeding Up the Data Science Pipeline](#)
- [Machine Learning Workflow: Streamlining Your ML Pipeline](#)

## About Run:ai

Run:ai is an AI management platform for MLOps, Data Science, and DevOps teams. In addition to helping these teams access and utilize their GPU resources more effectively, it also has a powerful set of features that can abstract infrastructure complexities and simplify the process of training and deploying models. With or without a GPU shortage, Run:ai enables data scientists to focus on innovation without having to worry about resource limitations.

Read more about how Run:ai supports  
data scientists here

[www.run.ai/runai-for-data-science](http://www.run.ai/runai-for-data-science)