****

Mesosphere Cluster on Ubuntu 14.04

****

Introduction

Mesosphere is a system that combines a number of components to effectively manage server clustering and highly available deployments on top of an existing operating system layer. Unlike systems like CoreOS, Mesosphere is not a specialized operating system and is instead a set of packages.

In this guide, we will go over how to configure a highly available cluster in Mesosphere. This configuration will set us up with failover in case any of our master nodes go down as well as a pool of slave servers to handle the tasks that are scheduled.

We will be using Ubuntu 14.04 servers for this guide.

Prerequisites and Goals

Before you complete this guide, it is strongly recommended that you review our introduction to Mesosphere. This is a good way to get familiar with the components that the system is comprised of and assist you in identifying what each unit is responsible for.

During this tutorial, we will be using six Ubuntu servers. This fulfills the Apache Mesos recommendation of having at least three masters for a production environment. It also provides a pool of three worker or slave servers, which will be assigned work when tasks are sent to the cluster.

The six servers we will be using will use `zookeeper` to keep track of the current leader of the master servers. The Mesos layer, built on top of this, will provide distributed synchronization and resource handling. It is responsible for managing the cluster. Marathon, the cluster's distributed init system, is used to schedule tasks and hand work to the slave servers.

For the sake of this guide, we will be assuming that our machines have the following configuration:

Hostname	***Function***	***IP Address***
master1	***Mesos master***	***192.0.2.1***
master2	***Mesos master***	***192.0.2.2***
master3	***Mesos master***	***192.0.2.3***
slave1	***Mesos slave***	***192.0.2.51***
slave2	***Mesos slave***	***192.0.2.52***
slave3	***Mesos slave***	***192.0.2.53***

Each of these machines should have Ubuntu 14.04 installed. You will want to complete the basic configuration items listed in our Ubuntu 14.04 initial server setup guide.

When you are finished with the above steps, continue on with this guide.

Install Mesosphere on the Servers

The first step to getting your cluster up and running is to install the software. Fortunately, the Mesosphere project maintains an Ubuntu repository with up-to-date packages that are easy to install.

Add the Mesosphere Repositories to your Hosts

On **all** of the hosts (masters and slaves), complete the following steps.

First, add the Mesosphere repository to your sources list. This process involves downloading the Mesosphere project's key from the Ubuntu keyserver and then crafting the correct URL for our Ubuntu release. The project provides a convenient way of doing this:

```
#  sudo apt-get install software-properties-common
#  sudo add-apt-repository ppa:openjdk-r/ppa
#  sudo apt-get update
#  sudo apt-get install openjdk-8-jdk
#  java -version
```

```
#  sudo apt-key adv --keyserver keyserver.ubuntu.com --recv E56151BF
#  DISTRO=$(lsb_release -is | tr '[:upper:]' '[:lower:]')
#  CODENAME=$(lsb_release -cs)
#  echo "deb http://repos.mesosphere.io/${DISTRO} ${CODENAME} main" | sudo tee
/etc/apt/sources.list.d/mesosphere.list
```

Install the Necessary Components

After you have the Mesosphere repository added to your system, you must update your local package cache to gain access to the new component:

```
# sudo apt-get -y update
```

Next, you need to install the necessary packages. The components you need will depend on the role of the host.

For your **master** hosts, you need the `mesosphere` meta package. This includes the `zookeeper`, `mesos`, `marathon`, and `chronos` applications:

```
# sudo apt-get install mesosphere
```

For your **slave** hosts, you only need the `mesos` package, which also pulls in `zookeeper` as a dependency:

```
# sudo apt-get install mesos
```

Set up the Zookeeper Connection Info for Mesos

The first thing we are going to do is configure our `zookeeper` connection info. This is the underlying layer that allows all of our hosts to connect to the correct master servers, so it makes sense to start here.

Our master servers will be the only members of our `zookeeper` cluster, but all of our servers will need some configuration to be able to communicate using the protocol. The file that defines this is `/etc/mesos/zk`.

On **all** of your hosts, complete the following step. Open the file with root privileges:

```
# sudo nano /etc/mesos/zk
```

Inside, you will find the connection URL is set by default to access a local instance. It will look like this:

```
zk://localhost:2181/mesos
```

We need to modify this to point to our three master servers. This is done by replacing `localhost` with the IP address of our first Mesos master server. We can then add a comma after the port specification and replicate the format to add our second and third masters to the list.

For our guide, our masters have IP addresses of `192.0.2.1`, `192.168.2.2`, and `192.168.2.3`. Using these values, our file will look like this:

```
zk://192.0.2.1:2181,192.0.2.2:2181,192.0.2.3:2181/mesos
```

The line must start with `zk://` and end with `/mesos`. In between, your master servers' IP addresses and `zookeeper` ports (`2181` by default) are specified.

Save and close the file when you are finished.

Use this identical entry in each of your masters and slaves. This will help each individual server connect to the correct master servers to communicate with the cluster.

Configure the Master Servers' Zookeeper Configuration

On your **master** servers, we will need to do some additional `zookeeper` configuration.

The first step is to define a unique ID number, from 1 to 255, for each of your master servers. This is kept in the `/etc/zookeeper/conf/myid` file. Open it now:

```
# sudo nano /etc/zookeeper/conf/myid
```

Delete all of the info in this file and replace it with a single number, from 1 to 255. Each of your master servers must have a unique number. For the sake of simplicity, it is easiest to start at 1 and work your way up. We will be using 1, 2, and 3 for our guide.

Our first server will just have this in the file:

```
1
```

Save and close the file when you are finished. Do this on each of your master servers.

Next, we need to modify our `zookeeper` configuration file to map our `zookeeper` IDs to actual hosts. This will ensure that the service can correctly resolve each host from the ID system that it uses.

Open the `zookeeper` configuration file now:

```
# sudo nano /etc/zookeeper/conf/zoo.cfg
```

Within this file, you need to map each ID to a host. The host specification will include two ports, the first for communicating with the leader, and the second for handling elections when a new leader is required. The `zookeeper` servers are identified by "server" followed by a dot and their ID number.

For our guide, we will be using the default ports for each function and our IDs are 1-3. Our file will look like this:

```
server.1=192.168.2.1:2888:3888
server.2=192.168.2.2:2888:3888
server.3=192.168.2.3:2888:3888
```

Add these same mappings in each of your master servers' configuration files. Save and close each file when you are finished.

With that, our `zookeeper` configuration is complete. We can begin focusing on Mesos and Marathon.

Configure Mesos on the Master Servers

Next, we will configure Mesos on the three master servers. These steps should be taken on each of your master servers.

Modify the Quorum to Reflect your Cluster Size

First, we need to adjust the quorum necessary to make decisions. This will determine the number of hosts necessary for the cluster to be in a functioning state.

The quorum should be set so that over 50 percent of the master members must be present to make decisions. However, we also want to build in some fault tolerance so that if all of our masters are not present, the cluster can still function.

We have three masters, so the only setting that satisfies both of these requirements is a quorum of two. Since the initial configuration assumes a single server setup, the quorum is currently set to one.

Open the quorum configuration file:

```
# sudo nano /etc/mesos-master/quorum
```

Change the value to "2":

```
2
```

Save and close the file. Repeat this on each of your master servers.

Configure the Hostname and IP Address

Next, we'll specify the hostname and IP address for each of our master servers. We will be using the IP address for the hostname so that our instances will not have trouble resolving correctly.

For our master servers, the IP address needs to be placed in these files:

- */etc/mesos-master/ip*
- */etc/mesos-master/hostname*

First, add each master node's individual IP address in the `/etc/mesos-master/ip` file. Remember to change this for each server to match the appropriate value:

```
# echo 192.168.2.1 | sudo tee /etc/mesos-master/ip
```

Now, we can copy this value to the hostname file:

```
# sudo cp /etc/mesos-master/ip /etc/mesos-master/hostname
```

Do this on each of your master servers.

Configure Marathon on the Master Servers

Now that Mesos is configured, we can configure Marathon, Mesosphere's clustered init system implementation.

Marathon will run on each of our master hosts, but only the leading master server will be able to actually schedule jobs. The other Marathon instances will transparently proxy requests to the master server.

First, we need to set the hostname again for each server's Marathon instance. Again, we will use the IP address, which we already have in a file. We can copy that to the file location we need.

However, the Marathon configuration directory structure we need is not created automatically. We will have to create the directory and then we can copy the file over:

```
# sudo mkdir -p /etc/marathon/conf
# sudo cp /etc/mesos-master/hostname /etc/marathon/conf
```

Next, we need to define the list of `zookeeper` masters that Marathon will connect to for information and scheduling. This is the same `zookeeper` connection string that we've been using for Mesos, so we can just copy the file. We need to place it in a file called `master`:

```
# sudo cp /etc/mesos/zk /etc/marathon/conf/master
```

This will allow our Marathon service to connect to the Mesos cluster. However, we also want Marathon to store its own state information in `zookeeper`. For this, we will use the other `zookeeper` connection file as a base, and just modify the endpoint.

First, copy the file to the Marathon zookeeper location:

```
# sudo cp /etc/marathon/conf/master /etc/marathon/conf/zk
```

Next, open the file in your editor:

```
# sudo nano /etc/marathon/conf/zk
```

The only portion we need to modify in this file is the endpoint. We will change it from `/mesos` to `/marathon`:

```
zk://192.0.2.1:2181,192.0.2.2:2181,192.0.2.3:2181/marathon
```

This is all we need to do for our Marathon configuration.

Configure Service Init Rules and Restart Services

Next, we will restart our master servers' services to use the settings that we have been configuring.

First, we will need to make sure that our master servers are only running the Mesos master process, and not running the slave process. We can stop any currently running slave processes (this might fail, but that's okay since this is just to ensure the process is stopped). We can also ensure that the server doesn't start the slave process at boot by creating an override file:

```
# sudo stop mesos-slave
# echo manual | sudo tee /etc/init/mesos-slave.override
```

Now, all we need to do is restart `zookeeper`, which will set up our master elections. We can then start our Mesos master and Marathon processes:

```
# sudo restart zookeeper
# sudo start mesos-master
# sudo start marathon
```

ZOOKEEPER FOR MULTI-MASTER CONFIGURATION

Check your zookeeper whether is it running or not,

```
#  sudo netstat -anplt | grep 2181
```

check which system is leader and follower,

```
#  find / -name zkServer.sh
#  {zookeeper-home-directory}//bin/zkServer.sh status
```

output:

```
JMX enabled by default
Using config: /etc/zookeeper/conf/zoo.cfg
Mode: follower
```

we can connect to the another zookeeper machine also,

```
#  /usr/share/zookeeper/bin/zkCli.sh -server 192.168.1.253:2181
```

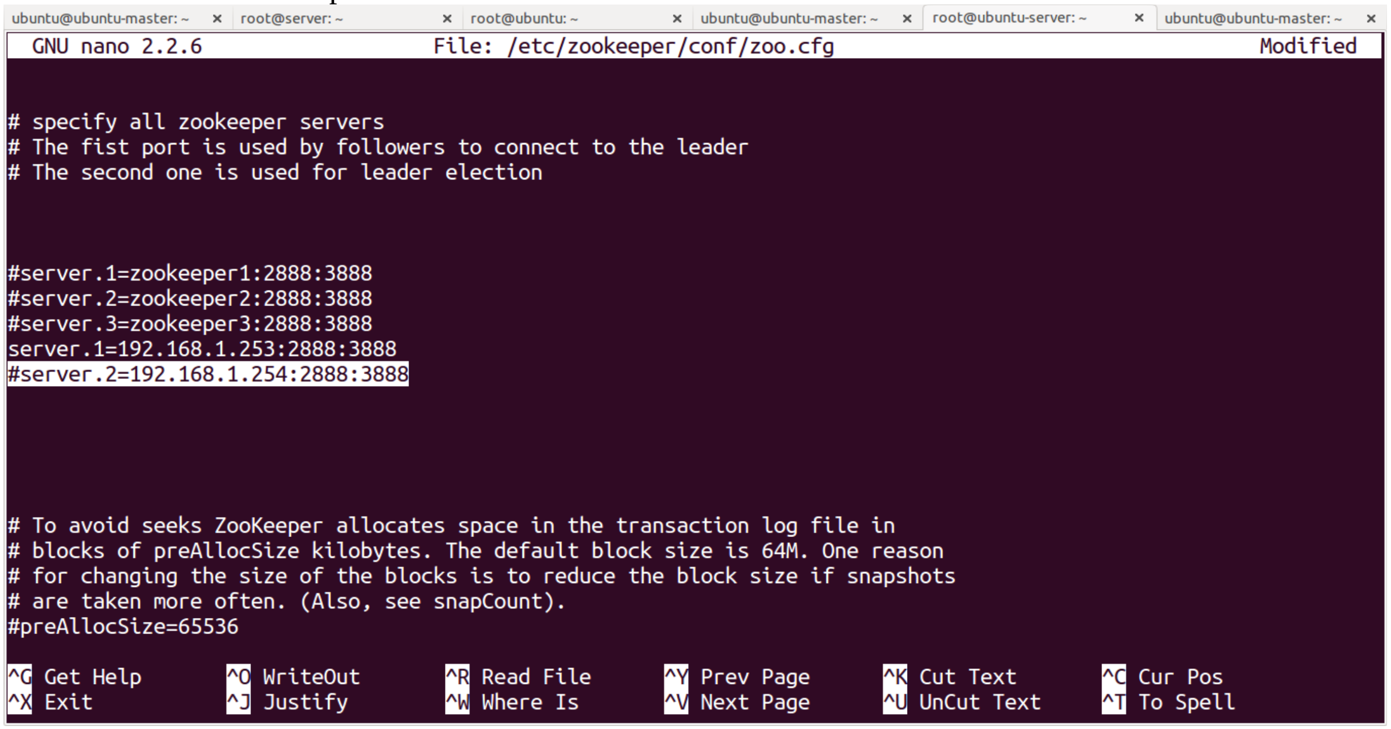
output:

```
Connecting to 192.168.1.253:2181
Welcome to ZooKeeper!
JLine support is enabled

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: 192.168.1.253:2181(CONNECTED) 0]
```

if one zookeeper server is down, then need to change the configuration , othervise WUI willnot work for mesos,

```
# sudo nano /etc/zookeeper/conf/zoo.cfg
```

commend out the server ip which is down



```
  GNU nano 2.2.6                File: /etc/zookeeper/conf/zoo.cfg                     Modified


# specify all zookeeper servers
# The fist port is used by followers to connect to the leader
# The second one is used for leader election



#server.1=zookeeper1:2888:3888
#server.2=zookeeper2:2888:3888
#server.3=zookeeper3:2888:3888
server.1=192.168.1.253:2888:3888
#server.2=192.168.1.254:2888:3888




# To avoid seeks ZooKeeper allocates space in the transaction log file in
# blocks of preAllocSize kilobytes. The default block size is 64M. One reason
# for changing the size of the blocks is to reduce the block size if snapshots
# are taken more often. (Also, see snapCount).
#preAllocSize=65536

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

restart the zookeeper service,

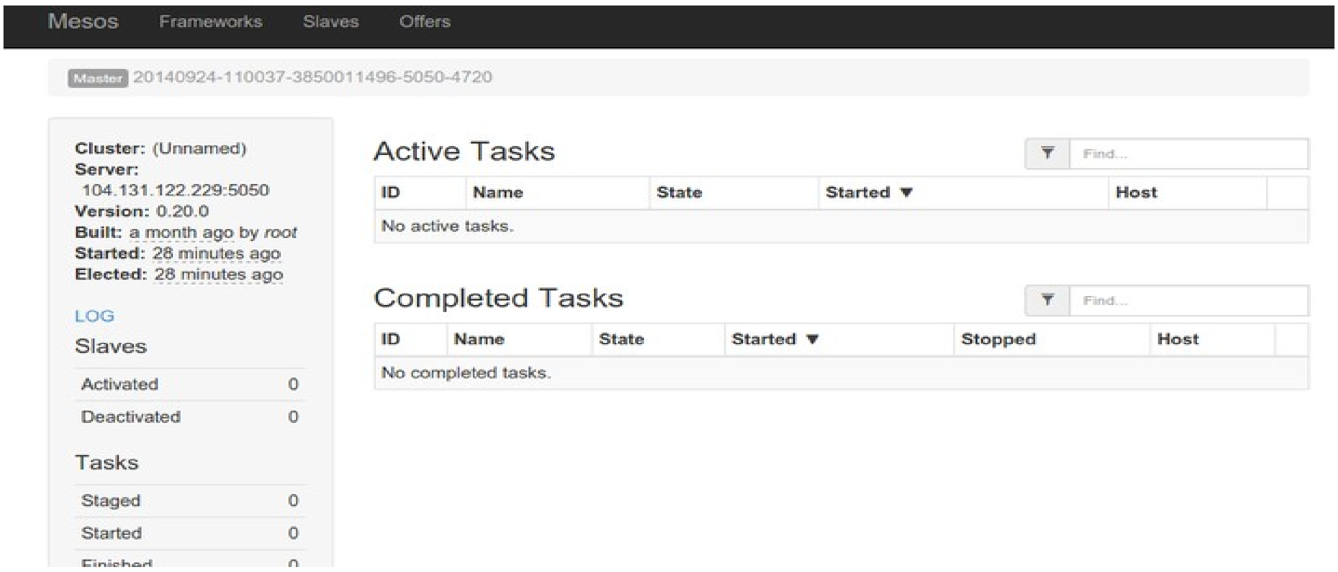
# sudo restart zookeeper

access the WUI, now it will work

To get a peak at what you have just set up, visit one of your master servers in your web browser at port `5050`:

`http://192.168.2.1:5050`

You should see the main Mesos interface. You may be told you are being redirected to the active master depending on whether you connected to the elected leader or not. Either way, the screen will look similar to this:



20140924-110037-3850011496-5050-4720

Cluster: (Unnamed)
Server: 104.131.122.229:5050
Version: 0.20.0
Built: a month ago by *root*
Started: 28 minutes ago
Elected: 28 minutes ago

LOG

Slaves

Activated	0
Deactivated	0

Tasks

Staged	0
Started	0
Finished	0

Active Tasks

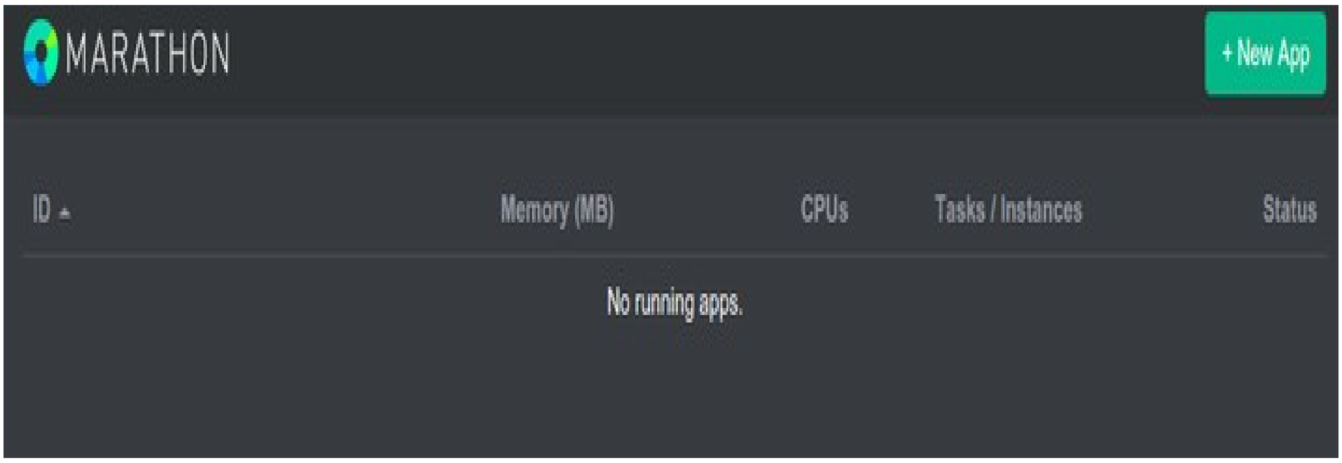
ID	Name	State	Started ▼	Host
No active tasks.				

Completed Tasks

ID	Name	State	Started ▼	Stopped	Host
No completed tasks.					

This is a view of your cluster currently. There is not much to see because there are no slave nodes available and no tasks started.

We have also configured Marathon, Mesosphere's long-running task controller. This will be available at port `8080` on any of your masters:



No running apps.

We will briefly go over how to use these interfaces once we get our slaves set up.

Configure the Slave Servers

Now that we have our master servers configured, we can begin configuring our slave servers.

We have already configured our slaves with our masters servers' `zookeeper` connection information. The slaves themselves do not run their own `zookeeper` instances.

On **all** of your hosts, complete the following step. Open the file with root privileges:

```
# sudo nano /etc/mesos/zk
```

Inside, you will find the connection URL is set by default to access a local instance. It will look like this:

```
zk://localhost:2181/mesos
```

We need to modify this to point to our three master servers. This is done by replacing `localhost` with the IP address of our first Mesos master server. We can then add a comma after the port specification and replicate the format to add our second and third masters to the list.

For our guide, our masters have IP addresses of `192.0.2.1`, `192.168.2.2`, and `192.168.2.3`. Using these values, our file will look like this:

```
zk://192.0.2.1:2181,192.0.2.2:2181,192.0.2.3:2181/mesos
```

We can stop any `zookeeper` process currently running on our slave nodes and create an override file so that it will not automatically start when the server reboots:

```
# sudo stop zookeeper
# echo manual | sudo tee /etc/init/zookeeper.override
```

Next, we want to create another override file to make sure the Mesos master process doesn't start on our slave servers. We will also ensure that it is stopped currently (this command may fail if the process is already stopped. This is not a problem):

```
# echo manual | sudo tee /etc/init/mesos-master.override
# sudo stop mesos-master
```

Next, we need to set the IP address and hostname, just as we did for our master servers. This involves putting each node's IP address into a file, this time under the `/etc/mesos-slave` directory. We will use this as the hostname as well, for easy access to services through the web interface:

```
# echo 192.168.2.51 | sudo tee /etc/mesos-slave/ip
# sudo cp /etc/mesos-slave/ip /etc/mesos-slave/hostname
```

Again, use each slave server's individual IP address for the first command. This will ensure that it is being bound to the correct interface.

Now, we have all of the pieces in place to start our Mesos slaves. We just need to turn on the service:

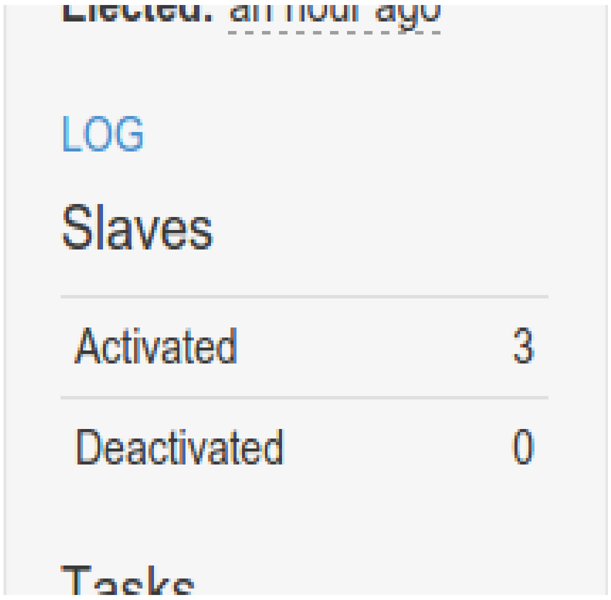
```
# sudo start mesos-slave
```

Do this on each of your slave machines.

To see whether your slaves are successfully registering themselves in your cluster, go back to your one of your master servers at port `5050`:

http://192.168.2.1:5050

You should see the number of active slaves at "3" now in the interface:



Elected: an hour ago

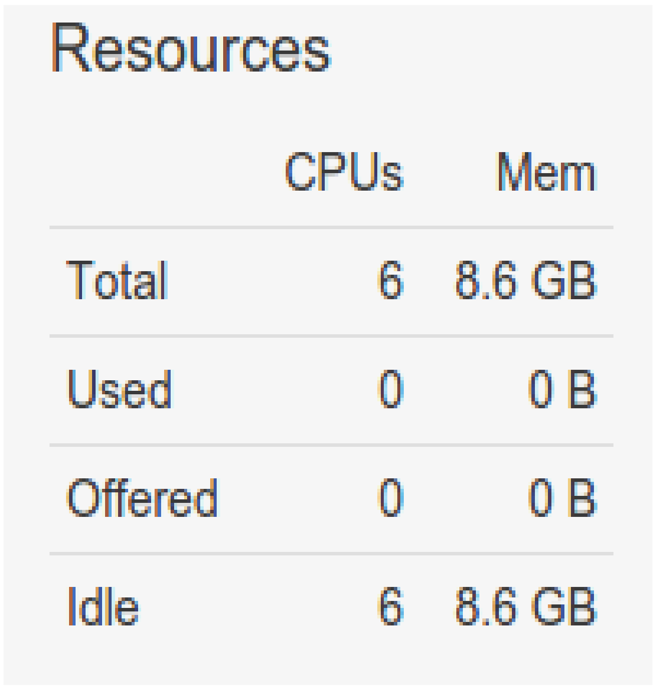
LOG

Slaves

Activated	3
Deactivated	0

Tasks

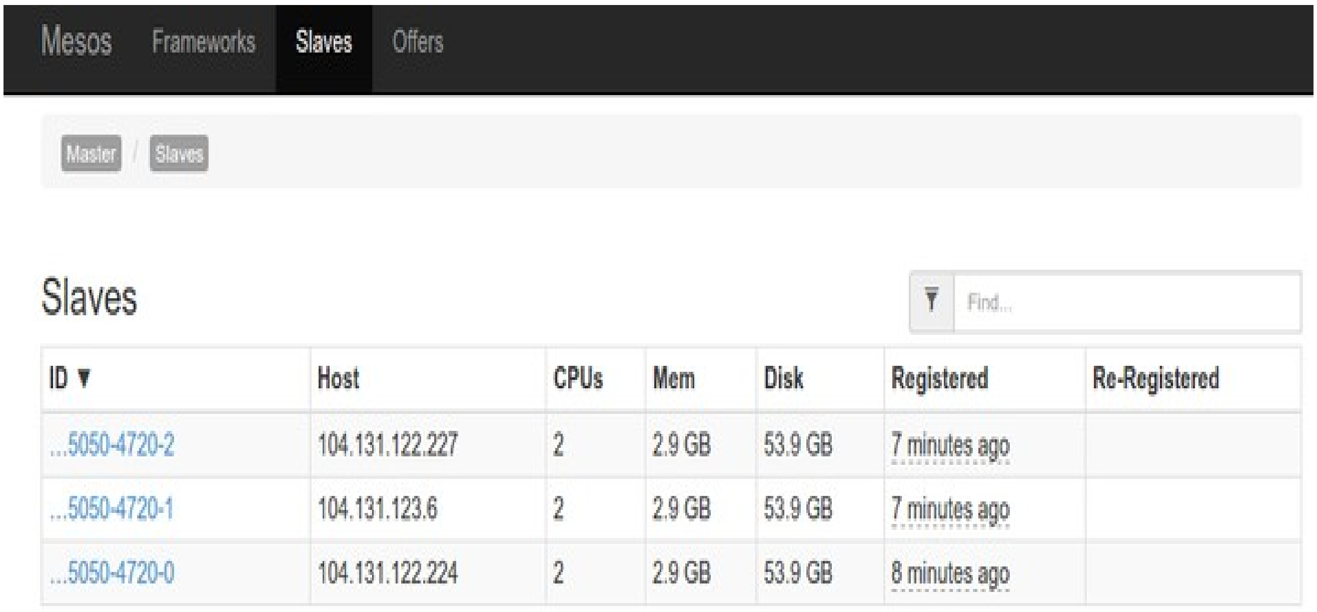
You can also see that the available resources in the interface has been updated to reflect the pooled resources of your slave machines:



Resources

	CPUs	Mem
Total	6	8.6 GB
Used	0	0 B
Offered	0	0 B
Idle	6	8.6 GB

To get additional information about each of your slave machines, you can click on the "Slaves" link at the top of the interface. This will give you an overview of each machine's resource contribution, as well as links to a page for each slave:



Slaves

ID ▾	Host	CPUs	Mem	Disk	Registered	Re-Registered
...5050-4720-2	104.131.122.227	2	2.9 GB	53.9 GB	7 minutes ago	
...5050-4720-1	104.131.123.6	2	2.9 GB	53.9 GB	7 minutes ago	
...5050-4720-0	104.131.122.224	2	2.9 GB	53.9 GB	8 minutes ago	

Starting Services on Mesos and Marathon

Marathon is Mesosphere's utility for scheduling long-running tasks. It is easy to think of Marathon as the init system for a Mesosphere cluster because it handles starting and stopping services, scheduling tasks, and making sure applications come back up if they go down.

You can add services and tasks to Marathon in a few different ways. We will only be covering basic services. Docker containers will be handled in a future guide.

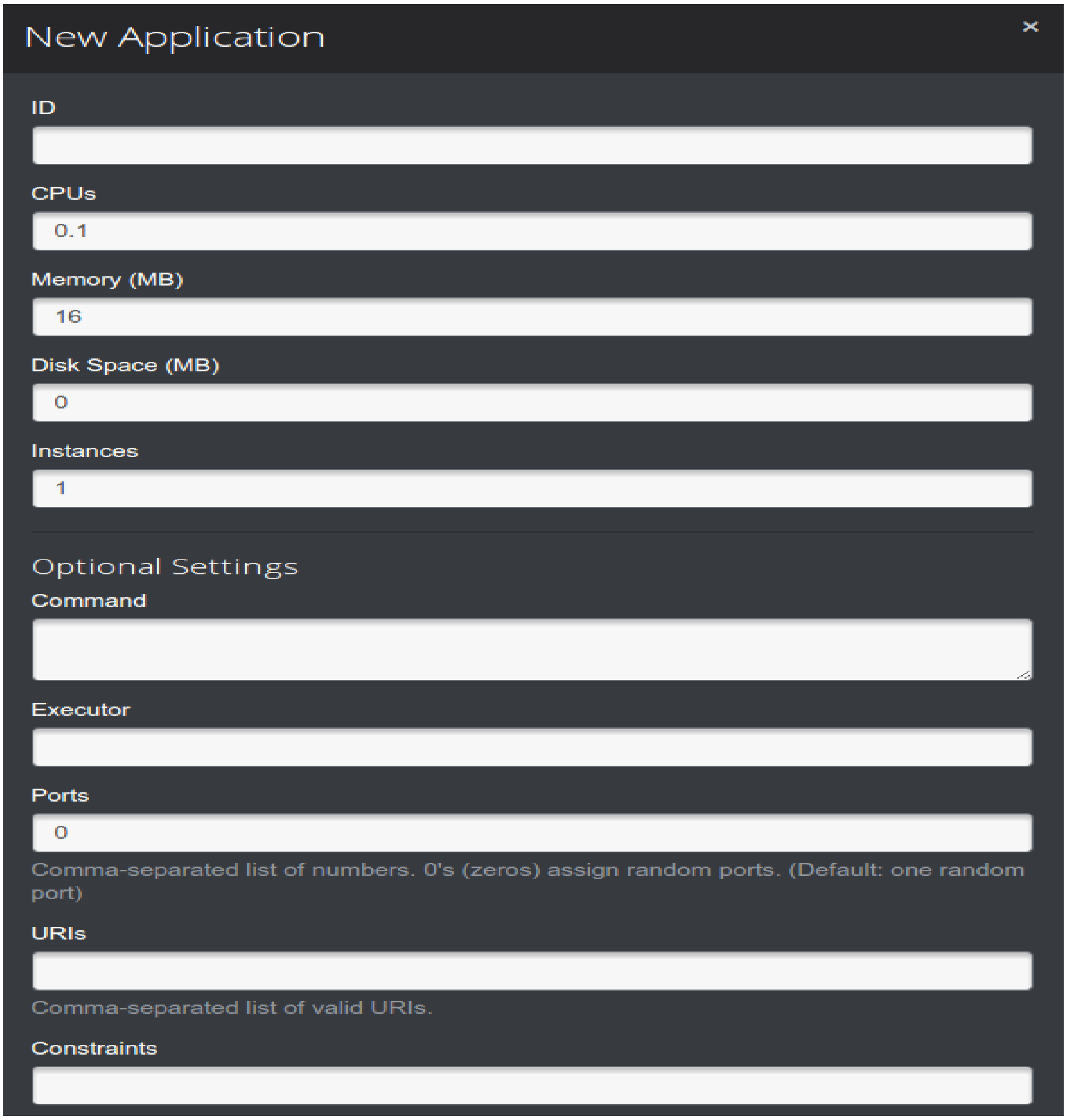
Starting a Service through the Web Interface

The most straight forward way of getting a service running quickly on the cluster is to add an application through the Marathon web interface.

First, visit the Marathon web interface on one of your the master servers. Remember, the Marathon interface is on port `8080`:

```
http://192.168.2.1:8080
```

From here, you can click on the "New App" button in the upper-right corner. This will pop up an overlay where you can add information about your new application:



New Application

ID

CPUs
0.1

Memory (MB)
16

Disk Space (MB)
0

Instances
1

Optional Settings

Command

Executor

Ports
0
Comma-separated list of numbers. 0's (zeros) assign random ports. (Default: one random port)

URIs

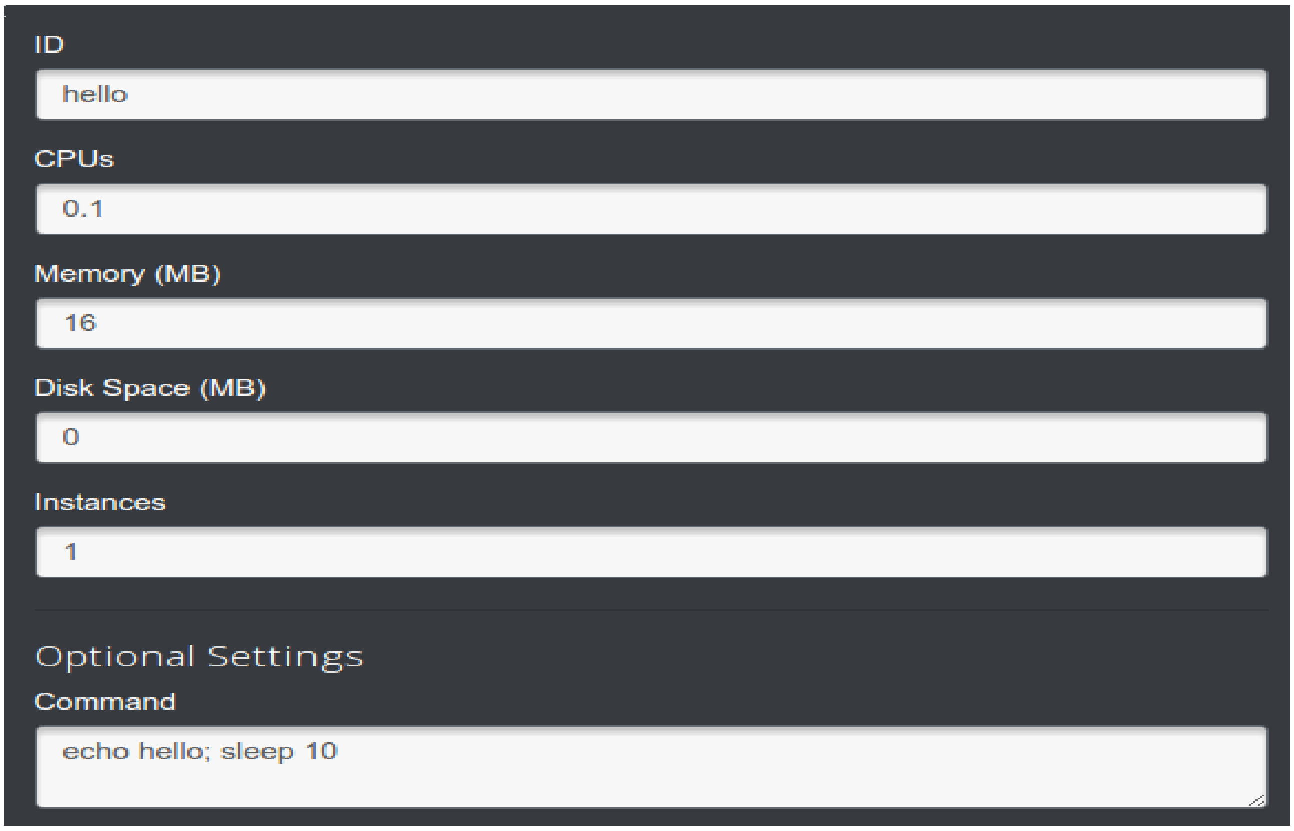
Comma-separated list of valid URIs.

Constraints

Fill in the fields with the requirements for your app. The only fields that are mandatory are:

- **ID**: A unique ID selected by the user to identify a process. This can be whatever you'd like, but must be unique.
- **Command**: This is the actual command that will be run by Marathon. This is the process that will be monitored and restarted if it fails.

Using this information, you can set up a simple service that just prints "hello" and sleeps for 10 seconds. We will call this "hello":


ID

hello

CPUs

0.1

Memory (MB)

16

Disk Space (MB)

0

Instances

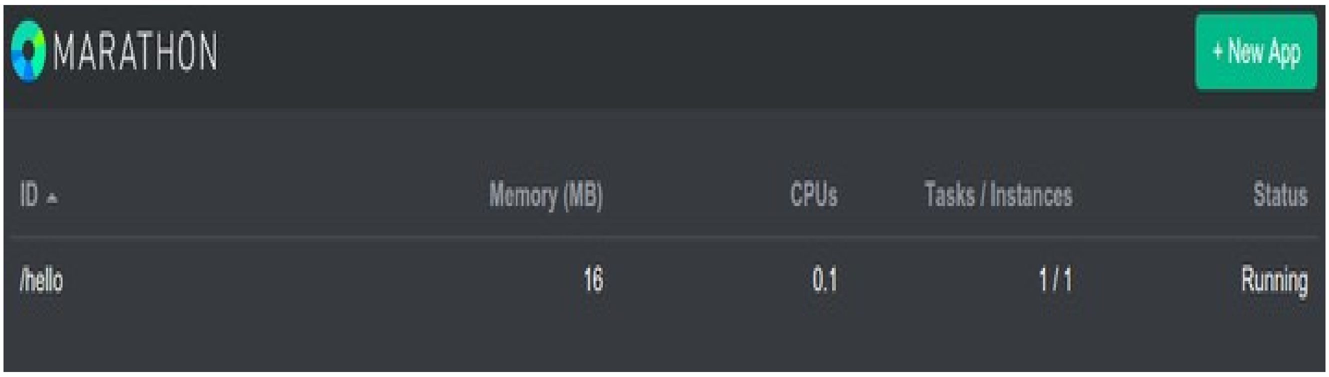
1

Optional Settings

Command

echo hello; sleep 10

When you return to the interface, the service will go from "Deploying" to "Running":



ID ▴	Memory (MB)	CPUs	Tasks / Instances	Status
/hello	16	0.1	1 / 1	Running

Every 10 seconds or so, the "Tasks/Instances" reading will go from "1/1" to "0/1" as the sleep amount passes and the service stops. Marathon then automatically restarts the task again. We can see this process more clearly in the Mesos web interface at port `5050`:

http://192.168.2.1:5050

Starting a Service through the API

We can also submit services through Marathon's API. This involves passing in a JSON object containing all of the fields that the overlay contained.

This is a relatively simple process. Again, the only required fields are `id` for the process identifier and `cmd` which contains the actual command to run.

So we could create a JSON file called `hello.json` with this information:

```
# nano hello.json
```

Inside, the bare minimum specification would look like this:

```
{
    "id": "hello2",
    "cmd": "echo hello; sleep 10"
}
```

This service will work just fine. However, if we truly want to emulate the service we created in the web UI, we have to add some additional fields. These were defaulted in the web UI and we can replicate them here:

```
{
    "id": "hello2",
    "cmd": "echo hello; sleep 10",
    "mem": 16,
    "cpus": 0.1,
    "instances": 1,
    "disk": 0.0,
    "ports": [0]
}
```

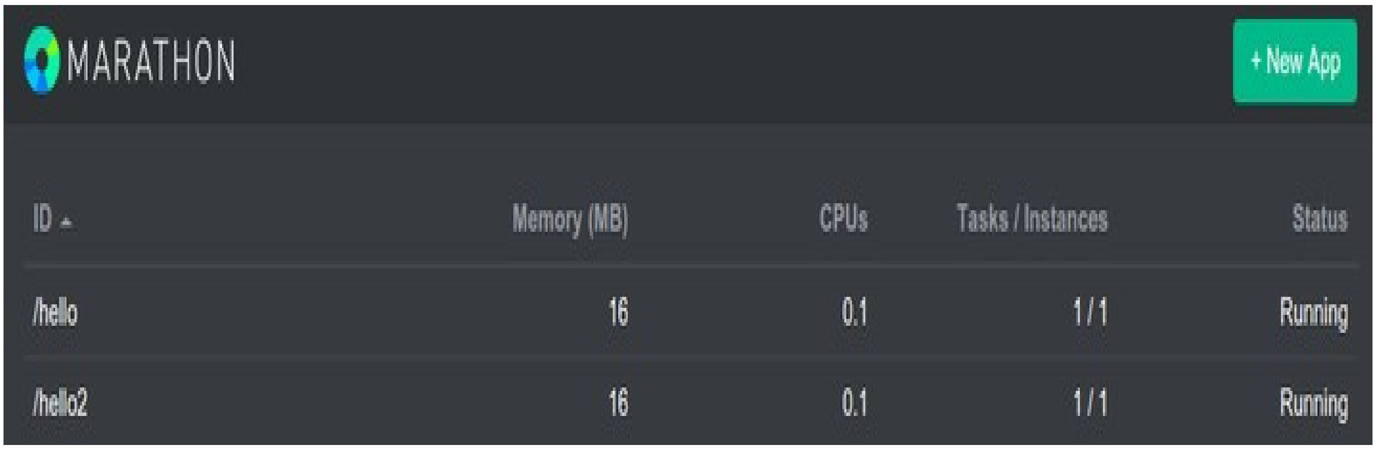
Save and close the JSON file when you are finished.

Next, we can submit it using the Marathon API. The target is one of our master's Marathon service at port `8080` and the endpoint is `/v2/apps`. The data payload is our JSON file, which we can read into `curl` by using the `-d` flag with the `@` flag to indicate a file.

The command to submit will look like this:

```
# curl -i -H 'Content-Type: application/json' -d@hello2.json 192.168.2.1:8080/v2/apps
```

If we look at the Marathon interface, we can see that it was successfully added. It seems to have the exact same properties as our first service:



ID	Memory (MB)	CPUs	Tasks / Instances	Status
/hello	16	0.1	1 / 1	Running
/hello2	16	0.1	1 / 1	Running

The new service can be monitored and accessed in exactly the same way as the first service.

Json file ::

```
{
 "container": {
   "type": "DOCKER",
   "docker": {
     "image": "eboraas/apache",
     "network": "BRIDGE",
     "portMappings": [
       { "containerPort": 80, "hostPort": 0, "protocol": "tcp", "servicePort": 80 },
       { "containerPort": 443, "hostPort": 0, "protocol": "tcp", "servicePort": 443 }
       ]
 }
 },
 "id": "tomcat",
 "instances": 1,
 "cpus": 1,
 "mem": 512
}
```

conditions ::

1) starting a block with "{" and must end with "," unless it is a single sub-block.

Eg:1 >>
{ blah...........blah..................blah }

Eg:2 >>
{ blah...........blah..................blah },
{ blah...........blah..................blah }

2) don't provide "," at the last in a sub-block if it is last sub-block.

Eg >>
{ blah...........blah..................blah },
{ blah...........blah..................blah },
{ blah...........blah..................blah },
{ blah...........blah..................blah }

3) don't provide particular port in "portMappings" for "hostPort" if done then this task will not execute anbd showing just scheduling.

Eg >>

```
 "containerPort": 80, "hostPort": 0, "protocol": "tcp", "servicePort": 80
```

****
Links ::

installation>>
https://www.digitalocean.com/community/tutorials/how-to-configure-a-production-ready-mesosphere-cluster-on-ubuntu-14-04
http://frankhinek.com/setup-mesos-multi-node-cluster-on-ubuntu/

json file creation>>
http://frankhinek.com/deploy-docker-containers-on-mesos-0-20/
https://mesosphere.github.io/marathon/docs/native-docker.html
https://mesosphere.github.io/marathon/docs/ports.html

errorwaiting forever>>
https://mesosphere.github.io/marathon/docs/troubleshooting.html

****