Exploring Logs Through kubectl

In this lesson, we will explore logs through the kubectl command and determine whether it's effective in knowing which pods are misbehaving or not.

WE'LL COVER THE FOLLOWING

Overview of logging

The first contact most people have with logs in Kubernetes is through `kubectl`. It is almost unavoidable not to use it.

As we're learning how to tame the Kubernetes beast, we are bound to check logs when we get stuck. In Kubernetes, the term "logs" is reserved for the output produced by our and third-party applications running inside a cluster. However, those exclude the events generated by different Kubernetes resources. Even though many would call them logs as well, Kubernetes separates them from logs and calls them events. I'm sure that you already know how to retrieve logs from the applications and how to see Kubernetes events. Nevertheless, we'll explore them briefly here as well since that will add relevance to the discussion we'll have later on. I promise to keep it short, and you are free to skip this section if a brief overview of logging and events baked into Kubernetes is too basic for you.

Generate logs

We'll install the already familiar `go-demo-5` application. It should generate enough logs for us to explore them. Since it consists of a few resources, we are

bound to create some Kubernetes events as well.

Off we go.

```
GD5_ADDR=go-demo-5.$LB_IP.nip.io

echo $GD5_ADDR

kubectl create namespace go-demo-5

helm install go-demo-5 \
    https://github.com/vfarcic/go-demo-5/releases/download/0.0.1/go-demo-5
-0.0.1.tgz \
    --namespace go-demo-5 \
    --set ingress.host=$GD5_ADDR

kubectl -n go-demo-5 \
  rollout status deployment go-demo-5

curl "http://$GD5_ADDR/demo/hello"
```

We rolled out `go-demo-5` and sent a `curl` request to confirm that it is indeed working.

> 🔍 The outputs and screenshots in this chapter are taken from minikube, except inside the sections dedicated exclusively to GKE, EKS, and AKS. There might be slight differences between what you see here and what you can observe on your screen.

Retrieve the events

To see "logs" generated by Kubernetes and limited to a specific resource, we need to retrieve the events.

```
kubectl -n go-demo-5 \
  describe sts go-demo-5-db
```

The **output**, limited to the messages in the `Events` section, is as follows.

```
...
Events:
```

```
... Message
... -------
... create Claim go-demo-5-db-go-demo-5-db-0 Pod go-demo-5-db-0 in Statefu
lSet go-demo-5-db success
... create Pod go-demo-5-db-0 in StatefulSet go-demo-5-db successful
... create Claim go-demo-5-db-go-demo-5-db-1 Pod go-demo-5-db-1 in Statefu
lSet go-demo-5-db success
... create Pod go-demo-5-db-1 in StatefulSet go-demo-5-db successful
... create Claim go-demo-5-db-go-demo-5-db-2 Pod go-demo-5-db-2 in Statefu
lSet go-demo-5-db success
... create Pod go-demo-5-db-2 in StatefulSet go-demo-5-db successful
```

The events you see in front of you are, in a way, Kubernetes logs generated by, in this case, the `go-demo-5-db` StatefulSet.

Events are insufficient

While those events are useful, they are often insufficient. More often than not, we do not know in advance where the problem is. If one of our Pods misbehaves, the cause might be in that Pod, but it might also be in the ReplicaSet that created it, the Deployment that created the ReplicaSet, the node got detached from the cluster, or it might be something completely different.

> 🔍 For any but the smallest systems, going from one resource to another and from one node to another to find the cause of an issue is anything but practical, reliable, and fast.

Simply put, looking at events by describing a resource is not the way to go and we need to find an alternative. But, before we do that, let's see what happens with logs from applications.

We deployed a few replicas of the `go-demo-5` API and a few replicas of the MongoDB. How can we explore their logs if we suspect that there is a problem with one of them? We can execute `kubectl logs` command like the one that follows.

```
kubectl -n go-demo-5 \
    logs go-demo-5-db-0 -c db
```

The **output** shows the logs of the `db` container inside the `go-demo-5-db-0` Pod.

While the previous output is limited to a single container and a single Pod, we can use labels to retrieve logs from multiple Pods.

```
kubectl -n go-demo-5 \
    logs -l app=go-demo-5
```

Which pods are misbehaving?

This time, the output comes from all the Pods with the label `app` set to `go-demo-5`. We broadened our results, and that is often all we need. If we know that there is something wrong with, let's say, `go-demo-5` Pods, we need to figure out whether the issue is present in multiple Pods or if it is limited to a single one. While the previous command allowed us to broaden our search, if there were something suspicious in those logs, we would not know where it comes from. Retrieving logs from multiple Pods does not get us any closer to knowing which Pods are misbehaving.

Using labels is still very limiting. They are by no means a substitute for more complex querying. We might need to filter the results based on timestamps, nodes, keywords, and so on. While we could accomplish some of those things with additional `kubectl logs` arguments and creative usage of `grep`, `sed`, and other Linux commands, this approach to retrieving, filtering, and outputting logs is far from optimum.

> 🔍 More often than not, the `kubectl logs` command does not provide us with enough options to perform anything but the simplest retrieval of logs.

We need a few things to improve our debugging capabilities. We need a potent query language that will allow us to filter log entries, we need sufficient information about the origin of those logs, we need queries to be fast, and we need access to logs created in any part of the cluster.

Q

Retrieving logs from multiple Pods does not get us any closer to knowing which Pods are misbehaving

which Pods are misbehaving.

In the next lesson, we'll try to accomplish that, and a few other things, by setting up a centralized logging solution.