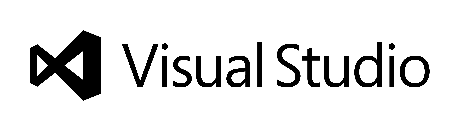
|  |  |
| --- | --- |
| PROJECT UNICORN |  |

|  |
| --- |
| **Docker-izing a legacy app**  **running in Azure** |
| **SAMPLE DEMO DOCUMENT** |

Version: 0.1 - 2015-03-05



**Project Unicorn – Docker and Azure Resource Manager**

*Azure Resource Manager and Linux VMs*

For this demo, we’ll take a quick walkthrough of:

1. Deploy a Docker Host in Azure - using portal Docker Gallery VM
2. Review of the Docker Build process
3. Containerize the Order Service for the Web front end (Tomcat / Spring)
4. Containerize the back end (Mongo)
5. Link the two together

You will see how you can leverage Docker with a pre-built Azure VM from the gallery to quickly deploy your solution in Azure.

For this demo you will see how flexible Azure is in support for Docker containers and leverage existing Templates that exist within Azure for a Docker host, along with the Docker build and deployment process.

This demonstrates direct use of Docker – skipping the Azure Resource manager deployment steps. This is here do demonstrate the existence and partnership with Docker that Microsoft has for support of Docker, and the use of the Marketplace VM image.

**Why Docker? We can build repeatability, known configuration – versioned config, dependency, re-use, and isolation.**

Prerequisites:

* Access to Active Azure subscription
* **Prepare another VM prior to the Demo so 1 is ready for Docker usage**. The VM provisioning can take 15 minutes or longer; Once an image is in the ReadState and Docker up & Running, all actions are generally fast at that point within Docker
* You need an SSH client – recommend using Putty or the “git” (<http://git-scm.com/> ) client from GitHub will install SSH support as well – execute from the Git shell – official Putty site: <http://www.chiark.greenend.org.uk/~sgtatham/putty/> - from GitHub either <http://windows.githhub.com> or <https://mac.github.com/>

|  |  |  |
| --- | --- | --- |
| Screenshots | Click Instructions | Talking Points |
|  | 1. Logon to the Preview Portal – <https://portal.azure.com>   New -> Compute -> “Azure Market Place”  Enter “Docker”   This should return a list of matching images.  What is “docker” <https://www.docker.com/whatisdocker/> | You’ll see several items and 2 specific gallery images that are Docker Hosts  We’ll be choosing “Docker on Ubuntu Server” from Canonical and MS Open Tech  We choose this image as it allows the use of Username/Password for logging on via SSH – the other image now only allows SSH credentials via certificates.  Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. |
|  | 1. For the VM settings enter things you’ll remember for username/password: | Make sure you choose a storage account that is in the same region.   Choose your Pricing Tier – A2 makes a nice starting point with 2 CPU’s and 4G  Make sure you go into Network settings  Choose a region “close” |
|  | 1. For the VM – choose a Resource Group (or new) | We need port 22 opened to the internal machine and the internet. So, make private port and public port the same. |
|  | 1. Now we wait-> Transition to the pre-created Docker Linux VM | Note, if there are portal issues, revert to the Azure CLI: the following is all one line.  azure vm docker create -e 22 -l "East US2" -z "Medium" <VMNAME> " b39f27a8b8c64d52b05eac6a62ebad85\_\_Ubuntu-14\_04-LTS-amd64-server-20140724-en-us-30GB" <USERNAME> <PASSWORD> |
| C:\Users\Shawn\AppData\Local\Temp\SNAGHTMLad7c99.PNG | 1. Open up your SSH client and provide the DNS name | Grab the DNS name from the Portal – it will be some DNS name based upon the Image name with some randomness appended. Such as:  docker2-v6vk67py.cloudapp.net |
|  | 1. You can also list the VM’s for your subscription | Use this to list easily the running state |
| Any issues with the Azure Docker image, you can easily spin up a plain Ubuntu image – then install docker  **‘sudo apt-get update’**  **‘sudo apt-get install docker.io’**  **‘sudo usermod -a -G docker <username>’**  **Login again (restart the Putty/SSH ression).** | 1. These commands are all on one line | **ONLY do this if the Portal or the Azure CLI fails** |
|  | 1. Login to the Docker Host |  |
|  | 1. After Login ensure that Docker is available by executing:  **docker info** | You should see something similar, although no containers or images would be there in a fresh Docker Host.  Containers: 3  Images: 69  Storage Driver: aufs  Root Dir: /var/lib/docker/aufs  Backing Filesystem: extfs  Dirs: 75  Execution Driver: native-0.2  Kernel Version: 3.13.0-36-generic  Operating System: Ubuntu 14.04.2 LTS  CPUs: 2  Total Memory: 3.362 GiB  Name: docker2  ID: YJDH:YXUB:P7XT:QAC5:N7KQ:VFGZ:BQQB:2SUE:MNAZ:FRBX:SBE4:L7MF |
| **“NOTE: an alternative is to just start a Ubuntu image, and install Docker”** |  | Discuss that this is an Ubuntu image and it comes with useful things already, however each Docker container won’t have default access to these things.   Containers in Docker start “clean” and you add required ‘things’ or start from an existing Docker image that has some foundation. |
|  | 1. Next we will clone a GitHub repo   <https://github.com/cicorias/IgniteARMDocker> | We’re using GitHub here as it’s quick and easy.  We need to get the various Dockerfile and associated content onto the Docker Host were we will build then run the containters.  You could use ‘scp’ or some means to get these to the machine, or use the Docker client. |
|  | 1. In the ~/ home directory, clone the git repo:   ‘git clone <https://github.com/cicorias/IgniteARMDocker.git>’  Then change to the created directory “IgniteARMDocker” | This just “copies” the content from GitHub – next we’ll review the Dockerfile files and scripts.  We will be “building” 2 tiers – the MongoDB and OrderService tiers. |
| Do this either on the GitHub repo itself, or use VIM, NANO or your chosen editor or ‘cat’ to the screen | 1. Review “buildboth.sh” | This script executes the docker build commands from 2 sub directories |
|  | 1. Review /mongoseed/Dockerfile | This file describes the build for the Mongo DB.  It “borrows” / based upon a <http://hub.docker.com> image called “Ubuntu” using the tag “latest.  The ADD command add the script to the “package”.  RUN executes these things during build to pull or make changes.  COPY places the content in the /tmp directory FOR THAT RUNNING container – which is virtualized.  ENTRYPOINT is where it starts.  VOLUME indicates what will be “shared” or actually for this script/Dockerfile how MongoDB data and logs will persist across restarts of a container. |
|  | 1. /mongorun.sh | Does some prep work, and starts the Mongo Daemon – then polls the log file for a “ready” message.  When Mongo is “done” it then populates with “seed” data.  Note that this is a demo and this would not be a way to normally populate seed data – as there is no validation done here.  WAIT forces the script to wait till MondoD stops – so, Docker will keep the container in a running state. |
|  | 1. ./orderService/Dockerfile | This Dockerfile builds the “middle tier” Spring/Java app that will handle REST requests.  It’s a simple Dockerifle to run a Java Jar file.  Note that while EXPOSE indicates port 8080 – you still must specify on the “run” command in docker to expose that port. Consider this “informational”. |
|  | 1. Review the orderservice startup script | Note that the script will wait till “mongodb” host is ready.  This is the same “host” that was specified or will be specified on the “run” command for the MongoDB container. And this will be the –link parameter connecting the 2 containers. |
| At this point, we will now “run” the 2 containers | 1. From the ~/IgniteARMDocker directory run:   ./runboth.sh | This runs the containers.  Make note of line 3 the “OrderService” container requires a “Link” to the Mongo container as otherwise it would not be able to initiate calls to that container. Docker permits external calls into containers via the “-v” switch – cross container communication is provided via “—link”.  Internally, Docker provides NAT and other dynamic hostnames, etc. eg. The name “mongo” on line 3 is how the hostname for the MongoDB container is referenced from within OrderService |
| Now we will do a “docker” buiild. | 1. Run “./buildboth.sh” |  |
|  |  |  |
|  |  | You can safely ignore these following error messages. |
|  |  | You can ignore. |
|  |  | You can ignore. |
|  | 1. Now we will look at the images in Docker Host. Run:   ‘docker images’ | The images will be shown. There are dependancies that are added by Docker along with the “built” images from the script. |
| Now, we’re going to test. | 1. First let’s test to see if MongoDB is up and running.   Initiate until you get a message “it looks like you are trying to …”  curl <http://localhost:27017>  Any other message MongoDB is not running | We need to validate if MongoDB is ready as this version of the OrderService Dockerfile doesn’t validate IF MongoDB is up and running. It is not resilient (the Java APP) to network failure.  Note, if you need to restart this, remove the files in /data/db via “sudo rm –r /data/db” |
| Once MongoDB is up and running, let’s test the order service: | 1. Initiate a http command.   ‘curl http://localhost:8080/ping | Wait till the “curl … /ping” command comes back with success. |
| ‘curl <http://localhost:8080/catalog> | 1. Initate a ‘curl http://localhost:8080/catalog | This should dump some json to the output. |
|  | 1. If you map an external port using Azure portal or script, you can query via the DNS name. |  |
| **At this point, you have the ‘MRP’ OrderService and the MongoDB layer running in Docker Containers.** |  |  |