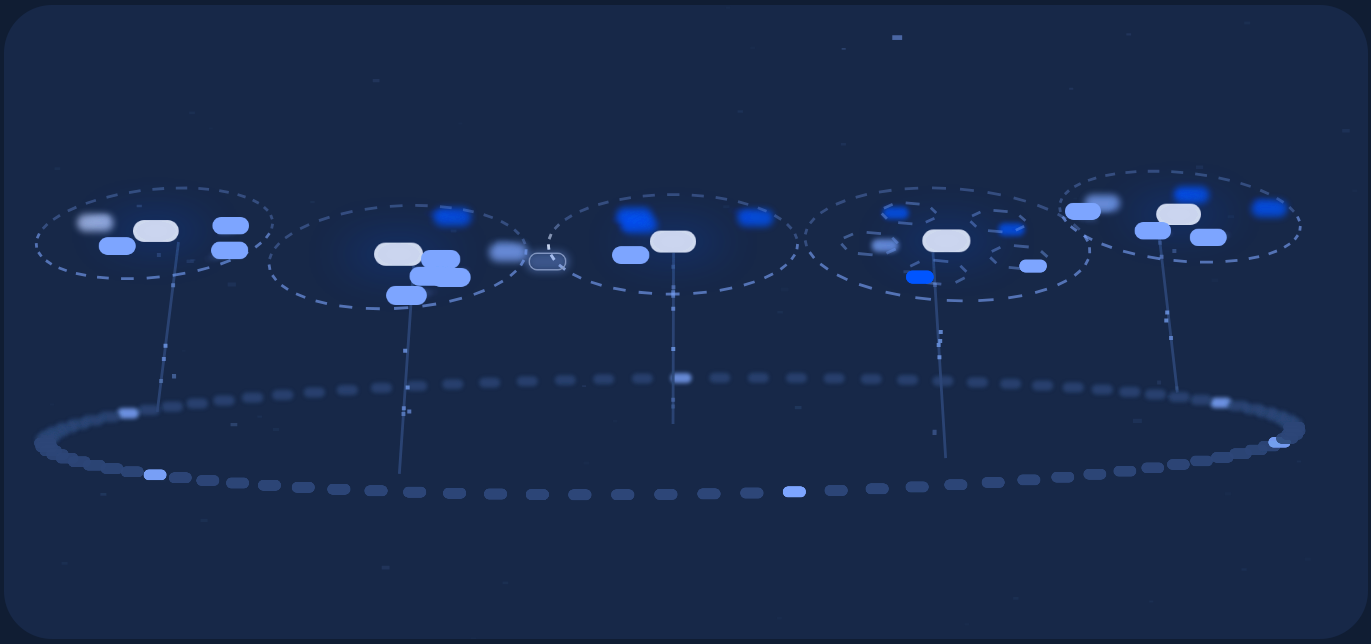
Rillor

EXPLAINER

Slurm or Kubernetes for GPU clusters: how to choose

Slurm vs Kubernetes for GPU clusters: the unit each scheduler places, how a training job starts all at once, where each fits and how one cluster runs both.



Rillor

Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

Summary

Slurm and Kubernetes both place work on GPU nodes from opposite assumptions, and the three facilities whose designs this report reads run both side by side.

- Slurm vs Kubernetes starts with the unit each one places. Slurm allocates nodes to a job for a set time and runs the job's tasks inside that allocation, all of them or none. The default Kubernetes scheduler places one pod, its basic unit of scheduling, at a time [1, 3, 17].
- Gang scheduling means two different things. In Slurm's documentation it is time slicing: several jobs share the same resources and take turns, each suspended while another runs [4]. In Kubernetes it means all-or-nothing placement: a group of pods starts together, or none of them is bound to a node [20].
- Kubernetes gets all-or-nothing starts for training from added projects. Kueue, a Kubernetes queue manager, holds jobs until quota is free and requeues a job whose pods are not all ready within a timeout [24, 26]. JobSet groups the Jobs of one training run and can recreate them so the run resumes from its last checkpoint [28].
- Core Kubernetes gang scheduling, through the Workload and PodGroup APIs, graduated in v1.37, released on 26 August 2026 [21]. It is off by default, and an administrator turns it on with the GenericWorkload feature gate [20, 21].
- Slurm comes with batch policy built in: partitions with time limits, a priority formula that includes fair share, backfill that starts lower-priority jobs when that delays no higher-priority job, and accounting that can record every job and job step [3, 5, 6, 9, 10].
- The two can share one cluster. Slinky, from Slurm's developer SchedMD, runs Slurm clusters as Kubernetes workloads or lets Slurm schedule Kubernetes pods [14, 15].
- A Swiss national supercomputing centre keeps large-scale pre-training and fine-tuning on Slurm and runs inference, agentic systems and retrieval applications on Kubernetes [32].
- Failures grow with job size on either scheduler. On two research clusters running Slurm, 1,024-GPU jobs had a mean time to failure of 7.9 hours, against 47.7 days for 8-GPU jobs [34].

Background

Two schedulers from two traditions

Slurm is an open-source workload manager [3, 16], documented here at version 26.05 [3]. A central manager, slurmctld, runs beside a slurmd daemon on each compute node, with an optional slurmdbd that records accounting for several clusters in one database [3]. Plugins add accounting, advanced reservation, gang scheduling, backfill, topology-optimized resource selection, limits by user or account and multifactor job priority, and Slurm runs unprivileged OCI containers for jobs [3].

NVIDIA announced on 15 December 2025 that it had acquired SchedMD, Slurm's developer, and said it would keep developing and distributing Slurm as open-source, vendor-neutral software [16]. The announcement says Slurm is used in more than half of the top 10 and top 100 systems of the TOP500 list [16]. That share is NVIDIA's own statement.

Kubernetes is built around controllers: control loops that move the cluster's current state toward the desired state written in each object's spec [18]. Its default scheduler, kube-scheduler, finds a node for each new pod, and a cluster can run its own scheduling component instead [17].

Where each is used

The CNCF's announcement of its 2025 Annual Cloud Native Survey, published on 20 January 2026, reports that 82% of container users run Kubernetes in production, and that 66% of organizations that host generative models use it for some or all of their inference [35]. It gives no figure for training and none for Slurm [35]. SchedMD's Kubernetes guide says Kubernetes is being adopted into HPC clusters to orchestrate deployments and run certain workloads such as inference, and describes the area as evolving [12].

Who this is for

Platform engineers and HPC administrators choosing how a GPU cluster schedules training, evaluation and inference, and ML engineers who submit work to either scheduler.

Method

What was examined. Ten questions a GPU cluster's scheduler answers, one per finding below.

Sources. Slurm's documentation for version 26.05 [3, 4, 5, 6, 7, 8, 9, 10, 11], SchedMD talks and tools [12, 14, 15], a NERSC talk [13], NVIDIA's acquisition announcement [16], Kubernetes documentation and blog posts [17, 18, 19, 20, 21, 22, 23], Kueue [24, 25, 26, 27], JobSet [28], Volcano [29], Kubeflow Trainer [30], CNCF publications [1, 2, 35] and four papers [31, 32, 33, 34].

How. Every page was opened on 8 November 2026. Features are described by what they do and by the version in which they graduated, became stable or are on by default. Release dates come from each project's GitHub release and tag pages. Figures keep the scope their source gives them, and documentation examples are labelled as examples. No benchmark was run, and no tool is ranked. Facility papers are cited as research. Cloud providers' own Slurm-on-Kubernetes products and model families are left out.

Findings

1. Built for a fixed machine or for a declared state

Slurm's overview lists three functions: it allocates access to compute nodes to users "for some duration of time so they can perform work", provides a framework for starting and monitoring work on those nodes, and arbitrates contention by managing a queue of waiting work [3].

A 2023 talk by SchedMD's chief technology officer frames the difference [12]. HPC systems assume a fixed system size and an infinite workload, so queue prioritization is critical, and in a colleague's words "Slurm is a policy engine" [12]. Kubernetes was built to manage long-running processes. Cloud-native systems assume "infinite" resources and a finite workload, and all workload is expected to run at once by default [12]. HPC jobs are usually ad-hoc scripts submitted on the command line, and cloud systems are used programmatically [12]. Its background slide says that Slurm, Kubernetes or the two combined "may be appropriate stacks", depending on the site, its users and its systems [12].

The README of SchedMD's operator for Slurm on Kubernetes draws the same line. Kubernetes suits workloads that run for an indefinite time, with vague resource requirements, on a single node and with loose policy, on a pool that can grow to meet demand. Slurm suits finite workloads with well-defined resource requirements and topology, on multiple nodes and with strict policy, on a known pool [15].

SchedMD warns that older presentations may contain outdated information [12]. This report uses the 2023 talk's framing and its models of combined clusters (Finding 9), not its list of Kubernetes limits, which predates the features in Findings 4 and 5.

2. The unit of scheduling: an allocation or a pod

	Slurm	Kubernetes
What a request names	The tasks, CPUs, GPUs and nodes a job needs [1]	One pod's resources, with GPUs in its limits [22]
What is placed	The job's whole allocation, as one atomic unit [1]	One pod at a time [1, 17]
When it cannot fit	The job waits in its partition's queue [3]	The pod stays unscheduled until a node can take it [17]

Table 1. The unit each scheduler places.

Slurm. A CNCF whitepaper of 4 November 2026 describes Slurm as scheduling at the task level [1]. A job declares its tasks, CPUs, GPUs and nodes, and its tasks are scheduled as a single atomic unit: either all are allocated resources, or the job does not start [1]. Partitions act as job queues, each with limits on job size, time and permitted users. Once a job holds its nodes, its user starts parallel work as job steps in any configuration within the allocation [3].

Kubernetes. The same paper calls the pod the fundamental unit of scheduling in Kubernetes. The default scheduler places each pod independently and does not understand that several pods belong to one distributed training job [1]. It filters the nodes, scores the ones that pass and binds the pod to the node it ranks first [1, 17].

The consequence for training. A training job cannot make progress unless all its workers run at once, and a partial allocation wastes resources and may block other work [1]. A second CNCF whitepaper, of 5 November 2026, gives an example: a job needs 8 GPUs, 6 are free, and the default scheduler starts 6 pods and leaves 2 waiting [2]. The 6 GPUs sit idle, and several partial jobs can deadlock [2]. The default scheduler was designed for services that can start with partial capacity and scale up, and the first paper argues that building job-level scheduling into it "would add complexity that most users do not need" [1]. For inference, the effective unit is often a single request, typically an HTTP or gRPC call [1].

3. Two meanings of gang scheduling

In Slurm, time slicing. Slurm's documentation describes timesliced gang scheduling: two or more jobs are allocated to the same resources in the same partition and suspended in turn, so that one job at a time has dedicated access for a configured period [4]. Every job must fit into memory at the same time [4], and slurm.conf sets the length of each slice with SchedulerTimeSlice [11]. Preemption is built on the same logic: a preempted job can be cancelled, requeued on other resources, or suspended and resumed [4].

In Kubernetes, all-or-nothing placement. The Kubernetes documentation defines gang scheduling as scheduling a group of pods on an all-or-nothing basis. If the cluster cannot place the whole group, or a minimum number of its pods set by minCount, none of the pods is bound to a node [20]. Volcano's gang scheduling makes all tasks of a job start at the same time [29], and Kueue calls its version "All-or-nothing with ready Pods" [24].

Reading the term. The CNCF paper uses "gang scheduling" in the Kubernetes sense and says Slurm's task-level model supports it naturally [1]. On a Slurm cluster, a request for gang scheduling asks for time sharing, since an all-or-nothing start is how every allocation begins [1, 3, 4].

4. What Kubernetes adds for training jobs

The first CNCF paper describes a layered stack: workflow orchestrators, then queue managers or batch schedulers, then device plugins or Dynamic Resource Allocation (DRA) for the hardware [1]. It names Armada, KAI Scheduler, Kueue, Slinky and Volcano as projects that add job queues, fair sharing and gang scheduling [1].

Project	What it adds for training	Release on 8 November 2026
Job (core)	Indexed completion, a retry limit, pod failure policies, suspension	Part of Kubernetes [19]
JobSet	A group of Jobs with several pod templates, pod-to-pod networking, failure policies, exclusive placement per topology domain	v0.13.0, 12 October 2026 [28]
Kueue	Admission, quotas, cohorts, fair sharing, preemption, all-or-nothing with ready pods, topology-aware scheduling	v0.20.5, 5 November 2026 [24]
Volcano	Its own scheduler with gang, queue quota and topology policies	v1.16.0, 14 October 2026 [29]
Kubeflow Trainer	The TrainJob API on top of JobSet and LeaderWorkerSet, MPI on Kubernetes	v2.4.0, 6 November 2026 [30]

Table 2. Public projects for training jobs on Kubernetes.

Job. Indexed Jobs, stable since v1.24, give each pod a completion index, and with a Service the pods reach each other through fixed hostnames in DNS [19]. A Job's backoff limit is 6 by default [19].

JobSet. JobSet models a distributed training or HPC workload as a group of Jobs. It adds separate pod templates for a driver and its workers, pod-to-pod communication through headless Services and startup order, and can give each child job exclusive use of one topology domain, typically a rack [28]. Submitted through Kueue, JobSets wait until capacity is free, which prevents partial scheduling and deadlocks [28].

Kueue. Kueue manages quotas and how jobs consume them. It decides when a job waits, when it is admitted, meaning its pods can be created, and when it is preempted [24]. It replaces no component: pod-to-node scheduling stays with kube-scheduler [24]. Its integrations include the batch Job, JobSet, LeaderWorkerSet, MPIJob and RayJob [24].

Volcano. Volcano's scheduler is the component responsible for pod scheduling. Each cycle runs enqueue, allocate, preempt, reclaim and backfill actions and binds each job to a node [29]. That is the difference behind the "Kueue vs Volcano" question: Kueue admits jobs and leaves placement to kube-scheduler [24], and Volcano places pods itself [29].

Kubeflow Trainer. Kubeflow describes Trainer as a Kubernetes-native project for LLM fine-tuning, hyperparameter optimization and training. It brings MPI to Kubernetes for multi-node, multi-GPU jobs and integrates with Kueue and with Slurm's bridge for hybrid clusters [30].

5. Gang scheduling in core Kubernetes graduated in v1.37

Native gang scheduling through the Workload API was introduced in Kubernetes v1.35 [2]. In v1.37, released on 26 August 2026, the core Workload and PodGroup APIs graduated, together with workload-aware preemption [21]. Among the changes the 8 September 2026 post describes [21]:

- **One queue entry per group.** Member pods of a PodGroup were queued one by one before. Now only the PodGroup is queued.
- **A mutable minimum.** Controllers can change minCount to resize a gang on the fly.
- **Groups as preemption victims.** Default preemption respects the PodGroup's disruptionMode field.

If the scheduler cannot place the minimum count, none of the pods is scheduled, and they wait in the unschedulable queue while other workloads are scheduled [20]. The guarantee covers the start: the running count can drop below the minimum if pods are later deleted or evicted [20].

Off by default. A cluster administrator enables the GenericWorkload feature gate on kube-apiserver, kube-controller-manager and kube-scheduler [20, 21]. The Job controller's integration with Workload APIs, introduced in v1.36, and hierarchical gangs, introduced in v1.37, are off by default behind their own feature gates [19, 20, 21].

Next steps. In the 8 September 2026 post, the working group listed moving these APIs to stable among its aims for v1.38, and envisioned Kueue using workload-aware scheduling as its engine for gang scheduling and topology-aware placement in the long term [21].

6. Queues, priority, fair share and backfill

Slurm's priority formula. The multifactor priority plugin is Slurm's default. Nine factors feed it, among them age, fair share, job size, partition and quality of service (QOS) [9]. Fair share is the difference between the share of the computing resource an account was promised and what it has consumed [9]. Every weight is 0 by default, so only the factors weighted in slurm.conf count [9]. In the documentation's worked example, a fair-share weight of 5,000 and weights of 1,000 for job size and age give a maximum priority of 7,000, about 70% of it from fair share [9]. QOS levels affect a job's priority, preemption and resource limits [9].

Backfill and time limits. The backfill plugin is loaded by default. Without it, each partition is scheduled strictly in priority order, which the documentation says typically gives significantly lower utilization [6]. Backfill starts lower-priority jobs if that does not delay the expected start of any higher-priority job, and it needs reasonably accurate time limits to work well [6]. A job's time limit defaults to its partition's default, a job that asks for more than the partition allows waits, possibly indefinitely, and a job that reaches its limit gets SIGTERM and then SIGKILL [5].

Kubernetes priority. Kubernetes priority works per pod: the scheduler preempts lower-priority pods to place a waiting one, and the victims get their graceful termination period [17]. The second CNCF paper argues that these workloads need priority per job, not only per pod, with production inference highest, then

production training, development and batch work [2].

Kueue quotas and fair sharing. A ClusterQueue governs a pool of resources, such as CPU, memory and accelerators, with quotas per resource flavor and fair sharing rules [25]. Each quota has a nominal amount, a borrowing limit and a lending limit, and ClusterQueues in one cohort borrow unused quota from each other [25]. Admission fair sharing orders workloads by the historical resource use of their queues, favouring those that have consumed less [25].

What fair share means for training. The second CNCF paper gives illustrations. With 60% quota for one team and 40% for another, resources divide that way when both have work waiting [2]. One job may use 80% of the cluster for a week as long as other teams get their share over the month [2]. A job that needs 256 GPUs can starve indefinitely while smaller jobs keep taking free GPUs, and backfill needs jobs to declare, or the system to estimate, their runtime [2]. Volcano's queues support borrowing, reclaiming and preemption, and its scheduler runs a backfill action [29].

7. GPUs, topology and the network

Slurm. GPUs are generic resources (GRES). None are enabled by default, so slurm.conf names the GRES it manages, and a job gets GPUs only when it asks with options such as --gpus or --gpus-per-node [8]. A node found to have fewer resources than configured is drained [8]. Without a topology plugin, Slurm treats nodes as a one-dimensional array [7]. The tree plugin allocates under the lowest-level switch that can satisfy a job, users can cap the number of leaf switches and the wait for them, and the block plugin places jobs to minimize fragmentation [5, 7].

Kubernetes. GPU scheduling through device plugins has been stable since v1.26 [22]. An administrator installs the vendor's drivers and device plugin, the cluster exposes a schedulable GPU resource, and GPUs go in a pod's limits [22]. DRA, which lets pods request and share devices such as accelerators, has been stable since v1.35 [22].

Kueue's topology-aware scheduling has been on by default since Kueue v0.14 [27]. Its documentation explains why: such workloads commonly need heavy pod-to-pod communication, so the bandwidth between pods translates into execution time and cost [27]. Node labels describe blocks and racks, set by a cloud provider or by the administrators of on-premise clusters [27]. JobSet's exclusive placement and Volcano's network topology-aware scheduling cover the same need [28, 29].

What a pod costs on the fabric. A UK national supercomputing service measured point-to-point communication on its own system. Small-message latency was about 3.11 microseconds from Kubernetes pods against about 2.22 microseconds on bare metal and in containers launched through Slurm, a fixed overhead of about 0.89 microseconds [31]. Peak bandwidth converged to the fabric's theoretical maximum of 200 Gb/s as messages grew, in every case, and the authors judge the fixed overhead unlikely to significantly affect throughput-bound workloads [31].

8. Failure handling and restarts

Slurm. A job submitted with --requeue may be requeued after a node failure or on preemption, and its batch script starts again from the beginning with the same job ID [5]. The --signal option sends a chosen signal a set time before the job's end time, and by default no signal is sent [5]. HealthCheckProgram runs a script on every responding compute node at a set interval, and the script can drain a node when it finds a problem [11].

Kubernetes. The pod failure policy, stable since v1.31, can end a Job as soon as a pod fails with an exit code that signals a software bug, or ignore failures caused by preemption or eviction so that they do not count against the backoff limit [19]. JobSet's failure policy recreates the whole JobSet when any child job fails, so the workload can resume from its last checkpoint [28]. Kueue's all-or-nothing setting evicts and requeues a workload whose pods are not all ready within a timeout, 30 minutes by default, because two such jobs can otherwise deadlock when physical resources do not match the quotas [26].

Why job structure matters. The second CNCF paper notes that preempting 1 of a job's 64 workers can lose all 64 GPUs' worth of work, so a scheduler needs to understand job structure [2].

How often large jobs fail. A study of two multi-tenant research clusters analysed 11 months of data, 4 million jobs and over 150 million GPU hours, on Slurm with bare-metal allocations [34]. Health checks ran every five minutes, a node that failed one was unavailable until fixed, and a job ended by a failed check was requeued automatically with the same job ID [34]. The authors write that health checks prevent "restart loops" on defective nodes [34].

Job size	Mean time to failure
8 GPUs	47.7 days
1,024 GPUs	7.9 hours
16,384 GPUs (projected)	1.8 hours

Table 3. Mean time to failure by job size on two research clusters [34].

One 1,024-GPU job hit node failures and was requeued 35 times, causing a total of 548 preemptions involving over 7,000 GPUs [34].

9. Four ways to run Slurm and Kubernetes together

SchedMD's 2023 talk names four models for a combined cluster, seen from Slurm's side [12].

Model	How it works	In the sources
Over	Slurm manages all resources, and Kubernetes clusters are created inside Slurm jobs	The talk judges it not especially useful beyond test or development environments [12]
Distant	Both run in the cluster, neither sees the other's demand, and a management tool shifts nodes	A UK service set nodes apart with a Slurm reservation that blocks user jobs [31]
Adjacent	Both control planes overlap, and Slurm schedules both through a scheduler plugin	slurm-bridge [14]
Under	Slurm clusters run inside Kubernetes, with long-lived login pods and an operator	slurm-operator [15], and in part NERSC's control daemons [13]

Table 4. The four models of SchedMD's 2023 talk, with an example of each [12, 13, 14, 15, 31].

slurm-bridge. Workloads arrive from Kubernetes as a Pod, PodGroup, Job, JobSet or LeaderWorkerSet, or from Slurm through salloc or sbatch [14]. Each Kubernetes workload becomes a representative Slurm job that the Slurm controller schedules, and a hybrid node must not run native Slurm work and bridged Kubernetes work at the same moment [14]. Its latest stable tag, v1.2.4, dates from 28 October 2026 [14].

slurm-operator. The operator manages Slurm cluster components through custom resources built for their creation, upgrade and graceful shutdown [15]. A NodeSet is a set of identical Slurm compute nodes run as pods, drained before they end for a scale-in or upgrade [15]. Release 1.2 supports Kubernetes 1.35 to 1.37, and its latest stable tag is v1.2.4 [15].

NERSC's control services. NERSC described in September 2023 how it runs slurmctld and slurmdbd as single-replica Kubernetes deployments [13]. When a node stops responding, Kubernetes moves its pods to other workers, and the team reports that this has happened and worked [13].

Flux inside Kubernetes. A 2023 paper's Flux Operator deploys another HPC workload manager inside Kubernetes on demand [33]. In its test of one simulation code at 8 to 64 nodes, mean times were about 5% faster than with the MPI Operator [33].

What combining costs. The UK paper finds that Slinky makes resource accounting across two scheduling domains harder to keep consistent, and describes the slurm-operator as struggling with latency overhead and with the mismatch between Kubernetes' declarative, eventually consistent model and Slurm's synchronous job lifecycle [31]. SchedMD's talk lists the drawbacks of the Under model: Kubernetes workloads run outside Slurm's view, and prioritizing between the two is difficult [12].

10. What facilities report

A Swiss national supercomputing centre. An April 2026 paper from the centre writes that large-scale pre-training has fit naturally on HPC clusters managed by queue-based workload managers, and that inference needs long-running, highly available services that conflict with the batch model [32]. Its design is a dual control plane. Slurm runs large-scale pre-training and tightly coupled simulations, and inference pipelines, agentic systems and retrieval-augmented generation (RAG) applications run on Kubernetes [32]. Hosting fine-tuning on both schedulers would duplicate engineering work, such as debugging GPU communication and managing concurrent access to datasets, so the Swiss centre concentrated fine-tuning on Slurm [32]. Kubernetes is the control plane for dataset work, evaluations and experiment logging, and Slurm the execution plane, joined by a REST API that submits Slurm jobs [32]. The paper sees expert HPC users who prefer low-level control through Slurm and newer users who want "one-click" abstractions [32].

A UK national supercomputing service. A paper of August 2026 states that Kubernetes "is not considered a replacement for batch scheduling systems" and presents it as a complementary service [31]. It finds that batch schedulers often lack the flexibility, service abstractions and multi-tenant isolation its newer use cases need, and expects Slurm "to schedule the largest tenants and partitions" [31].

NERSC. NERSC's 2023 talk shows a third pattern: Slurm schedules the compute nodes while Kubernetes keeps Slurm's own control services running [13].

Implications

Question	Points to Slurm	Points to Kubernetes
Team skills	Users who want low-level control through job scripts on the command line [12, 32]	Users who want "one-click" abstractions, and teams that manage services as declared state [18, 32]
Job shapes	Finite multi-node jobs with known resource needs and topology [15]	Long-running services and request-level inference [1, 15, 32], with batch jobs through Kueue or Volcano [24, 29]
Tenancy	Partitions that limit their users, accounts and QOS limits [3, 9]	Namespaces with role-based access and network policies scoped to them [23], and Kueue quotas [25]
Failure handling	Requeue with the same job ID, health checks that drain nodes [5, 11]	Pod failure policies, JobSet restarts from checkpoint, Kueue requeue after a timeout [19, 26, 28]
Accounting	Records for every job and job step once accounting is configured, reported by sacct and sreport [10]	Kueue's built-in Prometheus metrics [24]

Table 5. A decision table drawn from the findings.

For a cluster that mostly runs large training jobs. Slurm's atomic allocation, priority formula, backfill and accounting come with it [1, 3, 6, 9, 10], and backfill works well only with reasonably accurate time limits [6]. On Kubernetes the same needs come from a queue manager or batch scheduler [1, 2], or from core gang scheduling, which graduated in v1.37 and is off by default [20, 21].

For a cluster that mostly serves models and agents. Inference schedules requests and runs as long-lived services [1, 32]. Kubernetes was built for that kind of work [12, 15], and 66% of organizations that host generative models used it for some or all of their inference, in the CNCF's January 2026 announcement [35].

For a cluster that runs both. Choose one of the four models on purpose [12], plan accounting across both schedulers [31], and give each workload type one scheduler, as the Swiss centre did with fine-tuning [32].

For everyone reading the terms. Ask which meaning of gang scheduling a document uses [4, 20], and check each Kubernetes feature's version and feature gate [20, 21].

Platform work. Rillor designs and builds Kubernetes platforms for superintelligence (SI) work on GPUs, on servers the client controls, where workloads ask for GPUs and the scheduler places them on the nodes that have them.

Limits of this analysis

- This is an explainer of public documentation, talks and papers. No benchmark was run for it, and it ranks no tool.
- No source opened gives the share of training or GPU clusters that use either scheduler. The TOP500 statement is NVIDIA's own [16], and the CNCF survey covers inference and general use only [35].
- The only measured comparisons are scoped to their setups: point-to-point microbenchmarks on one UK system [31] and a 2023 test of the Flux Operator against the MPI Operator [33].
- SchedMD's talk dates from 2023 and predates core gang scheduling in Kubernetes [12].
- The CNCF fair-share, 8-GPU and 256-GPU cases are illustrations [2], and the 30-minute Kueue timeout is a default that each site can change [26].
- The reliability figures come from one study of two clusters [34].
- Versions are those of 8 November 2026: Slurm's documentation for 26.05 [3], slurm-operator and slurm-bridge v1.2.4, their latest stable tags [14, 15], and Kubernetes v1.37 [21].
- The CNCF announcement gives no sample size or fieldwork dates for its survey [35].

Sources

1. Scheduling Fundamentals for AI Workloads. Sabrina Di Leva, Kevin Hannon, Rajas Kakodkar and others, Cloud Native Computing Foundation, 4 November 2026, viewed 8 November 2026. https://www.cncf.io/reports/scheduling-fundamentals-for-ai-workloads/ (PDF: https://www.cncf.io/wp-content/uploads/2026/11/02_2Scheduling-Fundamentals-for-AI-Workloads_CNCF_Whitepaper.pdf)

2. Job Orchestration Challenges for AI Workloads. Sabrina Di Leva, Kevin Hannon, Rajas Kakodkar and others, Cloud Native Computing Foundation, 5 November 2026, viewed 8 November 2026. https://www.cncf.io/reports/job-orchestration-challenges-for-ai-workloads/ (PDF: https://www.cncf.io/wp-content/uploads/2026/11/02_3Job-Orchestration-Challenges-for-AI-Workloads_CNCF_Whitepaper.pdf)

3. Overview, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/overview.html (Containers Guide: https://slurm.schedmd.com/containers.html)

4. Gang Scheduling, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/gang_scheduling.html (Preemption: https://slurm.schedmd.com/preempt.html)

5. sbatch, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/sbatch.html

6. Scheduling Configuration Guide, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/sched_config.html

7. Topology Guide, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/topology.html

8. Generic Resource (GRES) Scheduling, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/gres.html

9. Multifactor Priority Plugin, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/priority_multifactor.html (Fair Tree: https://slurm.schedmd.com/fair_tree.html, QOS: https://slurm.schedmd.com/qos.html)

10. Accounting and Resource Limits, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/accounting.html

11. slurm.conf, Slurm Workload Manager documentation 26.05. SchedMD, viewed 8 November 2026. https://slurm.schedmd.com/slurm.conf.html

12. Slurm and/or/vs Kubernetes. Tim Wickberg, SchedMD, SC23, November 2023, viewed 8 November 2026. https://slurm.schedmd.com/SC23/Slurm-and-or-vs-Kubernetes.pdf (Kubernetes Guide: https://slurm.schedmd.com/kubernetes.html)

13. Never use Slurm HA again: Solve all your problems with Kubernetes. Chris Samuel and Doug Jacobsen, NERSC, Slurm User Group 2023, 12 September 2023, viewed 8 November 2026. https://slurm.schedmd.com/SLUG23/NERSC-SLUG23.pdf

14. Slinky documentation. SchedMD, viewed 8 November 2026. https://slinky.schedmd.com/docs/ (slurm-bridge, SlinkyProject on GitHub, tag v1.2.4, 28 October 2026: https://github.com/SlinkyProject/slurm-bridge)

15. Kubernetes Operator for Slurm Clusters (slurm-operator). SchedMD, SlinkyProject on GitHub, tag v1.2.4, 28 October 2026, viewed 8 November 2026. https://github.com/SlinkyProject/slurm-operator

16. NVIDIA Acquires Open-Source Workload Management Provider SchedMD. NVIDIA Blog, 15 December 2025, viewed 8 November 2026. https://blogs.nvidia.com/blog/nvidia-acquires-schedmd/

17. Kubernetes Scheduler. Kubernetes Documentation, last modified 16 February 2024, viewed 8 November 2026. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ (Pod Priority and Preemption, last modified 24 June 2026: https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/)

18. Controllers. Kubernetes Documentation, last modified 1 September 2024, viewed 8 November 2026. https://kubernetes.io/docs/concepts/architecture/controller/

19. Jobs. Kubernetes Documentation, last modified 12 October 2026, viewed 8 November 2026. https://kubernetes.io/docs/concepts/workloads/controllers/job/

20. Gang Scheduling. Kubernetes Documentation, last modified 24 July 2026, viewed 8 November 2026. https://kubernetes.io/docs/concepts/scheduling-eviction/gang-scheduling/

21. Kubernetes v1.37: Advancing Workload-Aware Scheduling. Antoni Zawodny, Bartosz Rejman, Heba Elayoty and others, Kubernetes Blog, 8 September 2026, viewed 8 November 2026. https://kubernetes.io/blog/2026/09/08/kubernetes-v1-37-advancing-workload-aware-scheduling/ (Kubernetes v1.37: Garhwal, Kubernetes v1.37 Release Team, 26 August 2026: https://kubernetes.io/blog/2026/08/26/kubernetes-v1-37-release/)

22. Schedule GPUs. Kubernetes Documentation, last modified 20 September 2024, viewed 8 November 2026. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/ (Dynamic Resource Allocation, last modified 27 July 2026: https://kubernetes.io/docs/concepts/resource-management/dynamic-resource-allocation/)

23. Multi-tenancy. Kubernetes Documentation, last modified 22 May 2025, viewed 8 November 2026. https://kubernetes.io/docs/concepts/security/multi-tenancy/

24. Overview, Kueue documentation. Kubernetes SIGs, last modified 9 October 2026, viewed 8 November 2026. https://kueue.sigs.k8s.io/docs/overview/ (Release v0.20.5, 5 November 2026: https://github.com/kubernetes-sigs/kueue/releases/tag/v0.20.5)

25. Cluster Queue, Kueue documentation. Kubernetes SIGs, last modified 19 August 2026, viewed 8 November 2026. https://kueue.sigs.k8s.io/docs/concepts/cluster_queue/ (Fair Sharing: https://kueue.sigs.k8s.io/docs/concepts/fair_sharing/, Preemption: https://kueue.sigs.k8s.io/docs/concepts/preemption/)

26. Setup All-or-nothing with ready Pods, Kueue documentation. Kubernetes SIGs, last modified 30 September 2026, viewed 8 November 2026. https://kueue.sigs.k8s.io/docs/tasks/manage/setup_wait_for_pods_ready/

27. Topology Aware Scheduling, Kueue documentation. Kubernetes SIGs, last modified 30 October 2026, viewed 8

November 2026. https://kueue.sigs.k8s.io/docs/concepts/topology_aware_scheduling/

28. Introducing JobSet. Daniel Vega-Myhre, Abdullah Gharaibeh and Kevin Hannon, Kubernetes Blog, 23 March 2025, viewed 8 November 2026. https://kubernetes.io/blog/2025/03/23/introducing-jobset/ (Release v0.13.0, 12 October 2026: https://github.com/kubernetes-sigs/jobset/releases/tag/v0.13.0)

29. Introduction, Volcano documentation. Volcano project, viewed 8 November 2026. https://volcano.sh/docs/home/introduction/ (Scheduler overview: https://volcano.sh/docs/scheduler/overview/, Release v1.16.0, 14 October 2026: https://github.com/volcano-sh/volcano/releases/tag/v1.16.0)

30. Overview, Kubeflow Trainer documentation. Kubeflow, last modified 5 November 2026, viewed 8 November 2026. https://www.kubeflow.org/docs/components/trainer/overview/ (Release v2.4.0, 6 November 2026: https://github.com/kubeflow/trainer/releases/tag/v2.4.0)

31. Multi-tenant Kubernetes Use Cases for AI, Secure Computing and Data Services, and More. Jake Watson, Sadaf R. Alam, Christopher Woods and others, arXiv:2608.00742, 1 August 2026, viewed 8 November 2026. https://arxiv.org/abs/2608.00742

32. Beyond Pre-Training: The Full Lifecycle of Foundation Models on HPC Systems. Dino Conciatore, Elia Oggian, Federico Da Forno and others, arXiv:2604.12599, 14 April 2026, viewed 8 November 2026. https://arxiv.org/abs/2604.12599

33. The Flux Operator. Vanessa Sochat, Aldo Culquicondor, Antonio Ojea and Daniel Milroy, arXiv:2309.17420, 29 September 2023, viewed 8 November 2026. https://arxiv.org/abs/2309.17420

34. Revisiting Reliability in Large-Scale Machine Learning Research Clusters. Apostolos Kokolis, Michael Kuchnik, John Hoffman and others, arXiv:2410.21680, version 2, 6 February 2025 (first version 29 October 2024), viewed 8 November 2026. https://arxiv.org/abs/2410.21680

35. Kubernetes Established as the De Facto 'Operating System' for AI as Production Use Hits 82% in 2025 CNCF Annual Cloud Native Survey. Cloud Native Computing Foundation, 20 January 2026, viewed 8 November 2026. https://www.cncf.io/announcements/2026/01/20/kubernetes-established-as-the-de-facto-operating-system-for-ai-as-production-use-hits-82-in-2025-cncf-annual-cloud-native-survey/

About this report

Published by Rillor on 10 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. Slurm or Kubernetes for GPU clusters: how to choose. 10 November 2026. https://rillor.com/insights/slurm-or-kubernetes-for-gpu-clusters

Notices

Rillor is not a registered investment adviser or commodity trading advisor and does not provide investment or trading advice. Research, forecasts, data and software described on this site are for research and engineering use. Nothing here is an offer or recommendation to buy or sell any security, commodity interest or digital asset. Past or simulated results do not indicate future results.

NVIDIA, AMD, Ceph, Kubernetes and related product names are trademarks of their owners and are used only to identify products. No affiliation or endorsement is implied.

Work with Rillor

Send the workloads, the GPUs and servers they run on, the teams that deploy to the cluster and who runs it afterward. Contact us at rillor.com/contact?type=kubernetes or contact@rillor.com. More on Kubernetes platforms: rillor.com/compute/kubernetes