**Pods**

Kubernetes pods by example

A pod is a collection of containers sharing a network and mount namespace and is the basic unit of deployment in Kubernetes. All containers in a pod are scheduled on the same node.

To launch a pod using the container [image](https://hub.docker.com/r/mhausenblas/simpleservice/) mhausenblas/simpleservice:0.5.0 and exposing a HTTP API on port 9876, execute:

$ kubectl run sise --image=mhausenblas/simpleservice:0.5.0 --port=9876

We can now see that the pod is running:

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

sise-3210265840-k705b 1/1 Running 0 1m

$ kubectl describe pod sise-3210265840-k705b | grep IP:

IP: 172.17.0.3

From within the cluster (e.g. via minishift ssh) this pod is accessible via the pod IP 172.17.0.3, which we’ve learned from the kubectl describe command above:

[cluster] $ curl 172.17.0.3:9876/info

{"host": "172.17.0.3:9876", "version": "0.5.0", "from": "172.17.0.1"}

Note that kubectl run creates a [deployment](http://kubernetesbyexample.com/deployments/), so in order to get rid of the pod you have to execute kubectl delete deployment sise.

**Using configuration file**

You can also create a pod from a configuration file. In this case the [pod](https://github.com/openshift-evangelists/kbe/blob/master/specs/pods/pod.yaml) is running the already known simpleservice image from above along with a generic CentOS container:

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pods/pod.yaml

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

twocontainers 2/2 Running 0 7s

Now we can exec into the CentOS container and access the simpleservice on localhost:

$ kubectl exec twocontainers -c shell -i -t -- bash

[root@twocontainers /]# curl -s localhost:9876/info

{"host": "localhost:9876", "version": "0.5.0", "from": "127.0.0.1"}

Specify the resources field in the pod to influence how much CPU and/or RAM a container in a [pod](https://github.com/openshift-evangelists/kbe/blob/master/specs/pods/constraint-pod.yaml) can use (here: 64MB of RAM and 0.5 CPUs):

$ kubectl create -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/pods/constraint-pod.yaml

$ kubectl describe pod constraintpod

...

Containers:

sise:

...

Limits:

cpu: 500m

memory: 64Mi

Requests:

cpu: 500m

memory: 64Mi

...

Learn more about resource constraints in Kubernetes via the docs [here](https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-ram-container/) and [here](https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/).

To remove all the pods created, just run:

$ kubectl delete pod twocontainers

$ kubectl delete pod constraintpod

To sum up, launching one or more containers (together) in Kubernetes is simple, however doing it directly as shown above comes with a serious limitation: you have to manually take care of keeping them running in case of a failure. A better way to supervise pods is to use [deployments](http://kubernetesbyexample.com/deployments), giving you much more control over the life cycle, including rolling out a new version.