Kubernetes Workshop - Step by Step Proctor Guide

1 - Run a Docker container

- Pull the redis image from the online Docker store

```
docker pull redis
```

- Inspect that the image is in your local repository

```
docker image ls
```

- Run the image, making sure to expose port 6379

```
docker run --name myredis -p 6379:6379 --rm -d redis

// use with --rm flag if desired to cleanup immediately after stop: docker run --name myredis -p
6379:6379 --rm -d redis
```

- Install a redis client to test with

```
// on windows, using chocolatey:
choco install redis-64

// on linux, using apt:
sudo apt install redis-tools

// on mac:
brew install redis
```

- Verify that redis is running

```
// list the containers in docker
docker ps

// see details
docker inspect myredis

// try connect with the redis cli, check if redis-cli connects and if counter increases
redis-cli
> incr mycounter
> incr mycounter
```

- Stop the container and start it again; notice that state is still there

```
docker stop myredis
docker start myredis
redis-cli
> incr mycounter
```

- Clean up

```
docker stop myredis
docker rm myredis
```

2 - Build a Docker container

For the second part, the source code for the container to build is published on GitHub here: https://github.com/Azure-Samples/azure-voting-app-redis/tree/master/azure-vote

The easiest way to get this source code on a local machine is to clone the repo, like this:

```
// when using ssh (make sure you have keys setup correctly):
git clone git@github.com:Azure-Samples/azure-voting-app-redis.git

// when using http:
git clone https://github.com/Azure-Samples/azure-voting-app-redis.git
```

The directory from which to work would be .\azure-voting-app-redis\azure-vote\. The source files to build our image with reside in azure-vote, next to which there is a Dockerfile with which the image can be build.

- Inspect the Dockerfile:

```
FROM    tiangolo/uwsgi-nginx-flask:python3.6
RUN     pip install redis
ADD     /azure-vote /app
```

> TODO WE COULD ASK PEOPLE TO MODIFY THE DOCKER FILE TO RUN UNDER A CERTAIN USER OR HAVE THEM ADD SOMETHING

- Build the image:

```
// build:
docker build -t myname/azure-vote .

// check if image is in local docker repo:
docker image list
```

- In order to run our web application, it needs to know how to find the Redis server, which is running in a different container on a different ip address. It detects this from an environment variable called REDIS. This means we need to do two things:
    - find the ip address for the REDIS container: docker inspect myredis. Let's assume the ip is: 172.17.0.2. (It will be different on your machine.)
    - start the web application with the REDIS environment variable: docker run --name myvoteapp -e REDIS=172.17.0.2 -p 8080:80 --rm -d xstof/azure-vote
- test the web application by going to http://localhost:8080
- stop both containers:

```
docker stop myvoteapp
docker stop myredis
```

> OPTIONAL: build both images and push them to a private repo in Azure Container Registry

3 - Deploy a Kubernetes cluster

In this section, we'll spin up our own Kubernetes cluster in Azure, so we can deploy applications on it. We'll use the Azure cross-platform command line interface to do so.

- create a Resource Group called gbbk8s (pick a suitable name yourself):

```
az group create -n gbbk8s -l northeurope
```

- spin up your Kubernetes cluster:

```
# using ACS:
az acs create --name gbbk8scluster -g gbbk8s --orchestrator-type kubernetes --generate-ssh-keys

# using AKS:
# az aks create -n ccaks -g ccaks -l westus2 -k 1.8.1 --generate-ssh-keys
```

- install the Kubernetes kubectl command line interface:

```
az acs kubernetes install-cli
```

- before you connect the new cluster to your local environment, inspect the available Kubernetes contexts: kubectl config get-contexts
- note that the new cluster is not in there
- now connect the new cluster to the kubectl cli: az acs kubernetes get-credentials -g gbbk8s -n gbbk8scluster
- verify that you are connected: kubectl config get-contexts and get the nodes: kubectl get nodes
- now browse to the web-based admin interface, which takes you to a local page like http://127.0.0.1:8001/ui:

```
// using the azure cli:
az acs kubernetes browse -n gbbk8scluster -g gbbk8s

// using the k8s cli:
kubectl proxy
```

4 - Deploying on Kubernetes

Now that we have our cluster up and running, it's time to start deploying a workload on there. As a sidenote, for easy Kubernetes yaml manifest generation, one might want to install a yoman generator like npm install -g generator-kubegenor Kubernetes support for VS Code from: https://marketplace.visualstudio.com/items?itemName=ipedrazas.kubernetes-snippets.

Deploying a single Pod

To warm up, we'll kick off with launching just a single pod onto the cluster, running Redis. Create your manifest like this:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: azure-vote
  name: myredis-pod
spec:
  containers:
    - name: myredis-container
      image: redis
```

Deploy the pod and inspect that it has been deployed:

```
kubectl create -f single-pod.yaml
kubectl get pods
kubectl describe pod myredis-pod
```

At this point, we just have it running but there's no port exposed for us to connect to. We don't want to publish this port wide open to the public though: it should just be accesible to our web site. We'll do that next during out deployment.

For now, if you like to test that the Redis pod works, use the kubectl port-forward feature, like this:

```
kubectl port-forward myredis-pod 6379:6379
```

Then try some Redis commands locally from your machine.

Deploying multiple pods using a Deployment

Use the following container images when creating the Kubernetes deployment:

- Redis: redis
- Frontend: microsoft/azure-vote-front:redis-v1

Use the environment variable "REDIS" and as its value the hostname for the Redis container to indicate how the frontend can communicate with Redis.

The yaml for the deployment could look like this:

- Backend:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name:  voting-app-backend
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app:  voting-app
        tier: voting-app-backend
    spec:
      containers:
      - image:  redis
        name: voting-app-redis
        env:
        - name:  ENVVARNAME
          value:  ENVVARVALUE
        ports:
        - containerPort:  6379
          name:  redis-port
```

- Backend Service:

```
kind: Service
apiVersion: v1
metadata:
  name:  voting-app-backend-svc
spec:
  selector:
    app: voting-app
    tier:  voting-app-backend
  type:  ClusterIP
  ports:
  - name:  redis-port
    port:  6379
    targetPort:  6379
```

```
kubectl create -f ./votingapp-backend.yaml
kubectl create -f ./votingapp-backend-svc.yaml
kubectl get pods
kubectl get svc
```

- Frontend:

```
apiVersion: extensions/v1beta1
kind: Deployment
```

```
metadata:
  name:  voting-app-frontend
spec:
  replicas: 3
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app:  voting-app
        tier: voting-app-frontend
    spec:
      containers:
      - image:  microsoft/azure-vote-front:redis-v1
        name: voting-app-frontend
        env:
        - name:  REDIS
          value: "voting-app-backend-svc"
        ports:
        - containerPort:  80
          name:  frontend-port
```

- Frontend Service

```
kind: Service
apiVersion: v1
metadata:
  name:  voting-app-frontend-svc
spec:
  selector:
    app:  voting-app
    tier: voting-app-frontend
  type:  LoadBalancer
  ports:
  - name:  http
    port:  80
    targetPort:  80
```

```
kubectl create -f ./votingapp-frontend.yaml
kubectl create -f ./votingapp-frontend-svc.yaml
kubectl get pods
kubectl get svc

// go into a pod and check environment vars for REDIS:
kubectl exec -p voting-app-frontend-4212741653-4z8wm -ti /bin/bash
printenv

// try out the web app:
kubectl port-forward voting-app-frontend-4212741653-0t6n2 8080:80
```

> Side note: For troubleshooting dns resolution, you might need nslookup To install this into a pod: apt-get update apt-get install dnsutils -y

You may want to stream the logs of the frontend while you're testing the application, like this: kubectl logs deployment/voting-app-frontend -f

Troubleshooting Guide

Working with the Kubernetes Dashboard:

- Run the following command: kubectl proxy
- Open the ip given 127.0.0.1:8001/ui
- Append to the redirected URL that opens, a forward slash: /

> If Kubernestes Dashboard does not open, break the kubect proxy connection and reestablish it and try again with / and the end of the URL

Working with the Azure Free Tier:

Have people register the right Azure Resource Providers in the Azure portal:

- Microsoft.Compute
- Microsoft.ContainerService
- Microsoft.Storage
- Microsoft.Network

```
az acs create --orchestrator-type kubernetes --resource-group workshopno --name myK8SCluster --
agent-count 1 --generate-ssh-keys --verbose
```

> The free account does *not* work in West UK

Sharing your C Drive:

Please see: https://blogs.msdn.microsoft.com/stevelasker/2016/06/14/configuring-docker-for-windows-volumes/

Dealing with Anti-Virus

Symantec AV protection can block az acs get-credentials command while running inside Linux sub system for Windows. Sudo apt-get update also fails.