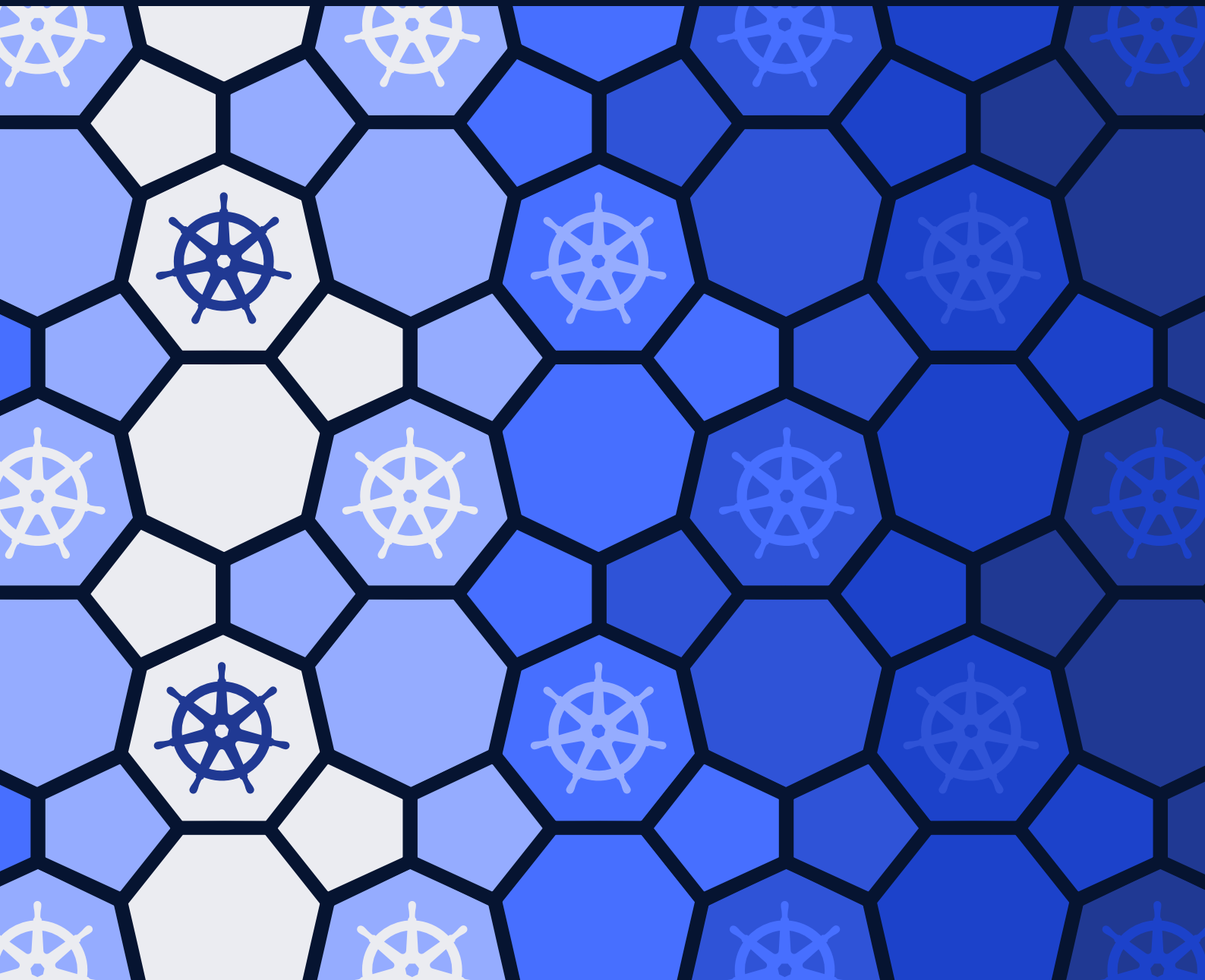
komodor

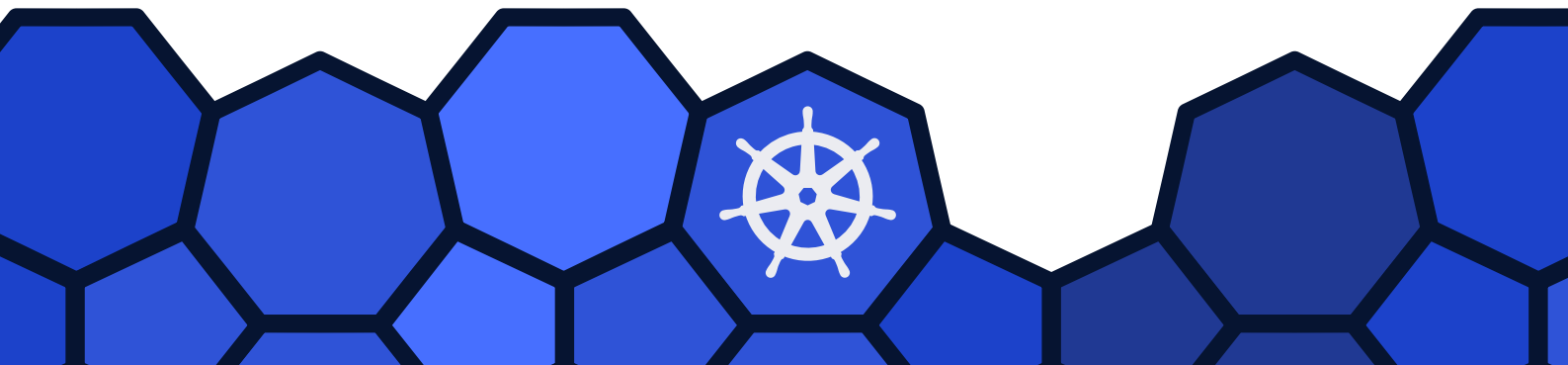
Introduction to ML on Kubernetes: Data Preparation, Model Training, and Model Serving



komodor

komodor.com

Table of contents



Introduction to ML on Kubernetes: Data Preparation, Model Training, and Model Serving

Kubernetes has been facilitating container orchestration for around a decade for both stateful and stateless application workloads. With the recent rise of AI and the advent of tools like Kubeflow and Argo Workflows, Kubernetes is also becoming a first-class citizen when it comes to running AI workloads.

When you are training a model on K8s, you may be tweaking many parameters and have to test each of them one by one. With the help of large-scale orchestrated platforms like Kubeflow, MLflow, Airflow, etc., you can run multiple experiments and model training exercises in parallel. This can also help you prepare and fix data via multiple processes at the same time.

This ebook will present the steps involved in running AI workloads on Kubernetes. We will explore the different steps, from data preparation to serving AI models, and see how several tools can help as well as discuss their drawbacks.

Preparing the Data

Data preparation involves multiple steps: gathering the data, cleaning the data, fixing missing values, transforming it, labeling it, and splitting it. There can be even more steps based on your use case.

Most of the above actions are done via batch jobs with certain predefined pipelines or data flows. This is where directed acyclic graph (DAG) tools are useful.
The following can all perform batch processing fairly easily, with each tool optimized for specific use cases.

Apache Airflow

Apache Airflow, first released in 2015 by Airbnb, specializes in managing, scheduling, and monitoring workflows, such as data pipelines with different steps like those discussed above. Airflow can handle large data pipelines and has a proper GUI that you can programmatically control.

Pros:

- Ability to define complex workflows with Python, enabling the creation of data pipelines that can meet all needs for data preparation
- Support for multiple operators like Bash, Python, SQL, and Docker, allowing for the execution of different types of workflows with ease
- Ability to orchestrate workloads across multiple worker nodes and execute them in parallel, making it scalable for large workloads
- DAG, retry mechanism, and error handling for high reliability
- Flexible scheduling, cool user interface, and open-source community for support

Cons:

- For data scientists, code-driven pipelines can pose a steep learning curve.
- While Airflow is great with batch processing, it lacks real-time data processing
- Devs will have to deal with operational overhead in managing different components.

Best use: Apache Airflow is best if you don't want to be bound to Kubernetes constructs nor manage ETL workflows and data pipelines. It can also work as a general-purpose scheduler as well as run on traditional setups.

Argo Workflows

Argo Workflows is a native Kubernetes open-source workflow engine for complex parallel workflows. It is widely used in data science, CI/CD use cases, and other tasks like running cron jobs on K8s.

Pros:

- Native to Kubernetes thus can utilize K8s resources more efficiently; can integrate very easily with the Kubernetes native ecosystem and utilize them
- Runs everything as a container, rendering the system more scalable horizontally
- Support for DAG-based definitions for workloads, as well as modular and reusable workflows; ability to trigger other workloads from one, making it reusable
- Great to use the same module with multiple different data sets based on attributes due to the templating feature

Cons:

- Argo Workflows is bound to Kubernetes and cannot work without it.
- It is defined in YAML definitions, which can be complex to manage. Also, it has a limited set of features in GUI to manage the workflow, which adds even more complexity
- Argo Workflows' community is less focused on AI and more on orchestrating workloads.

Best use: Argo Workflows is ideal when you're working in a Kubernetes-based environment and need to orchestrate batch processing jobs or ML workflows.

Kueue

Kueue is a job scheduling system designed by the special interest group of the Kubernetes community. It can manage large-scale, heavy, and complex batch jobs on Kubernetes

Pros:

- Uses basic Kubernetes constructs like namespaces, resource quotas, and CRDs
- Very resource-optimized, decides which job to execute based on cluster resource availability
- Priority-based scheduling and support for parallel execution, great features for high data volumes
- Cost efficiency due to task scheduling based on resources and open-source community support

Cons:

- Kueue is dependent on Kubernetes.
- It has a steep learning curve and a still-evolving ecosystem, making it less mature than Airflow or Argo Workflows.
- Maintenance and upgrades can be problematic, as you have to update the CRDs and follow the lifecycle of both the operators and CRDs.

Best use: Kueue is a great choice if you're running batch workloads and need options like prioritization, fair resource allocation, and multi-tenancy.

Training the Model

Training an AI model is the most important step. In this phase, the model is fed massive amounts of data. Models ingest this data, analyze and form patterns, and then perform algorithmic adjustments for optimization.

The models presented below are not mutually exclusive and do have overlapping features. You may have to run multiple models at once with different parameters to get the best possible model with the highest accuracy. Kubernetes can contribute by running these models in parallel as well as by utilizing and automating the GPU, which lies at the heart of training them.

Kubeflow

Kubeflow as an ML platform is capable of executing your entire ML learning workflow pipeline. Its various components deploy, manage, and scale AI models. Whether or not you should use Kubeflow can depend on a lot of things like the learning curve, ease of deployment, features, etc.

Pros:

- Versioned experimentation is versioned, with the ability to rerun an existing experiment, as well as multiple experiments simultaneously, which can be a major time saver
- Supports the complete ML lifecycle, i.e., data preparation, model training, and serving
- Native integration with Kubernetes, so it can scale better and has distributed training support for GPU
- Modular design, so you can pick and choose which modules to use in your model training.

Cons:

- Kubeflow installs Istio by default, which can be a major problem for maintenance; if Istio is injected in any other namespace by mistake, it can hamper traffic and have a major impact.
- The tool cannot run without Kubernetes, which is an issue if you want to move away from K8s.
- Kubeflow is still an evolving tool and support for security, auditing, and other functions is limited.

Best use: Kubeflow excels in end-to-end AI pipelines on Kubernetes with multi-step workflow orchestration. It has limited functionality for experiment tracking but is ideal for building large-scale cloud-native AI solutions.

MLflow

MLflow was open-sourced in 2018 and oversees machine learning lifecycles. You can implement it to experiment with different parameters, repeat experiments, and serve models. One of the early projects in the AI space, MLflow has evolved over time and been adopted by millions of users.

Pros:

- Versioned experiments you can rerun from the existing version by just changing some parameters
- Easy to use with frameworks like TensorFlow, scikit-learn, etc.; allows you to define a standard SOP to use all these libraries
- Features APIs and a mature UI ecosystem, making it easier to integrate with your ecosystem of tools and a great choice for implementing in your CI/CD
- Can be used to package the model for one deployment and makes it feasible to serve the models via APIs
- Works more efficiently due to Kubernetes default isolation constructs like namespaces and resource quotas

Cons:

- MLflow keeps all the versions, along with the artifacts and configurations related to them, demanding greater storage usage in Kubernetes.
- Its model-serving capability is not up to par with industry production standards and may not be valid for high-performance use cases.
- MLflow has limited capability for handling complex pipelines, as it is not built from a workflow-orchestration perspective to handle complex data processing jobs

Best use: MLflow is great for lightweight experiment tracking and model management. Also, it isn't bound to Kubernetes.

KAITO

Kubernetes AI toolchain operators, built by Azure, are native Kubernetes operators you can use to run and train models, along with other AI-related development.

Pros:

- Can use K8s' scaling capabilities very easily, is resource-efficient, and has native support for GPUs
- Enables inherently more secure workloads than if running multiple workloads in one machine due to network policies and running workloads in isolation via namespaces
- Strong support for CI/CD integration, making it very easy to run training model workloads as pipelines; also very easily integrated with other workflows like TensorFlow, PyTorch, etc.
- Full support for monitoring and observing workloads

Cons:

- The overhead of managing an extra Kubernetes operator can be a problem.
- KAITO can be costly compared to other platforms.
- It features very limited support on the user interface side; you cannot control any changes via the UI, as most of the UI is read-only.

Best use: KAITO is more focused on the Azure ecosystem and helps accelerate the usage of GPU and autoscaling. It can integrate with multiple Azure services very easily and assists with setting up an entire ecosystem on Azure.

Serving the Model

After you train the model, you need a way to expose it to your clients so they can use it. This is referred to as serving a model. There can be many ways to do this. Some organizations run their models in containers and build wrappers on top of them to serve them. Others may use more sophisticated tools designed to serve the models via an API.

Should You Use Kubernetes to Serve Models?

Kubernetes greatly simplifies this step. It lets you containerize your model as an app and then deploy it like a normal microservice, although you will have to write supporting code to expose it via an API. You can also implement tools like Hugging Face and OpenLLM to help you serve your model on Kubernetes.

Why should you choose Kubernetes?

- **Easy to scale.** Kubernetes can autoscale your application as and when it needs more resources. With higher loads, the application can scale up the number of pods to serve excess traffic.
- **Enhanced observability.** You can monitor and observe your workloads very easily with the help of open-source tools like ELK and Prometheus or managed solutions like Datadog, New Relic, and Dynatrace.
- **Better resource utilization and flexibility.** Deploy workloads on any cloud while using Kubernetes features like resource quotas to optimize your resource usage.

Kubernetes can be a great way to serve your models. This section discusses tools you can use to help.

Hugging Face

Hugging Face is a platform with a wide array of libraries and tools for developing, training, and serving AI models. You can simply choose the model to serve, and within a few minutes, it will be ready to deploy.

Pros:

- A great collection of libraries and data sets, as well as pre-trained models to serve
- Ability to deploy models using an inference API and easily access your models over APIs
- Support for NLP, speech recognition, and image processing, making it a very powerful platform for end-to-end product use cases
- Requires very minimal bandwidth and effort as a managed service provider

Cons:

- The solution is costly, as it is a managed service.
- Hugging Face has restrictions on the resources available for your workload; this limits the capability to serve large-size production-level deployments.
- Understanding the licenses of the models already present will be an extra step if you want to use predefined models.

Best use: Hugging Face is optimized for research projects where you need pre-trained models for testing and experimentation. However, it is unsuitable for production since you have no control over infrastructure.

BentoML's OpenLLM

OpenLLM was open-sourced by BentoML. It can very easily run multiple open-source models with few commands and serve them via an API on your chosen cloud.

Pros:

- Very easy to integrate with CI/CD pipelines, as it can be executed via commands
- Exposes the models via an API and also provides both a ChatGPT-like and command-line interface to test out the model
- Easy to deploy on Kubernetes, with numerous small and large open-source models available for use
- Can be easily deployed across multiple clouds and even has a managed cloud service

Cons:

- There is complexity in running and serving models during the inference phase and when serving actual queries. Models can take up large amounts of resources, which can be tricky with OpenLLM.
- Community support is not huge, so if you face issues, you may struggle to get any assistance. You can obviously opt for a managed cloud, but that can spike costs.
- Monitoring for your setup and load balancing are present, but these may require extra configuration and add to complexity.

Best use: OpenLLM is the top choice for deploying enterprise-grade, scalable models in an open-source ecosystem. You can also manage your infrastructure as needed.

Once you have selected your pipeline tools, created your models, and fine-tuned your data, you will need to start building pipelines for production. A few more challenges present themselves at this stage, primarily due to the comparatively young ecosystem of AI tools around us.

Let's look at the issues you will be facing during production deployments.

Kubernetes and Artificial Intelligence: The Problems

Despite the advantages, running AI workloads on Kubernetes—and using the above tools—does come with a few drawbacks you will need to keep in mind.

Resource-Hungry

Unlike normal microservices, AI/ML workloads are resource-intensive. They require a huge amount of memory and CPU in the learning phase and access to complex resources like GPUs. All this makes it complex to run these workloads on Kubernetes.

Proper resource utilization is also tough with Kubernetes, as there can be cases where large nodes will only be running one pod simply because all the pods need 32GB/64GB of memory.

With tools like Kubeflow, you'll also need to execute additional deployments such as Istio, rendering the cluster vulnerable to other possible issues.

Lack of Mature Guardrails

Most of the tools for running AI workloads on K8s are new to the various safeguards required, while others have minimal support for them. Basic security constructs like RBAC, authentication, and minimal permissions are not present in these tools.

A few of these tools also lack the capability to use Kubernetes isolation constructs like namespaces and network policies.

You can manage some guardrails via Kubernetes admission controller policies; however, covering everything is not easy. These tools have components that may be privately/publicly exposed without the ability to control them via admission controller policies.

Troubleshooting

Debugging issues in Kubernetes is already a tough task due to the many components running to power K8s. With the extra components needed to support the workloads of AI/ML tools, all of which are emitting logs and events, it becomes exponentially difficult to debug issues at the networking and storage levels.

Lack of Experience

Most of the tools discussed have evolved in the last four to five years, and the rise of AI has been in the last two years, so not many people in the industry have much experience running them in production—or at least not enough time to gain a level of expertise. You also need to know how to run these on Kubernetes, which is an added level of complexity.

Understanding the logs, events, and patterns of AI/ML tools, and if your team doesn't have this knowledge, you may struggle with managing them.

Komodor for AI

As discussed in the last section, troubleshooting in Kubernetes is a major problem with so many extra AI components and a lack of engineers with expertise in this domain.

Better visibility into what is happening in the cluster is the key to facilitating debugging issues.

With Komodor, you can see everything related to your AI workloads on Kubernetes happening in one place, empowering you to assess errors and remediate issues faster.

Boost your K8s ops for AI workloads, **try out Komodor today.**

GET STARTED