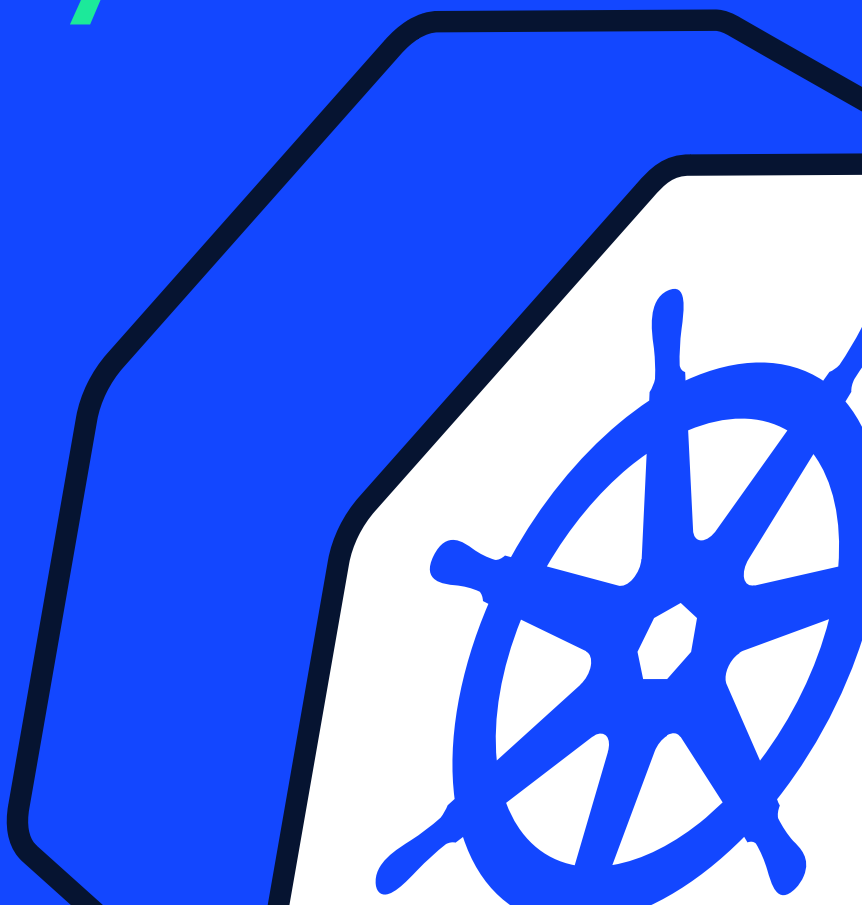
komodor

6 Kubernetes Best Practices to Empower Developers to **Troubleshoot Independently**



komodor.com

6 Kubernetes Best Practices to Empower Developers to Troubleshoot Independently

With great power comes great responsibility. While Kubernetes is an extremely powerful system, with nearly infinite scale and distribution capabilities, it is also a double-edged sword from an operational management perspective. For the most part, managing Kubernetes falls on platform engineering teams (i.e. DevOps and SRE) but there are actually quite a few areas that could and should be shifted left to enable improved troubleshooting and maintenance in the long-term.

Even before deploying to Kubernetes there are some proactive steps that can be taken to democratize the management and maintenance of these systems, and make it possible for ops and devs to troubleshoot quickly and more efficiently.

For this, we've rounded up some hard-earned tips that, when applied from the design phases of your Kubernetes systems, can save much of the troubleshooting pains. By baking these in, from the very beginning, you will be able to distribute the management efforts, and provide a greater sense of ownership through the entire engineering organization.

Hacks of Kindness: How to Shift Your Kubernetes Troubleshooting Left in Practice

Below we're going to take a dive into the quick wins that you can implement right away in your engineering organization, that will make your Kubernetes system easier to troubleshoot for everyone. As with all code, there will always be issues, and things will always break - so you want to make the processes of getting to the root cause as quickly as possible.

1. Stateless-First Design

When Kubernetes was initially conceived, it was built for stateless applications, although it has made a lot of progress with supporting stateful applications. That's why it remains a good practice to build your applications to be stateless from the start when deploying them to Kubernetes. This allows you to remove a lot of the risk around your applications, and also gain the benefits of elasticity.

When you are required to restart your application, it is not dependent upon any external information or data to run the same way as before the restart, making your application a lot easier to manage.

Therefore, applications that rely on external state for initialization and startup will have a lot more difficulty adapting to Kubernetes operations, as this requires significantly more engineering expertise to enable similar capabilities (elegantly restarting after a crash, for example). In many instances, this cannot be done at all.

On the flip side, stateless applications can scale up or down through simple definitions in the code. It also helps with troubleshooting. For example, when K8s detects a problem, it will just restart your application and container forever if something goes wrong, and enable you to resume operations as usual, seamlessly. This type of approach is much more prone to breakage with stateful applications.

2. How to Segregate Environments

Much has been said about managing environments on Kubernetes - a quick search will bring up plenty of great resources. But for the TL;DR, a commonly recommended practice is to create an environment for each stage of development: development, QA, staging, production.

With Kubernetes, it is also possible to separate environments logically using the namespaces resource, which segregates environments by 'names' that point to specific objects in isolation, while still sharing the same underlying resources and infrastructure.

This also maintains these different environments on the same shared resources, such as nodes, meaning that if one of the environments is using too many system resources - such as CPU or memory, your production environment will have less to work with.

One way to work around this is with node taints and tolerations, which enables you, according to your configuration, to let the K8s scheduler decide which node it should live in. This makes it possible to choose which node or specific environment lives on which machine, based on the amount of resources it usually consumes.

For someone just getting started with Kubernetes, separating the cluster entirely is likely a better practice. While it is more expensive, it will deliver greater safety overall, in addition to being easier to launch and manage.

A

```
Contexts(all)[10]
NAME↑                                                    CLUSTER
Mickael@test-cluster.us-east-2.eksctl.io                 test-cluster.us-east-2.eksctl.io
arn:aws:eks:us-east-1:63…  …34:cluster/buildkite-cluster  arn:aws:eks:us-east-1:63  …34:cluster/buildkite-cluster
arn:aws:eks:us-east-1:63…  …34:cluster/ci-cluster        arn:aws:eks:us-east-1:63  …34:cluster/ci-cluster
arn:aws:eks:us-east-1:63…  …34:cluster/demo             arn:aws:eks:us-east-1:63  …34:cluster/demo
arn:aws:eks:us-east-1:63…  …34:cluster/komodor-staging-eks arn:aws:eks:us-east-1:63  …34:cluster/komodor-staging-eks
arn:aws:eks:us-east-1:63…  …34:cluster/main(*)          arn:aws:eks:us-east-1:63  …34:cluster/main
arn:aws:eks:us-east-2:63…  …34:cluster/test-cluster     arn:aws:eks:us-east-2:63  …34:cluster/test-cluster
gke_komodor-load-test_us-central1-c_load-test            gke_komodor-load-test_us-central1-c_load-test
gke_komodor-load-test_us-central1-c_load-test-1          gke_komodor-load-test_us-central1-c_load-test-1
gke_komodor-load-test_us-central1_load-test-1            gke_komodor-load-test_us-central1_load-test-1
```

B

```
Namespaces(all)[7]
NAME↑                STATUS
all+                 Active
default              Active
komodor              Active
komodor-staging      Active
kube-node-lease      Active
kube-public          Active
kube-system          Active
```

You can either create an environment for each development stage (A) or separate using the namespaces resource (B)

3. Proper YAML Management (AKA Your K8s Deployment Manifest)

When working with YAML files, including helpful metadata can significantly simplify troubleshooting in the long run.

Some good practices include setting the right labels and annotations, environment variables, secrets, and config maps that point to the proper objects and volumes, configuring liveness and readiness probes. In this way, K8s will know when your app is healthy and ready to accept traffic, or otherwise alert you when there is an issue.

Other important data includes cloud-specific configurations (e.g. node taints, tolerations, readiness gates etc.) depending on your cloud provider you choose to use.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: data-processing-api
  labels:
    app.kubernetes.io/stack: python
    app.kubernetes.io/team: data
    app.kubernetes.io/group: backend
    app.kubernetes.io/db: postgresql
    app.kubernetes.io/name: data-processing-api
    app.kubernetes.io/version: "1.4.7"
    app.kubernetes.io/commit: 76959a71a82240348b8e818d97a085d45d3ad8d6
```

```yaml
        readinessProbe:
            httpGet:
              path: /healthz/ready
              port: 8080
            initialDelaySeconds: 3
            periodSeconds: 5
            failureThreshold: 3
            successThreshold: 1
```

YAML file containing labels and readiness probe

4. Kubernetes-Aware Logging

With K8s' inherent complexity, aggregating and collecting logs from a multitude of containers, clusters, and machines is a daunting operation.

Do "future you" a favor and take some proactive actions to simplify troubleshooting processes by making sure to tag and label your logs properly. Some good practices include:

- the proper service name (and not rely on the pod names that are volatile)
- the version, and;
- cluster environment information

Also note that there are many K8s-specific tags that define the application's production or runtime environment. This can help with troubleshooting and understanding where an issue originated from.

So for example, if you have an application with 500 replicas, in one of these replicas an error occurs and you'd like to try and get to the root cause, understanding which container this came from will be a nearly impossible feat if you don't properly tag and label your containers and logs.

5. Invest in Proper Monitoring

Prometheus with Grafana have become very popular ways to monitor applications on Kubernetes due to being open source projects as well. While open source projects provide a lot of flexibility and are built for customizability, they do come with a high learning curve.

It's often difficult to get started with configuring them properly to receive the relevant and actionable information for monitoring your cluster. This can come with the cost of not having accurate information in real-time when there is an issue, or can even cause you to miss critical failures.

There are commercial options that provide extensive documentation and a lot of features out of the box to help get started rapidly, such as Datadog and New Relic.

For instance, with the Datadog agent on your cluster, you'll get an integration with your cloud provider out of the box the moment you deploy, and be greeted with a default dashboard with useful metrics about your cluster, as well as relevant data about your cloud provider.

It's absolutely possible to get your open source tooling to do this, but takes more effort and familiarity with configuring this tooling.

Tooling aside, the three main things you'll want to start with monitoring on K8s are:

- Resources - CPU / Memory Usage
- Container Status - Up / Down / Errors / Probe Data / Restart count
- Application Metrics - APMs - Per-Request or Action Metrics / Latency / Stack Traces / Custom Metadata

The first two bullets provide critical information about your cluster, the third point and likely the most important one - APMs, provide you business critical information about your application

Due to Kubernetes' innate scalability, if you go from a couple to 10s to hundreds of servers, running hundreds to thousands of applications, finding the root cause of an issue is going to make you pull some haris.

At a certain scale, the manual approach simply stops working and that's when you'll need the help of these monitoring tools to help you to detect, alert, and understand the business logic of your applications.

6. Knowledge Sharing & Transparency

When it comes to applications running on K8s there are certainly core concepts that developers should be aware of, and this is where your DevOps teams can help. DevOps teams can and should empower development teams to learn about the platforms and environments on which they deploy their applications. Knowing about the platform that the application lives on is critical to help developers respond to incidents more quickly.

Understanding the nuances of containers and pod configurations, health checks, cluster orchestration, load balancing and more helps developers troubleshoot issues rapidly and effectively without escalating to the DevOps team when something goes wrong.

Takeaway: the more you empower and entrust to your developers, the more efficiently your Kubernetes systems will run end to end.

Bringing it all Together

Kubernetes is often perceived as DevOps & SRE closed quarters. However, the truth is, that there are actually quite a few practices that can be implemented as early as the design phases by DevOps engineers to empower developers to be much more autonomous with troubleshooting down the line.

DevOps can provide developers a deeper understanding of the underlying infrastructure, even from a high level perspective, so that they can be more proactive and independent with error handling - something that can prove particularly valuable with 2AM pages and alerts, for everyone involved.

In addition, proper mapping of your system via metadata, tagging, labels and other good practices, provides the benefit of having the right information that is easy to understand in real-time. The value of such insights grows hand-in-hand with the scale of your operation, and becomes a must-have for troubleshooting large-scale deployments.

We hope these tips will help developers and DevOps engineers alike, when designing their applications and deployments, to be highly optimized for Kubernetes systems.

Produced by Komodor. September, 2021.

What is Komodor

Komodor takes the complexity out of K8s troubleshooting, providing all of the tools you need to troubleshoot with confidence. For each service, Komodor displays a coherent view, including relevant deployments, config changes, and alerts.

- Full activity timeline with data insights that are most relevant for solving issues
- A complete drill-down to your K8s diff
- Easily understand cross-service changes

Free your devs-on-call to focus on their daily strategic work!

LEARN MORE