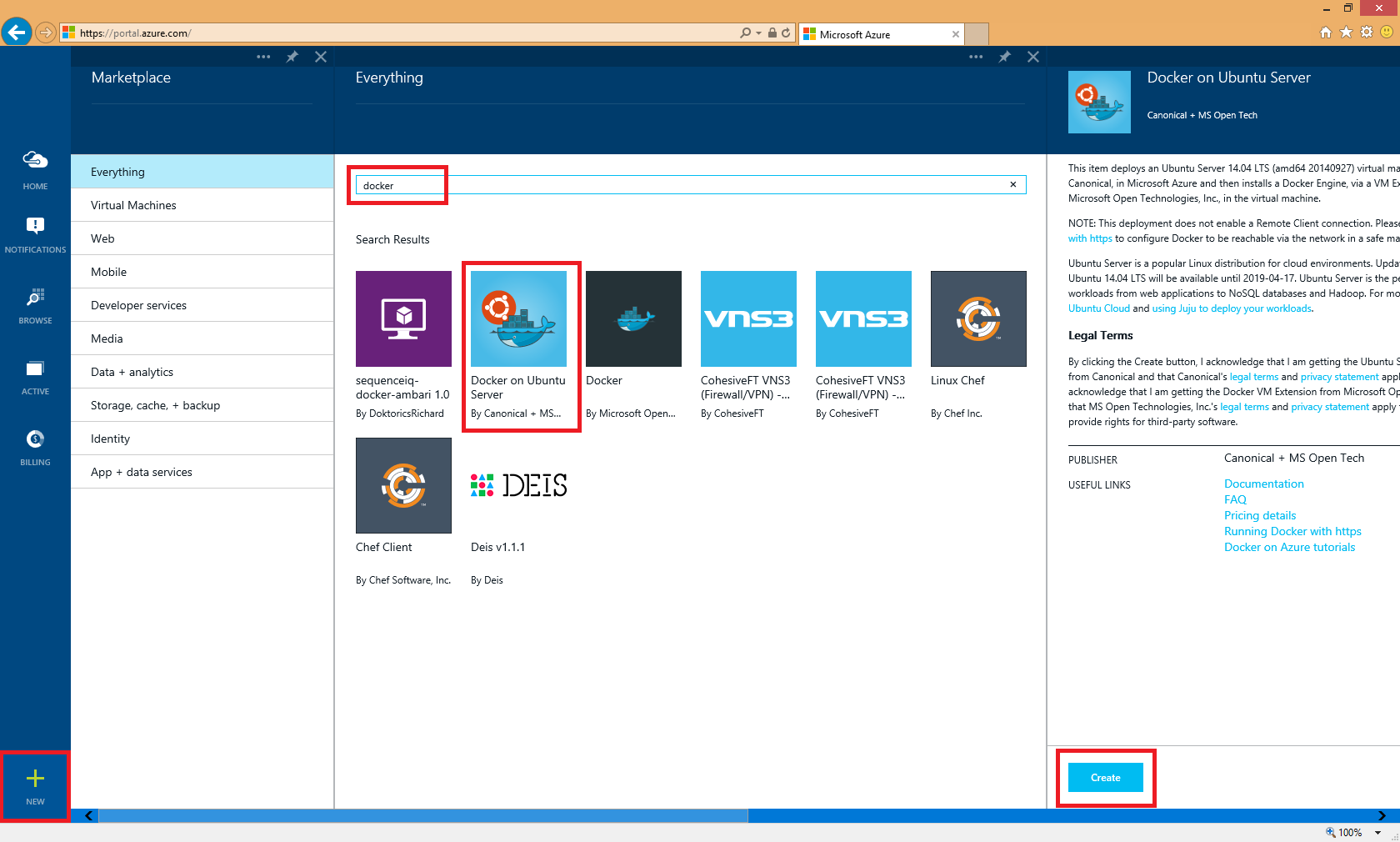
# TechReady 20 Docker Hands-On Lab

## Create your Docker VM

Logon on the Azure portal using your credential and create a new VM by using the Docker on Ubuntu VM that was built by MS Open Tech and Canonical.



Make sure you select the template called “Docker on Ubuntu Server”

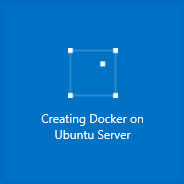
Click “**Create**” and fill the form as follows.

|  |  |
| --- | --- |
| Some of the following parameters have to be unique, use your own values:   * HostName: <Use\_your\_value> * UserName: azureuser * Password: P@ssw0rd! * Resource Group: TR20docker00 * Location: West US West US is used for performance reason during TR20. |  |

**Optional**: Review the network configuration of the resource group under “Optional Configuration”. You can review or modify the Domain Name, it will be used to connect later to the VM with SSH.

Click “**Create**” to start the creation of the VM.

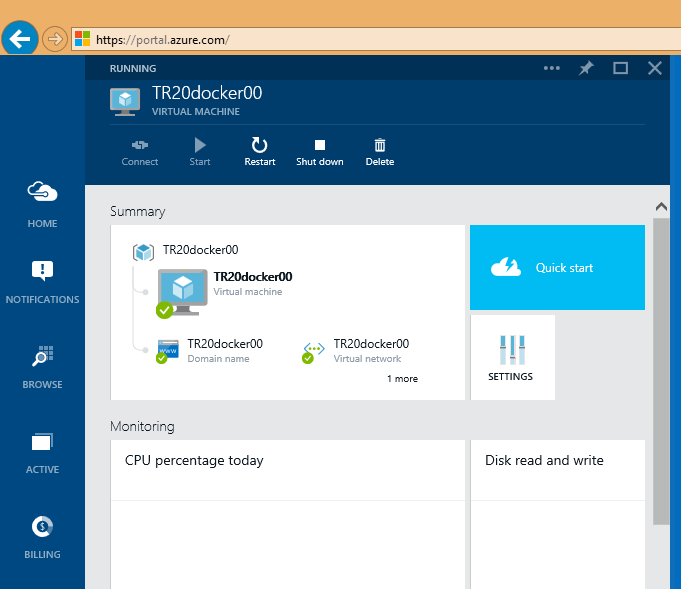
The creation of the VM shall appear as follows on the dashboard:



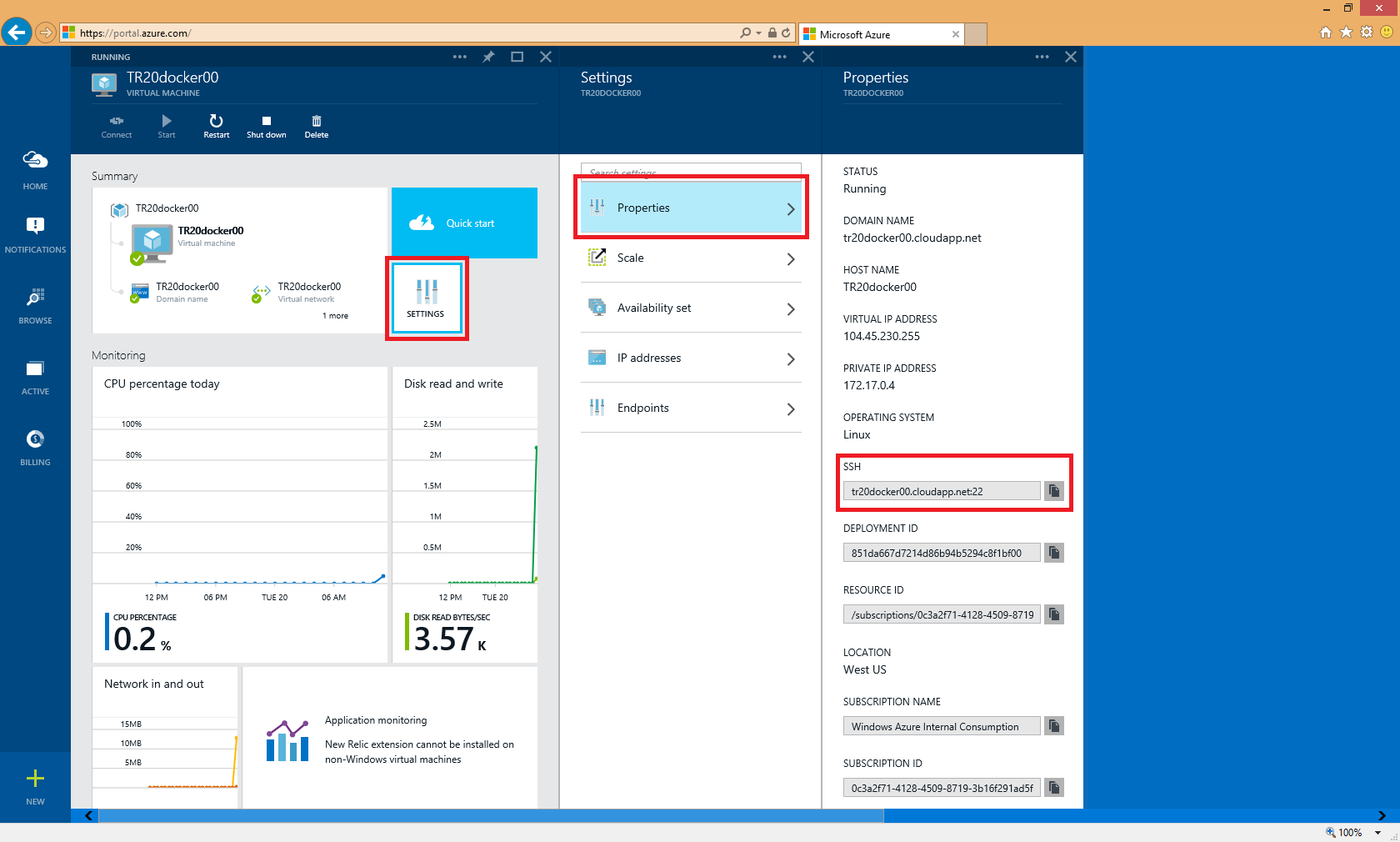
While the VM is being created, if you have not already done so, install Putty on your client machine.

Download Putty from here (official site): <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

Wait for the deployment to complete.

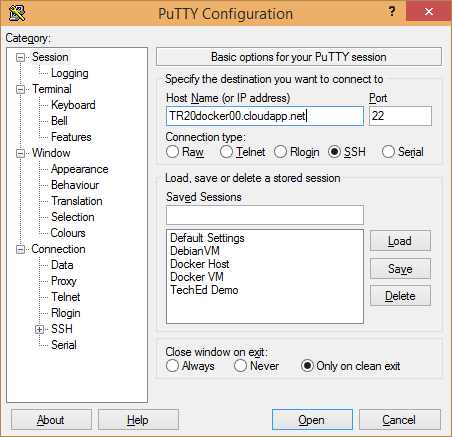
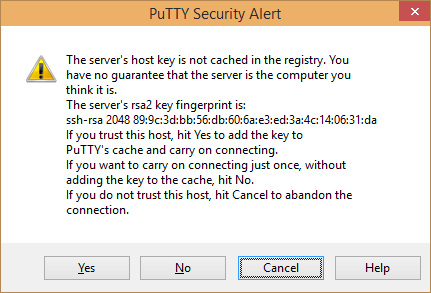


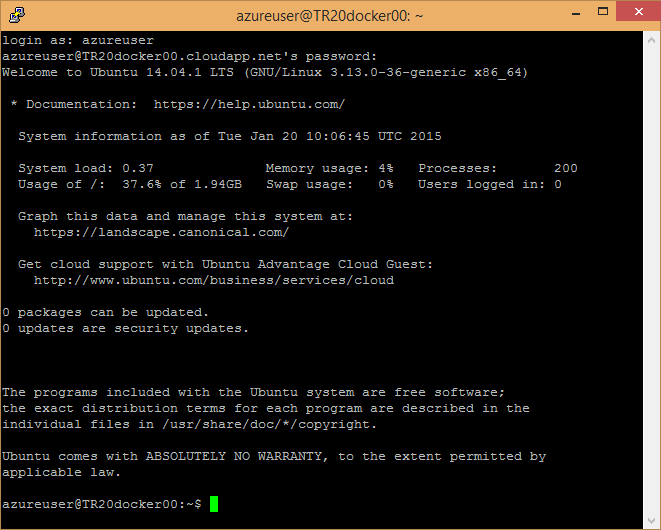
Click on **Settings** then **Properties** and note the **SSH** and the **Domain Name** values.



You can now connect to the VM using Putty by using the DNS Name and the port from the above.

Click **Yes** on the Security Alert.

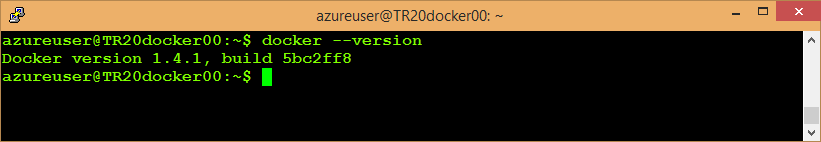
 



Verify that docker is installed and type:

**docker --version**

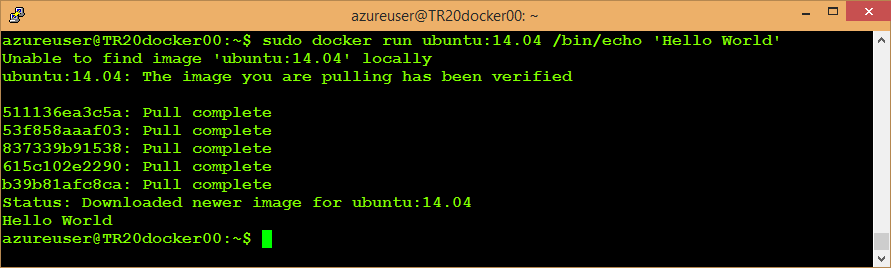
You should have the following result



## Lab#1 - Running your first container

We now have a Docker host, let’s use it to run the first image. From your SSH session, type the following command:

**$ sudo docker run ubuntu:14.04 /bin/echo 'Hello World'**



Note the following elements:

* The **docker run** command indicated to the docker host to run an image.
* We specified the image that we wanted to run, in our case **Ubuntu:14.04**.
* Docker looks for this image on the host, if not available, it will look for and download it from the public registry named “docker hub” (https://hub.docker.com/).
* The image will be downloaded (**Pull**) as well as the images that it depends on.
* Once the container launched Docker created a new Ubuntu 14.04 environment and executed the **/bin/echo** command inside it.

## Lab#2 - Running an ASP.NET application on Docker

In this lab we will run an ASP.Net application in a Docker container on the Ubuntu host that we have deployed. In order to do so, we will rely on the “base image” microsoft/aspnet (<http://registry.hub.docker.com/u/microsoft/aspnet/>) to build out application container.

Our container contains only the application and will rely on the other layers to run.

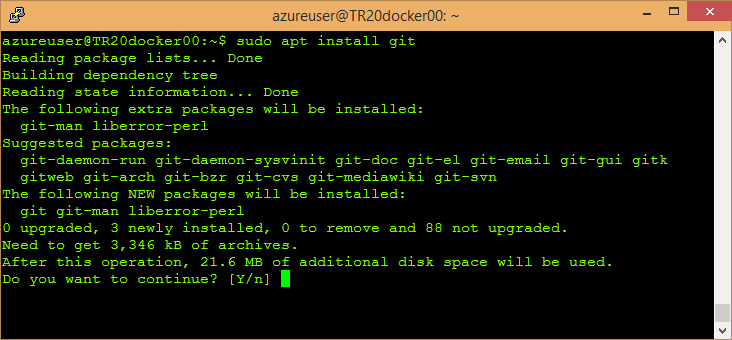
Update the packages on your VM:

**$ sudo apt-get update**

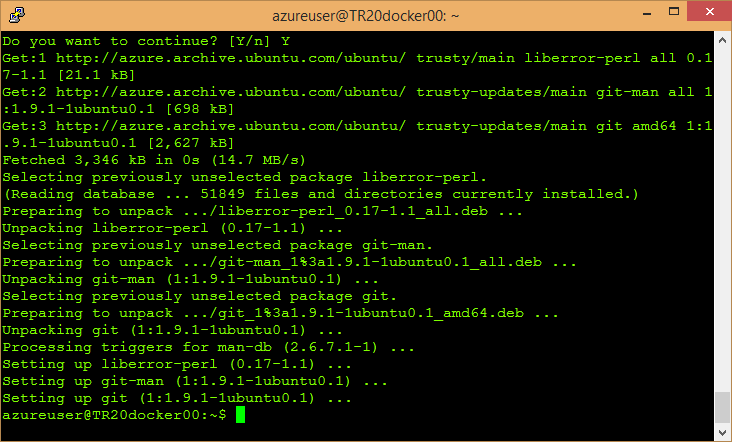
**$ sudo apt-get upgrade**

Install git (if not already done) by typing the following command from your SSH session:

**$ sudo apt install git**



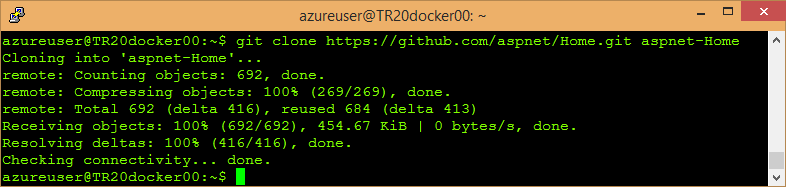
Type **Y** to accept the operation.



Now that we have git installed, start cloning the github repository

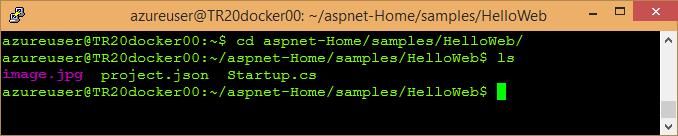
From the shell, type the following command:

**$ git clone https://github.com/aspnet/Home.git aspnet-Home**



Switch to the directory with our sample application by typing the following command.

**$ cd aspnet-Home/samples/HelloWeb/**



Create a new file called “**Dockerfile**” in the directory with the following command:

**$ vi Dockerfile**

Type "**i**" to start inserting text in the file. Then type the following text:

**FROM microsoft/aspnet**

**COPY . /app**

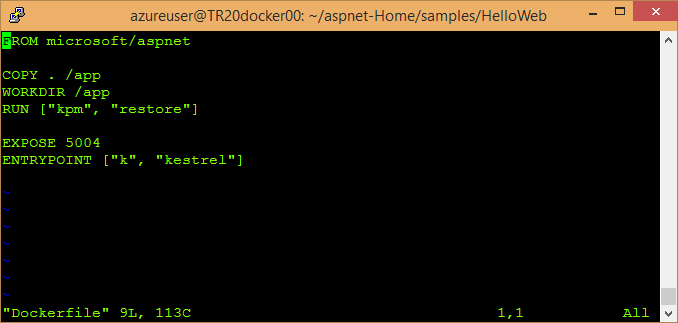
**WORKDIR /app**

**RUN ["kpm", "restore"]**

**EXPOSE 5004**

**ENTRYPOINT ["k", "kestrel"]**

The file will look like this:



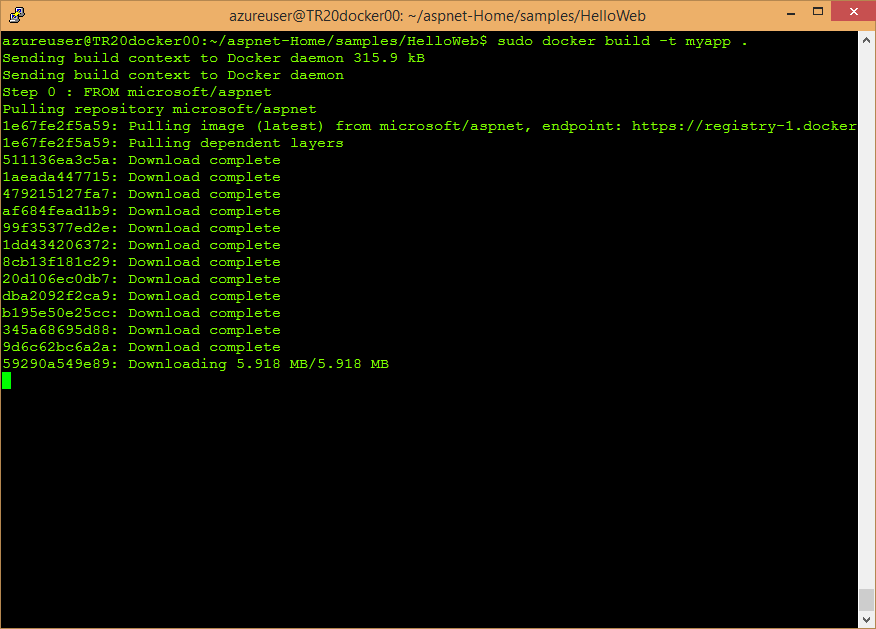
Save the file with the following set of commands:

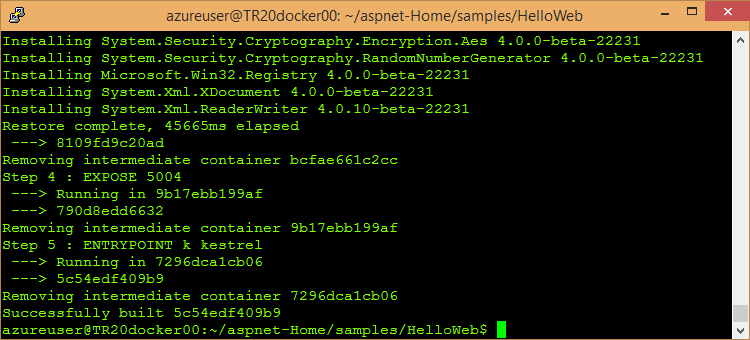
* press the “**Esc**” key
* type "**:wq**"
* press “**Enter**”

Once the file has been saved, type the following command to build the container of your application.

**$sudo docker build -t myapp .**

Note: Pay attention to the dot at the end of the command.



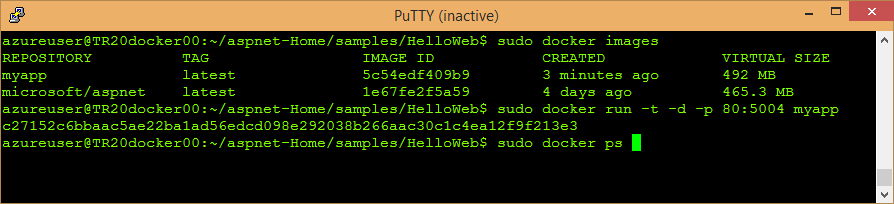


Now let’s check that we have the images on our host by typing the following command:

**$ sudo docker images**

Let’s run our application. The following command indicates to docker to run the container that we have just built.

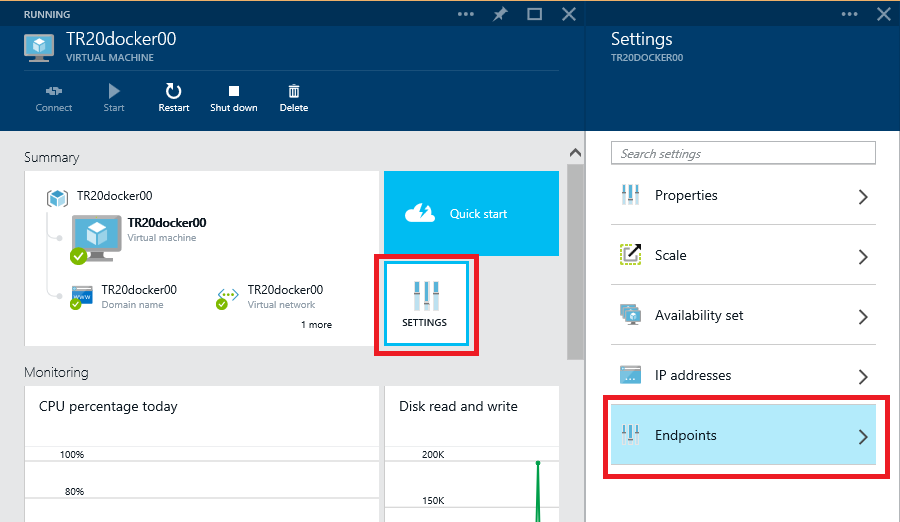
**$ sudo docker run -t -d -p 80:5004 myapp**



The application runs on port 5004 of the container which we map to port 80 of the host.

We now need to add the HTTP endpoint to our Azure deployment.

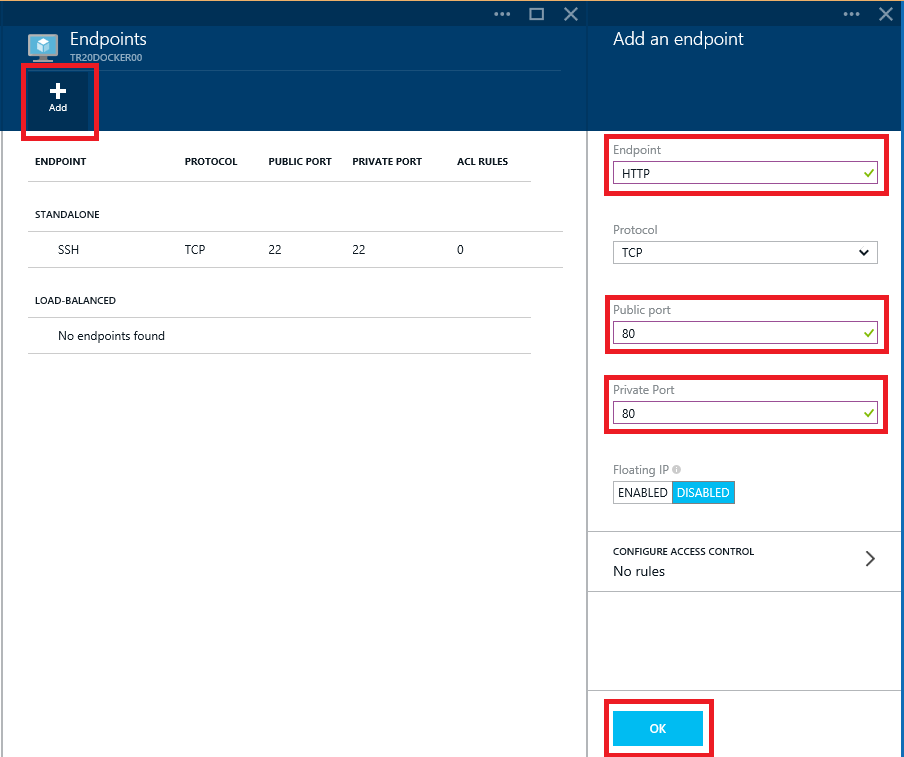
Switch to the Azure portal, select your Azure deployment and click “**Settings**”. In the Settings blade, click “**Endpoints**”.



In the “**Endpoints**” blade click “**Add**” and fill the fields as follows:

* Endpoint: HTTP
* Public port: 80
* Private port: 80

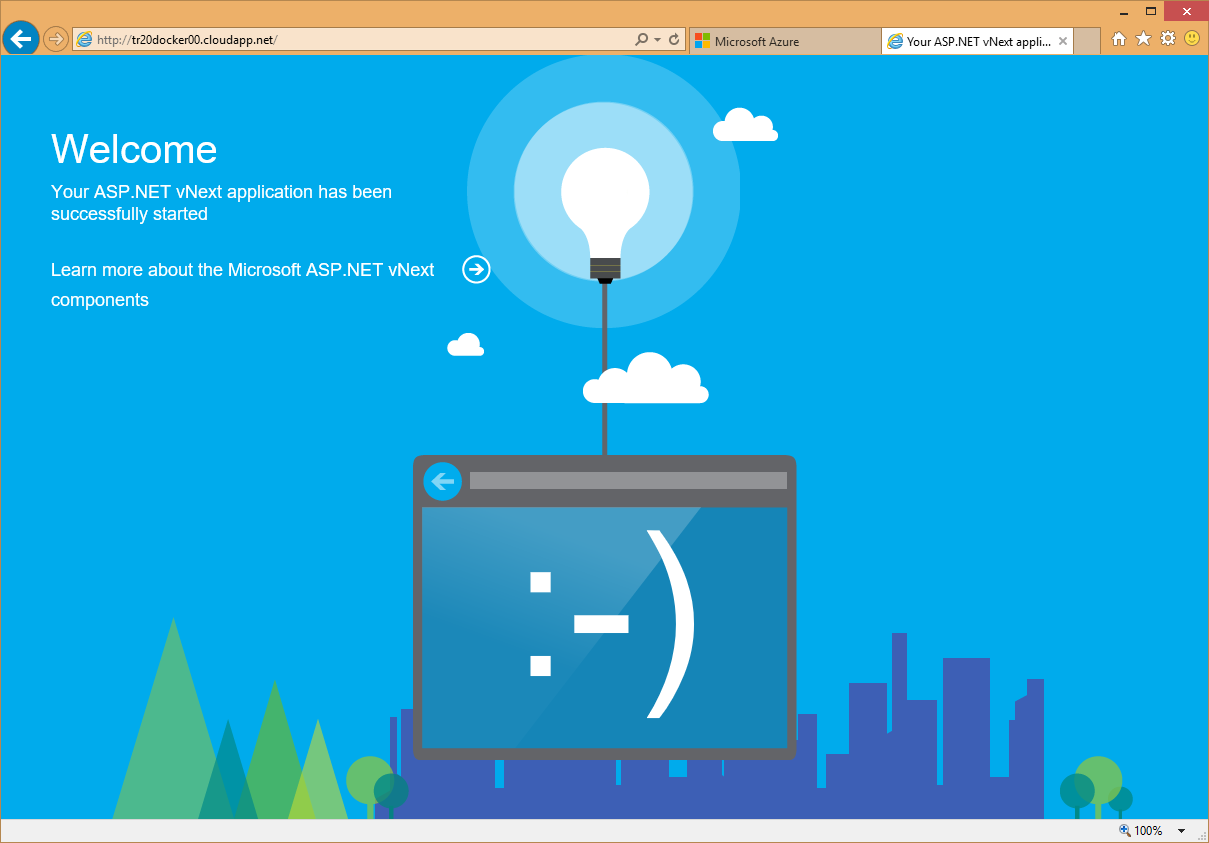
Then click “**OK**”



Wait until the endpoint appears in the “**Endpoints**” blade or that the Notification blade shows a success.

Launch a browser and navigate to the Domain Name value that you have noted previously.

The following page should be displayed:



This application runs on ASP.Net 5 in a Docker container. This container itself is hosted on Ubuntu.

## Build Docker client for Windows

From: <https://ahmetalpbalkan.com/blog/compiling-docker-cli-on-windows/>

**Step 1: Install Go**

Download the [Go MSI Installer](https://golang.org/doc/install#windows) from golang.org. This installation will add the go program to your PATH environment variable and you should be able to run the go command in cmd.exe. If that does not work, you may need a restart.

**Step 2: Check out the code**

Assuming you have Git installed on your system, you need to clone the [docker/docker](https://github.com/docker/docker/) repository locally:

git clone https://github.com/docker/docker.git c:\gopath\src\github.com\docker\docker

**Step 3: Compile!**

The rest is just as simple. Run cmd.exe and run the following commands in order:

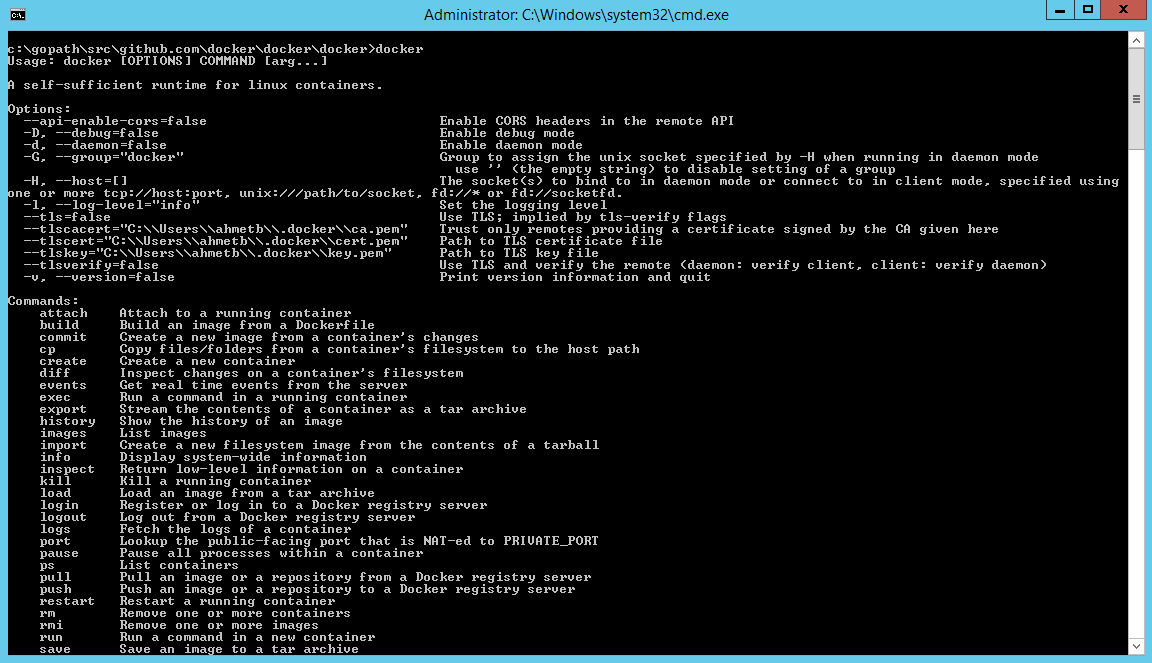
set GOPATH=c:\gopath;c:\gopath\src\github.com\docker\docker\vendor

set DOCKER\_CLIENTONLY=1

cd c:\gopath\src\github.com\docker\docker\docker

go build -v

If all goes well, you will end up with a lovely **docker.exe** on the directory you are at! ♥

[](https://ahmetalpbalkan.com/blog/static/images/2014/11/docker-win.png)

(If you happen to run in problems about emulating TTY (linux terminal) in cmd.exe, you need to use ConEmu or install ANSICON. Please see the [pull request](https://github.com/docker/docker/pull/9113) description for detailed info.)

**Summary**

This is just an intro to building the Docker CLI for Windows. Normally, these binaries are built in a Linux environment, inside a Docker container (yes, [Go](http://golang.org/) is crazy like that, it can cross compile Windows binaries on Linux)! However, for demonstration purposes we are detailing the procedure for building these Windows binaries on its own turf.

In the meantime, you can use this tutorial to build Docker on your Windows machine to test the changes. We are looking forward to your contributions on making the Docker’s Windows support even better!

If you happen to find a bug, please open an issue on [Docker repo] and mention me cc: @ahmetalpbalkan in the description.

I would like to acknowledge help of the Docker developer community on getting this work reviewed and merged to Docker. We are looking forward to make it more stable and shippable soon!

There is some Windows code now living in Docker! ♥

(This article has appeared on [Azure Blog](http://azure.microsoft.com/blog/2014/10/15/new-windows-server-containers-and-azure-support-for-docker/), [Docker Weekly](http://blog.docker.com/docker-weekly-archives/), [InfoQ](http://www.infoq.com/news/2014/11/docker-client-windows), [The Register](http://www.theregister.co.uk/2014/11/18/windows_docker_client/).)

If you like to watch a step-by-step video of this tutorial, there’s one in German by Joern Boeger [here](https://www.youtube.com/watch?v=3-wDaN2PqJY).

# Appendix

<http://blogs.msdn.com/b/webdev/archive/2015/01/14/running-asp-net-5-applications-in-linux-containers-with-docker.aspx>