# 

# **Global Azure Bootcamp 2017 – Introduction to Azure Container Service**

## Overview

In this hands-on lab you can re-enforce your knowledge by doing it yourself and gain hands-on experience on the things you learned. Before we can start we first need to have a look at setting up a machine and Azure.

The Exercise files are located at <https://github.com/XpiritBV/GABC2017_HandsOnLabs>

Clone the repo locally to get started.

## Setup

#### Desktop PC for container creation

To work with containers you need a Windows PC with Windows 10 Aniversery edition or higher + Hyper-v, or a Windows PC with Windows server 2016.You can use an Azure virtual machine to run the labs, if you choose for that option, please use a VM with Windows Server 2016 + containers.

#### Azure setup

If you already have a Microsoft Azure subscription, you can skip this section. Otherwise, follow these steps to create a free trial subscription. You will need to provide a valid credit card number for verification, but you will not be charged for Azure services – for more information, see the [frequently asked questions](https://azure.microsoft.com/en-us/pricing/free-trial-faq/) on the Azure sign-up page.

1. If you already have a Microsoft account that has not already been used to sign up for a free Azure trial subscription, you’re ready to get started. If not, don’t worry, just [create a new Microsoft account](https://signup.live.com/).
2. After you’ve created a Microsoft account, create your [free Microsoft Azure account](http://aka.ms/edx-dat202.2x-az). You’ll need to sign-in with your Microsoft account if you’re not already signed in. Then you’ll need to:
   * Enter your cellphone number and have Microsoft send you a text message to verify your identity.
   * Enter the code you have been sent to verify it.
   * Provide valid payment details. This is required for verification purposes only – your credit card won’t be charged for any services you use during the trial period, and the account is automatically deactivated at the end of the trial period unless you explicitly decide to keep it active.

## Exercise 1: Run a docker container on Windows

In this exercise you will run a container to inspect the concepts of inmutabillity.

Docker.

Using steps on the command-line

Verifiy thay docker is correctly installed by running

* docker version

Start by running a simple windows containr with the commandprompt in it.

* docker run -it microsoft/windowsservercore cmd

Then inside the newly started container run the following commands

* dir
* md demo
* cd demo
* copy con myfile.txt
* Hello this is in my docker container
* ctrl+c
* dir

Now you should see the text file you created in the steps above

Exit the container by running

* ctrl+p ctrl+q

When back on the host run the following command

* docker ps

Now you should see a container running with a generated name

Verify that you don’t see the file on your host by going to c: and running dir.

To stop the container run “docker stop name-of-the-container”. Then use

“docker ps” to view all running container and use “docker ps -a” to view all containers even stopped ones.

Now run “docker run -it microsoft/windowsservercore cmd” again. Inside the container run “dir”.

Verify that the folder and file your created earlier in not there anymore.

## Exercise 2: Build a new Container Image

In this exercise you will go through the steps of creating a new Container image first from the commandline using Docker commit and in a second step by using a docker file and the docker build command.

Using steps on the command-line

* docker run -it --name mycontainer microsoft/windowsservercore cmd
* md builddemo
* cd buidldemo
* copy con myfile.txt
* this is a text file
* ctrl+c
* exit
* docker ps -a
* docker stop mycontainer
* docker commit -m mycontainerimage mycontainer
* docker images
* docker run -it --name mycontainer microsoft/windowsservercore cmd
* dir
* exit
* docker run -it mycontainerimage cmd
* dir

you see the demo folder again

#### Now using the docker build file

Start with creating a folder. Inside the folder create 2 files. One called dockerfile and another called index.html

Edit both files. The index.html can contain any valid html you like. The dockerfile should contain the following content.

#### FROM microsoft/iis

#### WORKDIR /inetpub/wwwroot

#### ADD index.html index.html

Ensure you are in the folder you just created that contains the docker file

* Docker build -t mynewiiscontainer .
* Docker images
* Docker run -d -p 80:80 mynewiiscontainer
* Docker ps -> find the running container ID
* Docker inspect containerID
* Find the IP address

Start a browser to the found IP adres and see the new website running on port 80 serving the index.htm file that is in the current folder.

## Exercise 3: **Using docker-compose to orchestrate multiple containers**

In this exercise you will create multiple container instances that have a dependency. You will create a blog website that has a dependency on a database that also run’s in a container. The required files for this demo can be found in the folder Exercise 3 of the Github repository you cloned earlier.

In the folder exercise3 run the following commandline:

* Docker-compose up

Start a new command prompt to inspect the running containers

Inspect the running containers and find the ip address of the web frontend

* Docker ps
* Docker inspect containnerID

Use a browser to goto the blogsite running in the containers (<http://containerIP/blogsite>)

Ensure you are in the exercise3 folder so we can clean up the containers

* Docker-compose down
* Docker ps
* Docker ps -a

**This is the end of the exercises for part 1. We will proceeed with the next labs after the next theoretical introduction.**

## Exercise 4: Create an ACS Cluster

In this exercise, you create a Docker Swarm cluster by using Azure Container Service and an Azure Resource Manager template.

**You need the files that are vailable for you at the folder exercise4\acs-swarm-linux.**

To start off, we need to fill out the parameters.json file. Some important parameters are:

* (click) "dnsNamePrefix" must be unique for the region you are deploying your cluster into, it will become the DNS name of your cluster. Here we'll use 'acsswarmcluster'.
* (click) "orchestratorType" specifies what cluster orchestrator to use, Docker Swarm, DC/OS or Kubernetes. Here we'll go with Swarm, to indicate Docker Swarm.
* (click) "sshRSAPublicKey" is the public part of a public/private key pair, that will be used to authenticate the administrator to the cluster when using SSH. To generare an SSH key on windows use the instructions at <http://bit.ly/1LhUhPj>.
* (click) "enableDiagnostics" indicates whether or not a diagnostics extension should be configured for the Virtual Machines in the cluster. Learn more about VM extensions in chapter 2.4. In this demo we'll enable diagnostics.

## Run

We are now ready to begin the deployment of the cluster. We can do this by executing a PowerShell script. Usually this script will be called "deploy.ps1". This script takes some additional parameter values that indicate the (click) subscription, (click) region and (click) resource group to create the cluster in. It also takes a (click) deployment name, which we can use to track progress.

I have configured the deployment to run in Verbose mode, so you can see the resources while they are being created.

Now let's go ahead and run the script, and see what happens.

(alt + tab to switch to PowerShell ISE and F5 to execute)

After some time you can see that the cluster creation has succeeded. You can also see some url's at which you can reach your cluster, using HTTP or SSH.

## Validate

If we now go into the Azure Portal, and navigate to the resource group DockerSwarm-ACS - the name we passed to the script earlier -, you can see your brand-new cluster.

**This concludes this part. We will continue after the next theoretical part.**

## Exercise 5: using docker-compose on the previously created cluster

## Intro

In this exercise you will use Docker Compose to deploy an application to a Docker Swarm cluster.

## Using putty to connect

In the Azure portal got to the resourcegroup that was created in Exercise 4. Click the container service resource, and click Overview. The Master FQDN of the cluster appears under Essentials. Save this name for later use.

Now start Putty.

1. Enter a host name that is comprised of the cluster admin user name and the public DNS name of the master in the cluster. The Host Name looks similar to adminuser@PublicDNSName. Enter 2200 for the Port.
2. Select SSH > Auth. Add a path to your private key file (.ppk format) for authentication. You can use a tool such as PuTTYgen to generate this file from the SSH key used to create the cluster.
3. Select **SSH > Tunnels** and configure the following forwarded ports:
   1. Use port 2375 for Swarm
   2. Use localhost:2375 for Swarm.

Next use the command “docker ps” to show the master node has 2 containers running. The first container is called swarm. Note that this container has a port mapping from container port 2375, to host port 2375. This means that commands on the host that use that port, are processed by the Swarm container.

The second is called progrium/consul. This is used by Docker Swarm, to register running containers, so they can be discovered by other services.

## Connecting to Swarm

Let's now use the Docker cli to connect to Swarm. Remember that Swarm is compatible with the regular Docker CLI. To quickly demo this, list the containers that are running in the cluster, by executing docker -H:2375 ps. The argument 'H' indicates the target Docker deamon to connect to.

**run docker -H:2375 ps**

You see no containers are running, because this is a newly created cluster.

To show information about the cluster agent nodes type: docker -H:2375 info.

**run docker -H:2375 info**

Now you can see that this cluster has 2 agent nodes.

## Compose an application

Instead of using -H you can also specify an Environment Variable that indicates the deamon to use. I'll do this by typing export DOCKER\_HOST=:2375 After that, all commands will be directed at Swarm. **run export DOCKER\_HOST=:2375.**

Inside folder Exercise5 there is a simple compose file that will run a container from image "yeasy/simple-web", that hosts a simple web server. The yaml file describes what the deployment should look like, once deployed. We will now deploy this service to the cluster.

**docker-compose up -d**

You can see that both Agent nodes are now pulling the image. Once that completes, the web server will be run on one of the nodes.

**run docker ps**

One container is running!

## Validate

Because the container is running, we can now navigate to our Agent DNS name to see the web server in action.

**Open browser, navigate to:** <http://yourclusternameclusteragents.westeurope.cloudapp.azure.com>

## Scale container instance count

You can scale the deployment out by adding more instances of the container. This is done by using the command docker-compose scale web=2

**run docker-compose scale web=2**

When I now list the running containers, you'll see two running containers. One on each node.

## Remove deployment

To remove an application, simply execute docker-compose down. This will stop the containers and remove them

## Exercise 6: Monitor an ACS Cluster

In this exercise you learn how Azure Monitor can be used to send an email to the subscription owner, if the CPU load percentage crosses a configured threshold of 80%.

## Prepare

On the Azure portal, navigate to the Resource Group that contains the ACS Docker Swarm cluster we created earlier.

We want to be notified if the Agent nodes become too busy, so navigate to the Agent node VM Scale Set.

In the 'monitoring' node, select 'alert rules'.

**select monitoring/alert rules**

Because we have selected the resource group and vm scaleset before, those values are now already filled out.

On the new blade, click on 'add metric alert'.

**click add metric alert**

On the 'Add rule' blade, configure a new rule, based on metrics from the scale set. Let's call it 'cpu-load-warning'. Have it alert on 'metrics', for the selected resource. The metric we will use is 'Percentage CPU', but there are some other options; network and disk related.

**show dropdown**

We configure 80% as the threshold, and 15 minutes as the period. So if the CPU usage exceeds 80 percent for more than 15 consecutive minutes, we will be notified.

We can select email recipients based on RBAC settings, and by adding email addresses. It's also possible to configure a webhook. Doing this, will make Azure do an HTTP POST to the configured URL. The POST will contain JSON data with some details about the alert.

If we want to trigger automation based on this alert, we can select a 'logic app' to execute. I can use this to post a notification to Slack, for example.

When we click 'OK', the rule will be created.

**click ok**

## Azure Resource Manager

The Azure Portal has used the Azure Resource Manager to configure the alert rule. So if we navigate to the 'Automation script' node and select the 'template', you can see that there is a resource of type 'microsoft.insights/alertrules'. This is the definition or the rule we have created in this demo.

**Click Automation script, template, parameters('alert..., highlight type**

## Remove

To delete the alert rule, navigate to its details page and select 'delete', and confirm.

## Exercise 7: Scale the cluster

In this exercise you increase the Agent pools. First by using the Azure Portal and second by using the Azure CLI.

## Prepare

Create docker swarm linux cluster, if not there yet. Ensure 1 agent is in pool. Open putty terminal, load linux-acs, connect.

## Using the Azure Portal

On the Azure Portal, navigate to the ACS resource. Clicking on 'Agents' in the 'Settings' node, will show you the configuration of the Agent pool. Here you change the size of the VM's and the capacity of the VM scale set.

To change the Agent pool size, simply type a new value for 'vm count' between 1 and 100 and click on the Save button above.

When the deployment operation completes your agent pool is resized.

**Specify 2, and click save**

After the deployment completes, you can see that there are 2 active nodes, by running the following command on the master. docker -H:2375 info

**Switch to putty terminal. Type docker -H:2375 info, show 2 nodes are active.**

## Azure CLI (2.0)

You can also scale the cluster by using this command: az acs scale --resource-group dock erswarm-acs --name containerservice-DockerSwarm-ACS --new-agent-count 3

This will change the cluster named 'containerservice-DockerSwarm-ACS' out to 3 agent nodes.

**in powershell run acs-swarm-linux> az acs scale --resource-group dock erswarm-acs --name containerservice-DockerSwarm-ACS --new-agent-count 3**

After some time, the change has been applied. Your cluster will now have 3 agent nodes.

**Return to putty, repeat the command.**

This is also showing in the info returned by Swarm.