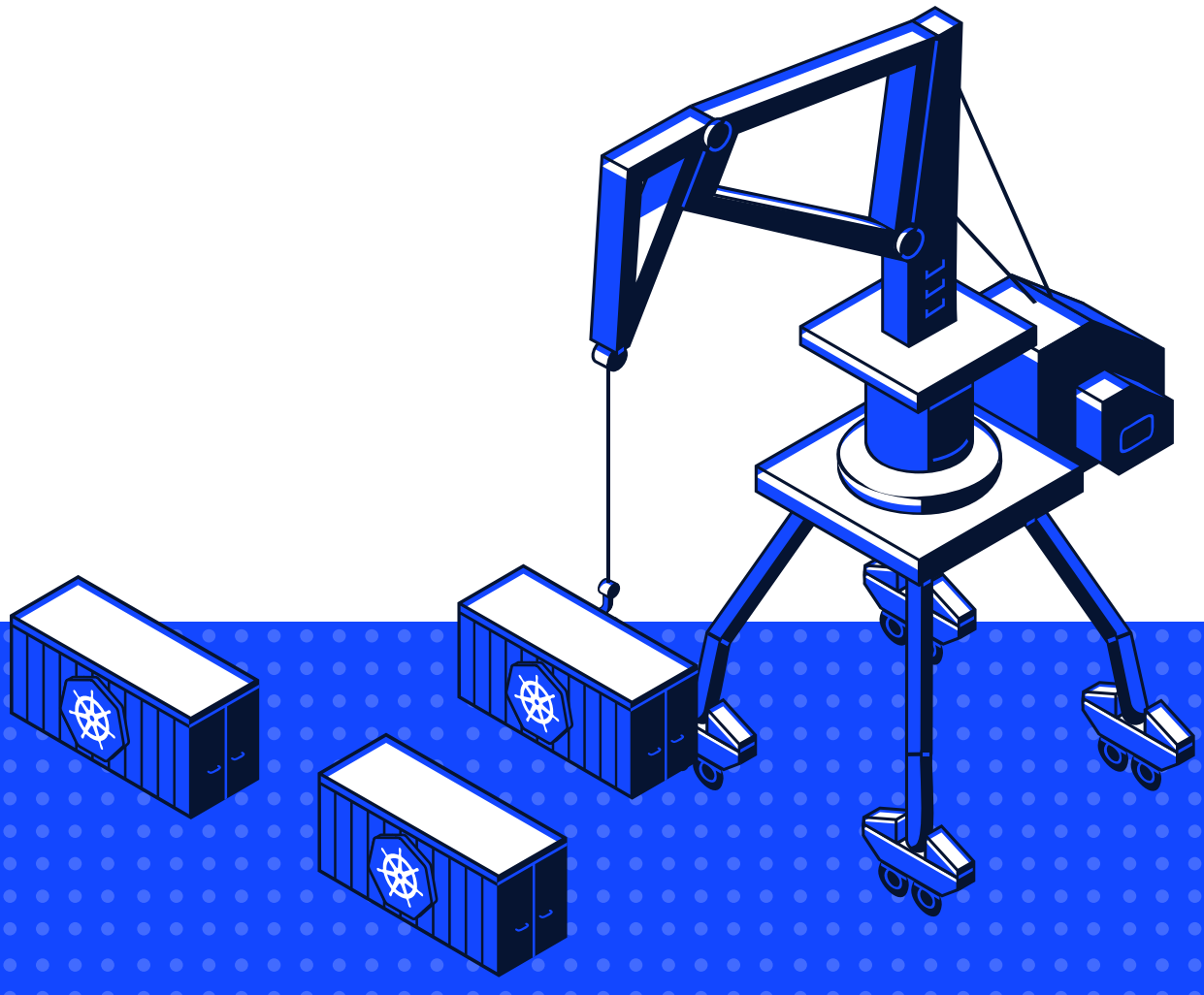
komodor

The Guide to Day-2 Operations in Kubernetes

Table of contents

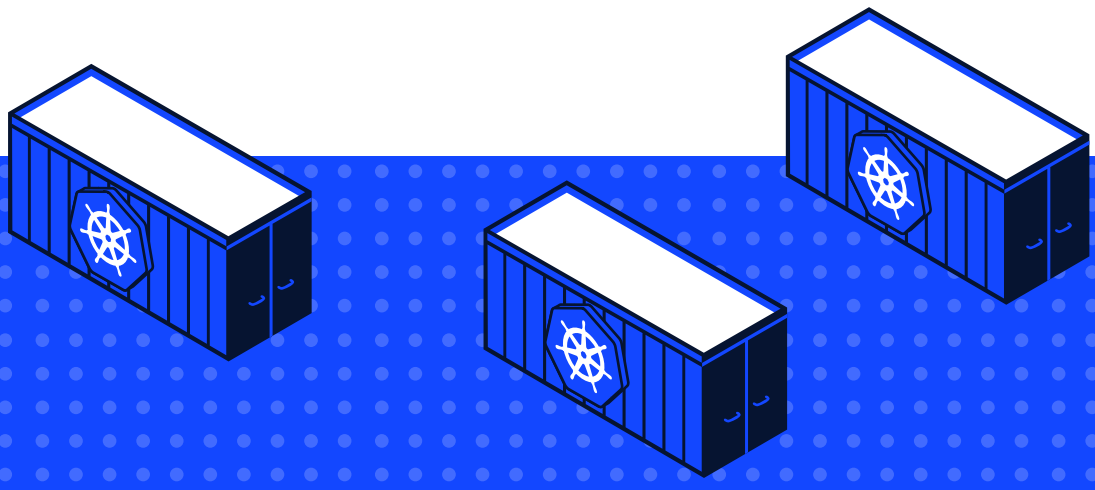


Introduction

The virtualization of a computer system is when a user combines several physical devices into one virtual environment, offering the user a single computing environment with flexibility over operating system choices. However, it would be challenging to manage multiple VMs simultaneously. You would need to ensure their maturity: production readiness, on-call alerts, node metrics, and so on.

Managing VMs or Kubernetes itself directly also involves other difficulties when it comes to application management. Automating the deployment process of applications and creating a reliable and secure network that can be used to communicate entail quite complex and error-prone processes. Especially when it comes to alerting against any failure in your cluster, Kubernetes does not provide any integration on Slack or other on-call systems.

However, thanks to Komodor, you can simplify the monitoring, troubleshooting, and intervention processes of applications that run on Kubernetes.



How Komodor Helps with Resource Utilization

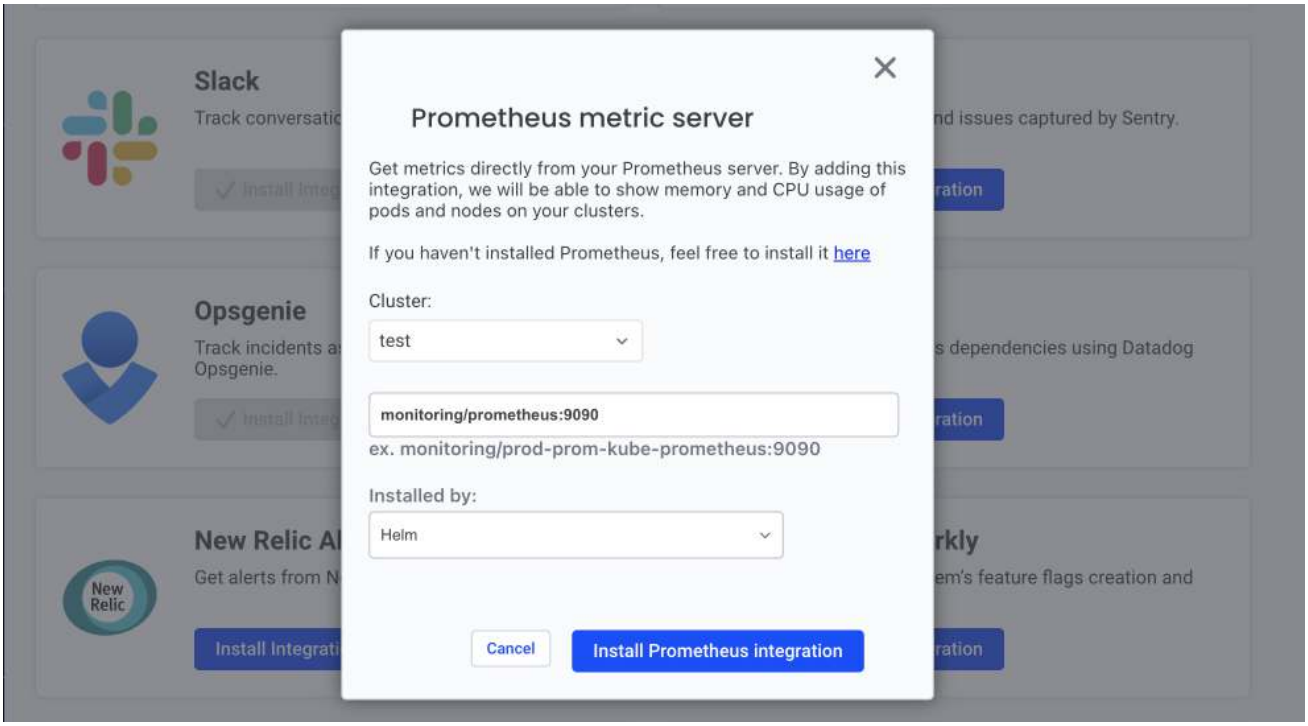
Resource utilization is critical in the Kubernetes world. CPU and memory are the two resources that are specified the most. When you specify a pod, you may optionally define how much of each resource a container requires.

A container may utilize more resources than what is specified in its request for that resource if the node on which a pod is executing has enough of that resource available. However, a container may not consume more resources than allowed by its resource limit.

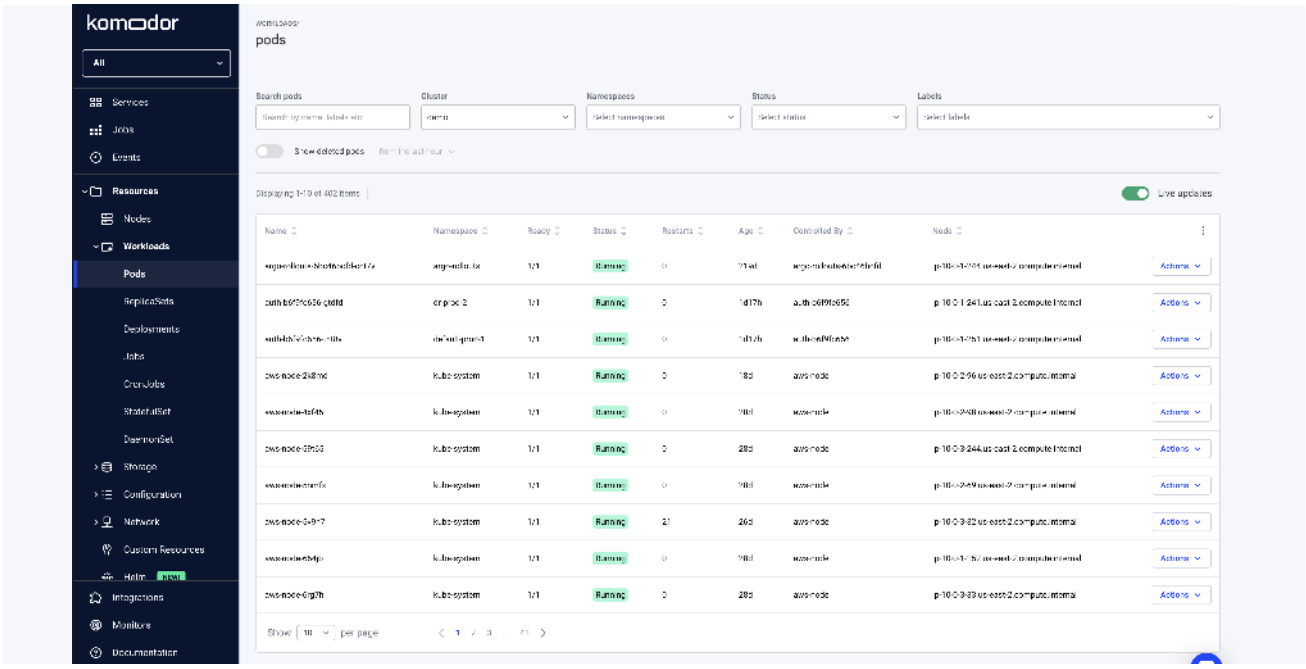
Normally, when you deploy an application to Kubernetes, you need to specify resource requirements for your pods. Those are hard-coded static values. But this creates an issue of unnecessary resource usage. Komodor's resource utilization feature provides suggestions on the optimum resource requirement values for your applications.

How does Komodor help with resource utilization?

- Creates a deployment
- Deploys Prometheus to your cluster
- Checks Komodor's dashboard



When we complete our monitoring setup steps, we can navigate to and see our resource usage on the Workloads -> Pods page:

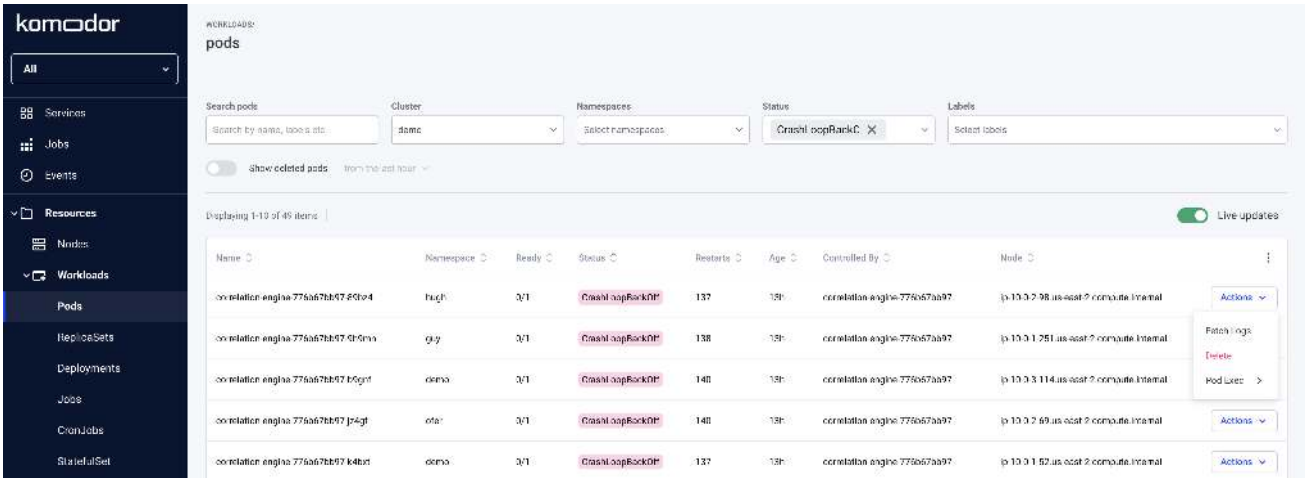


Scheduling

The act of matching pods to nodes so that Kubelet can run them is referred to as scheduling. Additionally, the scheduler keeps an eye out for freshly generated pods with no allocated workload. The Kubernetes scheduler is responsible for determining the best workload for each new resource it finds and assigns those pods to nodes to run on.

With the help of Komodor's monitoring system, users can easily find failed/ unhealthy pods via the UI, or get notified, and take actions against them.

You can easily view a list of your pods and their status on the Workloads/Pods page. You can gather the logs from pods or delete failed pods and let the scheduler reassign them:



START FREE

Scaling Up And Down

The number of resources being assigned to your application can be automatically scaled up or down based on CPU/memory utilization or any other specific metrics your application may expose. This technique is known as autoscaling. For instance, the number of workloads may be increased or decreased depending on the demand and the user count.

Komodor Actions enable users to take actions quickly on resources. These can include different interactions such as scaling services, deleting pods, restarting applications, rolling back a pod to previous versions, re-triggering jobs, cordoning/uncordoning nodes, and deleting resources.

You can use both VerticalPodAutoscaler (VPA) and HorizontalPodAutoscaler (HPA) in the Kubernetes environment to automatically scale your applications:

- HorizontalPodAutoscaler (HPA) automatically updates a workload resource to match demand. More pods will thus be deployed as a result of increased load.

- VerticalPodAutoscaler (VPA) automatically recreates your resources with the appropriate CPU/memory attributes. It can also automatically update attributes and suggest CPU/memory requests and limits.

Lastly, Komodor Dashboard allows users to easily see their nodes and workloads scaling up and down on the Event page. It visualizes, groups, and filters existing events in the system. You can dive into an event's details by simply clicking on it; you will see the related workload, configuration, and available actions to take.

Rolling Back

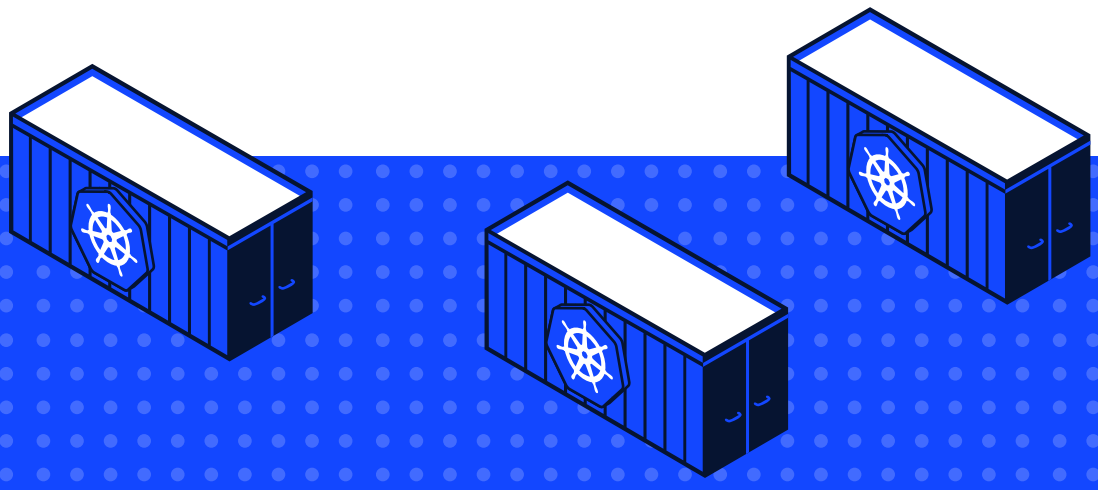
Kubernetes provides a quick way to undo changes made to resources like deployments, StatefulSets, and DaemonSets. This method is called a rollback. When the application is not stable, such as if it is in a crash loop, the rollback mechanism is useful. By default, the system saves the history of every deployment so that you can always roll back anytime.

How Komodor Actions Helps

Komodor aims to take the complexity out of Kubernetes troubleshooting. We set out to create a platform that would streamline root cause analysis, a platform that could quickly and intuitively identify when anything goes wrong.

Komodor Actions goal is straightforward. By consolidating all resources within the Komodor platform and making them easily accessible, we want to reduce the amount of time users spend moving between tools and break down the knowledge barrier that currently exists for non-Kubernetes specialists. With Actions, a proposed action to gain the needed insight is already prepared and can be carried out with only one click.

Take a look at the Rolling Back to Previous Versions to see how to accomplish rollback actions with Komodor.



From VM to Kubernetes: How Kubernetes Simplifies the Deployment Process

Organizations used to run applications on physical servers. This made allocating resources a challenge since physical servers don't allow you to define boundaries for your application's resources. Then Kubernetes came along.

Kubernetes enables the declarative configuration and automation of containerized workloads and services. It reduces the burden of manual maintenance and simplifies the deployment process.

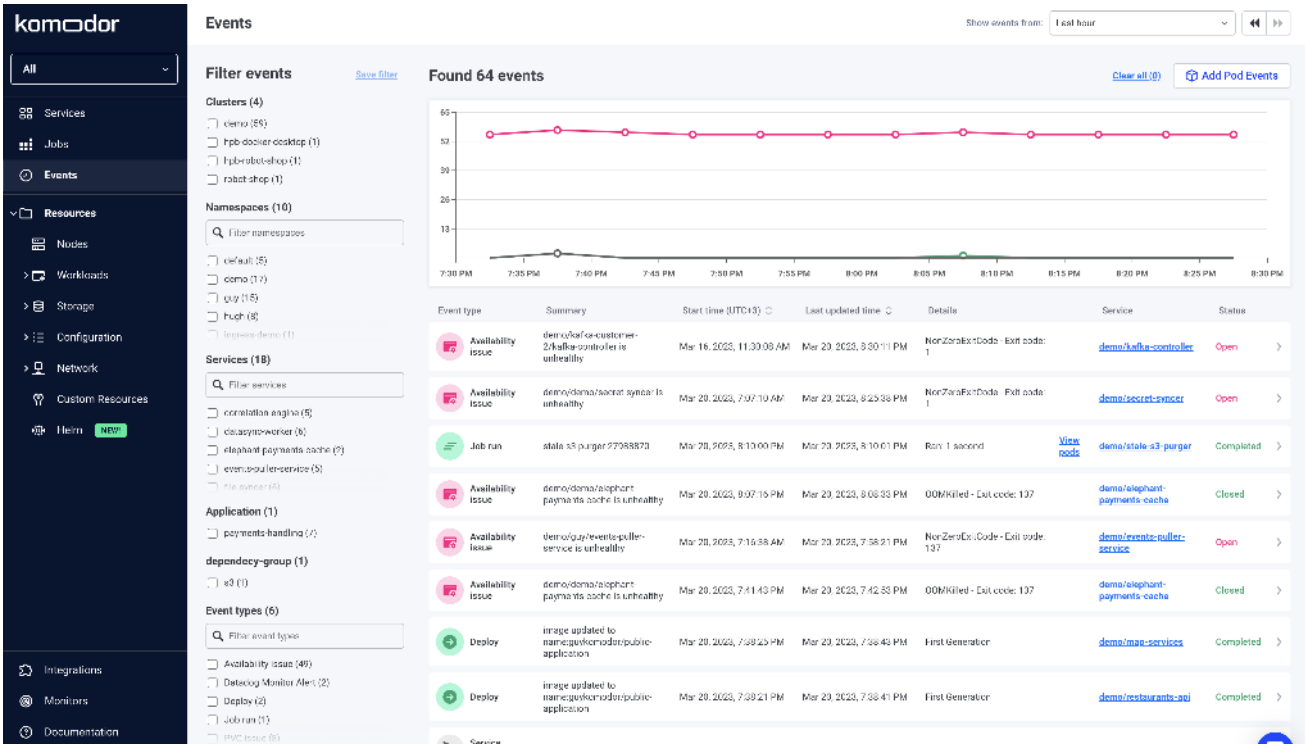
Monitoring Kubernetes Infrastructure With Komodor

Komodor is available as a hybrid application. The Komodor team manages and operates the web UI in the cloud. The Komodor agent, which comes in the form of a Helm chart and publishes data using the cloud UI, has to be installed in your cluster. This indicates that all communication for your cluster is outbound. You don't need to add any new port ranges to your allowlist or open any ports in your firewall.

Your dashboard will be instantly populated after your Komodor agent has been properly deployed. Once Komodor is monitoring different events in your cluster, the web UI will receive real-time updates.

Deployments, DaemonSets, and StatefulSets are all supported by Komodor. Furthermore, a list of each of your namespaces is included in the left sidebar, along with some additional details.

All your Kubernetes events are visible using Komodor event visualization, and you can filter cluster events according to various options:



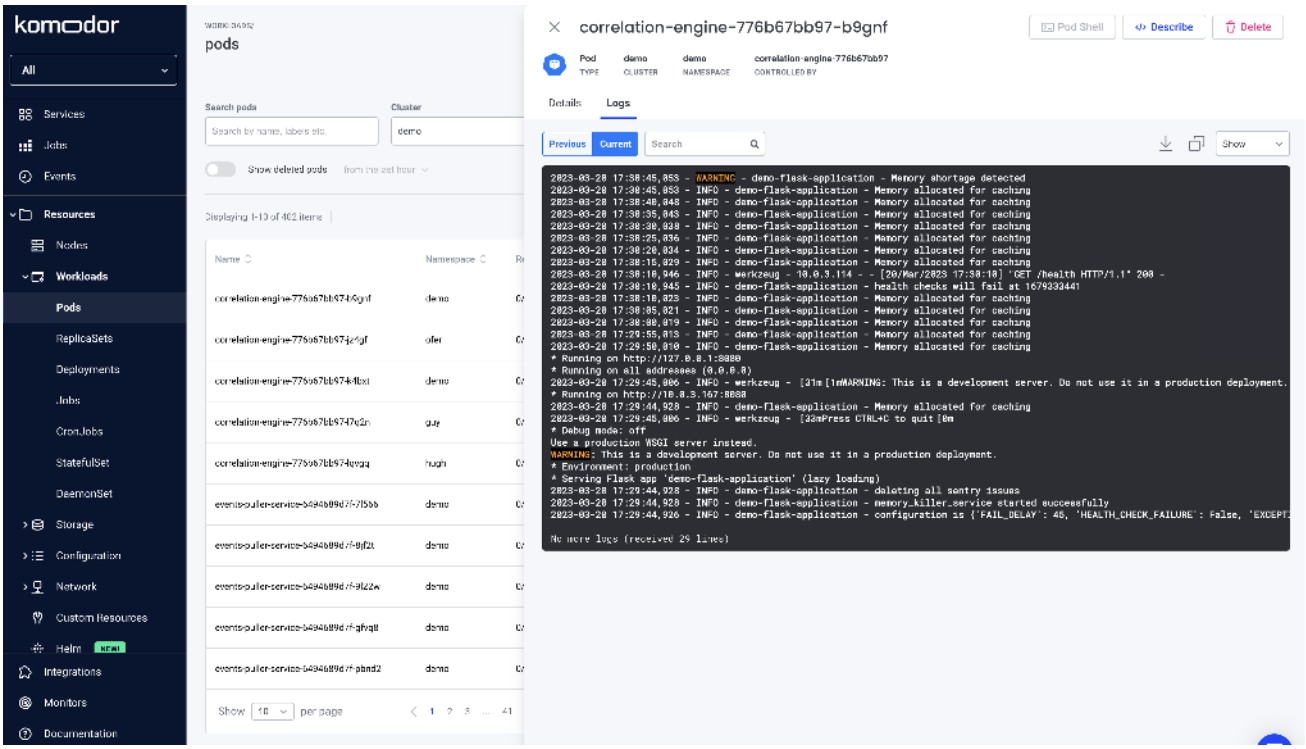
What Problems Are Solved By Kubernetes

The Kubernetes platform runs applications and services effectively. In addition to handling application scale and failover, it also provides deployment patterns. As a comprehensive container orchestration platform, Kubernetes offers an extensive range of services to enhance your application's performance and stability. These include service discovery and load balancing, storage orchestration, automated rollouts and rollbacks, self-healing, and efficient secret and configuration management, among many others.

How Komodor Helps With Application Troubleshooting

Consider a scenario in which a deployed app has failed. With Komodor, you can see all the logs to easily troubleshoot your pods:

Simply click the failed pod and jump to the logs panel:



Metrics Provided By Kubernetes

System component metrics can give a better look into what is happening inside applications, especially when visualized via dashboards and implementing alerts i. Kubernetes' components also emit some metrics you can monitor in Prometheus. You can set up Prometheus or another metrics scraper to collect these metrics on a regular basis and store them in a time-series database in a production environment.

You may also want to install a node-exporter to expose a wide variety of hardware- and kernel-related metrics across your cluster. cAdvisor would also be beneficial because it gives you knowledge about the resource usage and performance metrics of running containers.

How Komodor Metrics Help Monitoring Applications On Kubernetes

You can quickly identify at a high level how your services are being used across many clusters with Komodor. It gives you situational knowledge of what's going on in your particular cluster, informs you about new services that have been installed, and lets you know what other alarms are currently being generated. Komodor offers all of this in real time via its timeline view to help you hone in on the root cause of an unhealthy cluster or service.

You can access historical information as well and view all of a cluster's activity, past and present, in a single pane. With Komodor, you gain valuable insights into the factors that impact the health of your service and the underlying reasons behind any issues.

Managing Cluster Access

Authentication and authorization is one of the most important security topics in the Kubernetes environment. The resources that individuals, teams, and apps can access, as well as the actions they can perform, should be restricted as your organization grows in terms of the number of workloads, applications, and team members. This is possible with Kubernetes' role-based access control (RBAC) framework.

Using RBAC, you can control who has access to what computer or network resources based on their individual roles within your organization. You can dynamically configure policies using the Kubernetes API because it leverages the API to drive permission decisions.

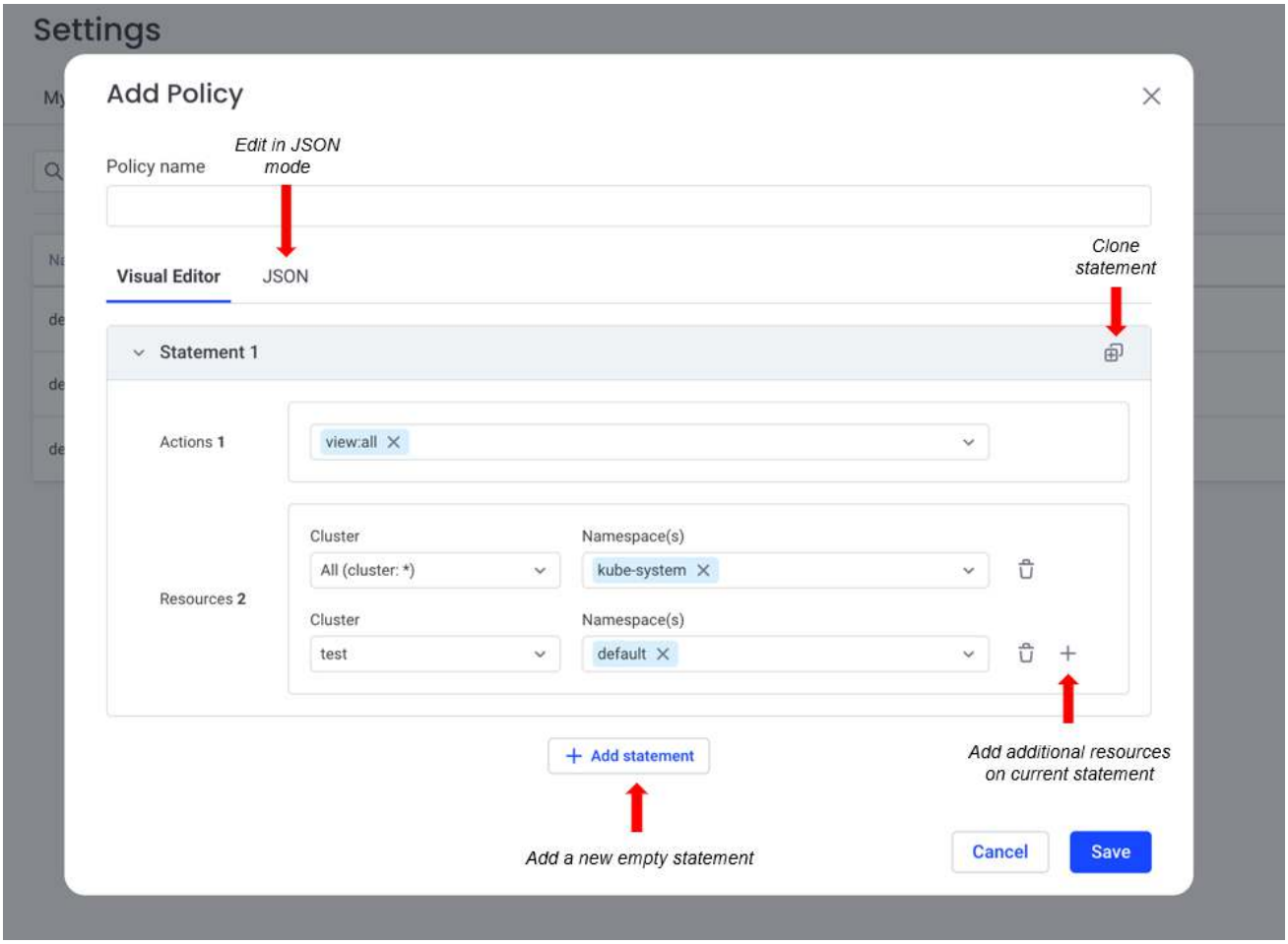
The actions a user may take inside a cluster or namespace are specified by ClusterRoles and Roles in the Kubernetes environment. Role and cluster role bindings let you give these responsibilities to subjects (users, groups, and service accounts).

Policy Example

The following policy allows you to view all resources, delete pods, and scale deployments on resources running on the namespace bar on the cluster named baz:

```
[
    {
        "actions": [
            "view:all"
            "delete:pod",
            "scale:deployment"
        ],
        "resources": [
            {
                "cluster": "baz",
                "namespaces": [
                "bar"
                ]
            }
        ]
    }
]
```

To create the above policy, simply jump to the settings page, find the policies tab, and click the JSON tab. Paste the policy, give it a name, and save. You can also create custom roles associated with other policies to assign the role to an existing/new user/s later on.



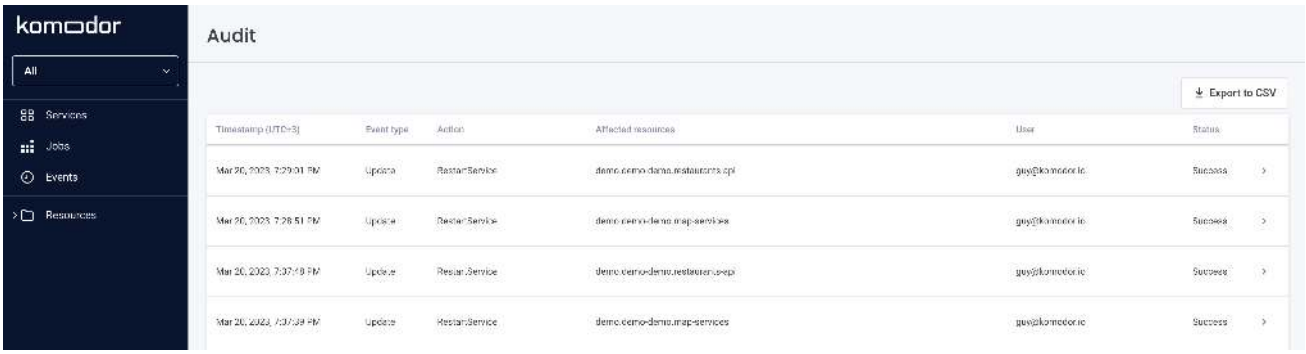
User Management And Cluster Security: Komodor Users & RBAC

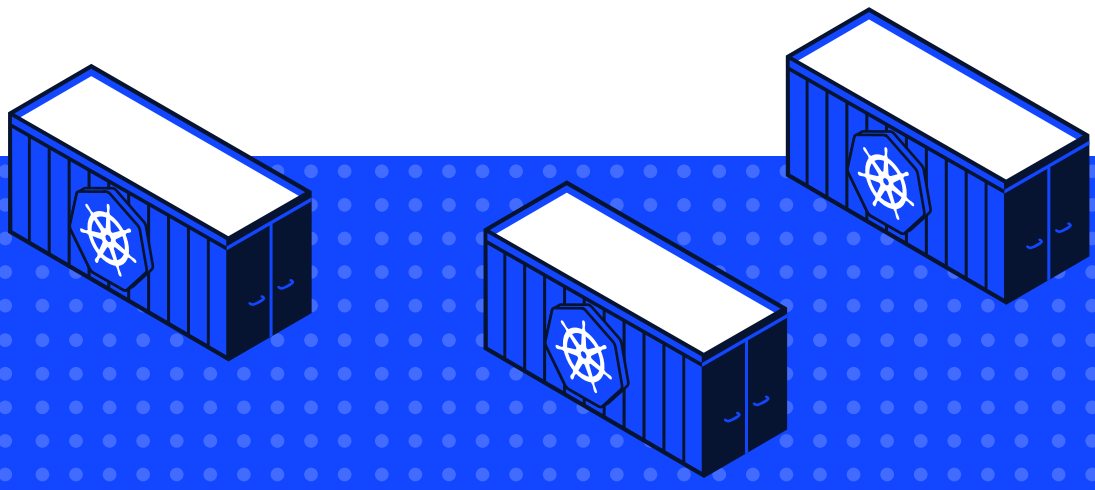
Komodor introduced RBAC support to enable DevOps, SRE, and platform teams to tailor access control and policy settings as needed without losing track of changes made to their Kubernetes infrastructure. This eliminates not knowing whether and when someone modified the cluster, who did it, and what the effects were.

DevOps and non-K8s professionals can now close the troubleshooting loop independently by identifying, looking into, and resolving any issue directly through Komodor's platform without any hassle.

In the Komodor dashboard you can see your cluster user's assigned roles:

You can also view your cluster user's actions with the Komodor audit feature:





Migrating a Go Application to Kubernetes

First, we start with containerizing our application using a very simple Go application:

```
package main

import (
 "fmt"
 "net/http"
)

func main() {
 http.HandleFunc("/test", func(w http.ResponseWriter, _ *http.Request) {
  fmt.Fprint(w, "Hello world!")
 },
 )
 http.ListenAndServe(":8000", nil)
}
```

Run the “go run” command on the terminal to try it for the first time. Also, check the curl at http://127.0.0.1:8000/test to see how it works.

Inside the Dockerfile, input:

```
FROM golang:alpine
RUN mkdir /app
COPY . /app
WORKDIR /app
RUN go build -o main .
CMD ["/app/main"]
```

Let's push the image to DockerHub first. Create the image with the repository specified:

```
$ docker build -t username/hello-world .
```

This should give the following output:

```
Successfully tagged username/hello-world:latest
```

Then run:

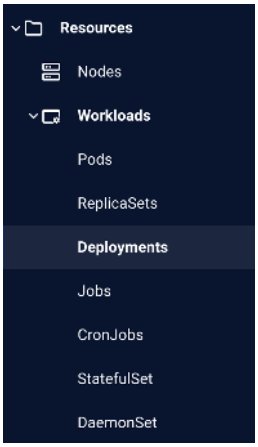
```
$ docker push username/hello-world:latest
```

We can pull the image we created and pushed to Docker Hub using the following registry address: [docker.io/username/hello-world]

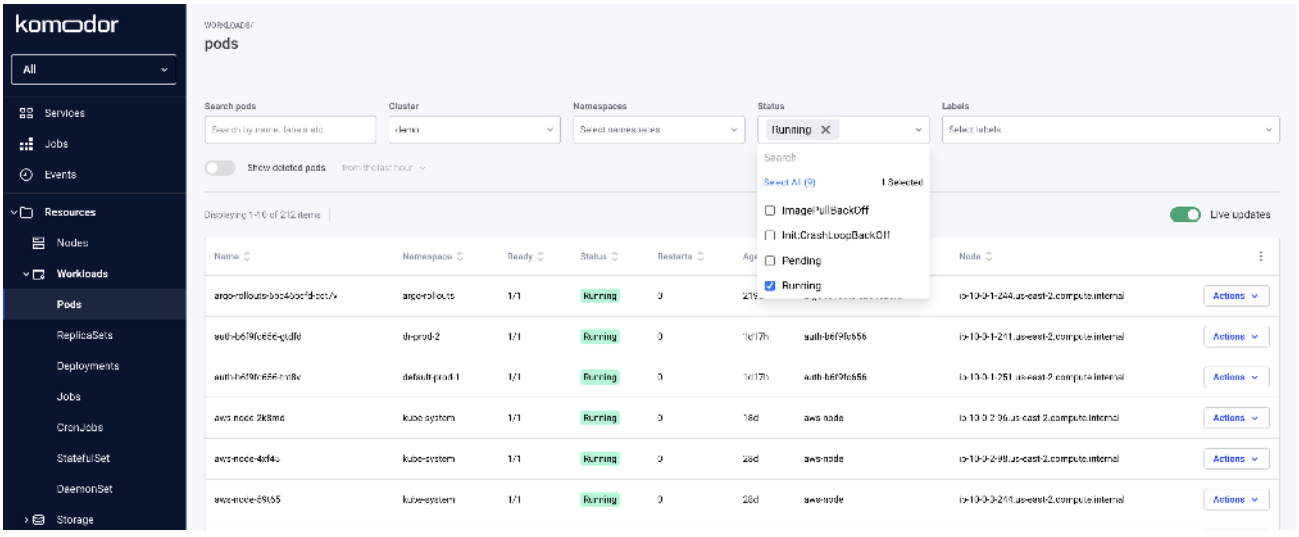
Next, create and deploy kubernetes manifests:

```
$ kubectl create deployment my-go-app
--image=docker.io/username/hello-world
$ kubectl expose deployment my-go-app --port=8000
```

Next, use the Workloads navigation to check your application status with Komodor:

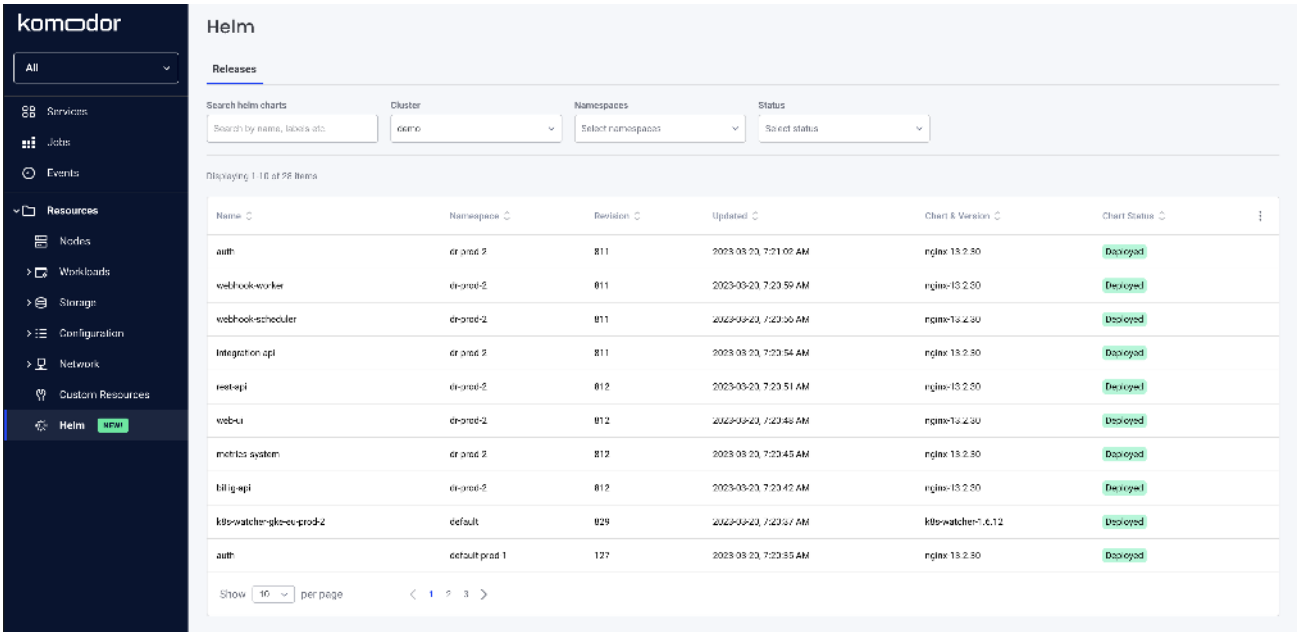


If you want to filter your pods by state, namespace, or labels, you can use the top filter to list your pods and check the health status of your applications:



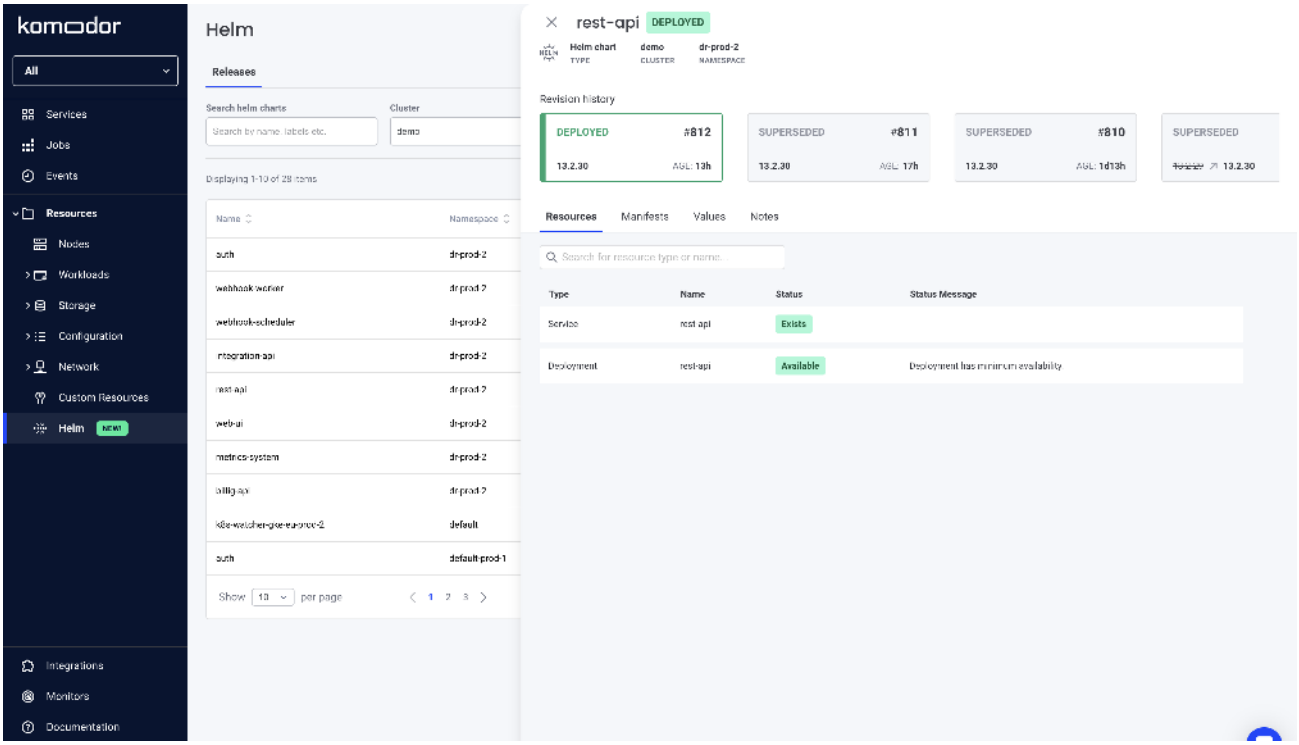
Monitoring Deployments with Komodor Helm Dashboard

Komodor creates tools that help you make sense of the complexity created by distributed cloud-native systems. It renders Kubernetes operations and troubleshooting simple and available to everyone. Another step toward realizing that objective is Helm Dashboard.

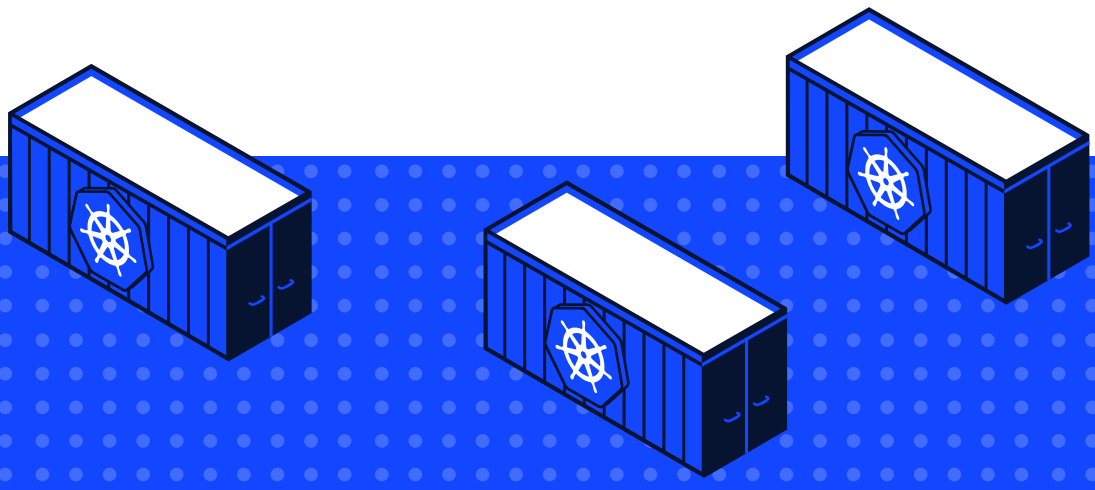


The first feature in the [Helm Dashboard] is the obvious one: displaying a list of installed charts along with their status. Its ability to preview manifest changes while updating or altering the chart is perhaps its most crucial feature. You can also quickly remove a chart with a single click, roll it back to a previous version, and upgrade charts to any current version.

Helm Dashboard enables users to see resources, Helm manifests, and values per revision:

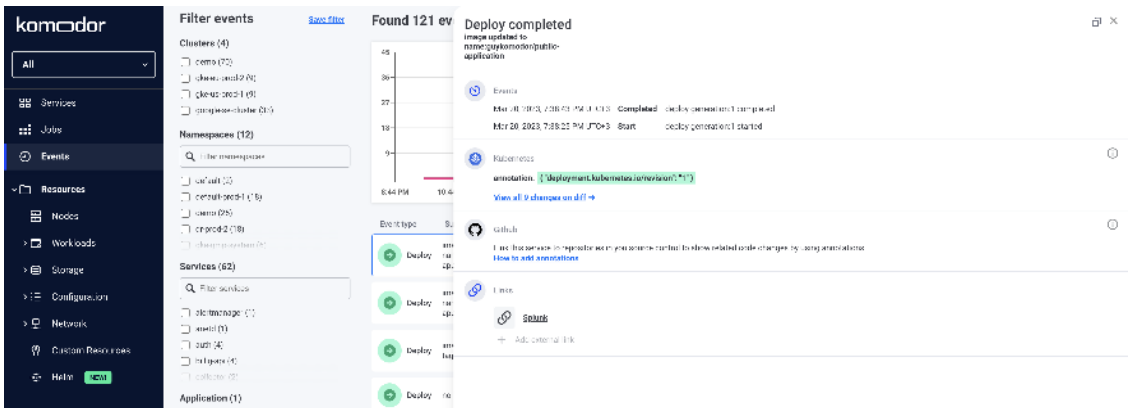


Komodor users can now see their Helm deployments by navigating to the [Helm page](#).



Monitoring Deployment Changes with Komodor

Komodor additionally shows deployment changes between versions in the Events dashboard:



If you click the “View all X changes on diff,” a popup will appear to make comparing resource versions easier for you:

START FREE

You can create a simple deployment manifest file (deployment.yaml) to test this feature by yourself:

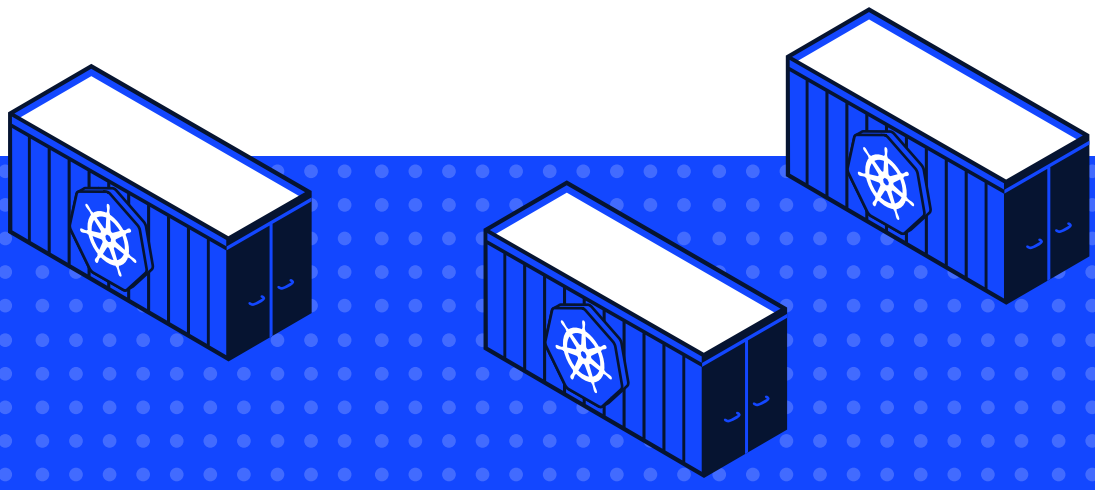
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  selector:
 matchLabels:
   app: hello-world
  template:
    metadata:
      labels:
     app: hello-world
     spec:
   containers:
   - name: hello-world-container
     image: username/hello-world
     resources:
         limits:
             memory: "64Mi"
           cpu: "256m"
         ports:
      - containerPort: 8080
```

Deploy the deployment.yaml file using the following command:

```
$ kubectl create -f deployment.yaml
deployment.apps/hello-world created

$ kubectl get deployments
NAME         READY     UP-TO-DATE   AVAILABLE   AGE
hello-world   1/1       1            1           9s

$ kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-7gy5671f6b-3tyqr   1/1       Running   0          17s
```



Configuring Application Lifecycle Requirements

Defining Health Checks

In a case where we run an application in Kubernetes, we need to add health checks: liveness and readiness probes. A liveness probe checks the application's operating status, whether the application is ready to service traffic. If it fails, the container will be taken out of service by the load balancers.

Many programs that are used for extended periods of time eventually transition to broken states and can only be fixed by restarting. To identify and address such issues, Kubernetes provides liveness probes.

A basic HTTP handler that returns the HTTP status code “OK” may be enough to set up a readiness probe. However, an event, for instance, the database being available, may need to occur before being able to serve traffic.

In your deployment manifest, in the spec.template.spec.containers[] field, you can use the livenessProbe and readinessProbe to define your requests and limits as follows:

```
livenessProbe:
      httpGet:
          path: /healthz
          port: 8000
readinessProbe:
      httpGet:
          path: /readyz
          port: 8000
```

Resource Quotas: Limiting Memory And CPU Usage

There is a chance that one team will consume more resources than it ought to when numerous users or teams share a cluster with a predetermined number of nodes. Administrators can use resource quotas to address this issue. This is a default feature in most Kubernetes distributions. A ResourceQuota imposes restrictions on the total amount of resources consumed by a namespace.

A container will get the memory it requests, but it is not permitted to consume more than what it is allocated. It is advantageous to have a default value set for the memory limit if your namespace has a preset memory resource quota. The control plane sets the default memory limit for every container that is used to create a pod but does not specify a memory limit of its own.

In your deployment manifest, in the spec.template.spec.containers[] field, you can use the resources property to define your requests and limits as follows:

```
resources:
      limits:
         cpu: 10m
         memory: 30Mi
      requests:
         cpu: 10m
            memory: 30Mi
```

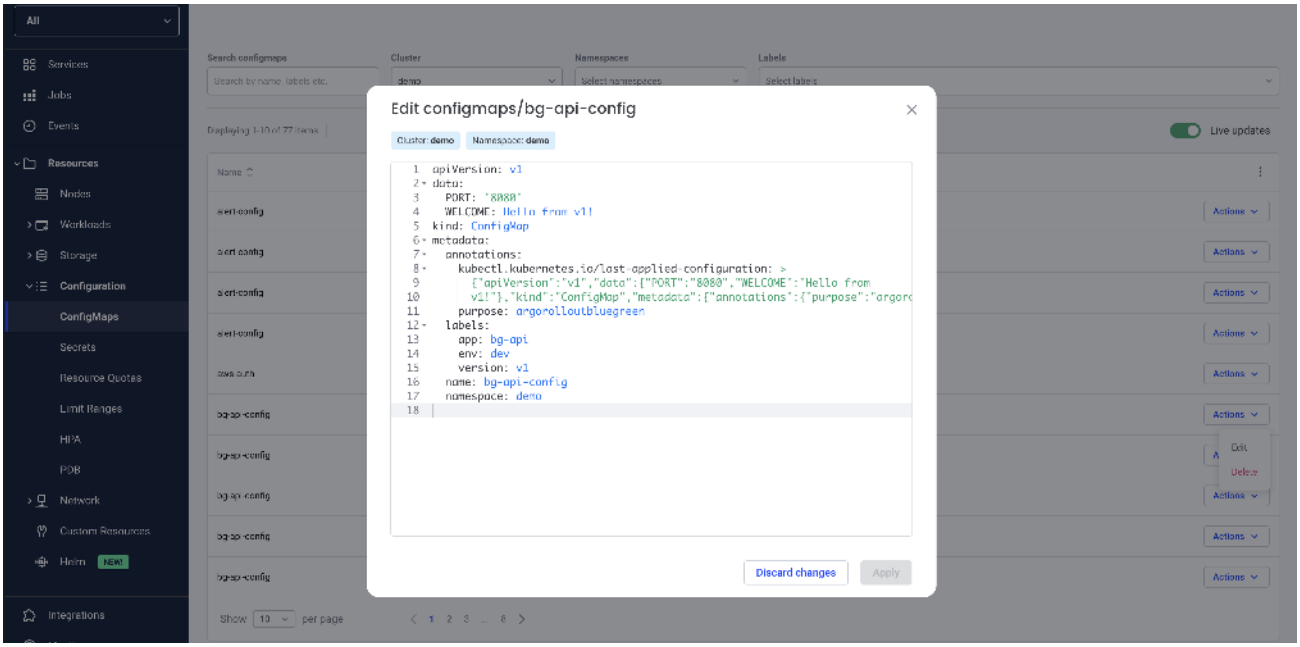
How Komodor's Static Checking Helps Application Configuration

As manual configuration modifications may be the cause of unsuccessful deployments, finding them is crucial. Developers frequently waste time worrying about problems with their code after a failed deployment when the underlying issue was an unapproved modification to the cluster or application.

Using ConfigMap For Application Configs

Users can send configuration changes (from internal tools and infrastructure) and receive them as part of the Komodor Service view thanks to the configuration change API.

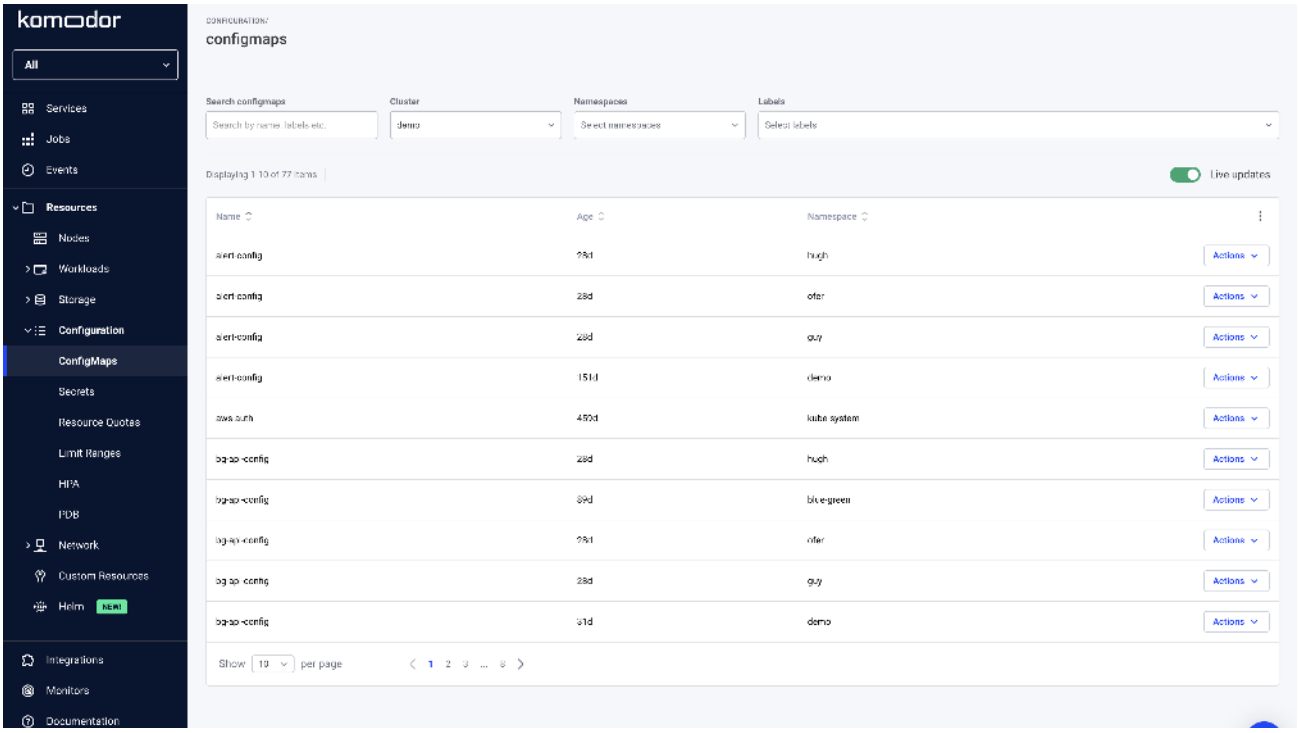
Komodor's beautiful dashboard enables you to easily view and edit ConfigMap resources under the Configuration -> ConfigMaps page:



Detecting ConfigMap Changes With Komodor

Komodor gives you insight into your Kubernetes deployments on a timeline with relevant information such as what changed, what code was pushed, and who pushed it.

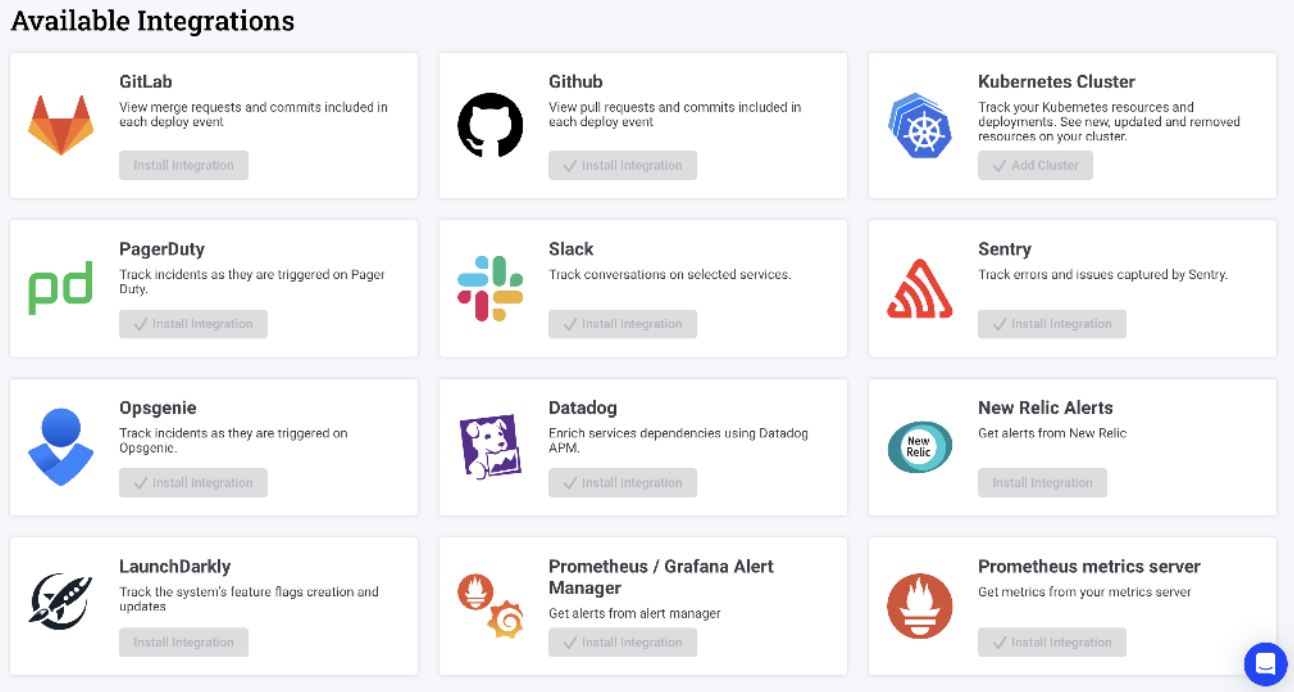
To detect any ConfigMap changes, you can use the Configuration/Config Maps panel to easily see the actual config data with live updates enabled:



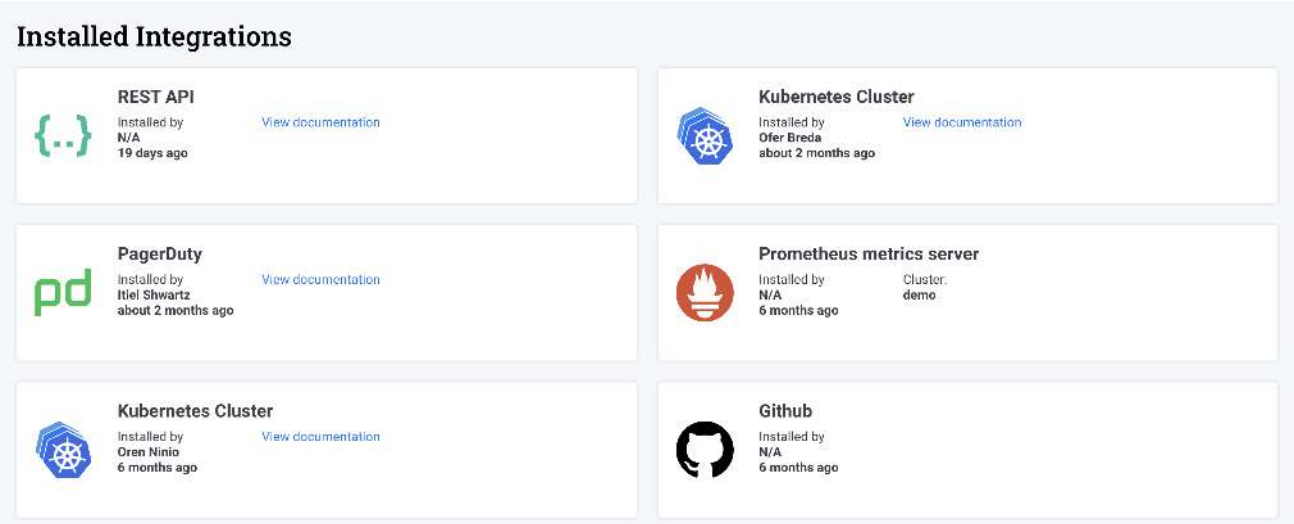
Defining Alerts Using Komodor

Komodor tracks changes across your entire K8s stack, analyzes their ripple effect, and provides you with the context you need to troubleshoot efficiently and independently. Komodor supports various alerting providers using their webhook service.

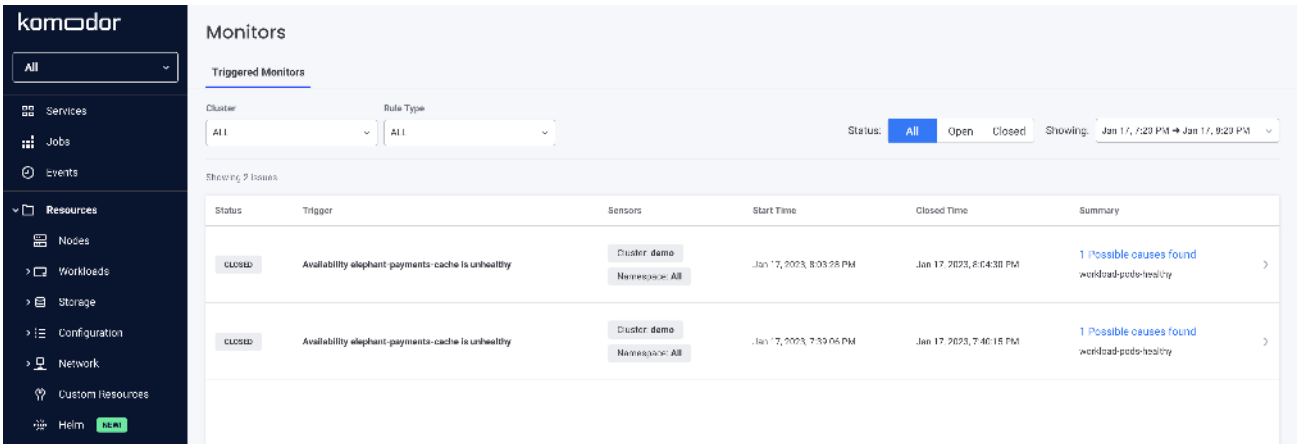
You can jump to available integrations in your dashboard to install any integration:



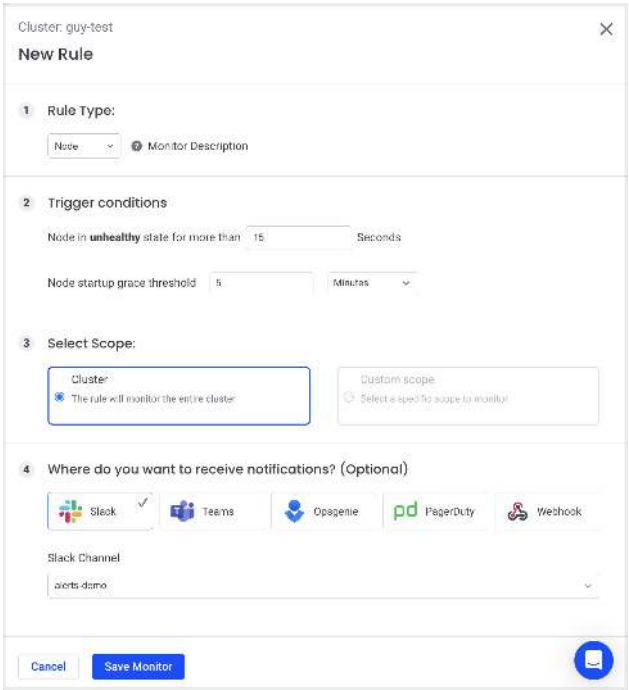
Installing a new integration is quite easy. Simply click the install button on the integrations page:



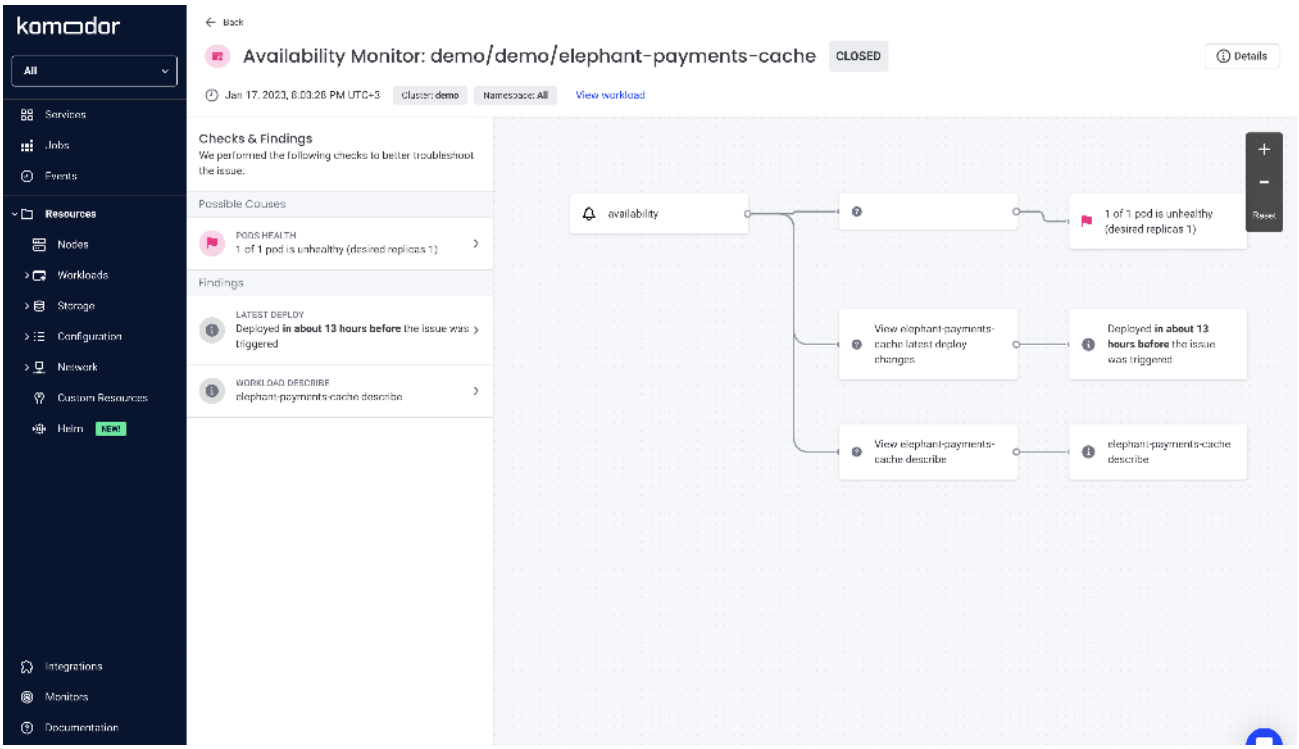
You can also add custom alerts to your monitors:



Just click the “Add Rule” button in the upper right and define your own rule:



You can also check triggered alerts on the Triggered Monitors tab. It shows you possible causes of a failure and is one of the most important features of Komodor to troubleshoot your applications:



Komodor also addresses the issue of alert fatigue with its own guide to managing alerts.

Rolling Back To Previous Versions

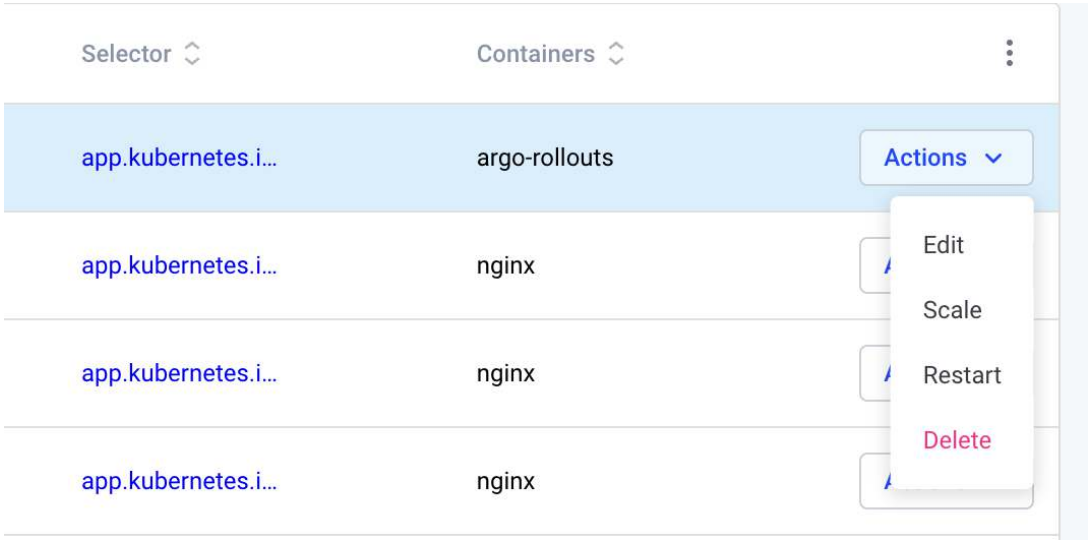
You should have a strategy in place in case you introduce a significant change that breaks production services.

A rollback mechanism is already included in Kubernetes. If the application deployment fails and an application process contains a built-in rollback step, you can quickly return the deployment model to a prior state so that you do not serve unhealthy responses to your customers.

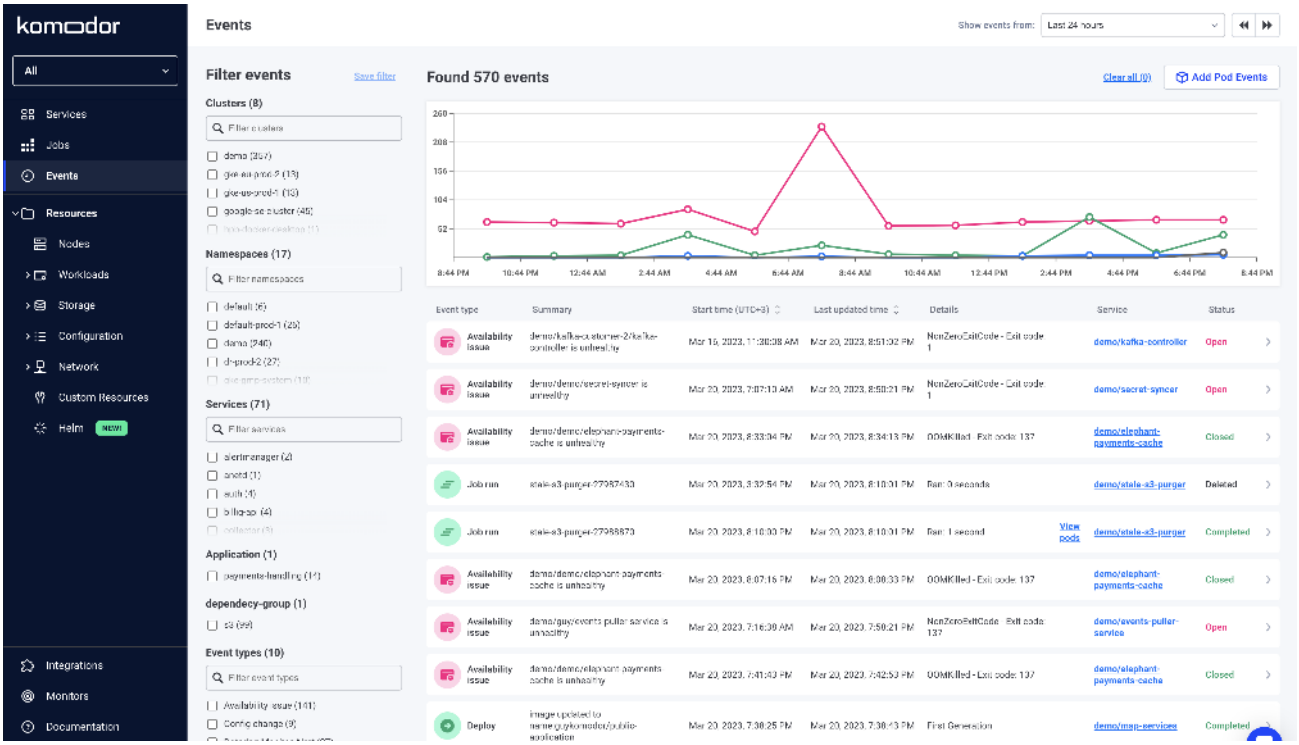
Komodor Actions: Rollback to Previous Version

Every time a Komodor Monitor notices a problem, whether it's with the infrastructure or an app, Komodor playbooks immediately perform a number of checks and offer helpful information for fixing the problem.

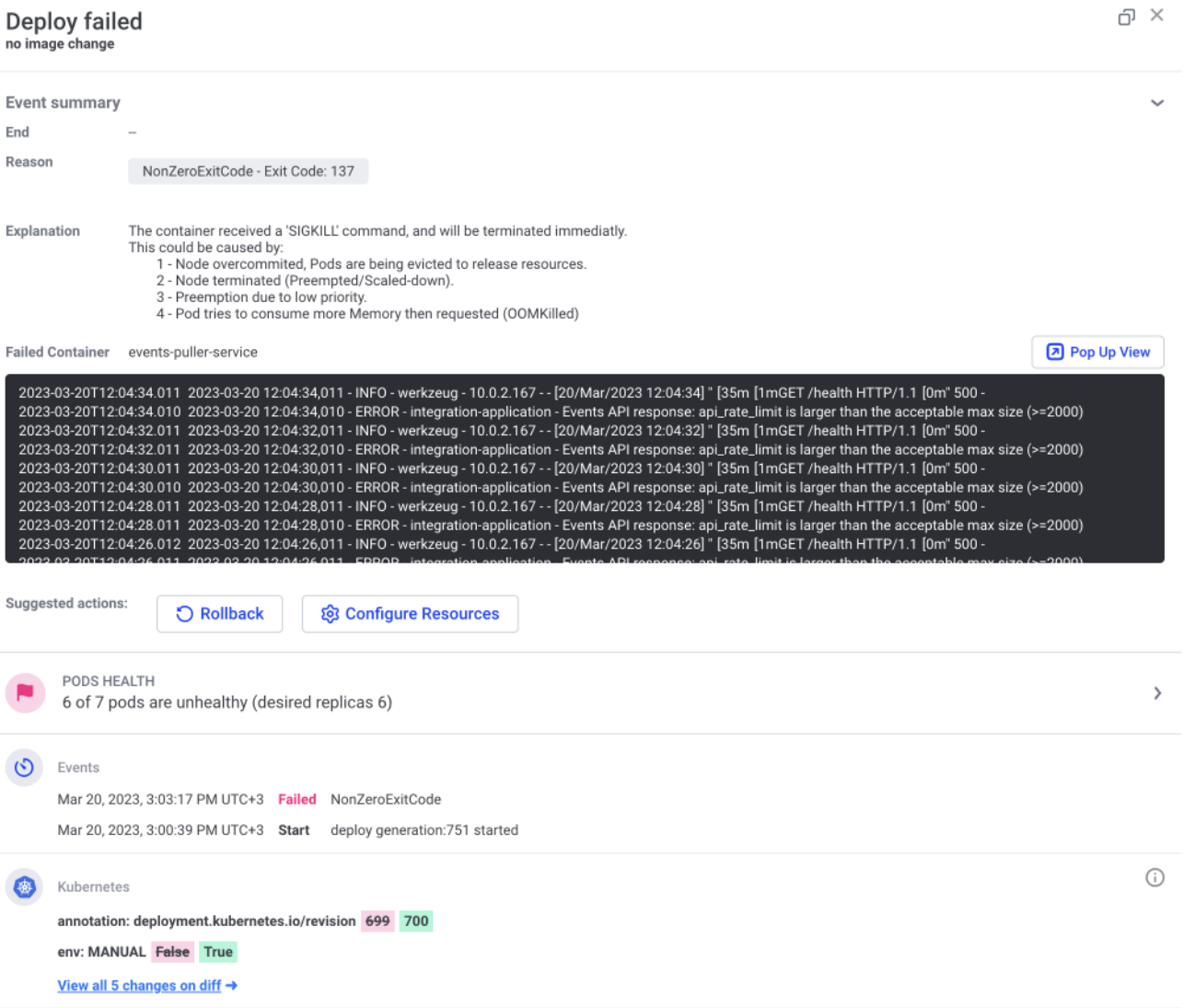
To complete the troubleshooting cycle of detection, investigation, and remediation inside the Komodor platform, it provides all necessary components. Actions available with this new functionality include drain/cordon node, restart, scale, delete, edit resource, and more:



Komodor Events is also helpful to identify your problem and roll back to a previous version:



As you can clearly see below, an error has occurred. By clicking the Rollback button, the deployment will be rolled back to the previous stable version of the application:



Conclusion

We can now troubleshoot fast and independently on a single platform thanks to Komodor's significant simplification of troubleshooting. It provides an end-to-end solution with Kubernetes expertise baked in. You can save hours of precious dev time by using Komodor's powerful features: assisted remediation, automated investigation, and proactive health monitoring.

Turn troubleshooting chaos into clarity by using Komodor. Get started for free today.

Technology Partners

CLOUD NATIVE COMPUTING FOUNDATION · DATADOG · aws · Google Cloud · RANCHER

Get started for free today

START FREE