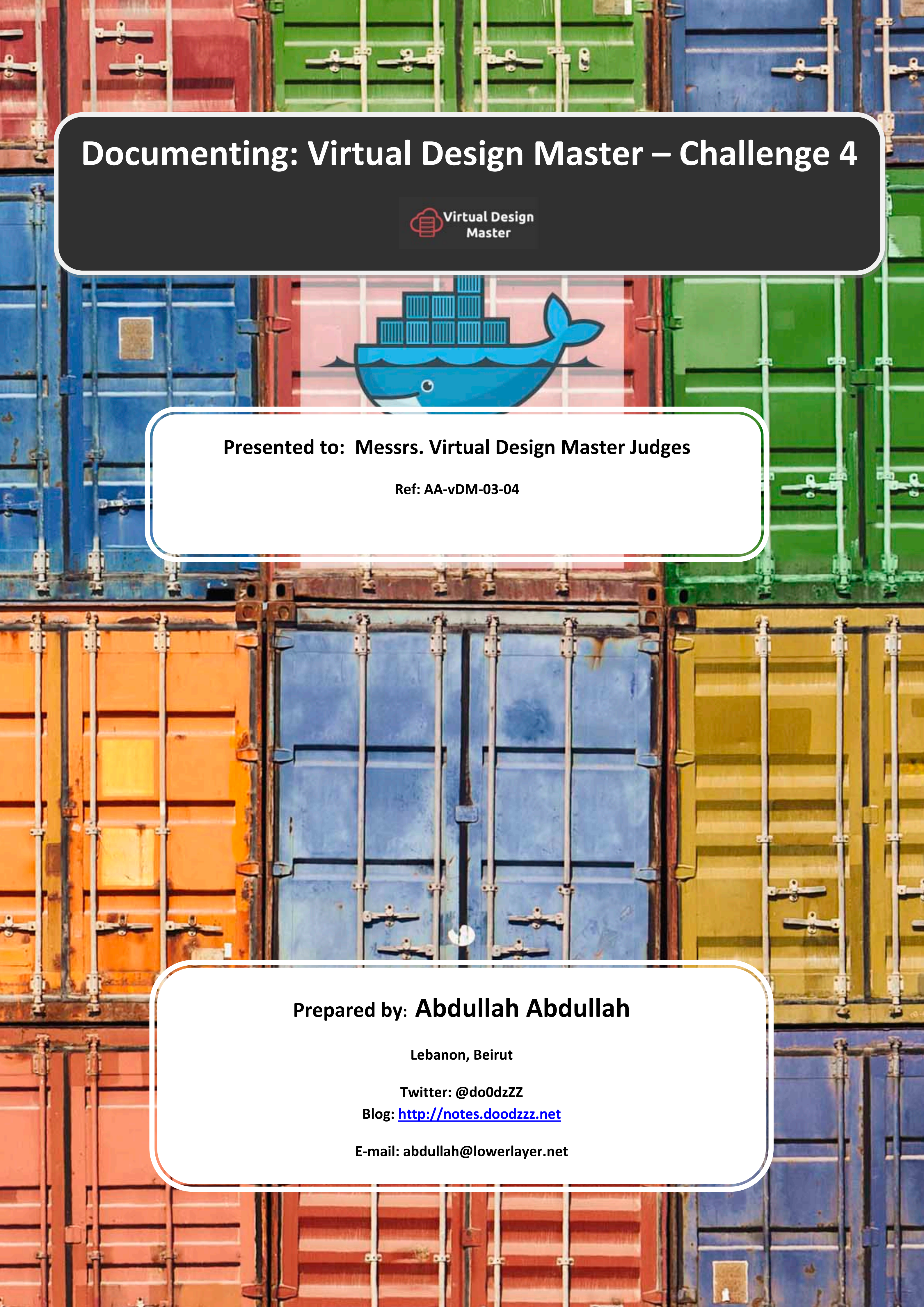
Documenting: Virtual Design Master – Challenge 4



Presented to: Messrs. Virtual Design Master Judges

Ref: AA-vDM-03-04

Prepared by: Abdullah Abdullah

Lebanon, Beirut

Twitter: @do0dzZZ
Blog: http://notes.doodzzz.net

E-mail: abdullah@lowerlayer.net

[Synopsis]

Now that we’ve established ourselves across the solar system, it is time to make plans for taking our home planet back from the zombies. Select teams of humans and robots will be sent back to Earth to attempt to secure different areas across the globe. We will of course need working infrastructures help us with this as we reclaim the planet. Chances of sending back IT people are pretty slim, so you will need to create an orchestrated system to rebuild the infrastructures. We know this is not an easy task so we are going to start small.

Our test will be a simple web application that displays “Welcome Back to Earth!”. Since we are evaluating the best way to accomplish orchestrating our infrastructure, you will need to use two different orchestration tools, two different operating system, and two different web servers. Because we don’t know what type of infrastructure we’re going to end up with (we could end up with a mix of equipment and operating systems), the web servers must run inside of a container. Share your code, and build instructions so that a complete walk through is available. Make sure to include dependencies your application may have.

Table of Contents

1. Executive Summary

1.1. Project Overview

Earth! It has been said that one cannot escape his origins nor one can lose the sense of returning back home! Even when we used to go on business trips back on Earth we always seemed to ache for getting back to our residence no matter how bad/good it was.

When we left Earth we always knew that at some point we will be returning, and well my friends NOW IS THAT TIME!

This project is all about infiltration, and sending messages. We know that on Earth there is a resistance still fighting but they are alone and almost resource-less, we need to tell them that we're here without the attention of the Zombies.

1.2. Intended audience

This document is intended for those whom will be building our pods,

1.3. Project Insights

For this to work out we need to go as low profile as much as possible, so we will be integrating different technologies that will enable us to achieve our goal.

1.3.1. Project Requirement

- R001: Build a mobile computing pod.
- R002: Run two different web application servers.
- R003: Web application servers must be different.
- R004: Web application servers must be running on containers.

1.3.2. Project Constrains

- C001: Console access in case of POD is unreachable.
- C002: Power.
- C003: Unexpected smartness of the Zombies.

1.3.3. Project Assumptions

- A001: A lot of hardware available in stock.
- A002: Docker hub was recreated on the moon.
- A003: Satellite network has been upgraded to support mobile units.
- A004: Administration team is familiar with Docker.
- A005: Plenty of hardware available on Earth's Moon.

2. Design Summary

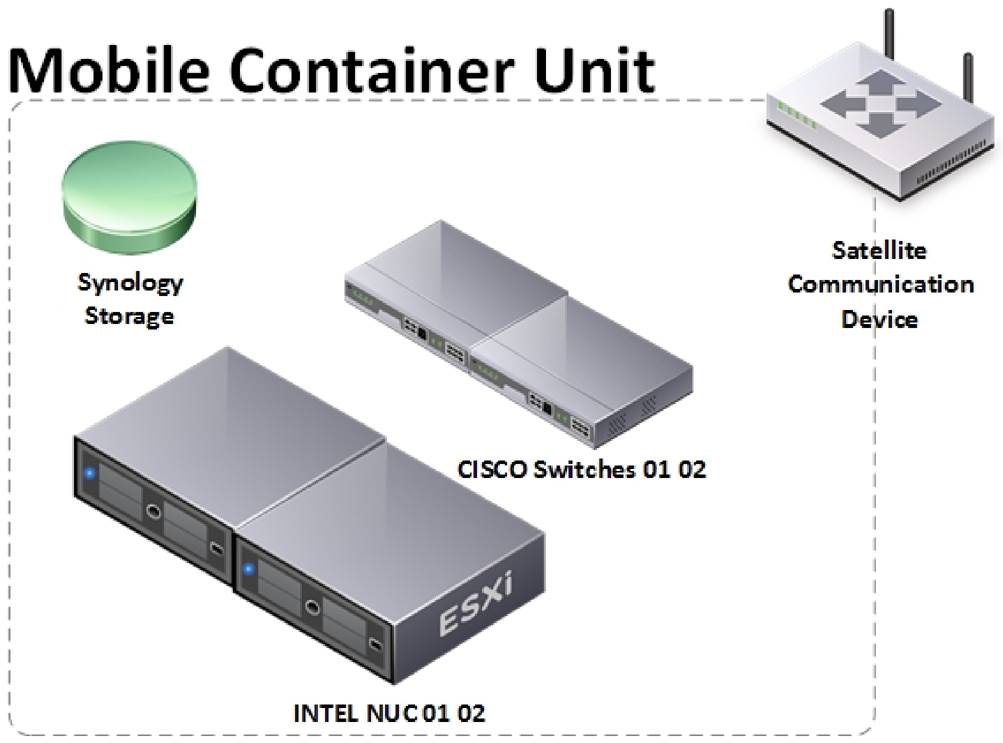
This is quite of a challenge to us, but we're full of hope that it will be successful and it will make a difference.

2.1. Physical Design

2.1.1. Pods

With the assistance of our beloved scientists and technicians back on the Moon we were able to devise a POD that looks like a shipping container and can wits tanned the drop from a space ship and can be used a mobile server rooms sort of speaking where it will be capable of hosting:

1- 2 Intel NUC machines.
2- 2 CISCO small-business gigabit switches.
3- 1 Synology DS214se.
4- 1 mobile satellite communication unit.



2.1.2. Power

Since this is going to be a self-sustaining unit in terms of power, we will be harnessing the energy from the Sun using solar panels covering the whole container.

All of our equipment except for the switches will be capable of WOL (Wakeup On LAN), thus it will be possible to shutdown the hosts and the storage when needed.

2.1.3. Servers

We want to use virtualization but we do not want to have a power consuming unit on the POD that would impose a major risk to the POD operability and sustenance, luckily we were able find a lot of Intel NUCs in the stock (apparently they fit well for travel and people managed to get them from earth). Our Intel NUCs are configured as follow:

1- Core I7 Processor.
2- 16GB DDR3 memory kit.
3- 8GB SD Card (used for OS installation).
4- An additional Mini PCIe Ethernet adapter.

2.1.4. Storage

In terms of storage we would have just placed two SSDs for the Intel NUC hosts but we do need high availability between the hosts and clustering suggests a shard storage. Building our own NAS would have been a waste of time knowing that we already have a lot of Synology devices available, our decision for this mission will be a Synology DS214+ with the following configuration:

1- System memory 1GB DDR3.
2- 2 x 1TB SATA III HDD (configured for RAID1).
3- 2 x RJ-45 1Gbps port (will configured for failover).

2.1.5. Networking

Because we need a reliable product and for simplicity purposes we will be using 2 Cisco SG 100D-08 8-Port Gigabit Switches, these are unmanageable and low on power and the ports will be divided as follow:

<u>Switch1:</u>

1- Host 1 – Ethernet Port 1.
2- Host 2 – Ethernet Port 1.
3- Storage – Ethernet Port 1.
4- Mobile Satellite Unit – Ethernet Port 1.

<u>Switch2:</u>

1- Host 1 – Ethernet Port 2.
2- Host 2 – Ethernet Port 2.
3- Storage – Ethernet Port 2.
4- Mobile Satellite Unit – Ethernet Port 2.

2.1.6. Security

Indeed this is Earth, this is home but nevertheless we have not fully regained it and we cannot easily disregard the fact that accidents do happen, we will be securing our PODs in such a way that:

1- It will be well camouflaged to cope with the terrain its landing on.

2- It will be configured to short-circuit all the equipment in case it was tampered to be accessed by force.

2.1.7. POD transfer method

Our trusty space ships will be having scheduled roundtrips to earth to place the PODs in their designated areas and will be coordinating with operations team at the Moon that the POD is reachable and ready.

2.2. Logical Design

2.2.1. POD Infrastructure (vSphere Cluster)

We will be utilizing VMware vSphere technology to support our POD setup.

2.2.1.1. ESXi Hosts

We will be using vSphere 6.0, where ESXi will be installed on the SD cards that we have equipped our Intel NUCs with, it should also be known that our ESXi image was customized for the Intel NUCs with custom VIBs as these drivers are not included within the standard ESXi distribution.

2.2.1.2. ESXi Networking

Since each host has 2 NICs on it, and for the sake of simplicity we will be using the NICs in failover mode.

In addition we will be having a VM running DHCP services for each POD, which will be needed for our container hosts.

2.2.1.3. ESXi Storage

Our shared storage is capable for being configured for NFS, for the sake of simplicity we will be configuring 2 NFS datastores that will be presented to both hosts.

It should also be known that the NICs on the Synology will also be configured in failover mode, again this is for the sake of simplicity and efficiency.

2.2.1.4. vCenter Server

Our vCenter Server won’t be residing within the POD ESXi cluster, the vCenter Server will be hosted on the Moon’s Datacenter as it is considered to be a management component since we have visibility through the Satellite network, this will leverage us to enable HA.

2.2.1.5. Container Technology

To fulfil this requirement we will be utilizing Docker, Docker containers wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries and anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

2.2.1.6. Container Administration Station

This is a new to us and because after experiencing a little bit with Docker and its components we have realized that to be able to properly administer a container driven infrastructure one will need to configure and customize a lot of things so that to have the confident grasp over the environment.

For that purpose and because not everyone will have the time to do the customization, we have decided to create what we call a CAS (Container Administration Station) which will be accessed by the administrators requiring to handle container related tasks.

This machine of course will be running on the Moon, our operation guys are fond of Linux but they are Windows oriented, so in order to get them the best of both we're going to setup a system that combines both and it is as follows:

1- Provision a Windows 2012R2 VM.

2- For Shell:
   a. Install Git Bash
   b. Run Git Bash.

3- Installing Docker on Git Bash:
   a. curl -L https://get.docker.com/builds/Windows/x86_64/docker-latest.exe > /bin/docker).

4- Installing docker-machine on Git Bash:
   a. (curl -L https://github.com/docker/machine/releases/download/v0.3.0/docker-machine_windows-amd64.exe > /bin/docker-machine).

5- Installing GoVC on Git Bash:
   a. Install Google Go https://storage.googleapis.com/golang/go1.4.2.windows-amd64.zip
   b. export GOPATH=$HOME/src/go
   c. mkdir -p $GOPATH
   d. export PATH=$PATH:$GOPATH/bin
   e. go get github.com/vmware/govmomi/govc

6- Configure Boot2Docker ISO Repository:
   a. Enable IIS Role with basic security checked.
   b. Run IIS manager and add IIS mimetype application/octet-stream for .iso
   c. Copy boot2docker.iso to the IIS wwwroot.

7- Install vSphere C# client if preferred.

8- SSH is also available via Git Bash.

As this point we have a ready administration station that has Docker on it along with utilities that we will be using for our first container solution.

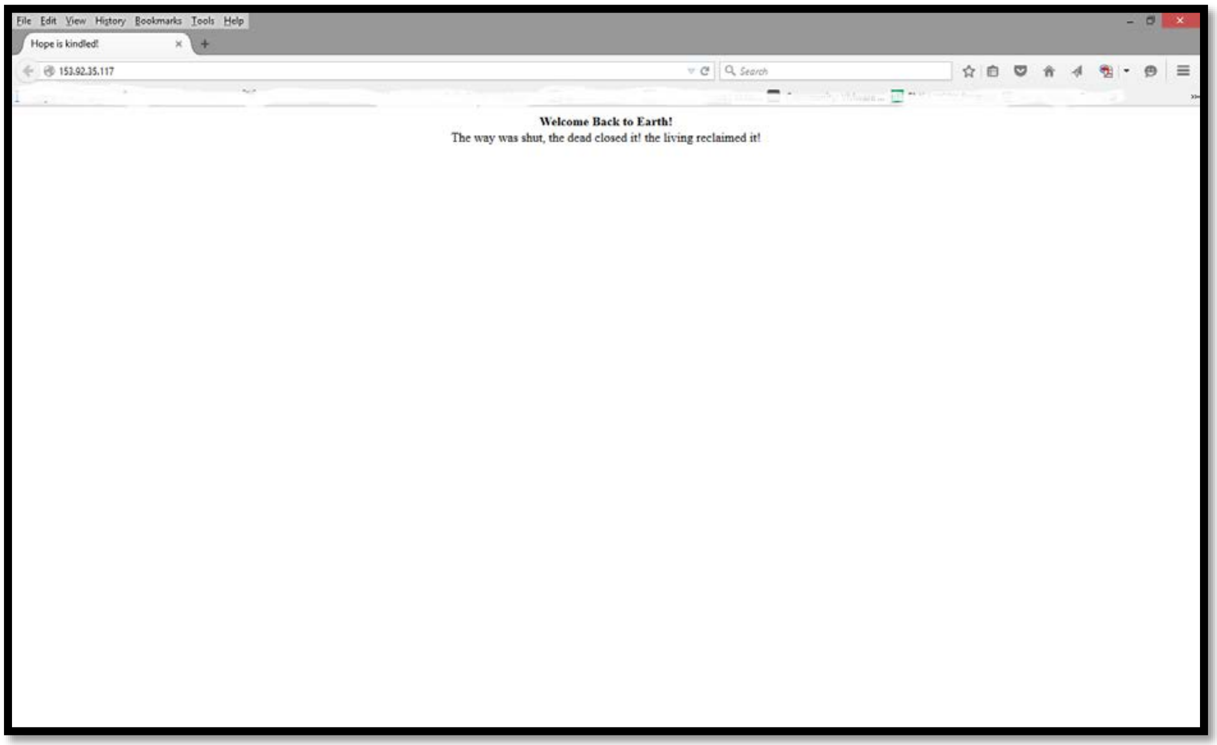
2.2.2. Container Solution 1

Our container host will be boot2docker which is a lightweight linux distribution make specifically to run Docker.

2.2.2.1. Webserver Container

Our choice for this solution will be a container of Ubuntu with Apache installed on it, we have already created that container and pushed it to **humankind/ubuntu-apache-1** hereunder you'll find the detailed configuration used when creating that container:

1- docker run -t -i ubuntu /bin/bash
2- apt-get update && apt-get install -y apache2 curl
3- apache2ctl start
4- cd /var/www/html
5- echo "<html><head><title>Hope is kindled!</title></head><body><center><b>Welcome Back to Earth!</b> <br />The way was shut, the dead closed it! the living reclaimed it!</center></body></html>" > index.html



Welcome Back to Earth!
The way was shut, the dead closed it! the living reclaimed it!

At this point we have created our desired container, and afterwards we will save it and push it to the Docker hub:

1- While the container is running -> docker ps << Thisis to get the container id.
2- docker commit containerid humankind/ubuntu-apache-1
3- docker push humankind/ubuntu-apache1

2.2.2.2. Orchestration

It’s all about Docker-Machine here, since we already have our vSphere cluster up and running we will be able to utilize the parameters found in docker-machine that integrate with VMware vSphere, and it breaks down the this (these commands are run from the administration station):

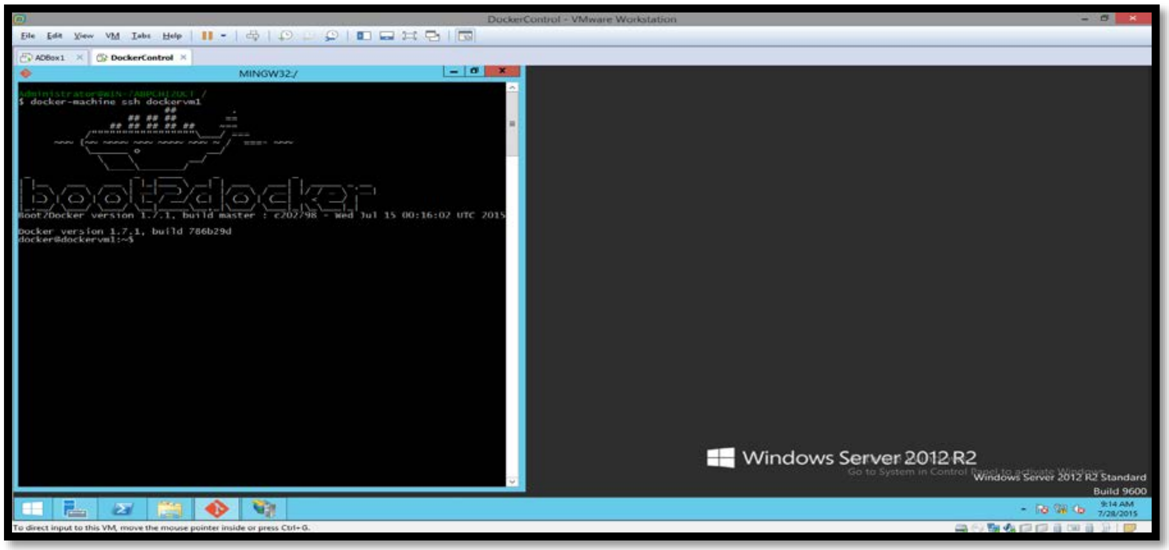
1- docker-machine create web-docker-vm1 -d vmwarevsphere --vmwarevsphere-compute-ip 192.168.x.x --vmwarevsphere-vcenter 192.168.x.x --vmwarevsphere-username username@domain.tlds --vmwarevsphere-password THISISACOMPLESXPASSWORD --vmwarevsphere-datacenter "Moon DC1" --vmwarevsphere-datastore=NFSDatastore01 --vmwarevsphere-network "VM Network" --vmwarevsphere-boot2docker-url http://localhost/boot2docker.iso

2- You’ll notice that the operation gets triggered quite fast and smooth, wait for the virtual machine to complete its startup and gain its DHCP IP address.

3- Now we need to get environment information about our docker host:
   a. docker-machine env web-docker-vm1

4- The up above command will place the docker system variables into this:
   a. eval "$(docker-machine env dockervm1)" << this actually sets:
      i. $DOCKER_HOST=tcp://192.168.x.x:2376
      ii. $DOCKER_CERT_PATH=C:\xx\.docker\machine\machines\dockervm1
      iii. $DOCKER_TLS_VERIFY=1

5- Now we can run our docker container on web-docker-vm1, we will be exposing the container to port 80, and we’ll be suing the restart=always parameter so that whatever happens to the container it will always start:
   a. sudo docker run --restart=always -d -p 80:80 humankind/ubuntu-apache-1 /usr/sbin/apache2ctl -D FOREGROUND


DockerControl - VMware Workstation
File Edit View VM Tabs Help
ADBox1
DockerControl
MINGW32:/
Windows Server 2012 R2
Windows Server 2012 R2 Standard
Build 9600
9:14 AM
7/28/2015
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```
Administrator@WIN-7ABPCHI2OCT /
$ docker-machine ssh dockervm1

Boot2Docker version 1.7.1, build master : c202798 - Wed Jul 15 00:16:02 UTC 2015

Docker version 1.7.1, build 786b29d
docker@dockervm1:~$
```

2.2.2.3. High Availability

Since we're utilizing vSphere we already have HA enabled and our container host VM is protected against ESXi failures.

Because Docker-Machine gives us control over creating the container host, we are capable of creating additional ones at any time depending on the workload.

2.2.3. Container Solution 2

Our container host in this scenario will be CoreOS which is another lightweight linux distribution designed to run Docker containers.

2.2.3.1. Webserver Container

For our second web server we will be using CentOS with Nginx, again this has been built and pushed to Docker hub under humankind/centos-nginx-1, hereunder are the details of building this container:

1- docker run -t -i centos /bin/bash
2- yum -y update; yum clean all
3- yum -y install epel-release; yum clean all
4- yum -y nginx
5- cd /usr/share/nginx/html
6- echo "<html><head><title>Hope is kindled!</title></head><body><center><b>Welcome Back to Earth!</b> <br />The way was shut, the dead closed it! the living reclaimed it!</center></body></html>" > index.html

At this point we have created our desired container, and afterwards we will save it and push it to the Docker hub:

1- While the container is running -> docker ps << This is to get the container id.
2- docker commit containerid humankind/centos-nginx-1
3- docker push humankind/ centos-nginx-1

2.2.3.2. Orchestration

Since CoreOS was designed for cloud infrastructure initially (to utilize tools such as cloud-init) there is no straight forward way to provision CoreOS virtual machines on our ESXi cluster other than using a third party script.

We'll be using a shell script originally written by William Lam, the script can be found in appendix A. What the script does is that:

1- Gets a bunch of parameters:
    a. CoreOS VMX URL < lastest from CoreOS download URL.
    b. CoreSO VMDK URL < latest from CoreOS download URL.
    c. IP or Hostname of ESXI host
    d. ESXi Username
    e. ESXi Password

f. Name of vSphere Datastore to upload CoreOS VM
g. Name of the VM Network to connect CoreOS VM
h. Name of the CoreOS VM
i. Hostname of CoreOS Instance
j. IP Address of CoreOS Instance
k. Username to enable on CoreOS Instance
l. Password hash of CoreOS Instance
m. Name of the CoreOS cloud-config ISO

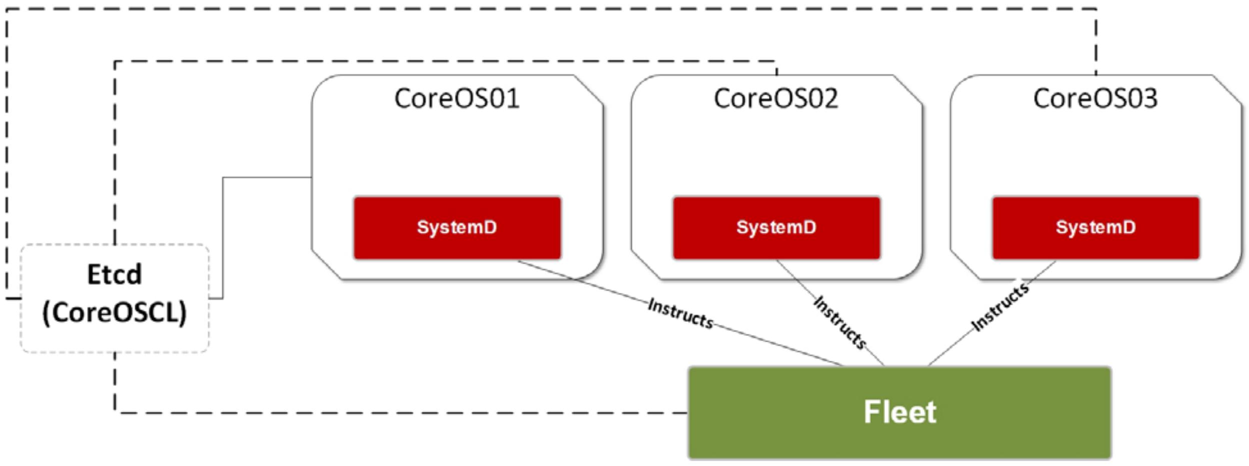
Based on the above information, the script build the CoreOS cloud-config file, which in the case of VMware can be placed into the virtual CD drive as an ISO and CoreOS will recognize it to pull its configuration from it on startup.

2.2.3.3. High Availability

Again here, we're still on a VMware vSphere cluster with HA enabled, but CoreOS has built-in clustering and service orchestration, since we're evaluating our options we will be configuring:

1- Etcd (service already configured within the cloud-config).

2- Fleet (service already configured within the cloud-config).



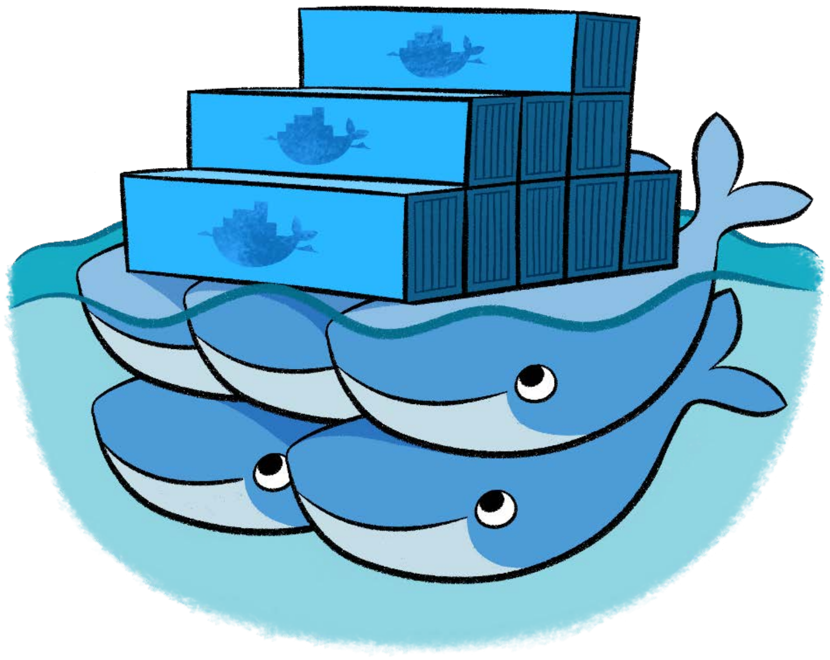
CoreOS Nodes:

1- CoreOS Cluster: 3 Nodes (CoreOS01, CoreOS02, CoreOS03).
2- Cloud-Config (Appendix B) same for all three nodes.
3- Fleet Unit Files:
   a. Centos-nginx@.service**: (Appendix C)** this will be used by fleet to update each of the services throughout our cluster.
   b. Centos-nginx-discovery@%i.service: (**Appendix D**) this unit file is responsible for the etcd updates on the status of the Docker service.
4- Send the services:
   a. fleetctl submit centos-nginx@.service centos-nginx-discovery@.service
   b. fleetctl list-unit-files << to view the files sent to the cluster.

5- **Loading the services to targets**:
   a. fleetctl load centos-nginx@80.service
   b. fleetctl load centos-nginx-discovery@80.service
   c. fleetctl list-unit-files << check the state of the service.
6- **Start the service**:
   a. fleetctl start centos-nginx@{1..3}.service << Here we've started 3 docker instances of our webserver.

3. Final Thoughts

Although this technology is new, DevOps is bringing those who develops to those whom operate closer and closer, there are still a lot of applications and projects surrounding Docker that are currently in beta, a lot of automation to it and also we'll be seeing Docker like containers in the future.



4. Appendix

4.1. Appendix A – deploy_coreos_on_esxi2.sh

4.1.1. https://github.com/lamw/vghetto-scripts/blob/master/shell/deploy_coreos_on_esxi2.sh

4.2. Appendix B – Cloud Config < Validated via https://coreos.com/validate/

```
#cloud-config
coreos:

  etcd2:
   name: CoreOS01
   discovery: https://discovery.etcd.io/4463a7cca5be970c31596be5df1d51a9
   advertise-client-urls: http://$public_ipv4:2379
   initial-advertise-peer-urls: http://$private_ipv4:2380
   listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
   listen-peer-urls: http://$private_ipv4:2380,http://$private_ipv4:7001

  units:
   - name: etcd.service
     command: start
   - name: fleet.service
     command: start



  fleet:
   public-ip: $public_ipv4
users:
 - name: coreadmin
   passwd:
$6$rounds=5000$GSIDSvEs.Cs0GHHW$bXV0TD7HJ388BWF6DLuAA5l3dn88dEAMYgI48EwNAuZlkCjcUtDxhuKV/oyo56siXBs6N433nG3ZZSyeJh1fz1
 - groups:
   - sudo
   - docker
hostname: CoreOS01
```

4.3. Appendix C – Centos-nginx@.service

```
[Unit]
Description=CentOS Nginx Service
After=etcd.service
After=docker.service
Requires=centos-nginx-discovery@%i.service

[Service]
TimeoutStartSec=0
KillMode=none
ExecStartPre=-/usr/bin/docker kill centos-nginx%i
ExecStartPre=-/usr/bin/docker rm centos-nginx%i
ExecStartPre=/usr/bin/docker pull humankind/centos-nginx-01
ExecStart=/usr/bin/docker run -d --name centos-nginx%i -p 80:80 humankind/centos-nginx-01

[X-Fleet]
X-Conflicts=centos-nginx@*.service
```

4.4. Appendix D – centos-nginx-discovery@%i.service

```
    [Unit]
    Description=CentOS-Nginx@%i Service
    BindsTo=centos-nginx@%i.service


    [Service]
    EnvironmentFile=/etc/environment
    ExecStart=/bin/sh -c "while true; do etcdctl set /announce/services/centos-nginx%i
    ${COREOS_PUBLIC_IPV4}:%i --ttl 40; sleep 35; done"
    ExecStop=/usr/bin/etcdctl rm /announce/services/centos-nginx%i


    [X-Fleet]
    X-ConditionMachineOf=centos-nginx@%i.service
```

4.5. Appendix D – Extra Resources

- https://coreos.com/fleet/docs/latest/launching-containers-fleet.html
- http://slopjong.de/2014/09/17/install-and-run-a-web-server-in-a-docker-container/
- https://coreos.com/os/docs/latest/installing-to-disk.html
- https://coreos.com/blog/boot-on-bare-metal-with-pxe/
- http://docs.docker.com/compose/
- http://www.projectatomic.io/docs/gettingstarted/
- https://docs.docker.com/machine/
- http://www.virten.net/2015/04/vmware-homelab-in-2015-systems-revised-with-vsphere-6/
- https://www.virten.net/2015/03/esxi-6-0-image-for-intel-nuc/
- http://slopjong.de/2014/09/17/install-and-run-a-web-server-in-a-docker-container/
- http://www.ducea.com/2006/08/07/how-to-change-the-hostname-of-a-linux-system/
- https://xivilization.net/~marek/blog/2014/08/04/installing-coreos/
- http://dannysu.com/2014/11/26/single-coreos-digitalocean/
- http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2104303
- https://coreos.com/validate/
- https://docs.docker.com/machine/
- http://www.virtuallyghetto.com/2014/09/govmomi-vsphere-sdk-for-go-govc-cli-kubernetes-on-vsphere-part-1.html
- https://docs.docker.com/machine/
- https://github.com/boot2docker/boot2docker
- https://coreos.com/os/docs/latest/cloud-config.html
- https://github.com/docker/machine/issues/996
- http://blog.xebia.com/2015/03/24/a-high-available-docker-container-platform-using-coreos-and-consul/
- http://vmiss.net/tag/boot2docker/
- https://www.google.co.uk/?gws_rd=ssl#q=docker-machine+container
- https://developer.rackspace.com/blog/using-docker-machine-to-deploy-your-docker-containers-on-rackspace/
- https://crate.io/blog/deploying-crate-with-docker-machine-swarm/
- http://www.projectnee.com/HOL/console/lab/HOL-SDC-1430-HOL/NEE-439369617073_939/

- https://www.snip2code.com/Snippet/410119/Docker-Machine-for-VMware-vSphere

- https://hub.docker.com/u/doodzzz/
- https://github.com/rancher/rancher#installation
- http://rancher.com/running-a-mesos-cluster-on-rancheros/
- https://github.com/dockerfile/nginx
- https://blog.docker.com/2015/04/tips-for-deploying-nginx-official-image-with-docker/
- http://docs.quay.io/solution/zero-downtime-deployments.html
- http://wiki.contribs.org/Docker_design_concept
- https://realpython.com/blog/python/docker-in-action-fitter-happier-more-productive/
- http://flurdy.com/docs/docker/docker_compose_machine_swarm_cloud.html
- https://github.com/coreos/fleet
- https://github.com/GoogleCloudPlatform/kubernetes/tree/master/contrib/mesos
- http://blog.scottlowe.org/2014/08/20/coreos-continued-fleet-and-docker/
- https://www.digitalocean.com/community/tutorials/how-to-use-fleet-and-fleetctl-to-manage-your-coreos-cluster
- https://coreos.com/fleet/docs/latest/examples/example-deployment.html
- http://www.recorditblog.com/post/how-to-create-a-web-scale-infrastructure-based-on-docker-coreos-vulcand-and-mesos-and-why-object-storage-becomes-the-de-facto-data-repository/
- https://labs.ctl.io/building-your-first-app-on-coreos/
- https://github.com/GoogleCloudPlatform/kubernetes/tree/master/contrib/mesos
- http://jwalczyk.blogspot.com/2015/01/my-experiences-using-coreos-fleet-to.html
- http://blog.carbonfive.com/2015/03/17/docker-rails-docker-compose-together-in-your-development-workflow/
- https://docs.docker.com/compose/yml/
- http://blog.arungupta.me/docker-compose-orchestrate-containers-techtip77/
- http://blog.arungupta.me/multicontainer-applications-docker-compose-swarm/
- https://github.com/vmware/govmomi/tree/master/govc
- https://github.com/docker/compose/releases
- http://translate.google.co.uk/translate?hl=en&sl=ru&u=http://habrahabr.ru/post/260329/&prev=search
- https://github.com/d11wtq/dockerpty/issues/7
- https://github.com/docker/compose/issues/1085
- https://github.com/d11wtq/dockerpty/issues/7

- https://coreos.com/os/docs/latest/customizing-docker.html
- https://coreos.com/os/docs/latest/booting-on-vagrant.html
- http://www.perehospital.cat/blog/six-container-orchestration-frameworks-to-keep-an-eye-on
- http://deis.io/get-deis/
- https://www.ykode.com/2014/08/22/building-vagrant-coreos-devbox.html
- http://www.virtuallyghetto.com/2014/07/how-to-quickly-deploy-coreos-on-esxi.html
- https://github.com/lamw/vghetto-scripts/blob/master/shell/deploy_coreos_on_esxi.sh
- https://access.redhat.com/articles/1353773
- https://github.com/CaptTofu/coreos_cluster_vmware
- https://devops.profitbricks.com/tutorials/getting-started-with-a-multi-node-kubernetes-cluster-on-ubuntu/
- https://docs.docker.com/articles/host_integration/
- http://serverfault.com/questions/633067/how-do-i-auto-start-docker-containers-at-system-boot
- http://stackoverflow.com/questions/18786054/coreos-how-to-auto-restart-a-docker-container-after-a-reboot
- http://www.virtxpert.com/sample-vagrantfile-vagrant-vsphere/
- http://blog.fosketts.net/2015/06/05/adding-a-second-ethernet-port-to-an-intel-nuc-via-mini-pcie/
- https://coreos.com/os/docs/latest/booting-on-vmware.html