Distributed Systems Administration

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Deployement of a K3s cluster on a dev environment with Vagrant over Virtualbox

Francisco Javier Pizarro Martínez 821259

19/04/2023

Abstract

This is the first approach to the use of tools like Vagrant which let us automate the deployment of a testing environment, as well as the first take off with K3s clusters and how to use them. This deployment will be running on Ubuntu 18.04 because it is one of the most validated ones and it is already inside of our toolset. Other objective is being able to understand how to make K3s manifests, that can be used to create certain elements in the cluster.

The commands we will be using with Vagrant are these:

- vagrant up
- vagrant halt
- vagrant destroy
- vagrant ssh ${machine}

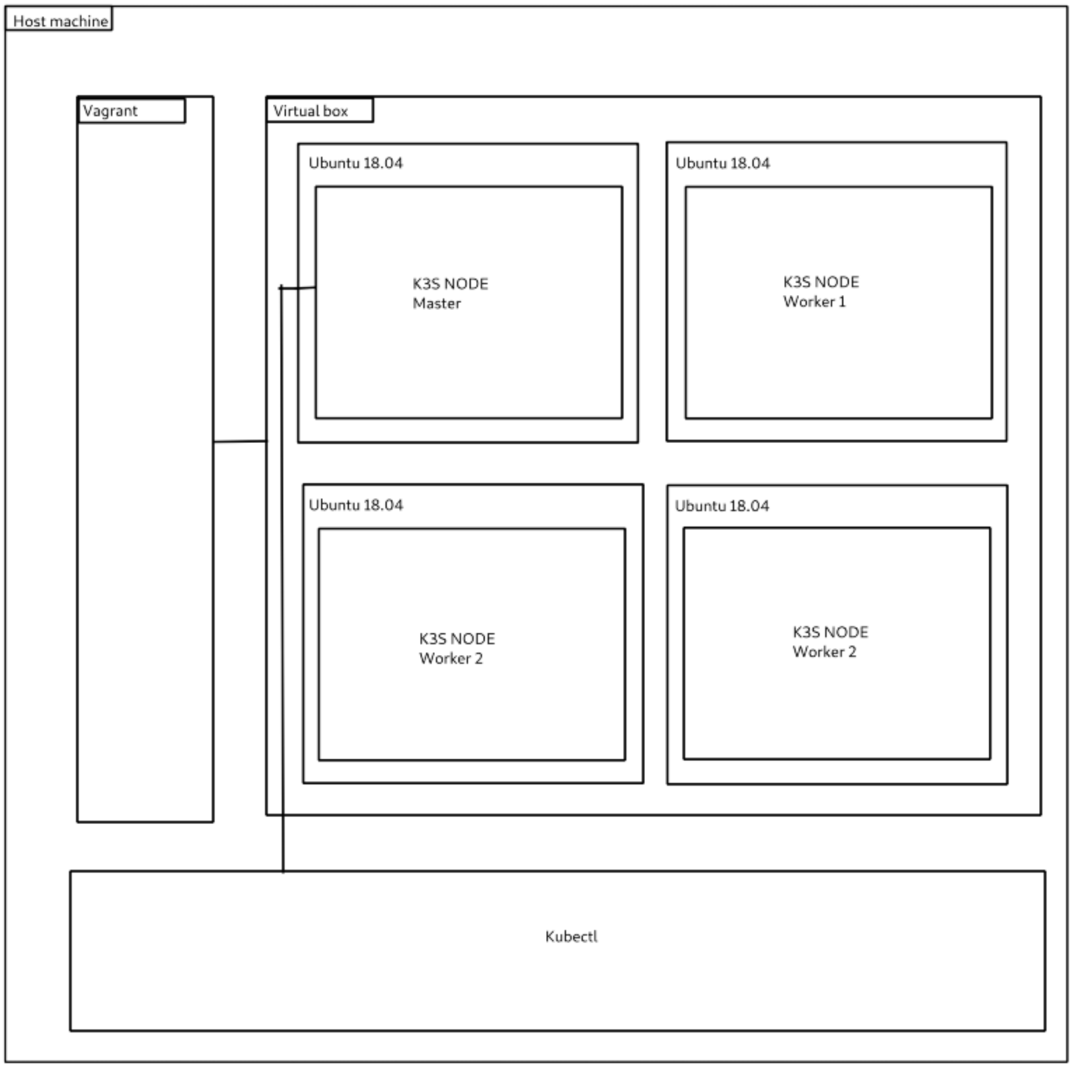
The components we will be playing inside the K3s cluster are this ones:

- Pods
- Manifests
- Labels
- Nodes
- Logs
- Proxy
- API
- Port forwarding
- Deployments
- Rollouts
- Services
- Namespaces
- Volumes
- Persistent volumes
- Statefulsets
- Jobs
- Dashboards
- Health checks
- Env variables

Another implicit task of this project is learn how to setup the correct enviroment inside a Linux machine in order to be able to perform all the other tasks.

Architecture

We are going to run a K3s scluster on 4 VMs hosted on VirtualBox, this VMs have Ubuntu 18.04 as OS. To setup, access, manage and destroy this VMs in a more automatic way we are going to use Vagrant. As usual to interact with the K3s cluster we will use kubectl. This is the schema that represents the desired architecture.



The elemental files that we need to configurate properly to do this deploy are the following ones:

- Vagrantfile
- provission.sh
- install.sh

This files can be found in the annexes.

Initial setup

In order to be able to start this cluster first we need to do some previous configurations:

- Install both vagrant and virtualbox

```
apt install virtualbox
apt install vagrant
```

- Change the values of the IPs and the network interface on the given Vagrantfile
- Change the values of the IPs that are inside of provision.sh
- Create the file `.config/Virtualbox/Virtualbox.xml` and write the following lines inside:

```
<VirtualBox xmlns="http://www.virtualbox.org/" version="1.16-linux">
  <Global>
    <SystemProperties defaultMachineFolder="/home/fjpizarro/Desktop/VMs/" />
  </Global>
</VirtualBox>
```

- Add to our bash.rc this `export VAGRANT_HOME=/home/user/Desktop/ProyectoParte1/vagrantk3s/`
- Set all the project files except the Vagrantfile with execution permission
- In the BIOS settings allow the Virtualization

After setting up all the configurations, we start the VMs by running `vagrant up` , to turn off all the VMs we must execute `vagrant halt`, if we want to delete all the VMs information we use `vagrant destroy`

To check that all is working properly we run `kubectl get nodes` on the master machine, the output should look like this:

```
kubectl get nodes
NAME   STATUS   ROLES                  AGE   VERSION
m      Ready    control-plane,master   40m   v1.23.5+k3s1
w1     Ready    <none>                 39m   v1.23.5+k3s1
w2     Ready    <none>                 39m   v1.23.5+k3s1
w3     Ready    <none>                 38m   v1.23.5+k3s1
```

A really usefull feature of Vagrant is the default shared folder, all the VMs share a folder called /vagrant which is the host folder where we have the Vagrantfile

Playing around with the K3S cluster

Before doing any advanced deployments or apps on top of the K3S we need to understand all the standard parts of K3S so we are going to do some tests using this cluster, all the tests done can be found in the annex III.

The most basic component is the Pod, this basic unit allow us to perform a task along the time for example running a basic http server. We can also launch terminals inside Pods, for example to test if the service that is running inside other Pod is working properly. If we need to login again inside our testing terminal we can also do it without the need of recreating the "terminal" Pod. If inside the same pod there are 2 or more containers they share the same "network" this can be used to send request across containers.

A essential part of using K3s is the use of the Manifests, this files allow us to define components in a "static" way, the advantage that we get from this is reusing component definitions.

Another very basic part of Kubernetes are the Labels which give us the opportunity to mark other components with certain tags, later we can use them to be more specific when we are running commands. If we add the flag -l we can filter the components by a specific Label or by a set of Labels. The labels can be added when we create the components or in runtime.

Under the hood all the things that run on K3S are being executed on which are known as Nodes , which are the (virtual) machines. We may need to setup a special schedule to do this we can use the labels as shown before.

A crucial part of the SysAdmin tasks is the use of logs which in case of failure are really important, we can get them in runtime or in a more "static" way, if a component crash and is relaunched we can also get the logs of the crashed instance by adding the flag -p or --previous.

In order to connect our host machine to certain API endpoints we may need to use a proxy. The API can also be reached in a raw way without using a proxy. The Deployments give us a higher level of abstraction, it ensures that the components that we need are properly launched. As it is shown in the test we can use the Deployments to "upgrade" the current deployment while we keep its disponibility, when the new version is running the old one is deleted, we can also keep track of the history and if something breaks we can go to a previous rollout version.

Usually we want to be as far as possible from the physical details like IP, we use a level of abstraction, for example we dont want to write IPs to access websites in this case the solution is using a DNS, in kubernetes we have something similar with more aspects, this are the Services. Apart from doing the "DNS" task it also acts as a load-balancer.

When we want to access some component without using a load balancer or a ingress the PortForwarding is the best option, this tool is designed to be used on dev enviroments and not in production.

To separate the resources, like if we had users on a OS, we use the namespaces to isolate different resources.

The volumes let the containers running inside a Pod share a directory, under the hood the implementation depends on the contents(node-local,file-sharing,cloud...). There is also a persistent version of the volumes which perdurrate along the time even if the Component crash. Unlike the normal volumes to use a persistent volume we need to claim what we are going to use.

If we have a more dynamic app a StatefullSet can bring us some intersting properties such as identifing each pod with a name known before even launching it, using a certain order when creating the Pods and giving each Pod a persistent volume.

There is also a Component that let us run batch process in a controlled way, this are the Jobs.

K3s offers us as a native tool doing healtch checks to do this we must specify the property livenessProbe in the manifest, for example:

```
livenessProbe:
      initialDelaySeconds: 2
      periodSeconds: 5
      httpGet:
        path: /health
        port: 9876
```

This property have various side effects, the first one is that the events related to the state of the service tested with the health check are listed when we execute a describe command the other principal side effect is that if K3s detects that the service of the container isnt working properly, it restarts the whole container.

In most modern apps there are variables that must be set when launching the app for example a URL in a webapp, kubernetes offers us something similar to set this env variables. To set them we use this property in the Manifest:

```yaml
env:
    - name: SIMPLE_SERVICE_VERSION
      value: "1.0"
```

K3s cheatsheet

The following commands are all preceed by `kubectl`

Command	Description
describe %	Gives information about %
run % --image &&	Creates on the fly a Pod called % usign && as base image
get % -o wide	List all the % components
apply -f file	Creates the components specified in the file
exec %	Commonly used to access a terminal inside a container
delete ¡¡ %	Deletes the component % which is of ¡¡ type
label ¡¡ % text	Labels the component % which is of ¡¡ type with the text
top ¡¡	Gives information like a normal top but in a static way
logs %	Prints the logs of the Pod %
proxy --port=8080	Creates a proxy to access the API of the cluster
rollout %	Controls the status and history of the versions, it also let us go back to previous versions
scale %	Its function is to change the number of active replicas
port-forward % 8080:80	Fast way to do a port forward for developing

The flag -l can be used to filter by using labels

To speed up the use of K3s we can use some abbreviatons in all the previous commands:

Complete name	Abbreviation
Pod	po
Replicaset	rs
Service	svc
ReplicationController	rc
Namespace	ns
PersistentVolume	pvc
StatefullSet	sts

Annex I Vagrantfile content

```
Ubu = 'ubuntu/bionic64'

# We define the names IPs roles and nodes
MASTER = '192.168.1.49'
NODES = [
  { hostname: 'm', type: "master", ip: MASTER, mem: 1000, m: MASTER },
  { hostname: 'w1', type: "worker", ip: '192.168.1.41', mem: 1000, m: MASTER },
  { hostname: 'w2', type: "worker", ip: '192.168.1.42', mem: 1000, m: MASTER },
  { hostname: 'w3', type: "worker", ip: '192.168.1.43', mem: 1000, m: MASTER },
]

# We config each one of the nodes
Vagrant.configure("2") do |config| # create the instance of the vagrant settings
    NODES.each do |node| # for each one of the predefined nodes do
        config.vm.define node[:hostname] do |nodeconfig| # for the name of the current node
            # Internal setup and configuration of the VM
            # Configuration of hostname and OS
            nodeconfig.vm.box = Ubu # we setup the self-contained image of the OS
            nodeconfig.vm.hostname = node[:hostname] # we setup the hostname of the VM
                        # Network configuration
            nodeconfig.vm.network :public_network, # we setup the type of network
                    bridge: "wlo1", # we choose the host interface that will be used
                    ip: node[:ip], # we setup the ip of the VM interface
                    # virtualbox__intnet: true,
                    nic_type: "virtio" # we choose the type of the ""physical"" network interface of the VM
                # Virtualbox external settings of the VM
            # RAM and CPU config
            nodeconfig.vm.provider "virtualbox" do |v|
                v.customize ["modifyvm",:id,"--memory",node[:mem],"--cpus","1"] # we setup the max RAM and CPUs of
the VM using a command
                v.default_nic_type = "virtio" # we choose the type of the ""physical"" network interface of the VM
            end
            # Timeour error config
            nodeconfig.vm.boot_timeout = 400
            # Call to provision.sh in order to obtain all the necesary elements.
            nodeconfig.vm.provision "shell",
                path: 'provision.sh',
                args: [ node[:hostname], node[:ip], node[:m], node[:type] ]

            if node[:type] == "master" # only in the master node
              nodeconfig.trigger.after :up do |trigger| # after booting up
                trigger.run = \ # execute the following
                        {inline: "sh -c 'cp k3s.yaml /home/user/.kube/config'"} # copy the k3s
conf(certificates....) to the host .kube conf
              end
            end
        end
    end
end
```

Annex II provision.sh

```bash
#!/bin/bash -x
# Get the arguments
HOSTNAME=$1
NODEIP=$2
MASTERIP=$3
NODETYPE=$4
# set time and folder
timedatectl set-timezone Europe/Madrid
cd /vagrant
# config hostname and hosts
echo $1 > /etc/hostname # writes the hostname into the conf file
hostname $1 # sets the hostname for the current execution of the MV
MASTER = '192.168.1.49'
{ echo 192.168.1.49 m; echo 192.168.1.41 w1; echo 192.168.1.42 w2
  echo 192.168.1.43 w3; cat /etc/hosts
} > /etc/hosts.new # rewrites the content of hosts and adds to it all the k3s nodes
mv /etc/hosts{.new,}

# get k3s executable
cp k3s /usr/local/bin/

# MASTER SETUP
if [ $NODETYPE = "master" ]; then
  INSTALL_K3S_SKIP_DOWNLOAD=true \
  ./install.sh server \ # use the installation script
  --token "wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5" \ #sets the token
  --flannel-iface enp0s8 \ #sets the name of the network interface used in the k3s cluster communications, this
option is only for the servers
  --bind-address $NODEIP \ # listen on the node IP
  --node-ip $NODEIP --node-name $HOSTNAME \ # sets the node ip and nodename
  --disable traefik \ # diabling the default http reverse proxy and load balancer
  --disable servicelb \ # disabling the default loadbalancer
  --node-taint k3s-controlplane=true:NoExecute # restricts the master node to no execute the Components that in
theory must run in worker nodes such as Pods
  #--advertise-address $NODEIP
  #--cluster-domain “cluster.local”
  #--cluster-dns "10.43.0.10"

  cp /etc/rancher/k3s/k3s.yaml /vagrant # copy the k3s conf to the shared folder
#WORKERS SETUP
else
  INSTALL_K3S_SKIP_DOWNLOAD=true \
  ./install.sh agent --server https://${MASTERIP}:6443 \ #use the installation script in agent mode telling where
the master node is
  --token "wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5" \
  --node-ip $NODEIP --node-name $HOSTNAME --flannel-iface enp0s8 # sets the ip,hostname and network interface
fi
```

Annex III Tests

Basic Pod

```
kubectl run sise --image=quay.io/openshiftlabs/simpleservice:0.5.0 --port=9876
kubectl get pods -o wide
```

Terminal inside a Pod

```
kubectl run -i --tty busybox1 --image=busybox -- sh
wget -q -O - 10.42.3.2:9876/info
#{"host": "10.42.3.2:9876", "version": "0.5.0", "from": "10.42.2.2"}
```

In order to enter again use:
`kubectl attach busybox1 -c busybox1 -i -t`

Manifests

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pods/pod.yaml
kubectl exec twocontainers -c shell -i -t -- bash
curl -s localhost:9876/info
#{"host": "localhost:9876", "version": "0.5.0", "from": "127.0.0.1"}`
kubectl delete pod twocontainers
kubectl delete pod sise
kubectl delete pod busybox1
```

Labels

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/labels/pod.yaml
kubectl get pods --show-labels
kubectl label pods labelex owner=michael
kubectl get pods --l owner=michael
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/labels/anotherpod.yaml
kubectl get pods -l 'env in (production, development)'
kubectl delete pods -l 'env in (production, development)'
```

Nodes

```
kubectl top nodes
kubectl label nodes w1 shouldrun=here
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/nodes/pod.yaml
kubectl describe node m
```

Logs

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/logging/pod.yaml
kubectl logs --tail=5 logme -c gen
kubectl logs -f --since=10s logme -c gen
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/logging/oneshotpod.yaml
kubectl logs -p oneshot -c gen
kubectl delete pod/logme pod/oneshot
```

Proxy

```
kubectl proxy --port=8080
curl http://localhost:8080/api/v1
```

Raw API

```
kubectl get --raw /api/v1
```

Deployments

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/deployments/d09.yaml
kubectl get deploy
kubectl get rs
kubectl get pods
kubectl run -i --tty busybox1 --image=busybox -- sh
wget -q -O - 10.42.2.5:9876/info
#{"host": "10.42.2.5:9876", "version": "0.9", "from": "10.42.3.7"}
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/deployments/d10.yaml
kubectl get pods
kubectl get rs
kubectl rollout status deploy/sise-deploy
kubectl attach busybox1 -c busybox1 -i -t
wget -q -O - 10.42.2.6:9876/info
#{"host": "10.42.2.6:9876", "version": "1.0", "from": "10.42.3.7"}
kubectl rollout history deploy/sise-deploy
kubectl rollout undo deploy/sise-deploy --to-revision=1
kubectl rollout history deploy/sise-deploy
kubectl delete deploy sise-deploy
```

Services

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/services/rc.yaml
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/services/svc.yaml
kubectl get pods -l app=sise
kubectl describe pod rcsise-rprhr | less
kubectl get svc
kubectl describe svc simpleservice | less
kubectl run -i --tty busybox1 --image=busybox -- sh
wget -q -O - 10.43.13.1:80/info
#{"host": "10.43.13.1:80", "version": "0.5.0", "from": "10.42.3.9"}
kubectl scale --replicas=2 rc/rcsise
kubectl get pods -l app=sise
kubectl delete svc simpleservice
kubectl delete rc rcsise
kubectl delete pod busybox1
```

Port-forwarding

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pf/app.yaml
kubectl port-forward service/simpleservice 8080:80
curl localhost:8080/info
#{"host": "localhost:8080", "version": "0.5.0", "from": "127.0.0.1"}
kubectl delete service/simpleservice
kubectl delete deployment sise-deploy
```

Namespaces

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/ns/ns.yaml
kubectl get ns
#NAME              STATUS   AGE
#default           Active   35h
#kube-system       Active   35h
#kube-public       Active   35h
#kube-node-lease   Active   35h
#test              Active   7s
kubectl apply --namespace=test -f https://raw.githubusercontent.com/openshift-
```

```
evangelists/kbe/master/specs/ns/pod.yaml
kubectl get pods --namespace=test
kubectl delete ns test
```

Volumes

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/volumes/pod.yaml
kubectl describe pod sharevol
kubectl exec -it sharevol -c c1 -- bash
[root@sharevol /] mount | grep xchange
#/dev/sda1 on /tmp/xchange type ext4 (rw,relatime,data=ordered)
[root@sharevol /] echo 'aaaaaaa' > /tmp/xchange/data
kubectl exec -it sharevol -c c2 -- bash
[root@sharevol /] mount | grep /tmp/data
#/dev/sda1 on /tmp/data type ext4 (rw,relatime,data=ordered)
[root@sharevol /] cat /tmp/data/data
#aaaaaaa
kubectl delete pod/sharevol
```

Persistent volume

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pv/pv.yaml
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pv/pvc.yaml
kubectl get pvc
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pv/deploy.yaml
kubectl get po
sudo kubectl exec -it pv-deploy-d86794765-k8m2n -- bash
[root@pv-deploy-d86794765-k8m2n /] touch /tmp/persistent/data
[root@pv-deploy-d86794765-k8m2n /] ls /tmp/persistent/
#data
kubectl delete po pv-deploy-d86794765-k8m2n
kubectl get po
kubectl exec -it pv-deploy-d86794765-74nfq -- bash
[root@pv-deploy-d86794765-74nfq /] ls /tmp/persistent/
kubectl delete deployment pv-deploy
kubectl delete pvc myclaim
```

Each node of the cluster has him own view of the storage, in other words if something happens in the node w3, this wont be seen by the pods deployed on other nodes.

StatefullSet

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/mehdb/master/app.yaml
kubectl get sts,po,pvc,svc
kubectl run -it --rm jumpod --restart=Never --image=quay.io/openshiftlabs/jump:0.2 -- curl mehdb:9876/status?
level=full
kubectl delete sts mehdb
```

Jobs

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/jobs/job.yaml
kubectl get jobs
kubectl get pods
kubectl describe jobs/countdown
kubectl logs countdown-wn5lw
kubectl delete job countdown
```

Health checks

```
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/main/specs/healthz/pod.yaml
kubectl describe pod hc
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/main/specs/healthz/badpod.yaml
kubectl describe pod badpod
#Events:
#  Type     Reason     Age                 From               Message
#  ----     ------     ----                ----               -------
# Normal   Scheduled  75s                 default-scheduler  Successfully assigned default/badpod to w3
#  Normal   Pulling    70s                 kubelet            Pulling image
"quay.io/openshiftlabs/simpleservice:0.5.0"
#  Normal   Pulled     34s                 kubelet            Successfully pulled image
"quay.io/openshiftlabs/simpleservice:0.5.0" in 35.63699003s
#  Normal   Created    34s                 kubelet            Created container sise
#  Normal   Started    34s                 kubelet            Started container sise
#  Warning  Unhealthy  19s (x3 over 29s)  kubelet            Liveness probe failed: Get
"http://10.42.3.2:9876/health": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
# Normal   Killing    19s                 kubelet            Container sise failed liveness probe, will be
restarted
kubectl get pods
#NAME     READY   STATUS    RESTARTS     AGE
#hc       1/1     Running   0            3m10s
#badpod   1/1     Running   1 (8s ago)   94s
kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/main/specs/healthz/ready.yaml
kubectl delete pod/hc pod/ready pod/badpod
```

Env variables

```
kubectl apply -f https://github.com/openshift-evangelists/kbe/raw/main/specs/envs/pod.yaml
kubectl exec envs -t -- curl -s 127.0.0.1:9876/info
kubectl exec envs -t -- curl -s 127.0.0.1:9876/env
kubectl exec envs -- printenv
kubectl delete pod/envs
```

Annex IV K3s Dashboard setup

https://docs.k3s.io/installation/kube-dashboard

```
GITHUB_URL=https://github.com/kubernetes/dashboard/releases
VERSION_KUBE_DASHBOARD=$(curl -w '%{url_effective}' -I -L -s -S ${GITHUB_URL}/latest -o /dev/null | sed -e
's|.*/||')
sudo k3s kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/${VERSION_KUBE_DASHBOARD}/aio/deploy/recommended.yaml


vi dashboard.admin-user.yml
#copypastear the following
apiVersion: v1
kind: ServiceAccount
metadata:
 name: admin-user
 namespace: kubernetes-dashboard

vi dashboard.admin-user-role.yml
#copypaste the following
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: admin-user
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: cluster-admin
subjects:
- kind: ServiceAccount
 name: admin-user
 namespace: kubernetes-dashboard

sudo k3s kubectl create -f dashboard.admin-user.yml -f dashboard.admin-user-role.yml
sudo k3s kubectl -n kubernetes-dashboard describe secret admin-user-token | grep '^token'
#sudo k3s kubectl proxy --address 127.0.0.1

#idk how to access it but is there i guess
```

```
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6ImlmTkhXem5RWVNMVGZpWHNSRkQwWjF1ZWJuYm1lWXE3ZWR0X0NZXzB4N3MifQ.eyJpc3MiOiJrdWJlcm5ldGV
zL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1Y
mVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLTJzdGNyIiwia3ViZXJuZXRlcy5pby9zZXJ2aWN
lYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb
3VudC51aWQiOiIxYTUwMmY5MC1iMGQ0LTQ0ZDItYWIyMy01NWUwMmRjODFlZDgiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZXJuZXR
lcy1kYXNoYm9hcmQ6YWRtaW4tdXNlciJ9.OlhINc2YS01F2F9N17CZXQPEX8YVJj-
XFZgF0kCjRYpz6uQ5ZD6WrYgN_W2bOUb9UZbiKDDul6XnIVg77HKtx-
aQEKyMuUu0h666TsnvX1OzWdxxwy6iL_VGxTT82LaqkfZC2oIKtcBW4sD4V2oyzC7sZPXFZFtcXYv6-
f6PeRtbZ1K2j4Pi8vpbU1y5_-4DPkXyT1WUMx5It8magPX8s0yft5jSQcnJQEQ_csAsTwB_ODP3TRHYW1hqONQllA79N5wgNKcsaS--TwVUWpvVx-
okbCQrX6OuugB9DEAv9gnjNLrjQhLtVbvOcjejIVq1kq13apHRaFMAhmtLA9rLRQ
```