Mesos and Marathon

Step 1: Installing Software

1. Install Vagrant and VirtualBox. MAKE SURE YOU HAVE THE LATEST ONE. Older versions of Vagrant do not work.
2. Install a simple Centos-7.1 Vagrant VM.
3. Configure your /etc/hosts file to have the correct hosts.
4. Add the Mesosphere RPM repository.
5. Install the mesos and marathon RPMs

Install Vagrant and VirtualBox. MAKE SURE YOU HAVE THE LATEST ONE. Older versions of Vagrant do not work. Create a virtual machine with Vagrant running the CentOS 7.1 Linux distro. This will download the image you need so it takes a little while:

```
$ mkdir vm-install
$ cd vm-install
$ vagrant init bento/centos-7.1
```

Edit the Vagrantfile to uncomment the line withconfig.vm.network "private_network", ip: "192.168.33.10"and add the line config.vm.hostname = "node1" right after that:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|
 config.vm.box = "bento/centos-7.1"

 config.vm.network "private_network", ip: "192.168.33.10"
 config.vm.hostname = "node1"
end
```

Start the VM and ssh into it:

```
$ vagrant up
$ vagrant ssh
```

Mesos and Marathon

- Designed by Yogesh Darji

Modify the /etc/hosts file to make the node1 name map to the IP address in the Vagrantfile:

```
[node1]$ sudo vi /etc/hosts
[node1]$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.33.10 node1
```

Install the Mesosphere software packages:

```
[node1]$ sudo rpm -Uvh http://repos.mesosphere.com/el/7/noarch/RPMS/mesosphere-el-repo-7-1.noarch.rpm
[node1]$ sudo yum -y install mesos marathon
```

Step2: Installing Zookeeper

Install Zookeeper and the Zookeeper server package by pointing to the RPM repository for ZooKeeper:

```
[node1]$ sudo rpm -Uvh http://archive.cloudera.com/cdh4/one-click-install/redhat/6/x86_64/cloudera-cdh-4-0.x86_64.rpm
[node1]$ sudo yum -y install zookeeper zookeeper-server
```

Initialize and start Zookeeper:

```
[node1]$ sudo -u zookeeper zookeeper-server-initialize --myid=1
[node1]$ sudo service zookeeper-server start
```

Use the interactive shell to test your installation:

```
[node1]$ /usr/lib/zookeeper/bin/zkCli.sh
# sometimes at this point you get an error that java is missing.  This means yum did NOT install java for...reasons.  Start over.
help
create /test 1
get /test
set /test 2
```

```
get /test
delete /test
quit
```

Validate that you can stop and restart ZooKeeper:

```
[node1]$ sudo service zookeeper-server stop
[node1]$ sudo service zookeeper-server start
```

Step3 : Using Apache Mesos

Apache Mesos is the foundation of the Mesosphere technology stack and powers all of the communications and system management. Mesos uses a simple architecture to give you intelligent task distribution across a cluster of machines without worrying about where they are scheduled. In this module Mesos is used to run and manage services on a four node cluster. To get started you must set up themesos-master on node1.

In this exercise:

Start the mesos-master and mesos-slave processes.
Test that the mesos-master and mesos-slave processes are working as expected.
Execute a sample command from the command line.
Observe the command's progress from both the command line and a web GUI.

Start the mesos-master and mesos-slave processes:

```
[node1]$ sudo service mesos-master start
[node1]$ sudo service mesos-slave start
[node1]$ sudo netstat -nlp | grep mesos
```

Access the Mesos user interface with your browser athttp://192.168.33.10:5050 and confirm that the IP address shown in the user interface is 192.168.33.10. If not, start over by usingvagrant destroy.

Test out mesos by using the mesos-execute command:

```
[node1]$ export MASTER=$(mesos-resolve `cat /etc/mesos/zk` 2>/dev/null)
[node1]$ mesos help
[node1]$ mesos-execute --master=$MASTER --name="cluster-test" --command="sleep 40"
```

With the mesos-execute command running, enter ctrl-z to suspend the command. You can see how it appears in the web UI and command line:

```
# hit ctrl-z
[node1]$ bg # this sends the process into the background
[node1]$ mesos ps --master=$MASTER
```

Step4: Starting Marathon

In Linux the init/systemd/upstart program manages all of the processes running on your system. The Mesosphere stack uses Marathon to manage the processes and services. Marathon is the technology that plays the role of init/systemd/upstart in the analogy of an operating system. Marathon provides both a simple GUI and an extensive REST API that you can work with if you need more capability.

1. Start Marathon.
2. Explore the Marathon GUI.
3. Install an app and start a simple Python web server by using Marathon.
4. Test that the Python web server is live.

Start Marathon by using the service command:

```
[node1]$ sudo service marathon start
```

Go to http://192.168.33.10:8080/ to view the Marathon GUI. From the GUI, install a new app that Marathon will run. In this example, we start by using the
python -m SimpleHTTPServer app.:

```
# view the python SimpleHTTPServer web server is running
[node1]$ netstat -nlp | grep 8000
# use curl to play with the server
[node1]$ curl http://192.168.33.10:8000/
```

Marathon and Mesos give you direct access to the stderr (standard error) and stdout (standard out) files for every process. You can usecurl to view these files and see that they are theSimpleHTTPServer's logs, which you would normally see on your terminal when you start it:

```
[node1]$ curl http://192.168.33.10:8000/stderr
[node1]$ curl http://192.168.33.10:8000/stdout
```

Step 5: Using Marathon GUI

You can use the Marathon GUI to configure and deploy your applications. In this exercise you will learn how the GUI works and how to configure and control a simple Python web service. This exercise is entirely video based. In this exercise:

1. Use the command line to kill the python web server.
2. Watch Marathon bring the Python web server back up.
3. Learn how to scale the app and halt the process.
4. Learn how to use $PORT to let Marathon assign random ports to your applications.
5. Scale the apps further than 1 node using randomly assigned ports.

Step 6: Marathon REST API

The GUI is the primary way to work with Marathon, but it's also good to understand the underlying REST interface for Marathon. This exercise will show you advanced tricks you can do right from the command line using the Marathon REST API. You don't need to know any programming languages to complete this exercise, just a basic understanding of the curl command line HTTP tool.
In this exercise you'll see how these REST calls map to the Marathon GUI and the Marathon command line tool.

```
# get metrics on the running apps

[node1]$ curl http://0.0.0.0:8080/metrics | python -m json.tool | less


# look at the apps you have installed

[node1]$ curl http://0.0.0.0:8080/v2/apps | python -m json.tool


# look at a specific app, named test from Ex4 and Ex5

[node1]$ curl http://0.0.0.0:8080/v2/apps/test | python -m json.tool


# delete that app

[node1]$ curl -X DELETE http://0.0.0.0:8080/v2/apps/test | python -m json.tool


# show that the app is gone
```

```
[node1]$ curl http://0.0.0.0:8080/v2/apps/test | python -m json.tool
```

Step 7: Creating a Slave Node

Using a single node is boring. You'll now create a second node to learn how the Mesosphere stack works with multiple machines, which is the entire point. Most of this exercise is a repeat of the first 8 exercises but compressed and simplified for the requirements of a slave node.

In this exercise you will:

1. Make sure all the node1 services will restart on reboot withchkconfig.
2. Halt the Vagrant for node1 and make a copy of it.
3. Use that copy to create a new master node with a new two nodeVagrantfile.

You need to do some cleanup before you can shut down and pacakgenode1. Make sure that all of the services are properly set to start on boot with chkconfig:

```
[node1]$ sudo chkconfig zookeeper-server on
[node1]$ sudo chkconfig mesos-master on
[node1]$ sudo chkconfig mesos-slave on
[node1]$ sudo chkconfig marathon on
 # if you are running chronos with marathon then do not do this
[node1]$ sudo chkconfig chronos on
```

Once you do that we need to make a Vagrant box out of it so we can copy it over to our new setup:

```
$ vagrant halt
$ vagrant package default
$ vagrant destroy default
$ vagrant box add mesos-master package.box
```

Next you need to add a two node configuration to your Vagrantfile. The line that has your network config now needs this:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

```
Vagrant.configure(2) do |config|
 config.vm.box = "bento/centos-7.1"
 config.vm.define "node1" do |node1|
     node1.vm.network "private_network", ip: "192.168.33.10"
     node1.vm.hostname = "node1"
     node1.vm.box = "mesos-master"
 end

 config.vm.define "node2" do |node2|
     node2.vm.network "private_network", ip: "192.168.33.11"
     node2.vm.hostname = "node2"
 end
end
```

Once you have that in your Vagrantfile you can then dovagrant up and it will recreate your original Vagrant master from thepackage.box file you created naming it node1, and start a new VM named node2 with no configuration in it.

Step 8: Installing Mesos

When the Vagrant VM is ready, you can install the necessary Mesos slave node software. In this exercise:

1. SSH into the new node that you created.
2. Add the Mesosphere RPM repository.
3. Install only the mesos RPM package.
4. Add the new IP for node2 to the /etc/hosts file.
5. Point the /etc/mesos/zk file at the node1 master.
6. Start the slave service and make sure it starts on reboot.

When the Vagrant VM is ready, you can ssh into it with this command:

```
$ vagrant up node2

$ vagrant ssh node2
```

On node2, install Mesos:

```
[node2]$ sudo rpm -Uvh http://repos.mesosphere.com/el/7/noarch/RPMS/mesosphere-el-rep
o-7-1.noarch.rpm

[node2]$ sudo yum -y install mesos
```

Update node2's /etc/hosts file to include entries for both nodes,and remove the "node2" name from the 127.0.0.1 entry on the first line. The file should look like this:

```
127.0.0.1   localhost [.. other localhosts ..]

::1         localhost [.. other localhosts ..]

192.168.33.10 node1

192.168.33.11 node2
```

Now, if you do a ping node2 then you should see 192.168.33.11:

```
[node2]$ ping node2

PING node2 (192.168.33.11) 56(84) bytes of data.

64 bytes from node2 (192.168.33.11): icmp_seq=1 ttl=64 time=0.043 ms

...
```

If node2 resolves to 127.0.0.1 then your first line is wrong. Remove"node2" from the first line.
Edit the /etc/mesos/zk file on node2 to point to the master node:

```
zk://192.168.33.10:2181/mesos
```

Start up Mesos as a slave with:

```
[node2]$ sudo service mesos-slave start
```

Ensure that mesos-slave will be kept running across reboots/failures:

```
[node2]$ sudo chkconfig mesos-slave on

[node2]$ sudo chkconfig mesos-master off

[node2]$ systemctl list-unit-files | grep mesos

mesos-master.service                    disabled

mesos-slave.service                     enabled
```

node2 is now configured.

Finally, switch over to node1 and update /etc/hosts to look the same as on node2, likewise ensuring that the "node1" name isn't present on the first line. Then verify the changes:

```
$ vagrant ssh node1
[node1]$ sudo vi /etc/hosts
[node1]$ ping node1
PING node1 (192.168.33.10) 56(84) bytes of data.
64 bytes from node2 (192.168.33.10): icmp_seq=1 ttl=64 time=0.08 ms
...
[node1]$ ping node2
PING node2 (192.168.33.11) 56(84) bytes of data.
64 bytes from node2 (192.168.33.11): icmp_seq=1 ttl=64 time=1.44 ms
...
```

Step 9: Scaling to Two Nodes

After you have the mesos-slave running on node2, you can use the Marathon GUI to expand the test Python web server out to node2and node1. This exercise is entirely video based.

In this exercise:

Use the Marathon GUI to scale up to 4 nodes.
Observe how node2 now has the test application running in the Mesos GUI.

1. View the two Mesos nodes in the Marathon GUI and see how they are displayed.
2. Use the Marathon GUI to scale down your nodes to 0, then with the newly added node2 active, scale it back up to 4 so you can see them be distributed across the little cluster.
3. Use mesos-dns to discover where they are and what their ports are.

Step 10: Deploying a Web App using Docker

This exercise uses the Go project Outyet to deploy a simple Go-based web application inside a Docker.

In this exercise:

1. Install Docker.
2. Compile the Outyet web application.
3. Build a Docker container that has the Outyet application in it.
4. Get the Docker container running in Marathon on `node1`.

For this exercise I use the instructions found at https://blog.golang.org/docker for deploying the simple Outyet web application. Borrowing from the Go instructions:

Install Docker:

```
[node1]$ sudo yum install -y golang git device-mapper-event-libs docker
[node1]$ sudo chkconfig docker on
[node1]$ sudo service docker start
[node1]$ export GOPATH=~/go
[node1]$ go get github.com/golang/example/outyet
```

The outyet project comes with a `Dockerfile` you can use, so `cd` to the source directory:

```
[node1]$ cd $GOPATH/src/github.com/golang/example/outyet
```

Use the `Dockerfile` to build your docker image:

```
[node1]$ sudo docker build -t outyet .
```

Test the `Dockerfile` before adding it to Marathon by running this command:

```
[node1]$ sudo docker run --publish 6060:8080 --name test --rm outyet
```

Then go to http://192.168.33.10:6060/ with your browser to confirm it works. Once it does you can hit CTRL-c to exit the outyet docker.

Create a Marathon application that runs this command, but using the Marathon Docker support. Once the `outyet` application is loaded onto the VM you can create a new app using JSON and `curl`. First make the file names `/vagrant/outyet.json`:

```
{
  "id": "outyet",
  "cpus": 0.2,
  "mem": 20.0,
  "instances": 1,
  "constraints": [["hostname", "UNIQUE", ""]],
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "outyet",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8080, "hostPort": 0, "servicePort": 0, "protocol": "tcp" }
      ]
    }
  }
}
```

You will also need to tell mesos that it should allow Docker:

```
[node1]$ echo 'docker,mesos' | sudo tee /etc/mesos-slave/containerizers
[node1]$ sudo service mesos-slave restart
```

This replicates the above `docker` command settings, but Marathon will configure and manage the container better. Once you have that run this command:

```
[node1]$ curl -X POST http://192.168.33.10:8080/v2/apps -d @/vagrant/outyet.json -H "Content-type: application/json"
```

Later in this tutorial you will use this method to easily sync your configuration to Marathon.

Step 11: Distributing Docker to Multiple Nodes

After building the Docker image in Exercise 12, you can easily deploy the Outyet web application to your `node2` server. You simply save the Docker and load it on `node2` and then tell Marathon to scale it. This simple procedure lets you automate the deployment of nearly any application that you can "dockerize".

In this exercise:

1. Save the Docker image to a file named `outyet.tar.gz`.
2. Copy the Docker image to the `/vagrant/` directory in `node1`.
3. Load the `outyet.tar.gz` docker container into `node2`.
4. Use Marathon to scale Outyet to two nodes.

Save the Docker image:

```
[node1]$ sudo docker save --output=outyet.tar.gz outyet
```

Send the tar.gz file to `node2`:

```
[node1]$ cp outyet.tar.gz /vagrant/
```

Install Docker on `node2`:

```
$ vagrant ssh node2
[node2]$ sudo yum install -y device-mapper-event-libs docker
[node2]$ sudo chkconfig docker on
[node2]$ sudo service docker start
[node2]$ echo 'docker,mesos' | sudo tee /etc/mesos-slave/containerizers
[node2]$ sudo service mesos-slave restart
```

Import the `outyet.tar.gz` file that you made:

```
[node2]$ sudo docker load --input=/vagrant/outyet.tar.gz
```

Test that Docker is now installed on the `node2` VM:

```
[node2]$ sudo docker run --publish 6060:8080 --name test --rm outyet
```

Test that Docker is running on http://192.168.33.11:6060/.

Enter CTRL-C and go to Marathon and expand this to 2 nodes. Watch the video to see me doing it. At this point, hopefully you know how to go into Marathon and instruct it to run more than one node.