Kubernetes Troubleshooting Workshop

Michael Bright, @mjbright Consulting

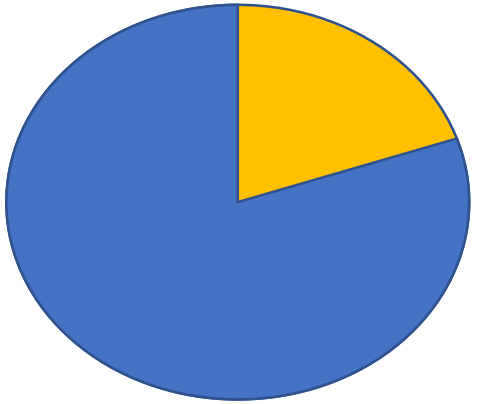
http://www.mjbright.net

https://linkedin.com/in/mjbright @mjbright @mjbright

$ who am i

$ who are you ?

Tell me, are you –



- A complete noob to k8s?
- You know the kubectl basics
  - how to deploy Deployments, Services
- You administer or develop Kubernetes at work
  - You’re a CKA/CKAD?

Troubleshooting - Groups

I propose to work in groups

- ideally of mixed Kubernetes experience so you learn together
- to limit the resources I need to create

Troubleshooting - Cluster

You will get access to a VM running in AWS EC2.

Each group will a share VM and use KIND « Kubernetes in Docker » to create a « 2 node » Kubernetes Cluster
https://kind.sigs.k8s.io/
https://github.com/kubernetes-sigs/kind

KIND allows us to create a lightweight cluster, with some limitations

Troubleshooting – BYOC ?

If you have access to a remote machine on which you want to run the workshop then you are welcome.

- The lab can be done on any cluster where you have full admin rights
- Don’t do this on a production cluster, though all scenarii should only affect the namespace ‘k8scenario’

You can also run this workshop from home after this session.

PLEASE DON’T USE A CLUSTER ON YOUR LAPTOP during this workshop, you will kill the network connection for everybody

Troubleshooting – BYOC ?

If you have access to a remote machine running OpenShift with full admin rights I'd love to know what works or not with this tool

Issues or PRs are welcome !

Troubleshooting exercises should work on
- Provided AWS EC2 VMs via KIND or kubeadm
- Managed cloud
- Local (Minikube, Docker Desktop, KIND, microk8s)

As long as sufficient resources, single-node is mostly ok.

Troubleshooting – k8scenario

There is an open source project, ‘k8scenario’ – a tool written in Go which will automatically install the selected scenario

https://k8scenario.github.io/

PRs, issues (ideas) for the tool are actively encouraged !!

- https://github.com/k8scenario/k8scenario

- or PRs, issues for the documentation https://github.com/k8scenario/k8scenario.github.io

Troubleshooting – k8scenario (priv)

Nevertheless we will be using a ‘closed source’ version of k8scenario which has some extra features, scenarii which I use in paid trainings

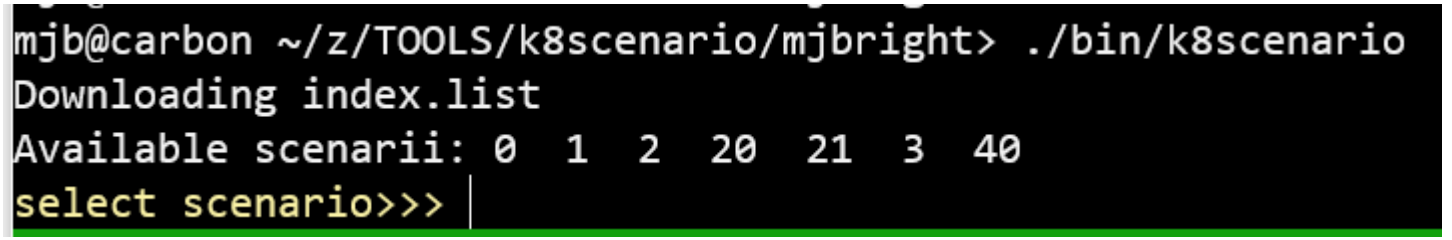
If you only want to use the ‘open source’ version then you’re welcome to do that

Next version: the tool will be completely open source, some scenarii will be ‘closed’

Troubleshooting – k8scenario

You will be presented with a basic menu

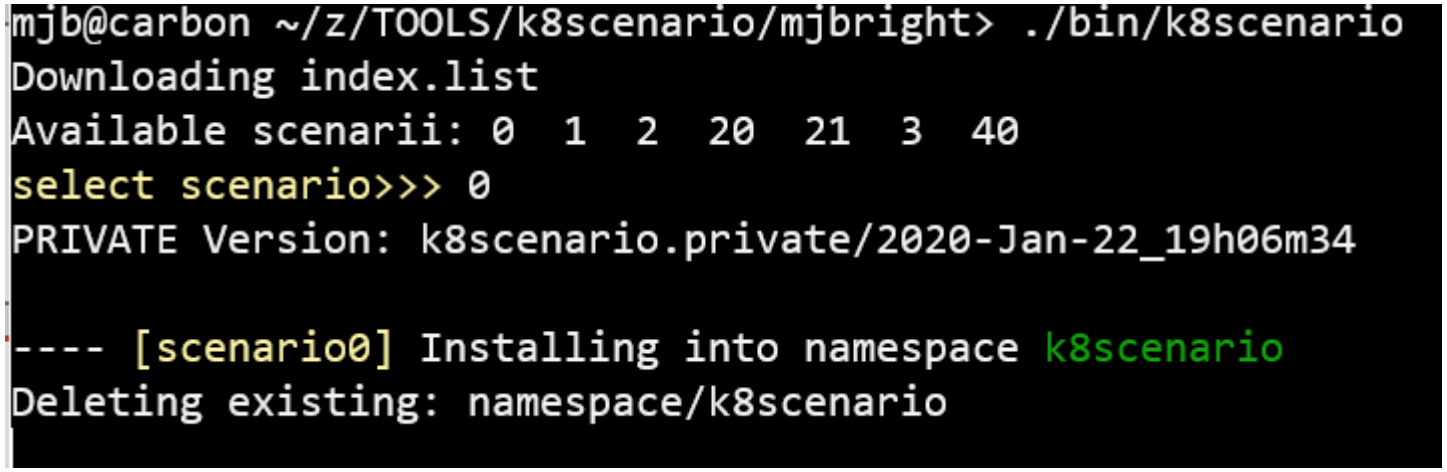
- select a scenario



Troubleshooting – k8scenario

Each scenario will be installed into the ‘k8scenario’ namespace

The namespace is deleted/recreated at the start of each scenario
So you need those namespace creation/deletion rights



Troubleshooting – k8scenario

```
mjb@carbon ~/z/TOOLS/k8scenario/mjbright> ./bin/k8scenario
Downloading index.list
Available scenarii: 0  1  2  20  21  3  40
select scenario>>> 0
PRIVATE Version: k8scenario.private/2020-Jan-22_19h06m34

---- [scenario0] Installing into namespace k8scenario
Deleting existing: namespace/k8scenario
(Re)creating namespace: k8scenario

# Task Instructions: Create a Pod in namespace 'k8scenario' whose name starts with 'basictest'

Note: There are 2 ways to do this
    - Explicitly create a Pod using 'kubectl run' with the appropriate name
or
    - Create a Deployment using 'kubectl create' with the appropriate name,
      it's Pods will also have a name derived from the Deployment name

[scenario0]/1 - task incomplete - Sleep 5s ... - exit status: 1
[scenario0]/2 - task incomplete - Sleep 5s ...
```

Troubleshooting – k8scenario

I recommend to download/install kubectx/kubens to be able to *easily* change your current namespace: https://github.com/ahmetb/kubectx/releases

```
mjb@carbon ~/z/TOOLS/k8scenario/mjbright> kubens k8scenario
Context "kind-kind" modified.
Active namespace is "k8scenario".
mjb@carbon ~/z/TOOLS/k8scenario/mjbright>
```

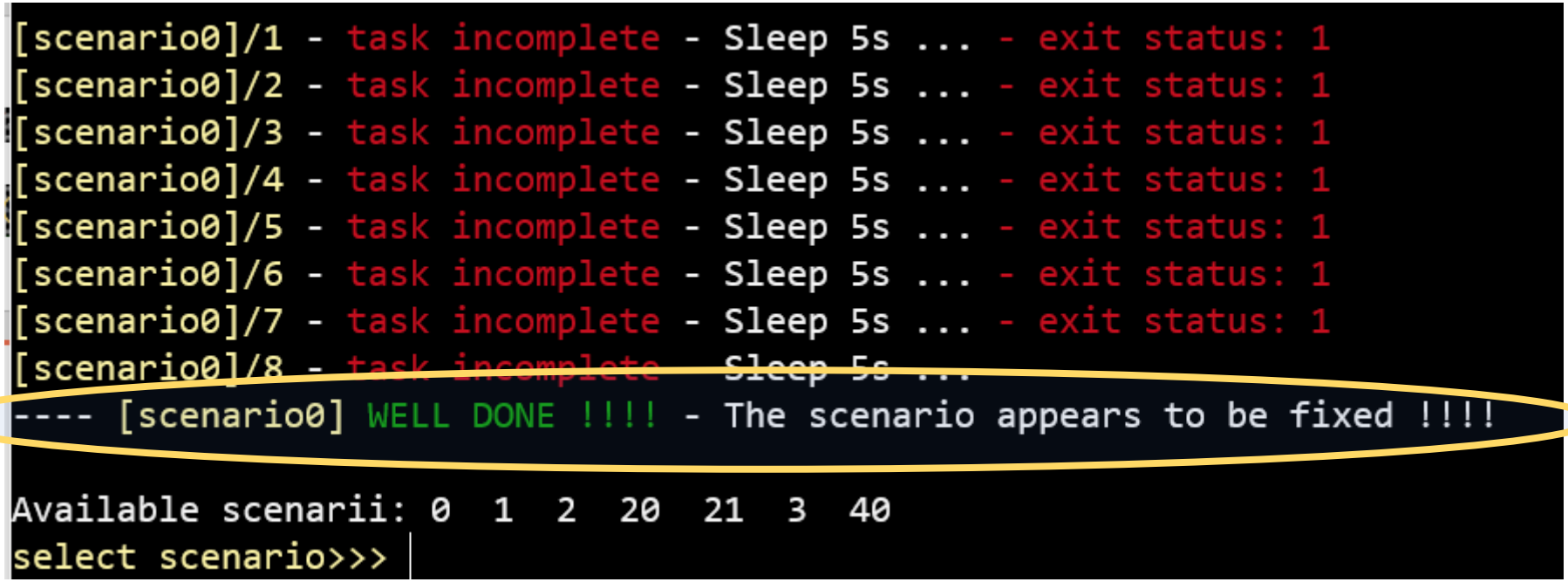
```
mjb@carbon ~/z/TOOLS/k8scenario/mjbright> kubectl config get-contexts
CURRENT   NAME          CLUSTER       AUTHINFO      NAMESPACE
          k8scenario    kind-kind     kind-kind     k8scenario
*         kind-kind     kind-kind     kind-kind     k8scenario
          kind-kind1    kind-kind1    kind-kind1
          kind-kind2    kind-kind2    kind-kind2
```

Troubleshooting – k8scenario

```
mjb@carbon ~/z/TOOLS/k8scenario/mjbright> kubectl create deploy --image mjbright/ckad-demo:1 basictest
deployment.apps/basictest created
mjb@carbon ~/z/TOOLS/k8scenario/mjbright>
```

```
mjb@carbon ~/z/TOOLS/k8scenario/mjbright> kubectl get all -A | grep basictest
k8scenario   pod/basictest-5779f4c99b-slx5c                   1/1     Running   0          16m
k8scenario   deployment.apps/basictest                  1/1     1            1           16m
k8scenario   replicaset.apps/basictest-5779f4c99b             1         1         1       16m
mjb@carbon ~/z/TOOLS/k8scenario/mjbright>
```

Troubleshooting – k8scenario



Troubleshooting – k8scenario

Scenarii are typically

- Tasks to perform (like scenario0 – create Pods adhering to some criteria)
- A problem to fix
- In the future quiz functionality will be added

Bored ?? !!

- Then start hacking on https://github.com/k8scenari/k8scenario
- Improve the tool
- Add scenarii
- Improve the documentation

Logistics

Questions?

Feedback ?

Please don’t leave without giving feedback

Google Forms:

http://bit.ly/2020JAN23

PRs/issues

https://github.com/k8scenario/k8scenario/issues

https://github.com/k8scenario/k8scenario.github.io/issues

Questions ?

Feedback

- What worked well ?
- What didn’t ?
- Suggestions

Let’s have some Kubernetes fun!

And learn a thing or two..

@mjbright https://linkedin.com/in/mjbright

Thank you !

 https://linkedin.com/in/mjbright

 @mjbright

 @mjbright

Troubleshooting

We will use the k8scenario tool for installing scenarii to be debugged into our cluster
https://k8scenario.github.io/
https://github.com/k8scenario/k8scenario

Some debug resources:
https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service

https://k8scenario.github.io/

Basic Troubleshooting

Some useful commands for command-line debugging

Refer to the kubectl cheat sheet for more detail

- https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Kubectl get pods

- Look for pod readiness/status, identify pods with problems
  - kubectl get pods -o wide
  - kubectl get pods -o yaml
  - kubectl get pods --show-labels
  - kubectl get pods –A # --all-namespaces
  - kubectl get pods -w # --watch (incremental)
  - watch –n 10 kubectl get pods -A

Basic Troubleshooting

Kubectl describe pod

- Look for events and container status/conditions/restarts

Kubectl get nodes

Kubectl get events [-w]

- Look for events and container status/conditions/restarts

Kubectl logs <pod> [-c <container>] [-p]

Basic Troubleshooting

An excellent resource to introduce you to the process of debugging applications on Kubernetes is the

"*Visual guide on troubleshooting Kubernetes deployments*“

on the @learnk8s blog at
https://learnk8s.io/troubleshooting-deployments

Troubleshooting

Describe any failing Pods, and look first at the

Events: section at the end which often describes what's wrong

$ kubectl describe pod <podname>

Look at labels on resources, e.g. Pods with

$ kubectl get pods --show-labels

Troubleshooting

Create a shell container inside a new or existing Pod to test DNS and other network issues

Check nodes logs for errors

Check Security Settings

- Security Contexts, Pod Security Policies, Network Policies
- RBAC rules
- SELinux, AppArmor settings if enabled

Troubleshooting

If Pods fail and redeploy too quickly (so log is always empty), you can view previous Container logs by using the –p option, e.g.

To view previous terminated ruby container logs from pod web-1

```
kubectl logs -p -c ruby web-1
```

You can open another console to watch as resources evolve:

```
$ kubectl get all –watch          # show incremental changes
$ watch –n 10 kubectl get all     # show current resources, every 10 secs
```

Or as events are reported:

```
$ kubectl get events [--watch]
```

Kubectl sub-commands

Selections from "*kubectl help*"

Basic Commands (Beginner):

create	Create a resource from a file or from stdin.
expose	Expose a replication controller, service, deployment or pod as a new Kubernetes Service
run	Run a particular image on the cluster
set	Set specific features on objects

Basic Commands (Intermediate):

explain	Documentation of resources
get	Display one or many resources
edit	Edit a resource on the server
delete	Delete resources by filename, stdin, resources & names, or resources & label selector

https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Kubectl sub-commands

Deploy Commands:

rollout	Manage the rollout of a resource
scale	Set replicas for Deployment, ReplicaSet or Replication Controller
autoscale	Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:

certificate	Modify certificate resources.
cluster-info	Display cluster info
top	Display Resource (CPU/Memory/Storage) usage.
cordon	Mark node as unschedulable
uncordon	Mark node as schedulable
drain	Drain node in preparation for maintenance
taint	Update the taints on one or more nodes

Kubectl sub-commands

Troubleshooting and Debugging Commands:

describe	Show details of a specific resource or group of resources
logs	Print the logs for a container in a pod
attach	Attach to a running container
exec	Execute a command in a container
port-forward	Forward one or more local ports to a pod
proxy	Run a proxy to the Kubernetes API server
cp	Copy files and directories to and from containers.
auth	Inspect authorization

Kubectl sub-commands

Advanced Commands:

diff	Diff live version against would-be applied version
apply	Apply a configuration to a resource by filename or stdin
patch	Update field(s) of a resource using strategic merge patch
replace	Replace a resource by filename or stdin
wait	Experimental: Wait for a specific condition on one or many resources.
convert	Convert config files between different API versions
kustomize	Build a kustomization target from a directory or a remote url.

Kubectl sub-commands

Settings Commands:

label	Update the labels on a resource
annotate	Update the annotations on a resource
completion	Output shell completion code for the specified shell (bash or zsh)

Other Commands:

api-resources	Print the supported API resources on the server
api-versions	Print the supported API versions on the server, in the form of "group/version"
config	Modify kubeconfig files
plugin	Provides utilities for interacting with plugins.
version	Print the client and server version information

Kubectl sub-commands

There's more: see "*kubectl options*" to see another 30 options which apply to all sub-commands

e.g. -v <verbosity>

kubectl get pods -v 10

> kubectl get pods -v 10 |& grep "GET http"
I0104 22:37:07.368597 21381 round_trippers.go:443] **GET** https://127.0.0.1:11997/api/v1/namespaces/default/pods?limit=500 200 OK in 10 milliseconds

Basic Troubleshooting – kubectl attach

We can attach to the stdout of a running container to see it's output

kubectl help attach
Attach to a process that is **already running** inside an **existing container**.

Examples:
Get output from running pod 123456-7890, using the first container by default
kubectl attach 123456-7890

Get output from ruby-container from pod 123456-7890
kubectl attach 123456-7890 -c ruby-container

Get output from the first pod of a ReplicaSet named nginx
kubectl attach rs/nginx

The usual -it (--interactive and --tty) options are also available

Basic Troubleshooting – kubectl exec

We can exec into a running container (creates a new process)

```
kubectl help exec
Execute a command in an existing container.

Examples:
# Get output from running 'date' command from pod mypod, using the first container by default
kubectl exec mypod date

# Get output from running 'date' command in ruby-container from pod mypod
kubectl exec mypod -c ruby-container date

# Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod mypod
# and sends stdout/stderr from 'bash' back to the client
kubectl exec mypod -c ruby-container -i -t -- bash -il
```

The usual -it (--interactive and --tty) options are also available

Basic Troubleshooting – kubectl exec

kubectl exec can be useful for

- Checking container functionality
- Checking connectivity from a Pod to
  - other Pods
  - Services
  - External sources
- Checking cluster DNS resolution

Break