Kubernetes Workshop Attendee Guide

Kubernetes Hands-on Lab developed by the Azure GBB Team

The objective of the workshop is to get hands-on with the content covered during the day . Attendees will get a set of resources delivered as a starting point, and then go from there to continue building functionality.

Technologies covered:

- Docker tooling
- Azure Container Services
- Kubernetes

Pre-Requisites

- *OS:* For people not using Linux or Mac please make sure you have the latest version of Windows and have enabled and tested the Subsystem for Linux. Install it from the store using these instructions: https://msdn.microsoft.com/en-us/commandline/wsl/install_guide
- *CLI:* please install the azure cli from here: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
- *Docker:* install Docker CE from here: https://docs.docker.com/engine/installation/
- *Git:* install git using your package manager of choice
- *VS Code:* install visual studio code: https://code.visualstudio.com/

1 - Run a Docker container

Scenario

In this section, you'll start off with running a simple docker container on your local machine. The container you'll run is backed by a simple Redis image which can just be referred to as "redis".

Steps

1. Pull the image from the public Docker registry and make sure you see it in your own local repo
2. Run a Redis container, exposing port 6379

> Tip: find Docker documentation for:
>
> - pulling images here
> - starting images here

Exit Criteria

- a working container, which you can test by installing a Redis client:

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

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

// on mac:
brew install redis
```

- verify you can connect and work with the Redis container:

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

- Stop and start the container again, make sure it retains the state where you left off
- Stop and delete the container

2 - Build a Docker container

Scenario

In this part, you'll build your own container image and run it.

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

Steps

1. Clone the repo locally on your machine

```
// when using ssh (make sure you have keys setup correctly; if not you might want to try http instead):
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
```

2. Go into this working directory: .\azure-voting-app-redis\azure-vote\
3. Look for the Dockerfile with which to build the image:
    - Inpect it
    - Build an image, tagged "yourname/azure-vote"
4. Spin up a Redis container (see Part 1), find the ip for it and run your freshly built "yourname/azure-vote". Make sure when you run it, to provide an environment variable with as its name REDIS and as its value the ip for the Redis container. This is how our frontend knows where to call into Redis.

> Tip: find Docker documentation for building images here

Exit Criteria

- Verify that you can see your image in your local repository
- You have a successful running web application on http://localhost:8080

3 - Deploy a Kubernetes Cluster

Scenario

We're now ready to spin up our own Kubernetes cluster in Azure to deploy our containers upon. As a first step, we'll deploy an empty cluster.

Steps

1. Create a resource group to deploy your cluster into.
2. Create a Kubernetes cluster, using ACS ("Azure Container Service")

> Tip: when deploying through the CLI, there's an option to have it create/reuse and store the ssh keys automatically, which is quite handy

3. Install the kubectl command line environment

> Tip: you can do so through the az azure command line interface

4. Configure kubectl to point to your freshly created cluster

> Tip: also this can be done from the az command line

Exit Criteria

- You have a working Kubernetes cluster and are able to access the web interface from Kubernetes on http://127.0.0.1:8001/ui
- kubectl get nodes provides you the list of nodes
- kubectl cluster-info gives you information on your cluster

4 - Deploying on Kubernetes

Scenario

Now that we have our cluster up and running, it's time to start deploying a workload on there.

> Tip: You might want to install Kubernetes support for VS Code from: https://marketplace.visualstudio.com/items?itemName=ipedrazas.kubernetes-snippets to ease you with yaml file creation.

4a - Deploying a single Pod

Steps

1. Create a manifest for deploying a single pod, containing just our Redis container. (Note that this is not done in production; but it's good for learning purposes to see how Kubernetes concepts stack up.)

2. Deploy the pod onto the cluster using kubectl

> Tip: find the documentation for Kubernetes Pod resources online here

Exit Criteria

1. Make sure you can see the new pod on the cluster
2. You are able to test it works by connecting to the pod with the Redis command line (redis-cli) and using kubectl port-forward, exposing port 6379 to your local machine

Advanced Scenario

1. Try deleting the pod or the container within it and see if it comes back up

4b - 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.

Steps

1. Create a yaml file for the backend (Redis) Deployment and deploy it, exposing port 6379 on the pod level
2. Create a yaml file for the backend (Redis) Service and deploy it. The service should not be accessible from outside the cluster and should expose Redis' port 6379
3. Create a yaml file for the frontend Deployment and deploy it, exposing port 80 on the pod level and making sure it knows how to find the Redis' backend
4. Create a yaml file for the frontend Service and deploy it, exposing port 80 to the public, making sure it is backed by a Load Balancer in Azure

> Tip: find the documentation for:
>
> - Kubernetes Deployment resources online here
> - Kubernetes Service resources online here

Exit Criteria

1. You have a fully working web application, exposed on the internet

Advanced Scenario

1. Try deleting the pod or the container and make sure it comes back up automatically