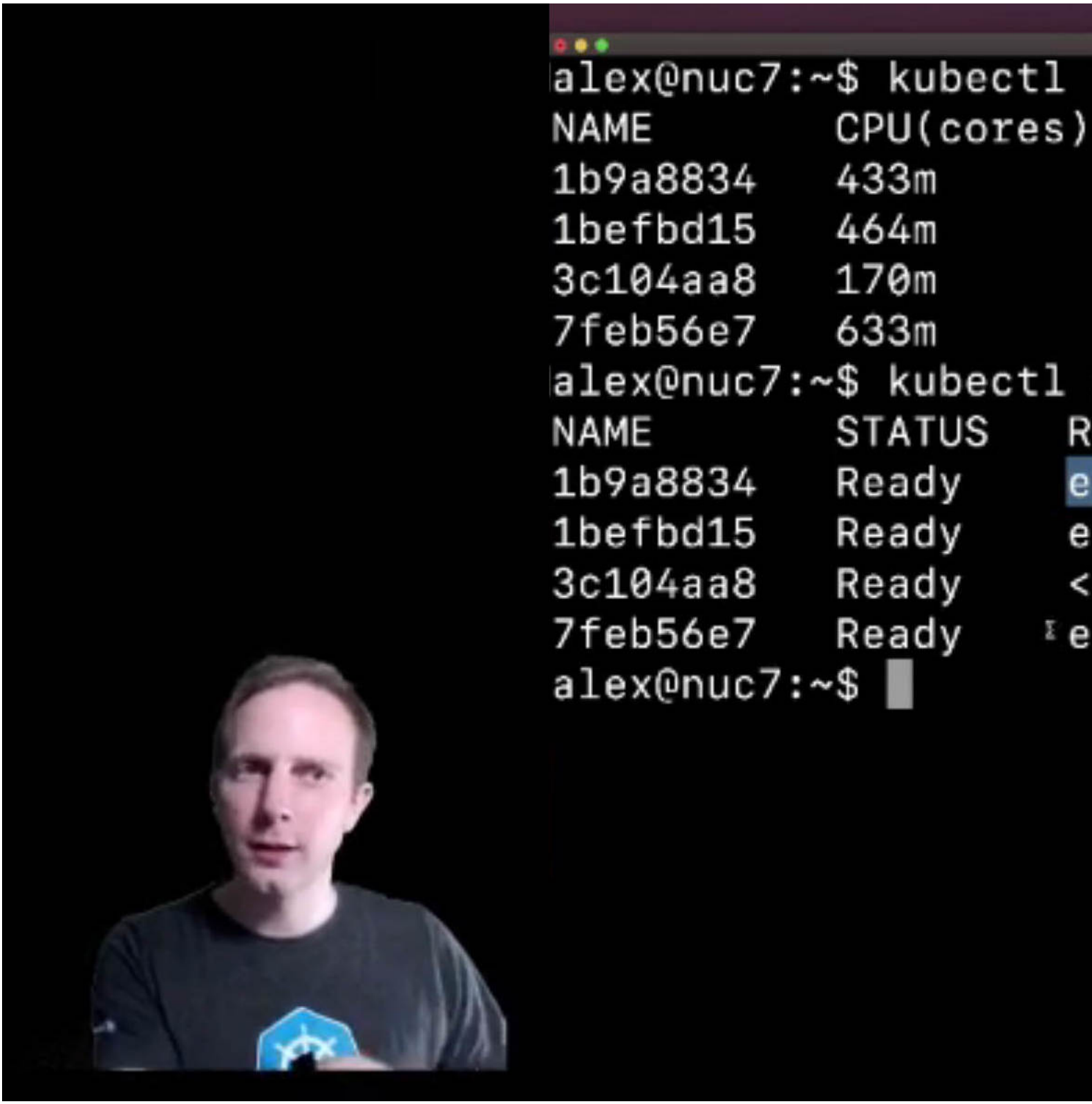
How to Troubleshoot Applications on Kubernetes

This article was fetched from an rss feed(01/06/2022)



This is my guide to troubleshooting applications deployed on a Kubernetes cluster. It's based upon my experience with the technology since I switched over from Docker in early 2017.

There are many third-party tools and techniques available, but I want to focus in on things that you'll find on almost every computer, or CLIs that you can download quickly for MacOS, Windows or Linux.

> What if you're a Kubernetes expert?
>
> If you're already supporting clusters for customers, or developing software, you may know some of these. So perhaps you may find this content useful to send over to your own users, and colleagues.
>
> In any case, stay with me and see whether there's a new flag or technique you may pick up. You may also like some of the more advanced supplementary material that I'm linking to throughout the post.

Got a comment, question or suggestion? Let's talk on Twitter

Is it there?

When your application isn't working, you may want to check that all its resources have been created.

The first command you learn is probably going to be "kubectl get pods"

But remember, that Kubernetes supports various namespaces in order to segregate and organise workloads. What you've installed may not be in the default namespace.

Here's how can change the namespace to look into the "kube-system" or "openfaas-fn" namespace for instance:

```
kubectl get pods --namespace kube-system
kubectl get pods -n openfaas
```

You can query all of the namespaces available with:

```
kubectl get pods --all-namespaces
kubectl get pods -A
```

The `-A` flag was added to kubectl in the 2-3 years, and means you can save on typing.

Now of course, Pods are just one of the things we care about. The above commands can also take other objects like `Service`, `Deployment`, `Ingress` and more.

Why isn't it working?

When you ran "kubectl get", you may have seen your resource showing as `0/1`, or even as `1/1` but in a crashing or errored state.

How do we find out what's going wrong?

> You may be tempted to reach for "kubectl logs", but this only shows logs from applications that have started, if your pod didn't start, then you need to find out what's preventing that.

You're probably running into one of the following:

- The image can't be pulled
- There's a missing volume or secret
- No space in the cluster for the workload
- Taints or affinity rules preventing the pod from being scheduled

Now `kubectl get events` on its own isn't very useful, because all the rows come out in what appears to be a random order. The fix is something you'll have to get tattooed somewhere prominent, because there's no shorthand for this yet.

```
kubectl get events \
  --sort-by=.metadata.creationTimestamp
```

This will print out events in the default namespace, but it's very likely that you're working in a specific namespace, so make sure to include the `--namespace` or `--all-namespaces` flag:

```
kubectl get events \
  --sort-by=.metadata.creationTimestamp \
  -A
```

Events are not just useful for finding out why something's not working, they also show you how pods are pulled, scheduled and started on a cluster.

Add "--watch" or "-w" to the command to watch an OpenFaaS function being created for instance:

```
kubectl get events \
  --sort-by=.metadata.creationTimestamp \
  -n openfaas-fn
```

And with the watch added on:

```
kubectl get events \
  --sort-by=.metadata.creationTimestamp \
  -n openfaas-fn \
  --watch
```

```
LAST SEEN   TYPE      REASON              OBJECT                          MESSAGE
0s          Normal    Synced              function/figlet                 Function synced successfully
0s          Normal    ScalingReplicaSet   deployment/figlet               Scaled up replica set figlet-5485896b55 to 1
1s          Normal    SuccessfulCreate    replicaset/figlet-5485896b55    Created pod: figlet-5485896b55-j9mbd
0s          Normal    Scheduled           pod/figlet-5485896b55-j9mbd     Successfully assigned openfaas-fn/figlet-5485896b55-j9mbd to k3s-pi
0s          Normal    Pulling             pod/figlet-5485896b55-j9mbd     Pulling image "ghcr.io/openfaas/figlet:latest"
0s          Normal    Pulled              pod/figlet-5485896b55-j9mbd     Successfully pulled image "ghcr.io/openfaas/figlet:latest" in 632.059147ms
0s          Normal    Created             pod/figlet-5485896b55-j9mbd     Created container figlet
0s          Normal    Started             pod/figlet-5485896b55-j9mbd     Started container figlet
```

There's actually quite a lot that's going on in the above events. You can then run something like `kubectl scale -n openfaas-fn deploy/figlet --replicas=0` to scale it down and watch even more events getting generated as the pod is removed.

Now, there is a new command called `kubectl events` which you may also want to look into, however my kubectl version was too old, and it's only an alpha feature at present.

You can upgrade kubectl using arkade whether you're on Windows, MacOS or Linux:

```
arkade get kubectl@v1.24.0
$HOME/.arkade/bin/kubectl alpha events -n openfaas-fn
```

Now as I understand it, this new command does order the events, to keep an eye on how it progresses to see if the Kubernetes community promote it to generally available (GA) status or not.

It starts, but doesn't work

So the application starts up with 1/1 pods, or starts up then keeps crashing, so you're seeing a lot of restarts when you type in:

```
kubectl get pod
NAME                                       READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-54d8b558d4-59lj2   1/1     Running   5          114d
```

This is probably where the age old favourite "kubectl logs" comes in.

Most people do not work with Pods directly, but create a Deployment, which in turn creates a number of Pods depending on the replicas field. That's true of the way I've installed ingress-nginx, which you can see has restarted 5 times. It's now been running for 114 days or nearly 4 months.

```
kubectl logs ingress-nginx-controller-54d8b558d4-59lj2|wc -l
```

Wow. I just saw 328 lines pass by, and that was too much information.

Let's filter down to just the past 10 lines, and who needs to be typing in pod names? That's no longer necessary in Kubernetes.

Find the deployment name and use that instead:

```
kubectl get deploy

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
ingress-nginx-controller   1/1     1            1           114d


kubectl logs deploy/ingress-nginx-controller \
  --tail 10
```

That's better, just the past 10 lines show up. But what if I want to monitor the logs as the application runs?

Well `kubectl logs` doesn't have a `--watch` flag, but it does have a `--follow` (`-f`) flag that we can use instead:

```
kubectl logs deploy/ingress-nginx-controller \
  --tail 10 \
  --follow
```

Pods can have more than one container, and when they do, it means more typing, because we have to pick one of them.

Did you notice that I added an extra flag here? `--container`

```
kubectl logs -n openfaas deploy/gateway

error: a container name must be specified for pod gateway-7c96d7cbc4-d47wh, choose one of: [gateway operator]
```

We can also filter down to logs that were emitted during a specific time-frame:

```
kubectl logs -n openfaas deploy/gateway \
  --container gateway \
  --since 30s
```

You can also enter a specific date in RFC3339 format by using the `--since-time` flag.

There are too many pods!

When your deployment has more than one replica, then the `kubectl logs` command will select only one of them, and you'll miss out on potentially important information.

```
kubectl logs -n openfaas deploy/queue-worker
Found 2 pods, using pod/queue-worker-755598f7fb-h2cfx
2022/05/23 15:24:32 Loading basic authentication credentials
2022/05/23 15:24:32 Starting Pro queue-worker. Version: 0.1.5   Git Commit: 8dd99d2dc1749cfcf1e828b13fe5fda9c1c921b6    Ack wait: 60s   Max inflight: 25
2022/05/23 15:24:32 Initial retry delay: 100ms  Max retry delay: 10s    Max retries: 100
```

We can see that there are two pods:

```
kubectl get pods -n openfaas|grep queue
queue-worker-755598f7fb-h2cfx           1/1     Running   1          28d
queue-worker-755598f7fb-ggkn9           1/1     Running   0          161m
```

So kubectl only attached us to the oldest pod, not the newer one created just over 2 hours ago.

The fix here is to either use a label selector, which matches common labels across both pods, or to use a third-party tool.

```
kubectl logs -n openfaas --l app=queue-worker
```

Using a label selector means that we will not get new pods created since we stat the command, so a third party tool is going to be better for anything that can auto-scale or crash and restart.

Recently, I would have recommended kail, but whilst working with an OpenFaaS customer, we discovered that the maintainer doesn't cater to Window users.

Instead, we switched to a very similar tool called stern.

You can install this tool with arkade onto Windows, MacOS and Linux:

```
arkade get stern
```

But can you `curl` it?

Whilst Kubernetes can run batch jobs and background tasks, most of the time, you will see teams deploying websites, microservices, APIs and other applications with HTTP or TCP endpoints.

So a good question to ask is "can I curl it?"

Accessing services requires its own article, so I wrote that up a few weeks ago: A Primer: Accessing services in Kubernetes

Why have we run out of RAM already?

The metrics-server project is an add-on for Kubernetes that can quickly show you how much RAM and CPU is being used by Pods across your cluster. It'll also show you how well balanced the workloads are across nodes.

Here's a complete installation of OpenFaaS on a 2x node Raspberry Pi cluster. It also includes a bunch of extras like OpenFaaS Pro, Grafana, an inlets tunnel for connectivity, the new UI dashboard and the metrics-server itself. I'm also running multiple replicas of some services like the queue-worker which has two separate pods.

```
kubectl top pod -A
NAMESPACE     NAME                                      CPU(cores)   MEMORY(bytes)
default       ingress-nginx-controller-54d8b558d4-59lj2   4m           96Mi
grafana       grafana-5bcd5dbf74-rcx2d                  1m           22Mi
kube-system   coredns-6c46d74d64-d8k2z                  5m           10Mi
kube-system   local-path-provisioner-84bb864455-wn6v5   1m           6Mi
kube-system   metrics-server-ff9dbcb6c-8jqp6            36m          13Mi
openfaas      alertmanager-5df966b478-rvjxc             2m           6Mi
openfaas      autoscaler-6548c6b58-9qtbw                2m           4Mi
openfaas      basic-auth-plugin-78bb844486-gjwl6        4m           3Mi
openfaas      dashboard-56789dd8d-dlp67                 0m           3Mi
```

```
openfaas      gateway-7c96d7cbc4-d47wh                  12m          24Mi
openfaas      inlets-portal-client-5d64668c8d-8f85d      1m           5Mi
openfaas      nats-675f8bcb59-cndw8                     2m           12Mi
openfaas      pro-builder-6ff7bd4985-fxswm              1m           117Mi
openfaas      prometheus-56b84ccf6c-x4vr2               10m          28Mi
openfaas      queue-worker-6d4756d8d9-km8g2             1m           2Mi
openfaas      queue-worker-6d4756d8d9-xgl8w             1m           2Mi
openfaas-fn   bcrypt-7d69d458b7-7zr94                   12m          16Mi
openfaas-fn   chaos-fn-c7b647c99-f9wz7                  2m           5Mi
openfaas-fn   cows-594d9df8bc-zl5rr                     2m           12Mi
openfaas-fn   shasum-5c6cc9c56c-x5v2c                   1m           3Mi
```

Now we can see how well the pods are balanced across machines:

```
kubectl top node
NAME          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k3s-agent-1   236m         5%     908Mi           23%
k3s-pi        528m         13%    1120Mi          14%
```

So we actually have plenty of headroom to deploy some more workloads.

Now you cannot add any kind of "--watch" or "--follow" flag to this command, so if you want to watch it whilst you scale some functions or deploy a bunch of new pods, you need to use a bash utility like "watch".

Try this for example:

```
# Terminal 1
watch "kubectl top pod -A"

# Terminal 2
watch "kubectl top node"
```

The metrics-server is an optional add-on, which you can install with arkade or helm:

```
arkade install metrics-server
```

Have you turned it off and on again?

It's tragic, but true, that turnings things off and on again can fix many errors that we run into on a daily basis.

Restarting a deployment in Kubernetes may fix issues due to forcing your code to reconnect to services, pull down an updated image, or just release memory and database connections.

The command you're looking for is `kubectl rollout restart`.

Let's restart the 2x OpenFaaS queue-workers, whilst watching the logs with stern, and the events from the namespace in another window.

```
# Terminal 1
kubectl get events \
  -n openfaas
  --sort-by=.metadata.creationTimestamp \
  --watch

# Terminal 2
stern -n openfaas queue-worker.* --since 5s

# Terminal 3
kubectl rollout restart \
  -n openfaas deploy/queue-worker
```

> Note the syntax for stern is a regular expression, so it'l match on anything that starts with "queue-worker" as a prefix. The `--since 5s` is similar to what we used with `kubectl logs`, to keep what we're looking at recent.

So we see our two new pods show up in stern, something that `kubectl logs` would not be able to do for us:

```
queue-worker-6d4756d8d9-km8g2 queue-worker 2022/06/01 10:47:27 Connect: nats://nats.openfaas.svc.cluster.local:4222
queue-worker-6d4756d8d9-xgl8w queue-worker 2022/06/01 10:47:30 Connect: nats://nats.openfaas.svc.cluster.local:4222
```

And the events for the queue-worker show the new pods being created and the older ones being removed from the cluster.

```
LAST SEEN   TYPE     REASON              OBJECT                                  MESSAGE
69s         Normal   Pulling             pod/queue-worker-6d4756d8d9-xgl8w        Pulling image "ghcr.io/openfaasltd/queue-worker:0.1.5"
68s         Normal   Pulled              pod/queue-worker-6d4756d8d9-xgl8w        Successfully pulled image "ghcr.io/openfaasltd/queue-worker:0.1.5" in 597
68s         Normal   Created             pod/queue-worker-6d4756d8d9-xgl8w        Created container queue-worker
68s         Normal   Started             pod/queue-worker-6d4756d8d9-xgl8w        Started container queue-worker
67s         Normal   ScalingReplicaSet   deployment/queue-worker                 Scaled down replica set queue-worker-755598f7fb to 0
67s         Normal   SuccessfulDelete    replicaset/queue-worker-755598f7fb      Deleted pod: queue-worker-755598f7fb-h2cfx
67s         Normal   Killing             pod/queue-worker-755598f7fb-h2cfx        Stopping container queue-worker
```

What we didn't talk about

I need to check on applications for two reasons. The first is that I actually write software for Kubernetes such as openfaas and inlets-operator. During development, I need most of the commands above to check when things go wrong, or to see output from changes I've made. The second reason is that my company supports OpenFaaS and inlets users in production. Supporting users remotely can be challenging, especially if they are not used to troubleshooting applications on Kubernetes.

There are so many things that need to be checked in a distributed system, so for OpenFaaS users, we wrote them all down in a Troubleshooting guide and you'll see some of what we talked about today covered there.

In my experience, application-level metrics are essential to being able to evaluate how well a service is performing. With very little work, you can record Rate, Error and Duration (RED) for your service, so that when the errors are more subtle, you can start to understand what may be going on.

Metrics are beyond the scope of this article, however if you'd like to get some experience, in my eBook Everyday Go I show you how to add metrics to a Go HTTP server and start monitoring it with Prometheus.

> [Adding Prometheus metrics to a HTTP server in Everyday Go]

You can learn more about how we use Prometheus in OpenFaaS in this talk from KubeCon: [How and Why We Rebuilt Auto-scaling in OpenFaaS with Prometheus]

I see Kubernetes as a journey and not a destination. I started my walk in 2017, and it was very difficult for me at the time. If, like me, you now have several years under your belt, try to have some empathy for those who are only starting to use the technology now.

You may also like these two free courses that I wrote for the Linux Foundation/CNCF:

- [Kubernetes at the Edge with K3s]
- [Introduction to Serverless on Kubernetes]

> Kubernetes, K3s, OpenFaaS, Docker, Prometheus, Grafana and many other Cloud Native tools are written in Go, learn what you need to know with [my eBook Everyday Golang]

Got a comment, question or suggestion? [Let's talk on Twitter]