

# Challenges in managing AI/ML workloads on Kubernetes



# Table of contents

## 03 Data Science And Engineering Challenges With Kubernetes

---

## 04 Kubernetes: Design And Troubleshooting

---

## 05 Data Science Platforms On Kubernetes

05 Apache Airflow

07 Argo Workflows

09 Kubeflow

11 How Do They Compare?

---

## 12 Kubernetes Troubleshooting Challenges

13 Large Number Of Moving Parts

14 Dynamic Configuration Changes

15 Custom Controllers And Automation

16 Control Plane Flags And Configuration

---

## 17 Everyday Kubernetes Issues

---

## 18 Kubernetes Issues For Data Scientists And Engineers

18 Scattered Workflows

20 Lack Of Kubernetes Expertise

21 Deleted Pods

22 Unavailable Nodes

---

## 23 Conclusion

# Data Science and Engineering Challenges with Kubernetes

Kubernetes is one of the trending cloud computing technologies of recent years. Kubernetes and its ecosystem enable you to run scalable and flexible applications while removing platform dependency in the cloud. Its ubiquitous features, such as self-healing, service discovery, and load balancing, make Kubernetes the leader in the container orchestrator market.

Similar to containerization, data collected has also increased substantially in the last decade. This has resulted in a high interest in data science, even making a career as a data scientist “the sexiest job of the 21st century,” according to [Harvard Business Review](#). Data science combines statistics, mathematics, programming, AI, and machine learning to reveal actionable insights from data. This means it is essential for decision-making and strategic planning to future-proof your organization.

This article will discuss the collaboration between Kubernetes and data science. Data scientists look for critical features in their software stack: repeatable experiments, portability, reproducible environments, and high scalability. These features ensure that analysis and learning steps are executed even for large data sets. Fortunately, Kubernetes offers all these features out of the box, creating a solid foundation for data science applications.

# Kubernetes: Design and Troubleshooting

Kubernetes offers many benefits, but it is a complex system with multiple layers of [API abstraction](#). Kubernetes applications are defined via these API layers and then deployed to clusters. Application instances are distributed over cluster nodes with redundancy, and their health is regularly checked. This makes them reliable and robust even in the case of disasters.

However, errors, downtimes, and connectivity issues are inevitable when it comes to cloud computing and distributed systems. Thus, debugging and troubleshooting are essential parts of the daily life of Kubernetes operators. When an application gets stuck and becomes unresponsive, the underlying cause could be at the cluster level, node level, or even in the container definition. Finding the root cause requires a high level of Kubernetes knowledge and containerization experience.

When it comes to running data science workloads, Kubernetes enables you to run them scalably and flexibly, but it also brings its Kubernetes-native complexity into the life of data scientists. Let's start by discussing the data science platforms on Kubernetes and then dive into its management and operational challenges.

# Data Science Platforms on Kubernetes

## Apache Airflow

[Apache Airflow](#) is a widely adopted open-source workflow orchestration platform. It allows users to define and run multi-step pipelines, such as [DAG \(Directed Acyclic Graph\)](#), in Python. It focuses on simplicity and does not require a high level of Python to create workflows. Each DAG consists of nodes that are designed to run pods on Kubernetes:

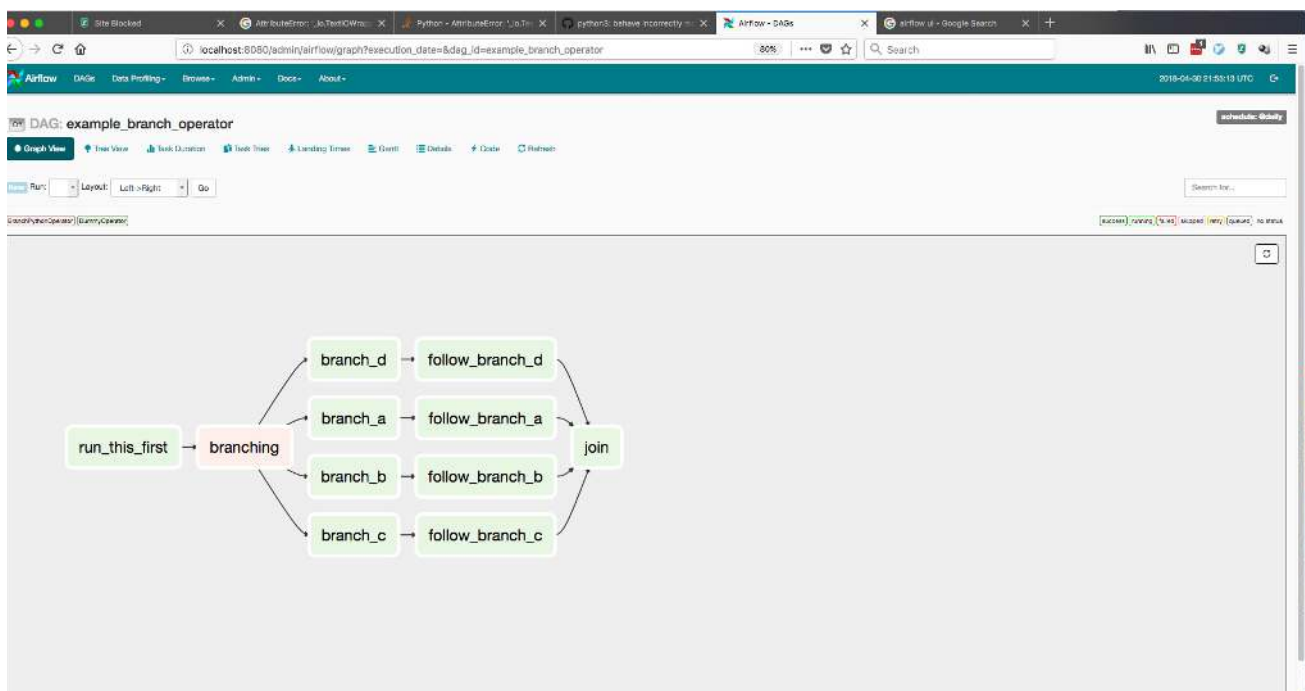


Figure 1: Airflow DAG (Source: [Kubernetes Blog](#))

Critical features of Airflow are as follows:

- Dynamic integration to cloud platforms such as AWS, Azure, or Google Cloud, as well as data engine platforms including Spark and HBase
- Customizability via combining operators, executors, hooks, and extended libraries
- User-friendly interface for creating workflows and checking their statuses
- Scalability with the simultaneous run of workflows on multiple platforms and clusters

Airflow's workflow approach makes it easy to automate data collection, processing, uploading, and creating reports. To run in Kubernetes, there is an integrated Kubernetes Operator that communicates via the Kubernetes API. The Scheduler in Airflow creates Jobs with the environment variables, secrets, and dependencies. Kubernetes handles the rest—scheduling pods to nodes, starting containers, and attaching volumes if necessary.

Airflow only needs to check the status of the jobs and collect logs to display in its user interface:

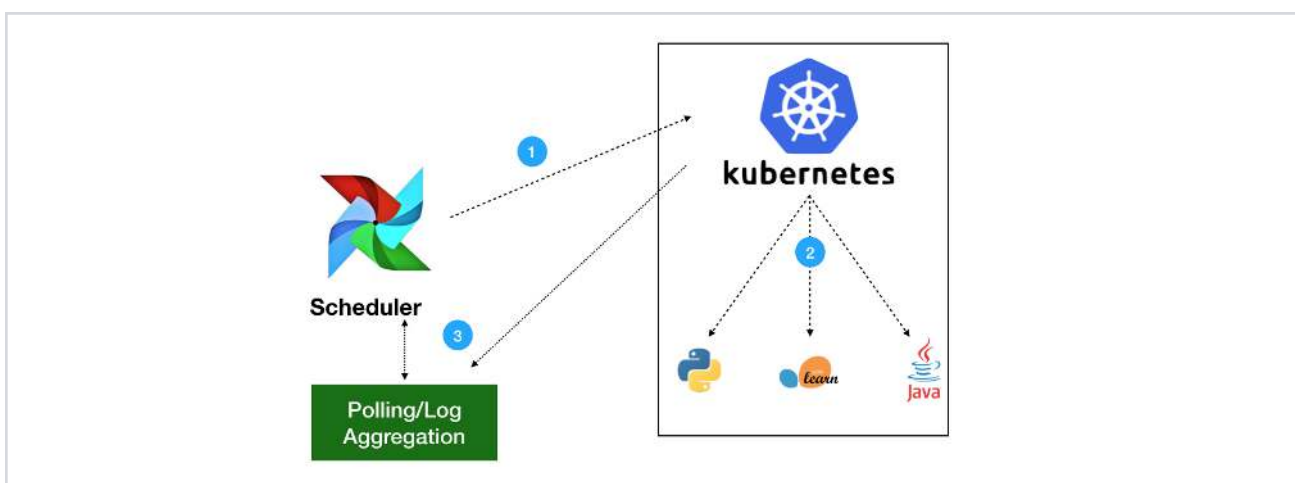


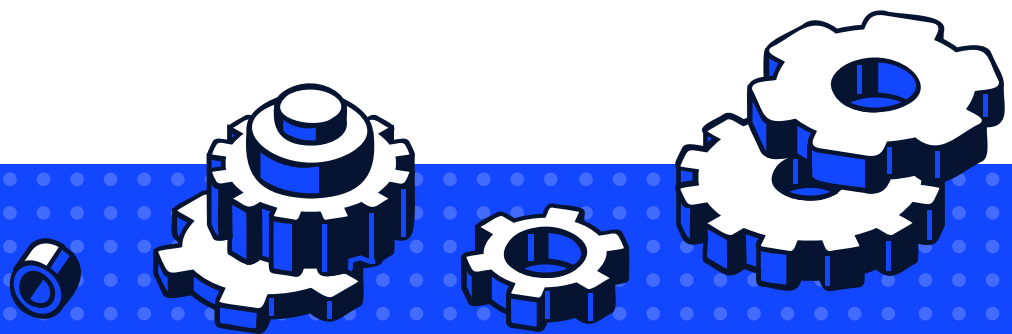
Figure 2: Airflow and Kubernetes integration (Source: [Kubernetes Blog](#))

## Argo Workflows

[Argo](#) was designed as a workflow engine for Kubernetes. Workflows in this open-source and container-native solution are defined as Kubernetes [custom resources](#), such as DAG, to specify dependencies, steps, and task sequences. Argo is primarily famous for CI/CD workflows, but you can also use it to define machine learning, data analytics, data science, and data processing pipelines.

Key features of Argo can be summarized as follows:

- Graduated as a CNCF incubating project with a dynamic community and strong connection to other CNCF projects
- Native cloud integrations to download, move, and upload files and data blobs
- Scalability to manage thousands of pods and parallel workflows
- Easy-to-use UI with events, widgets, and a log viewer





Argo is a Kubernetes-native platform, and each task step in the workflow runs as a separate pod:



Figure 3: Argo workflow UI (Source: [Argo Blog](#))

Tight integration into Kubernetes makes Argo highly parallel since it can scale horizontally and vertically with the cluster size. In other words, more Argo workflows will run in parallel when the [cluster autoscaler](#) automatically increases the cluster size. This makes managing clusters working with data science pipelines easier and cheaper.



## Kubeflow

[Kubeflow](#) is a machine learning (ML) toolkit to leverage machine learning pipelines with Kubernetes operations. It focuses on the complete data science workflow—from libraries and frameworks to pipelines and even notebooks.

Highlighted features of Kubeflow are:

- Comprehensive dashboard that enables the design, deployment, and monitoring of machine learning models
- Interoperability with data science libraries and frameworks such as TensorFlow, PyTorch, Scikit-learn, and Jupyter notebooks
- Monitoring and optimization with TensorBoard to visualize pipelines and tweak parameters

Kubeflow creates a bridge between Kubernetes and the leading ML platforms and libraries:

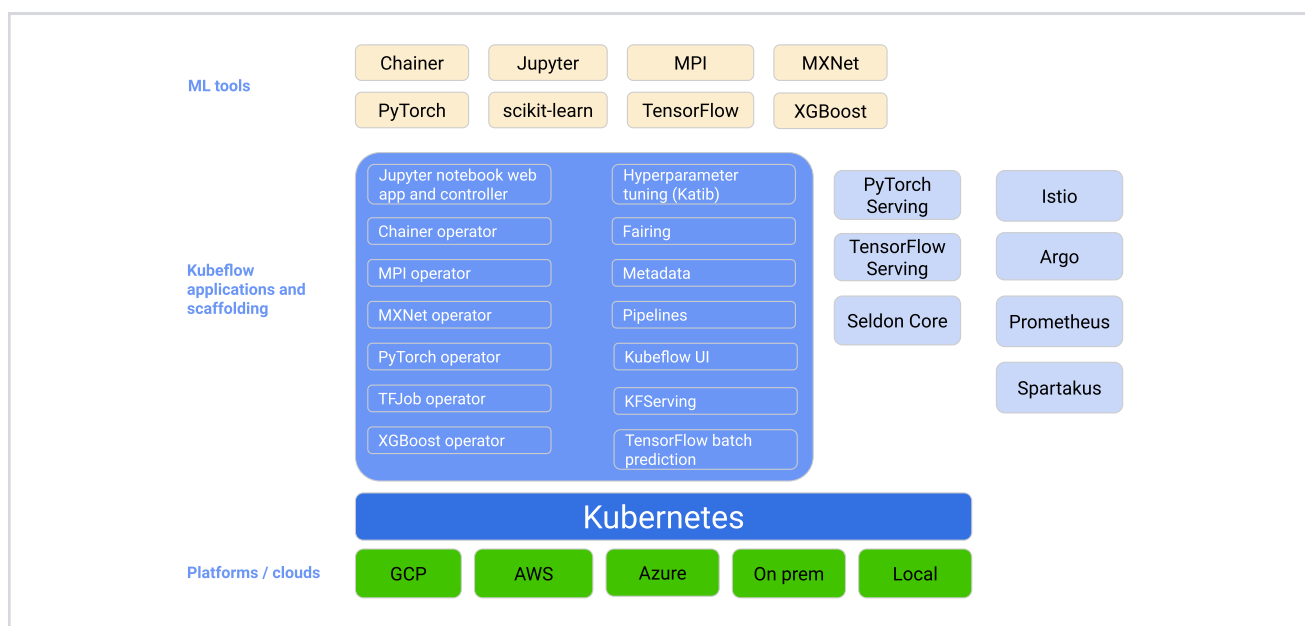


Figure 4: Kubeflow architecture (Source: [Kubeflow Docs](#))

With a central dashboard and [multi-tenancy](#), Kubeflow aims to implement an enterprise-level ML toolkit based on Kubernetes:

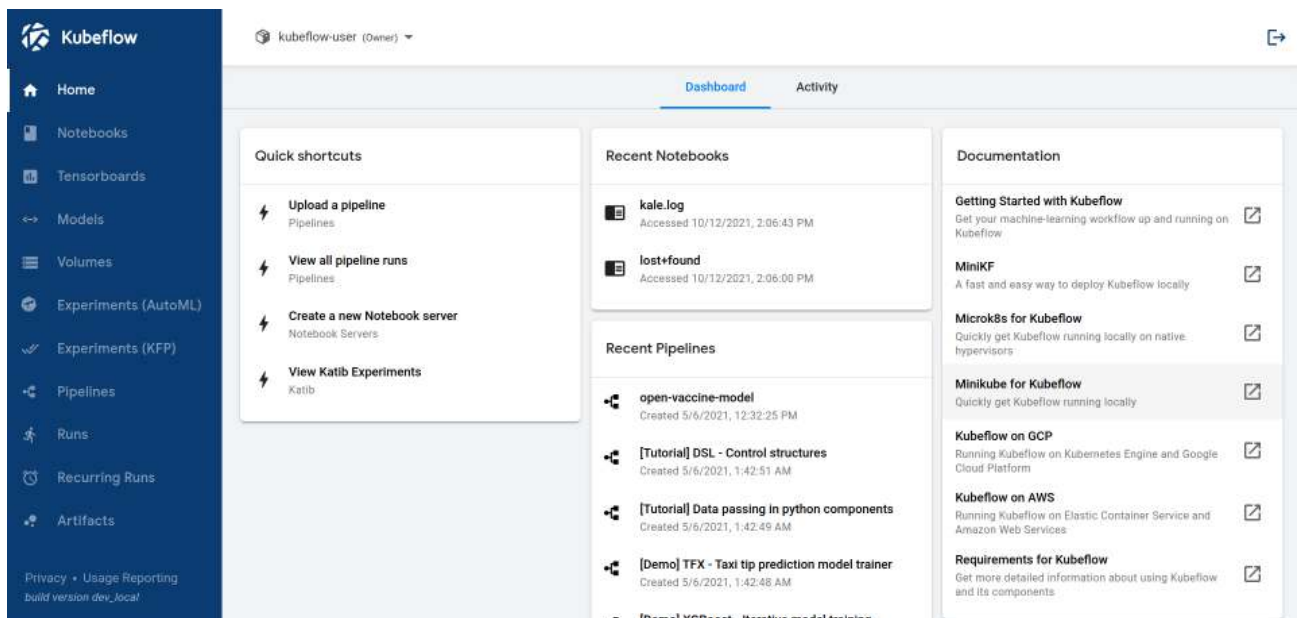
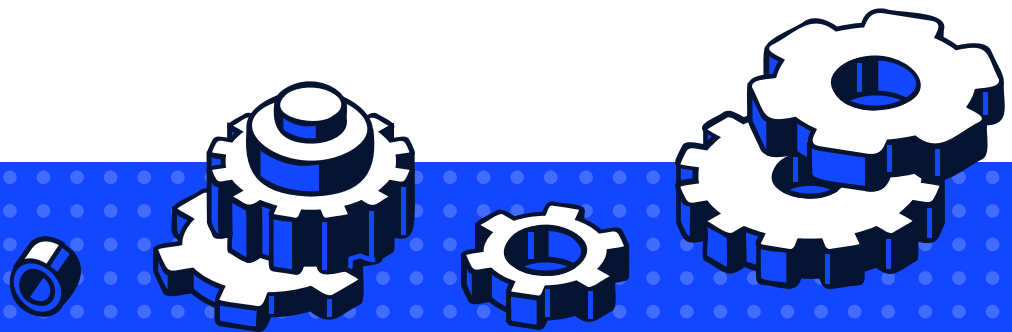


Figure 5: Kubeflow dashboard (Source: [Kubeflow Docs](#))

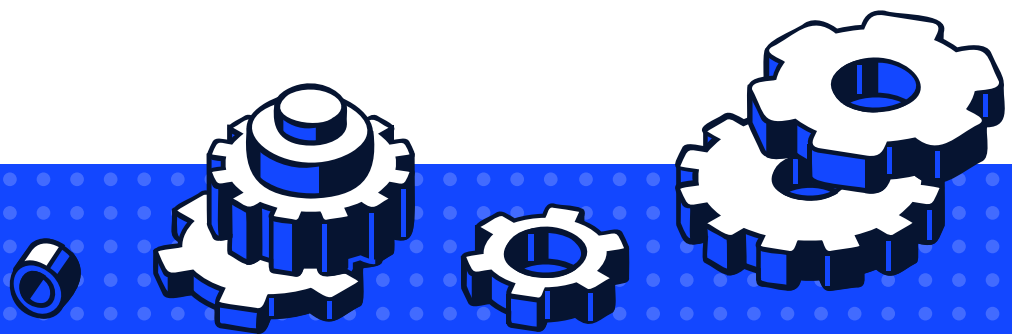


## How Do They Compare?

Upon looking at all three data science platforms, the main difference is how they interact and integrate Kubernetes into their operations:

- Airflow implements the Kubernetes Operator to run pods for pipeline steps.
- Argo enjoys full integration with Kubernetes resources by way of custom resources.
- Kubeflow is a Kubernetes-only platform to run ML projects while integrating pipelines, notebooks, and operators.

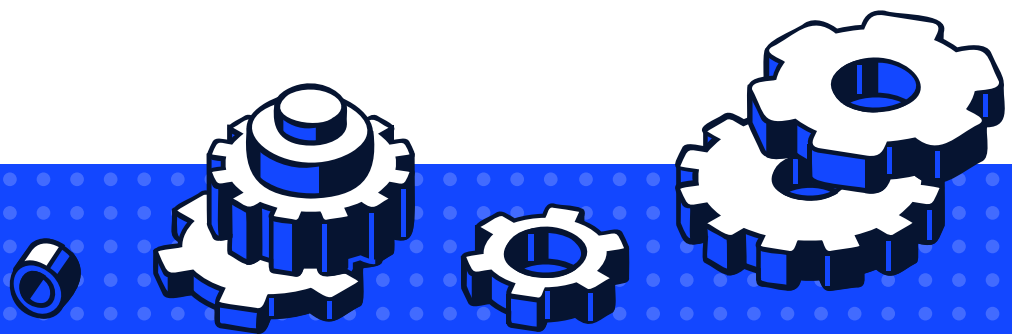
They all focus on flexibility, scalability, and rich user interfaces. In other words, they all aim to abstract away Kubernetes from the data scientists and create a scalable and easy-to-use environment on top of Kubernetes for data science workloads. However, none of these tools eliminate the troubleshooting and debugging requirements when problems arise. And when something does go wrong, the challenges of Kubernetes troubleshooting will be a critical hurdle for data scientists.



# Kubernetes Troubleshooting Challenges

Kubernetes is a complex system with multiple layers—infrastructure, nodes, containers, pods, and higher workload resources. On top of Kubernetes resources, custom applications such as databases, web frontends, and data science applications are designed and deployed. Thus, designing each layer with a cloud-native mindset is critical to benefit the most from Kubernetes and the underlying infrastructure. When there is an issue in any of these layers, troubleshooting and debugging activities take place. Unfortunately, this is not a straightforward task and requires a high level of Kubernetes and cloud computing knowledge.

There are four characteristics of Kubernetes that primarily make troubleshooting difficult.



## Large Number Of Moving Parts

Kubernetes clusters are made up of worker nodes, for running containerized applications, and a control plane. In the control plane, there are central components such as an API server, etcd database, scheduler, controller manager, and cloud controller manager:

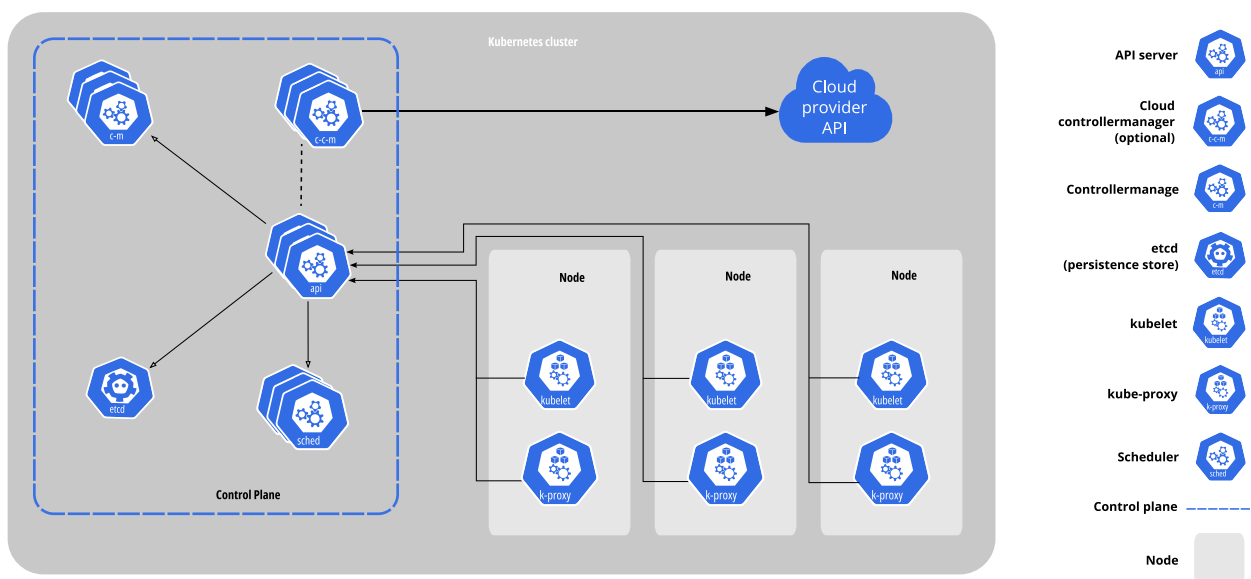


Figure 6: Kubernetes architecture (Source: [Kubernetes Docs](#))

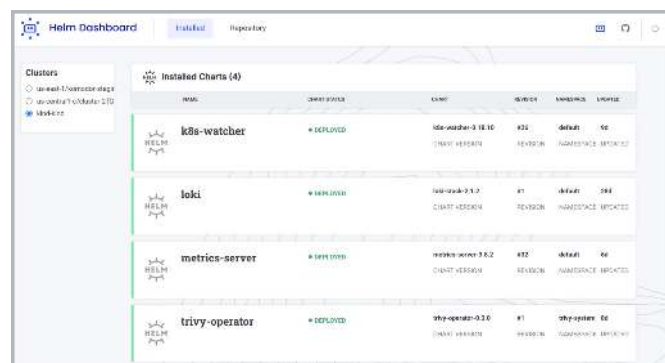
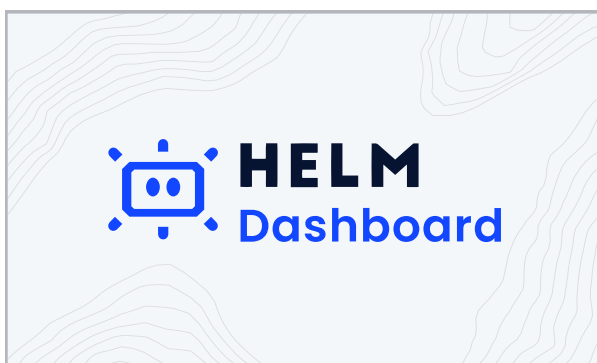
Even a single-node Kubernetes cluster has all these components to make the whole machinery work. For instance, when a job is created for the data science pipeline, the job controller—which is part of a controller manager—creates a pod with the job definition by communicating with the API server. Then, the scheduler finds a suitable node for the newly created pod. The components on the assigned node fetch the pod definition from the API server and handle creating a container and running it on the node. The agent in the node—kubelet—then checks the health of the pod and updates the pod status. Finally, the job controller updates the status of the job based on the pod status.

Even to complete just a single job, Kubernetes runs various components, making troubleshooting difficult if an error occurs in one of these parts.

## Dynamic Configuration Changes

Kubernetes offers a declarative API to define resource specifications and track their statuses. While designing and deploying applications, [declarative object configuration](#) with file directories is the suggested approach. Complex applications with multiple components, such as a PostgreSQL database, are packaged as [Helm Charts](#) and deployed with configuration files to the clusters. In addition to charts and files, there are also imperative methods to make changes directly on the resources—and in the cluster.

For instance, if you want to increase the number of PostgreSQL instances, you can run the `kubectl scale` command. The command sends data to the Kubernetes API and changes the number of replicas in the resource definition. However, it does not change the deployment files, which become obsolete after imperative commands. Therefore, tracking the source of changes and Kubernetes resources to make sure they're trustworthy is essential when debugging and troubleshooting.



## Did You Know?

Komodor provides A simplified way of working with Helm. Komodor makes operating helm-charts streamlined and visual, getting you started or speeding up your helm operations & troubleshooting.

You can review current state and its health, learn how versions changed over time, and make changes.

The helm dashboard was released as the missing UI for Helm.

[Helm @Komodor](#) is ultimately another type of resource you can inspect.

## Custom Controllers And Automation

Kubernetes is an extensible platform with [custom resources](#) and controllers to fulfill business requirements that are not part of core Kubernetes. For instance, you can define a new workflow instance in Argo Workflow with the following Kubernetes custom resource:

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-from-k8s
  labels:
    workflows.argoproj.io/archive-strategy: "false"
  annotations:
    workflows.argoproj.io/description: |
      This is a simple hello world example.
spec:
  entrypoint: whalesay
  templates:
    - name: whalesay
      container:
        image: docker/whalesay:latest
        command: [cowsay]
        args: ["hello world"]
```

Argo Workflow installation in the cluster watches for these kinds of custom resources and creates pods, jobs and all required Kubernetes resources based on the description. However, a misconfiguration in the workflow spec above could lead to a failed pipeline, which can be challenging to debug and find the root cause of.

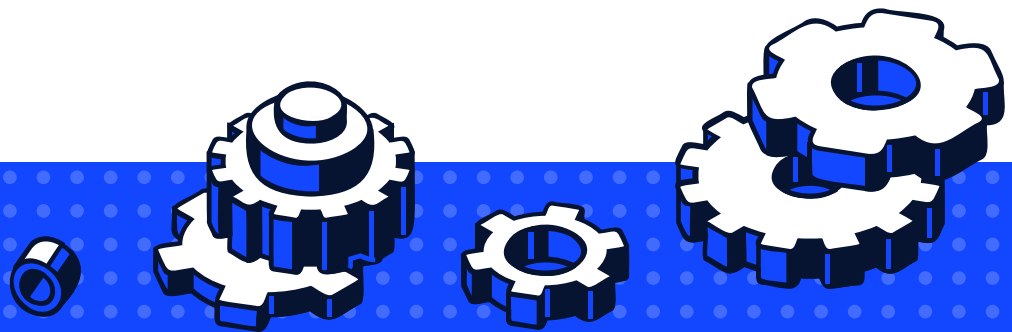


## Control Plane Flags And Configuration

The Kubernetes API server is the central place to deploy applications and check their status from outside. In addition, all internal components of Kubernetes communicate with the API server to read specifications, update a status, and take actions. In the latest version of the [kube-apiserver component](#), there are nearly 200 flags to configure and change the behavior of the whole cluster.

For example, if a security audit reveals that [privileged containers](#) are allowed in the cluster, the reason could be an `--allow-privileged` flag of the API server. Kubernetes offers high flexibility with flags and configuration files, making it difficult to trace the root cause of errors in the cluster.

Troubleshooting Kubernetes is challenging, but there are some common issues, discussed below, you can be aware of.



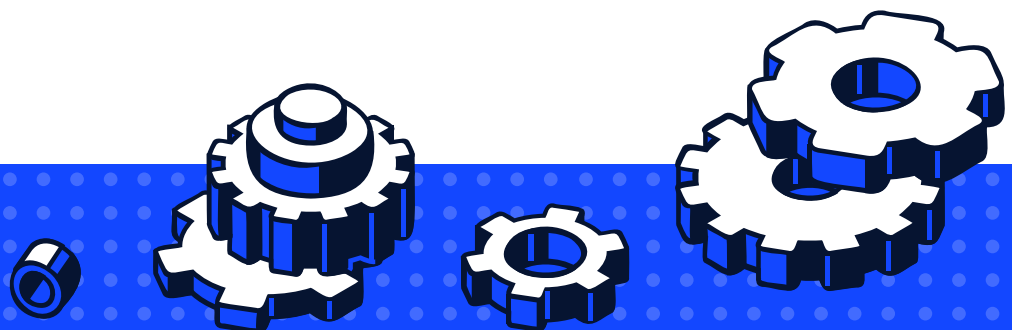
# Everyday Kubernetes Issues

Kubernetes relies on its underlying infrastructure to connect nodes and distribute workloads. Therefore, it is possible to see problems related to infrastructure and connectivity.

For example, you can lose the network connection between nodes and the control plane. You will primarily realize this when you can connect to the node and see all containers running. However, kubelet will not be able to update the status of the pods in the Kubernetes API.

Pods have a status field with the phases `Pending`, `Running`, `Succeeded`, `Failed`, or `Unknown`. In addition, an array of `PodConditions` shows which conditions have already been met: `PodSchedule`, `PodHasNetwork`, `ContainersReady`, `Initialized`, and `Ready`. Pods stuck in `Pending`, `Terminating`, or `Unknown` indicate that some underlying components are failing to complete their tasks.

Lastly, when the underlying infrastructure, such as networking or storage, has issues, you will be blocked from creating containers or attaching volumes, and pods will not achieve the `Running` phase.

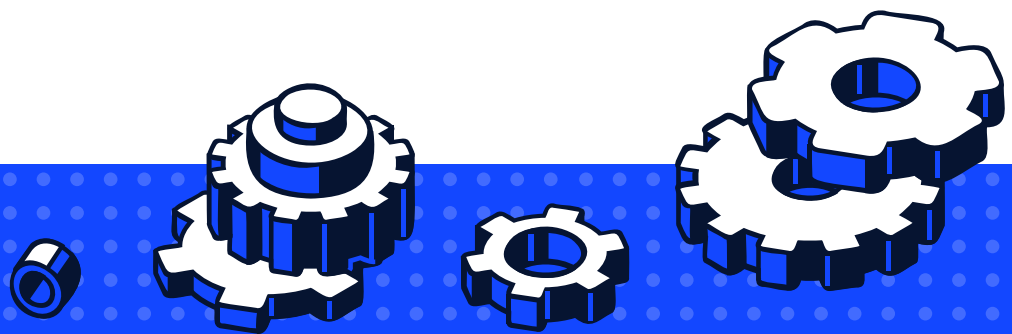


# Kubernetes Issues for Data Scientists and Engineers

When you run data science workloads and distributed pipelines, you may see different issues than everyday Kubernetes problems. For most organizations, Kubernetes is a platform to run their long-running applications in a scalable manner. Therefore, the main concern is the application's uptime and reliable infrastructure. However, when it comes to data pipelines and workflows, there are slightly different concerns and daily issues.

## Scattered Workflows

Kubernetes distributes pods to nodes in order to balance the resource usage and requirements across the cluster. This distribution, however, could lead to one step of the workflow being in node A from Availability Zone A, and the next step in node B from Availability Zone B. While scattered pods are not an issue for distributed applications, they can still pose issues if something goes wrong, as operators must connect multiple systems and debug different nodes with networking and storage configurations.



Komodor correlates all resources of a specific workflow or pipeline as shown in Figure 7 below:

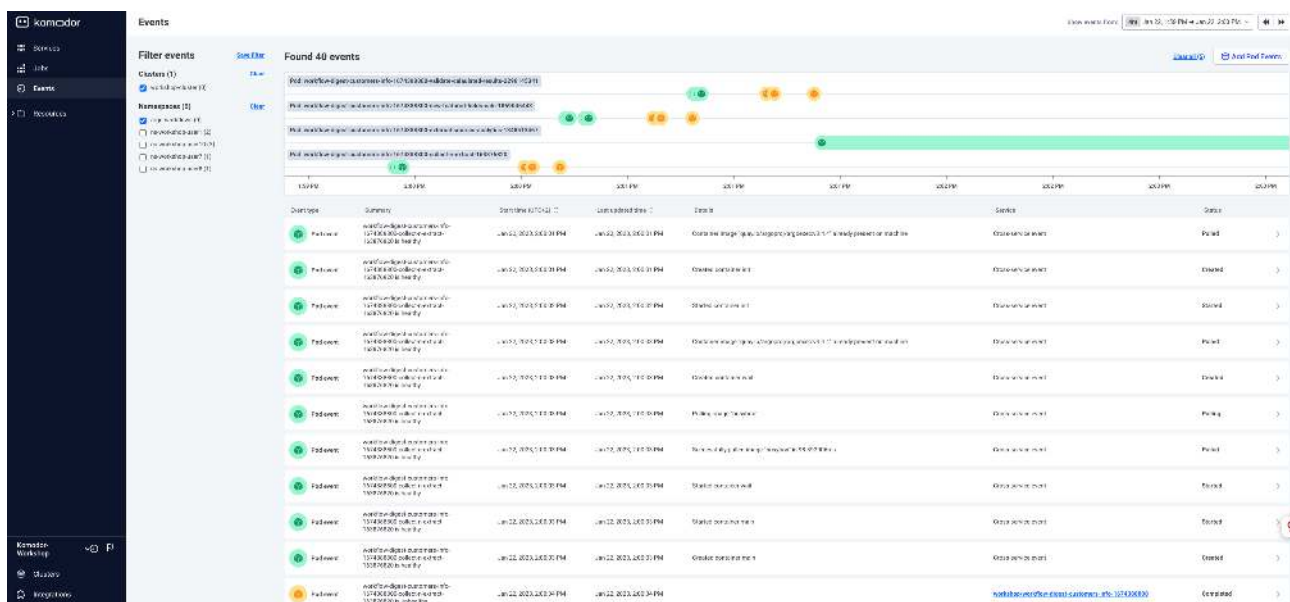


Figure 7: Komodor and Argo Workflow events

Furthermore, Komodor provides troubleshooting insights and failure information for each event:

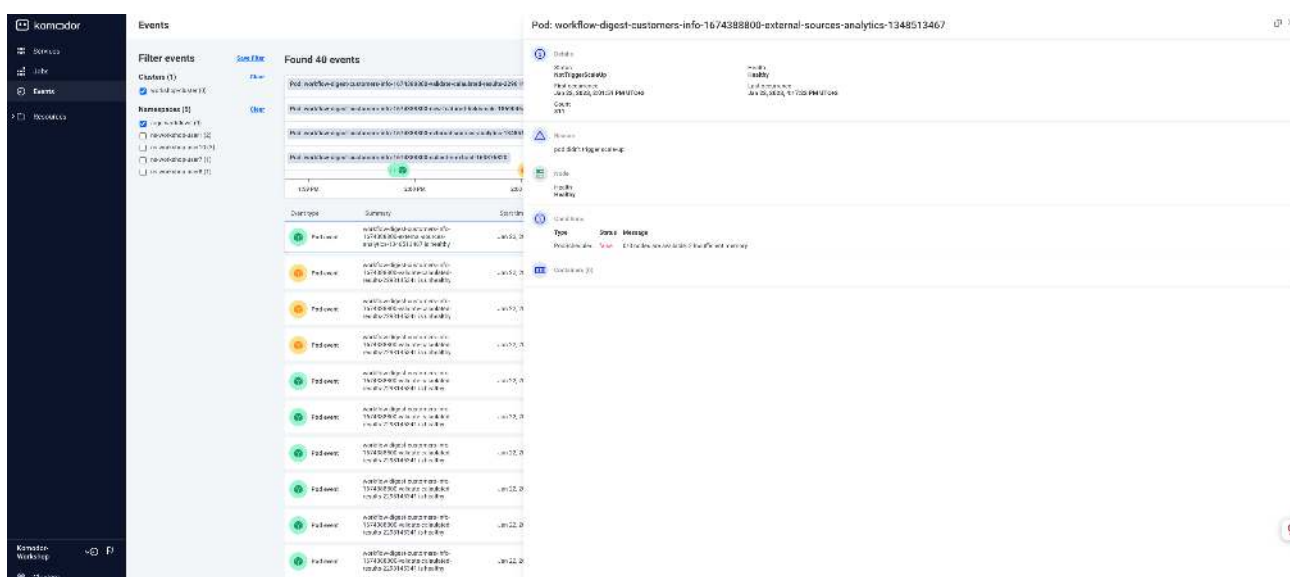


Figure 8: Komodor and failure information from pipeline

## Lack Of Kubernetes Expertise

Most DevOps engineers have either backend or operations backgrounds. Their cloud computing and system administration competencies are sufficient for dealing with the challenges of Kubernetes troubleshooting. But the same level of experience is not expected for data scientists and engineers, as Kubernetes is not their core competency.

Komodor is an intuitive Kubernetes-native platform that doesn't require deep K8s knowledge—even when it comes to troubleshooting an unhealthy task:

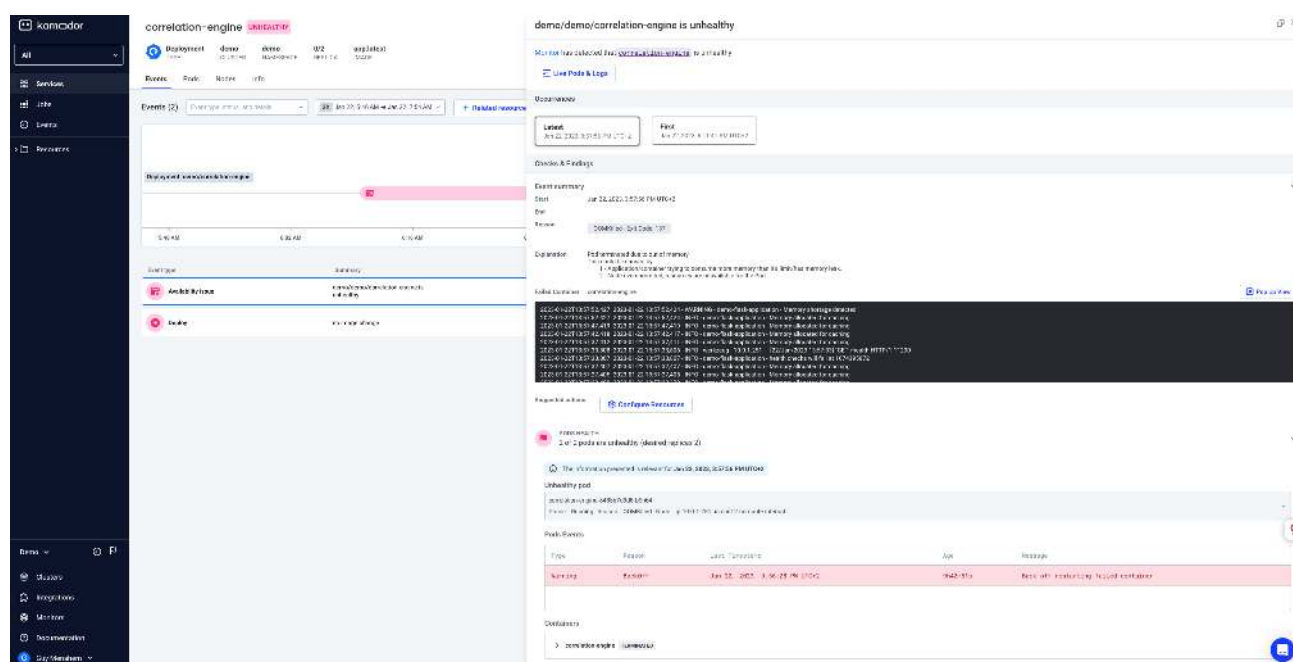


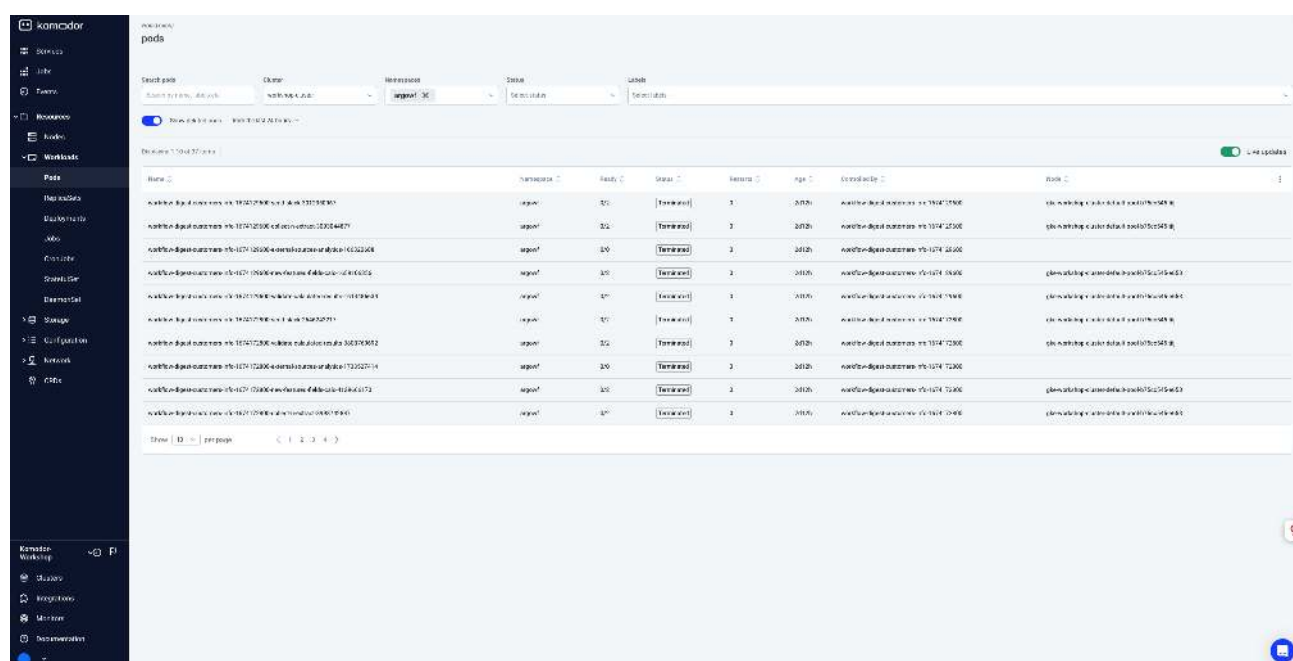
Figure 9: Komodor and troubleshooting explanations

## Deleted Pods

In Kubernetes, pods are ephemeral and scalable, with the applications designed accordingly. In other words, if there are four instances of a frontend application, it is acceptable to have three or five instances for some time. This is mainly caused by orphaned, uncontrolled, or deleted pods.

For data science pipelines, these issues are more critical since there should be, at most, only one data processor working on the same data set. Thus, ensuring and troubleshooting uncontrolled pods for data scientists is challenging.

While operating a Kubernetes cluster, it can be difficult to find information about the deleted resources since Kubernetes does not provide a long history of events. Komodor addresses this by providing visibility into resources that were already deleted from the cluster or that failed (Figure 10):



The screenshot shows the Komodor 'Deleted Pods' interface. On the left is a sidebar with navigation links: Home, Jobs, Tasks, Resources, and Workspaces. The 'Workspaces' section is expanded, showing 'Pods'. The main panel displays a table of deleted pods. The table has columns: Name, Namespace, Ready, Status, Reason, Age, Container ID, and Link. All pods in the table have a status of 'Terminated'.

Name	Namespace	Ready	Status	Reason	Age	Container ID	Link
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details
komodor-deploy-10000000000000000000	default	True	Terminated		10s	komodor-deploy-10000000000000000000	View pod details

Figure 10: Komodor and terminated pods of workflows



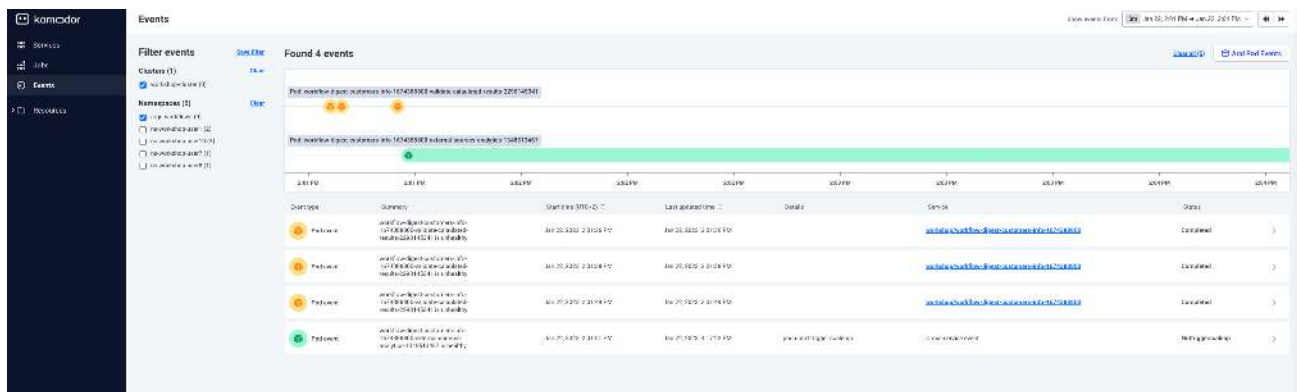


Figure 11: Visualize deleted pods on Komodor timeline

## Unavailable Nodes

Kubernetes distributes a workload to nodes and ensures there are enough replicas running inside the cluster. When some nodes are unavailable, Kubernetes recreates the pods on the available ones. However, data pipelines consist of steps that are dependent on each other. When some pods become unavailable due to their nodes, the workflow will have trouble continuing with the next steps.

Komodor auto-detects unavailable nodes or nodes under pressure and automatically troubleshoots and alerts when something goes wrong, with further playbook information:

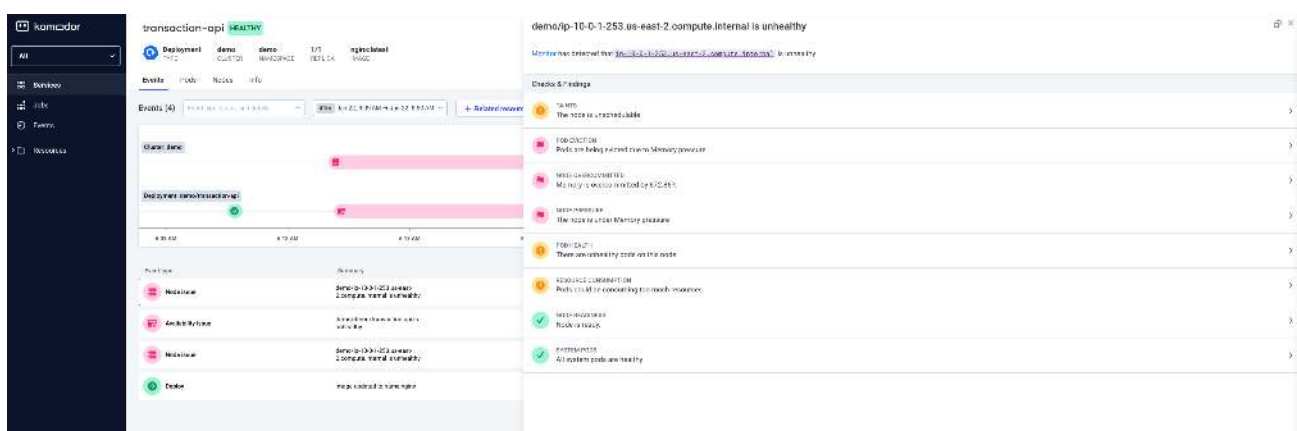


Figure 12: Komodor and node troubleshooting



# Conclusion

Kubernetes unquestionably provides many benefits for data science workloads and pipelines to run on cloud infrastructure. However, finding underlying problems when things go wrong is not straightforward. Therefore, a Kubernetes-native troubleshooting solution is more beneficial for data scientists and engineers in order to gain more from cloud computing.

[Komodor](#) is a Kubernetes-native platform to monitor your entire stack and identify issues along with their root causes. It is the troubleshooting platform to ensure you can run data science workloads on Kubernetes confidently.

## Technology Partners



# Create your free trial account

**START FREE**

