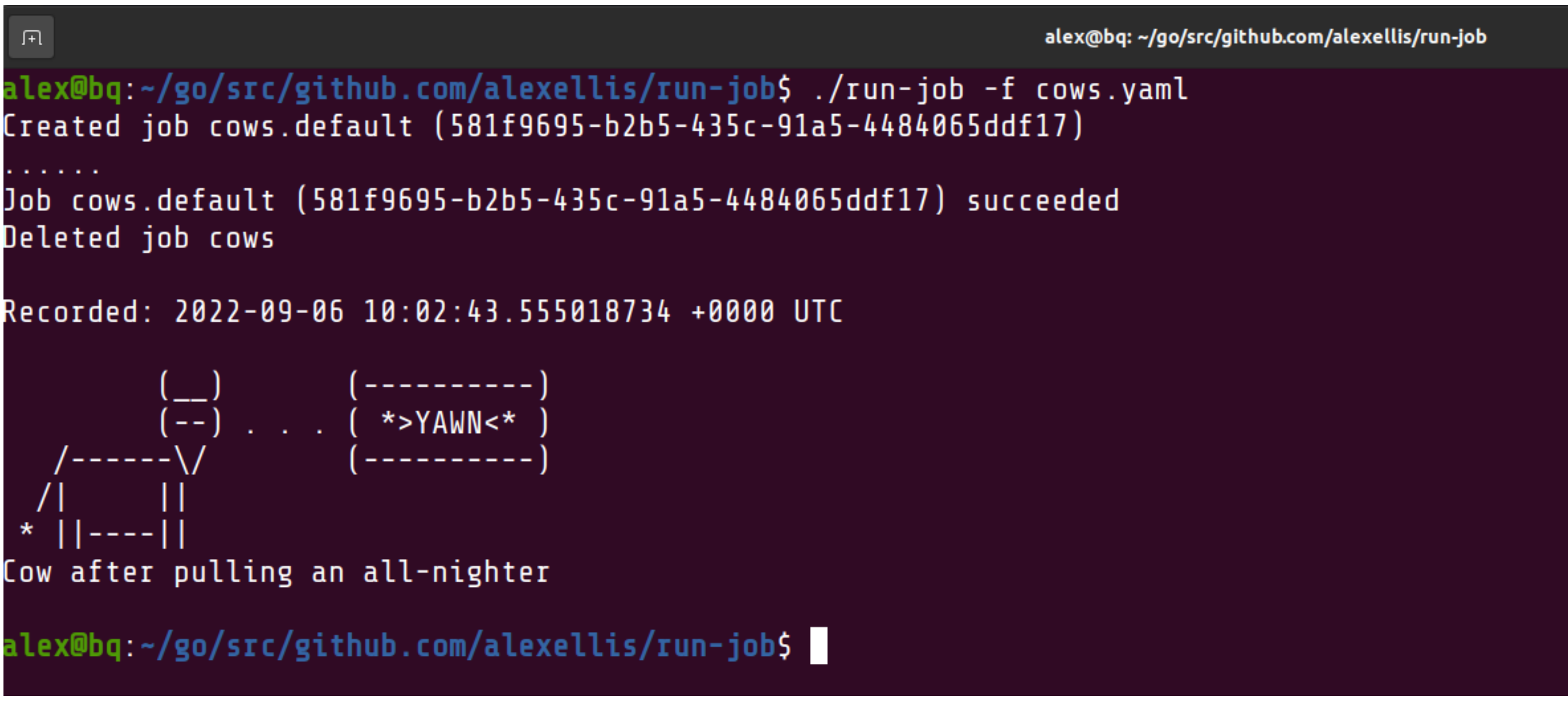
Fixing the UX for one-time tasks on Kubernetes

This article was fetched from an rss feed(06/09/2022)



```
alex@bq:~/go/src/github.com/alexellis/run-job$ ./run-job -f cows.yaml
Created job cows.default (581f9695-b2b5-435c-91a5-4484065ddf17)
......
Job cows.default (581f9695-b2b5-435c-91a5-4484065ddf17) succeeded
Deleted job cows

Recorded: 2022-09-06 10:02:43.555018734 +0000 UTC

         (__)        (----------)
         (--) . . .  ( *>YAWN<* )
   /------\/         (----------)
  /|      ||
 * ||----||
Cow after pulling an all-nighter

alex@bq:~/go/src/github.com/alexellis/run-job$ 
```

I wanted to run a container for a customer only once, but the UX just wasn't simple enough. So I created a new OSS utility with Go and the Kubernetes API.

A Deployment will usually restart, so will a Pod. You can set the "restartPolicy" to "Never", but it's still not a good fit for something that's meant to start - run and complete.

Why would you want to run a one-time job?

Here are a few ideas:

- Running kubectl within the cluster under a specific Service Account
- Checking `curl` works from within the cluster
- Checking inter-pod DNS is working
- Cleaning up a DB index
- A network scan with nmap
- Tasks that you'd usually run on cron
- Dynamic DNS updates
- Generating a weekly feed / report / update / sync
- Running a container build with Kaniko

Another use-case may be where you're running bash, but need to execute something within the cluster and get results back before going further.

I'll show you what working with a Job looks like today and how we can make it better with a little Go code.

What a Job looks like

So I looked again at the Kubernetes Job, which is an API that I rarely see used. They look a bit like this:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  suspend: true
  parallelism: 1
  completions: 5
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Always
  backoffLimit: 4
```

Example from the Kubernetes docs

So the overall spec is similar to a Pod, but it's unfamiliar enough to cause syntax errors when composing these by hand.

What's more: a Job isn't something that you run, then get its logs and continue with your day.

You have to describe the job, to find a Pod that it created with a semi-random name, and then get the logs from that.

```
kubectl describe jobs/myjob

Name:           myjob
...
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  12m   job-controller  Created pod: myjob-hlrpl
  Normal  SuccessfulDelete  11m   job-controller  Deleted pod: myjob-hlrpl
  Normal  Suspended         11m   job-controller  Job suspended
  Normal  SuccessfulCreate  3s    job-controller  Created pod: myjob-jvb44
  Normal  Resumed           3s    job-controller  Job resumed
```

So you end up with something like this:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: checker
  namespace: openfaas
spec:
  completions: 1
  parallelism: 1
  template:
    metadata:
      name: checker
    spec:
      serviceAccount: openfaas-checker
      containers:
      - name: checker
        image: ghcr.io/openfaas/config-checker:latest
        imagePullPolicy: Always
      restartPolicy: Never
```

And in my instance, I also wanted RBAC.

Then you'd:

- Apply the RBAC file
- Apply the Job
- Describe the Job
- Find the Pod name
- Get the Pod logs
- Delete the Job/Pod

  #!/bin/bash

  JOBUUID=$(kubectl get job -n openfaas $JOBNAME -o "jsonpath={.metadata.labels.controller-uid}") PODNAME=$(kubectl get po -n openfaas -l controller-uid=$JOBUUID -o name)

  kubectl logs -n openfaas $PODNAME > $(date '+%Y-%m-%d_%H_%M_%S').txt

  kubectl delete -n openfaas $PODNAME kubectl delete -f ./artifacts/job.yaml

Can we do better?

I think we can. And in a past life, all the way back in 2017, before Kubernetes won me over, I was using Docker Swarm.

I wrote a little tool called "jaas" and it turned out to be quite popular with people using it for one-time tasks that would run on Docker Swarm.

View the code: alexellis/jaas

I also remembered that Stefan Prodan who was a friend of mine and a past contributor to OpenFaaS had once had this itch and created something called kjob.

Stefan's not touched the code for three years, but his approach was to take a CronJob as a template, then to add in a few alterations. It's flexible because it means you can do just about anything you want with the spec, then you override one or two fields.

But I wanted to move away from lesser-known Kubernetes APIs and get a user-experience that could be as simple as:

```
kubectl apply -f ./artifacts/rbac.yaml
run-job ./job.yaml -o report.txt
```

Enter run-job

So that's how run-job was born.

My previous example became:

```
name: checker
image: ghcr.io/openfaas/config-checker:latest
namespace: openfaas
service_account: openfaas-checker
```

And this was run with `run-job -f job.yaml`.

Then I thought, let's make this more fun. The cows function in OpenFaaS is one of the most popular for demos and was also part of the codebase for jaas.

Here's its Dockerfile:

```
FROM --platform=${TARGETPLATFORM:-linux/amd64} alpine:3.16 as builder

ARG TARGETPLATFORM
ARG BUILDPLATFORM
ARG TARGETOS
ARG TARGETARCH

RUN mkdir -p /home/app

RUN apk add --no-cache nodejs npm

# Add non root user
RUN addgroup -S app && adduser app -S -G app
RUN chown app /home/app

WORKDIR /home/app
USER app
```

```
COPY package.json     .
RUN npm install --omit=dev
COPY index.js         .

CMD ["/usr/bin/node", "./index.js"]
```

It's multi-arch and you can either use buildx or faas-cli to compile it for various architectures.

```
$ cat <<EOF > cows.yaml
# Multi-arch image for arm64, amd64 and armv7l
image: alexellis2/cows:2022-09-05-1955
name: cows
EOF
```

Here's what it looks like:

```
$ run-job -f cows.yaml

        ()  ()
         ()()
         (oo)
  /-------UU
 / |     ||
*  ||w---||
   ^^    ^^
Eh, What's up Doc?
```

With `kubectl get events -w` we can see what's actually happening behind the scenes:

```
0s          Normal   SuccessfulCreate   job/cows         Created pod: cows-5qld5
0s          Normal   Scheduled          pod/cows-5qld5   Successfully assigned default/cows-5qld5 to k3s-eu-west-agent-1
0s          Normal   Pulling            pod/cows-5qld5   Pulling image "alexellis2/cows:2022-09-05-1955"
0s          Normal   Pulled             pod/cows-5qld5   Successfully pulled image "alexellis2/cows:2022-09-05-1955" in 877.707423ms
1s          Normal   Created            pod/cows-5qld5   Created container cows
0s          Normal   Started            pod/cows-5qld5   Started container cows
0s          Normal   Completed          job/cows         Job completed
```

An example to use at work

But we can also do for-work kinds of things with one-shot tasks.

Imagine that you wanted to get metadata on all your Kubernetes nodes.

You'd need an RBAC file for that:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kubectl-run-job
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app: run-job
  name: kubectl-run-job
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app: run-job
  name: kubectl-run-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kubectl-run-job
subjects:
  - kind: ServiceAccount
    name: kubectl-run-job
    namespace: default
---
```

Then you'd need a container image with kubectl pre-installed, so let's make that:

```
FROM --platform=${TARGETPLATFORM:-linux/amd64} alpine:3.16 as builder

ARG TARGETPLATFORM
ARG BUILDPLATFORM
ARG TARGETOS
ARG TARGETARCH

RUN mkdir -p /home/app

# Add non root user
RUN addgroup -S app && adduser app -S -G app
RUN chown app /home/app

RUN apk add --no-cache curl && curl -SLs https://get.arkade.dev | sh

WORKDIR /home/app
USER app

RUN arkade get kubectl@v1.24.1 --quiet

CMD ["/home/app/.arkade/bin/kubectl", "get", "nodes", "-o", "wide"]
```

We'd also create a YAML file for the job:

```
name: get-nodes
image: alexellis2/kubectl:2022-09-05-2243
namespace: default
service_account: kubectl-run-job
```

Finally, we'd run the job:

```
$ kubectl apply ./examples/kubectl/rbac.yaml
$ run-job -f ./examples/kubectl/kubectl_get_nodes_job.yaml

Created job get-nodes.default (4097ed06-9422-41c2-86ac-6d4a447d10ab)
....
Job get-nodes.default (4097ed06-9422-41c2-86ac-6d4a447d10ab) succeeded
Deleted job get-nodes

Recorded: 2022-09-05 21:43:57.875629 +0000 UTC

NAME           STATUS   ROLES                       AGE   VERSION        INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION
k3s-server-1   Ready    control-plane,etcd,master   25h   v1.24.4+k3s1   192.168.2.1   <none>        Raspbian GNU/Linux 10 (buster)   5.10.103-v7l+
k3s-server-2   Ready    control-plane,etcd,master   25h   v1.24.4+k3s1   192.168.2.2   <none>        Raspbian GNU/Linux 10 (buster)   5.10.103-v7l+
k3s-server-3   Ready    control-plane,etcd,master   25h   v1.24.4+k3s1   192.168.2.3   <none>        Raspbian GNU/Linux 10 (buster)   5.10.103-v7l+    c
```

Now, you can also specify the command and arguments for the command in the job file, so you could get the same output in JSON and pipe it through to jq to automate things in bash:

```
name: get-nodes
image: alexellis2/kubectl:2022-09-05-2243
namespace: default
service_account: kubectl-run-job
command:
 - kubectl
args:
 - get nodes
 - -o
 - json
```

Summing up

The point of run-job really is to make running a one-time container on Kubernetes simpler than the experience of crafting YAML and typing in half a dozen kubectl commands.

If you think it may be useful for you, then run-job is available via `arkade get run-job` along with over 100 other CLIs that download much quicker than brew.

We've also had the first contribution to the repo, which was to add the "command" and "args" options to the YAML file. The idea isn't to replicate every field in the Pod and Job specification, but just enough to make "run-job -f job-file.yaml` useful and convenient for supporting customers and running ad-hoc tasks.

Star on GitHub: alexellis/run-job

At OpenFaaS Ltd, we've used run-job with a customer to check their OpenFaaS Deployments and the configuration of their functions. The container is a Go program and you may like to take a look at it, to adapt it to check your own systems?

For those of us who are interested in manipulating and querying a Kubernetes cluster from code, the Go client provides a really good experience with hundreds of OSS examples available.

Read the code: openfaas/config-checker

If you're really into checking remote Kubernetes clusters for your customers, then Replicated have a similar, but more generic tool to openfaas/config-checker called "troubleshoot" which can be used to create bundles.

The bundles collect data from the customer's environment for you and you get to write in a DSL.