# Docker & Kubernetes workshop

## Install prerequisites

* Install Docker desktop
* Install minikube
* Create a DockerHub account.

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

On win10, check that HyperV is installed. If not install virtuabox.

## Getting to know Docker basics

Most used commands:

* “Docker images” - list all downloded images in local repository.
* “docker ps” – list all running processes.
* “docker run” – instanciate a docker image.
* “docker kill” – Stop a docker instance.
* “docker exec” – run a command in a running docker instance.
* “docker build” – make a docker image.
* “docker tag” – tag a image.
* “docker push”- upload a docker image to a central repository.
* “docker system prune” – Clean up temp docker “layers” to free space.

### Running a container

1. In a commandline window – (powershell/cmd/sh/bash/etc.)
2. “docker run hello-world”
3. Notice, layers. Every image is built of x layers based on a base image.

### Attaching to a container

1. In a commandline window
2. “docker run –it ubuntu bash”

Now you have a fully functioning Ubuntu minimal installation running.

1. Type “exit” to exit to host system ( kills the container)

### Running a container detached

1. In a cmd.
2. “docker run –d nodered/node-red”
3. Echoes the process id name.
4. Notice that this node app is exposed through port 1880.
5. Write “docker logAlso notice the folders.
6. Run “docker kill [process id]” to kill the running instance.

### Running with port mapping to host system

The container may contain services that might be reached from the host system. But these need to be exposed explicitly and mapped to a host port.

1. In a cmd.
2. Run “docker run –d nodered/node-red”
3. Echoes the process id name.
4. Notice that this node app is exposed through port 1880.
5. Also notice the folders.
6. Try to access the node application “localhost:1880”
7. This will not work because the port is not exposed to the host system.
8. Run “docker kill [process id]” to kill the running instance.
9. Run “docker run –d –p 1880:1880 nodered/node-red”
10. Try to access the node application “localhost:1880” again.
11. Now it works.
12. The format for the –p switch is :  
    “-p [hostport1]:[containerport1] [hostport2]:[containerport2]…….”.  
    Note: The order of the option switches must be before the other params.

### Running with folder mapping to host system

For earlier versions of Docker, the containers doesn’t persist their state. So anything done inside a running container is lost when the container is stopped/restarted or the host system is restarted. This is different in newer versions, but when running the container I Kubernetes, this is still true.

Try running node-red and make an API endpoint.

1. Make sure the node-red container is running (“docker ps”).
2. Go to the local address “localhost:1880”, and drag a “http in” and a “http response” box from the “network” part of the left menu onto the flow.
3. Drag a connection between the boxes.
4. Double click on the in box to set the url. Eks. “URL: /test”
5. Press the red “Deploy” button, and test the endpoint. “localhost:1880/test”
6. If you now kill the container and restart it again, the flow canvas is empty. So anything you do, is lost upon restart.
7. Kill the container again and start it with the following cmd line:  
   “docker run -it --rm -v C:\Users\Admin\dockerNk8s\node-red\data:/data -p 1880:1880 -d nodered/node-red”. ***Change the highlighted path to a local path that exist on your machine.***

## Making your own container

Make a docker image and push this to dockerhub.

1. Clone <https://github.com/kurt70/dockerk8sworkshop.git>
2. In cmd/bash/etc. go to the WebAppTest/ WebAppTest folder where the dockerfile is.
3. Compile the image by writing :  
   “docker build -t [yourdockeraccountname]/webapptest:latest .”
4. Run “docker images” to see your new image in your local repository.
5. To run it, “docker run -p 8080:80 -d [yourdockeraccountname]/webapptest”  
   If your port 8080 is used, change the port mapping to an available port.
6. In a browser go to “<http://localhost:8080>”.
7. Note that the program writes the hostname/machine name of the instance. We’ll use this later on in the k8s part.
8. Use “docker images” and note the size of the image. 261mb is huge.
9. Try finding a better and smaller base image. Ideally, the image should be 50-60 mb….
10. Try to push the image to your own dockerhub repository. Well need this later.

# Kubernetes & Minikube

### Exercise 2.1

First make sure that these applications are running:

* Minikube
* Docker desktop

Make also sure that the following addons is installed:

* Metrics-server
* Ingress
* Dashboard (optional)
* Ingress-dns (optional)  
  <https://github.com/kubernetes/minikube/tree/master/deploy/addons/ingress-dns>

1. Try to run the container we made earlier, now inside minikube.  
   “kubectl run webapptest --image=[imagename]--port=80 --expose=true”  
   NB! Use the full uri for the image.
2. Use “kubctl get all” to get information about the most relevant k8s artifacts.  
   Note the ip address on the service.
3. Try to curl the ip address for the webapptest service.   
   (This should not work, because the service is not attached to minikubes network adapter. The service is only available to other pods in the cluster)
4. Try using “kubectl port-forward svc/webapptest 8080:80”
5. Try “Curl <http://localhost:8080>”.
6. Exit port-forward by using CTRL+C.

### Exercise 2.2

Now we need to expose the service outside of our minikube vm and also change to using yaml files.

1. Expose the pod via a new NodePort service.  
   “kubectl expose deployment webapptest --type=NodePort --name=webapptest”.
2. Then create a template file by getting the yaml config from kubernetes.  
   “kubectl get svc/webapptest –o yaml > webapptest-svc.yaml”
3. This file include now all parameters including all default values. A lot in this file can be removed. Look at the service part of the “webapptest.yaml” file in the course repo to see what is needed.
4. Delete the old service : “kubectl delete svc/webapptest”  
   This is because You can’t mix modes in how you create a kubernetes artifact. You either create by using run/expose etc. Or declarativly by using apply on yaml files.
5. Add the new service by using the yaml file you just made.
   1. First test : “kubectl apply –f webapptest-svc.yaml –dry-run”
   2. Then run: “kubectl apply –f webapptest-svc.yaml”
6. Try “kubectl get svc/webapptest –o wide”
7. Try “kubectl describe svc/webapptest –o wide”
8. Try using a helper function in minikube “minikube service webapptest”.
9. Did it work?
10. Do the same thing for the deployment. Extract it to a “webapptest.yaml” file.
11. Clean up the file and copy the text for the svc file
12. In general, all kubernetes object related to one deployable item, should be in one file separated by “---“.

### Exercise 2.3

Now we need to expose the service outside of our minikube vm by using a hostname and an path.

1. Enable the ingresscontroller if you havn’t done this yet:  
   “minikube addons enable ingress”.
2. Run “kubectl get pods –all-namespaces”.
3. Note that there are several pods in other namespaces. These are system pods.
4. Make sure that the ingresscontroller is working fine.
5. Copy the ingress section below to your “webapptest.yaml”.
6. Apply the ingress by using:   
   “kubectl apply –f webapptest.yaml –dry-run”.
7. If all is ok, run apply.
8. Check the ingress with “kubectl get ing”. After a little while, the external ip for the ingress should be shown. So refresh a couple of times.
9. Add an entry to the host file with the ip address in the ingress.  
   “xxx.xxx.xxx.xxx webapptest.home”.
10. Test the url.
11. If this doesn’t work:
    1. If the ingress still hasn’t been assigned an ip address, something is wrong with the network setup in minikube/your machine. This might take several minutes
    2. If that is ok, it’s an issue with the svc/ing/pod setup.
       1. Check that the svc works by using port-forward.
       2. Check that the service name in the ingress is exactly the same as in the service.
       3. Check that the service’s port is the same as the one given in the ingress. Alternativly the services port can be given a name, and this can be used in the ingress instead.

apiVersion: networking.k8s.io/v1beta1 # for versions before 1.14 use extensions/v1beta1

kind: Ingress

metadata:

name: webapptest

spec:

rules:

- host: webapptest.home

http:

paths:

- path: /

backend:

serviceName: webapptest

servicePort: http

### Exercise 2.4

Add configuration to our app.

1. The webapptest api has a “/config” route. Use this to check if the config works.
2. Try adding config and secrets.
3. Use the files in “LoadTest/kubernetes/” folder to see how to ad ConfigMap and Secrets to your deployment template.
4. Redeploy by using apply.
5. Config map changes and env changes doesn’t trigger a recreation of the pods, so you need to delete the Pod(s) for the changes to appear.

### Exercise 2.5

Mount persistent storage.

1. Use “LoadTest/kubernetes/pi-calculator-api.yaml” as an example and Mount a folder to the webapptest deployment in your yaml file.
2. Reapply the yaml file. And check that everything is ok. (“kubectl describe [podname]”)
3. Use “kubectl exec –it [podname] bash” to ssh into the pod and try to add a file in the mapped folder.
4. Use “kubectl get pv” to get the name of the volume.
5. SSH into the minikube vm “minikube ssh”
6. Look in the “/mnt/sda1/hostpath-provisioner/[pv name]”, and you should find your files.

### Exercise 2.6

Try the loadtest.

1. In “\LoadTest\kubernetes” run the “apply.ps1” powershell script. If you don’t have powershell installed. Run the commands manually.
2. Chronograf is a timeline db frontend/dashboard, that the pi-calculator-api logs execution time to.
3. pi-calculator-api has a optional “?digits=xx” param. Try changing number of digits.
4. Try to run a load test