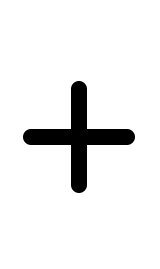
[**Building a Microservice Infrastructure using Mesos, Marathon and Zookeeper**](http://www.thedevpiece.com/building-a-microservice-infrastructure-using-mesos-marathon-and-zookeeper/)

[](http://www.thedevpiece.com/)

[**Gabriel Francisco**](http://www.thedevpiece.com/)

*Software Engineer at UOL, Bachelor of Computer Science and MBA in Service-oriented Software Engineering.*



This post will be the first of a "series" that I'm planning about an infrastructure for Microservices.

First of all, a few concepts:

* According to their [web site](http://mesos.apache.org/), **Mesos** is a distributed systems kernel. Mesos abstracts every machine resource of a cluster to build a "super computer", exposing it as an unique resource pool.
* **Marathon**, in the other hand, is a resource manager that runs on top of Mesos. Using Marathon you will be able to deploy containers and applications into a Mesos cluster.
* **Zookeeper** is an open source server that provides a highly reliable distributed coordination. In simple words, Zookeeper will build our cluster and Mesos will use it to operate.
* **Nginx** is a web server that can be also used as a reverse proxy.
* **Nixy** is an application written in Go that automatically configures Nginx for the applications deployed on Marathon/Mesos.

**Architecture:**

[alt](http://www.thedevpiece.com/content/images/2016/07/arch--2-.png)

In words:

* Totally, there will be 17 machines.
* Five machines will represent our "master pool", which will contain a Mesos master, a Zookeeper and a Marathon in each of them.
* Nine machines for our slave pool. These machines will only have a Mesos slave in each of them that will register in the Zookeeper cluster.
* Three machines as our proxy pool. They will be in charge for the service discovery and load balancing of our applications. These machines will have only Nginx and Nixy.
* The slave pool is where your applications will run. The master pool is the pool that will coordinate the execution of your applications in the slave pool.
* This architecture is highly scalable. You can have as many slave machines as you want without the need to change any configuration, just add more slave machines to the pool and your resource pool will grow.
* Each one of the three Nixy instances will register itself in each Marathon for the service discovery.
* The main load balancer can be a DNS round-robin, another Nginx, maybe a HAProxy, etc. It is up to you do decide.
* **Every machine of this tutorial uses a CentOS 7**

**Machines**

|  |  |  |
| --- | --- | --- |
| **Number** | **Pool** | **Hostname** |
| #1 | Slave | slave-1 |
| #2 | Slave | slave-2 |
| #3 | Slave | slave-3 |
| #4 | Slave | slave-4 |
| #5 | Slave | slave-5 |
| #6 | Slave | slave-6 |
| #7 | Slave | slave-7 |
| #8 | Slave | slave-8 |
| #9 | Slave | slave-9 |
| #10 | Master | master-1 |
| #11 | Master | master-2 |
| #12 | Master | master-3 |
| #13 | Master | master-4 |
| #14 | Master | master-5 |
| #15 | Proxy | proxy-1 |
| #16 | Proxy | proxy-2 |
| #17 | Proxy | proxy-3 |

**Configuring the master pool**

The following commands should be executed in every master machine.

**Installing the repository**

sudo rpm -Uvh <http://repos.mesosphere.com/el/7/noarch/RPMS/mesosphere-el-repo-7-1.noarch.rpm>

**Installing Zookeeper, Marathon and Mesos in the master pool**

As I said before, we will install Mesos, Zookeeper and Marathon in each machine.

sudo yum -y install mesos marathon && sudo yum -y install mesosphere-zookeeper

**Configuring Zookeeper**

Set **/etc/zookeeper/conf/myid** to the id of the current master.

Example for master one:

echo 1 > /etc/zookeeper/conf/myid

Configure **/etc/zookeeper/conf/zoo.cfg**, informing each machine our cluster will have.

server.1=master-1:2888:3888

server.2=master-2:2888:3888

server.3=master-3:2888:3888

server.4=master-4:2888:3888

server.5=master-5:2888:3888

**Configuring Mesos**

Now, you need to edit your **/etc/mesos/zk** informing the ZK url of your cluster:

echo zk://master-1:2181,master-2:2181,master-3:2181,master-4:2181,master-5:2181/mesos > /etc/mesos/zk

Since we have five master machines, the mesos quorum will be three:

echo 3 > /etc/mesos-master/quorum

**Disabling Mesos slave on the master nodes**

Since we have a separated slave pool, there is no need to use Mesos slave alongside our Mesos master instances, so, let's disable it:

systemctl stop mesos-slave.service

systemctl disable mesos-slave.service

**Configuring Marathon**

Add the following lines to **/etc/sysconfig/marathon**:

MARATHON\_EVENT\_SUBSCRIBER=http\_callback

MARATHON\_TASK\_LAUNCH\_TIMEOUT=600000

MARATHON\_TASK\_LOST\_EXPUNGE\_GC=60000

MARATHON\_TASK\_LOST\_EXPUNGE\_INITIAL\_DELAY=60000

MARATHON\_TASK\_LOST\_EXPUNGE\_INTERVAL=60000

Depending on the task you want to run in Marathon, it can take quite while to start, so, we increased the **task launch timeout** to 10 minutes. Also, we are enabling the http callback feature for the service discovery, thus, Nixy will register itself in Marathon for the service discovery and finally update Nginx if any service is healthy/unhealthy. Also, we decreased the interval time for Marathon to remove lost tasks that may happen sometimes.

**Restarting Mesos, Marathon and Zookeeper**

sudo systemctl restart zookeeper

sudo service mesos-master restart

sudo service marathon restart

That'd be all.

**Configuring the slave pool**

The following commands should be executed in every slave machine.

**Installing the repository**

sudo rpm -Uvh <http://repos.mesosphere.com/el/7/noarch/RPMS/mesosphere-el-repo-7-1.noarch.rpm>

**Installing Mesos**

sudo yum -y install mesos

**Configuring Mesos**

The same as the master nodes, you need to edit your **/etc/mesos/zk** informing the ZK url of your master cluster:

echo zk://master-1:2181,master-2:2181,master-3:2181,master-4:2181,master-5:2181/mesos > /etc/mesos/zk

**Disabling Mesos master in the slave nodes**

Since the mesos package is the same, we need to disable the master instance in each slave machine, so, there will be only mesos slave instances running:

sudo systemctl stop mesos-master.service

sudo systemctl disable mesos-master.service

**Restarting Mesos**

sudo service mesos-slave restart

**Validating our cluster**

If everything worked as expected, you should be able to access Mesos using:

<http://master-1:5050/>

The cluster will elect a Mesos master as a leader. If the master-1 is not the leader, it will redirect to the master leader.

You should be able to see the resources, such as how much memory, CPU and how many slaves you have in your resource pool.

Also, you can validate if Marathon is up, accessing:

<http://master-1:8080/>

You can access any Marathon from your master cluster.

**Configuring the proxy pool**

The following commands should be executed in every proxy machine.

**Installing Nginx**

First, add the Nginx repository:

sudo yum install epel-release

Then, install Nginx:

sudo yum install nginx

After that, enable it to start when your machine starts:

sudo systemctl enable nginx

Finally, you start it:

sudo systemctl start nginx

**Installing Nixy**

Get the latest Nixy .tar.gz from [their github release page.](https://github.com/martensson/nixy/releases/latest) Currently it is 0.6.0.

wget <https://github.com/martensson/nixy/releases/download/v0.6.0/nixy_0.6.0_linux_amd64.tar.gz>

After downloading, uncompress it:

tar -xvf nixy\_0.6.0\_linux\_amd64.tar.gz

And then move it to /opt/nixy

mv nixy\_0.6.0\_linux\_amd64/ /opt/nixy/

**Configuring Nixy**

Edit your **/opt/nixy/nixy.toml** to:

port = "8000"

marathon = ["[http://master-1:8080](http://master-1:8080/)", "[http://master-2:8080](http://master-2:8080/)", "[http://master-3:8080](http://master-3:8080/)", "[http://master-4:8080](http://master-4:8080/)", "[http://master-5:8080](http://master-5:8080/)"]

nginx\_config = "/etc/nginx/nginx.conf"

nginx\_template = "/opt/nixy/nginx.tmpl"

nginx\_cmd = "nginx"

After that, edit your **/opt/nixy/nginx.tmpl**:

worker\_processes auto;

pid /run/nginx.pid;

events {

use epoll;

worker\_connections 2048;

multi\_accept on;

}

http {

add\_header X-Proxy {{ .Xproxy }} always;

access\_log off;

error\_log /var/log/nginx/error.log warn;

server\_tokens off;

client\_max\_body\_size 128m;

proxy\_buffer\_size 128k;

proxy\_buffers 4 256k;

proxy\_busy\_buffers\_size 256k;

proxy\_redirect off;

map $http\_upgrade $connection\_upgrade {

default upgrade;

'' close;

}

# time out settings

proxy\_send\_timeout 120;

proxy\_read\_timeout 120;

send\_timeout 120;

keepalive\_timeout 10;

{{- range $id, $app := .Apps}}

upstream {{index $app.Hosts 0}} {

{{- range $app.Tasks}}

server {{ . }};

{{- end}}

}

{{- end}}

server {

listen 80;

server\_name app.org;

location / {

return 503;

}

{{- range $id, $app := .Apps}}

location /{{index $app.Hosts 0}}/ {

proxy\_set\_header HOST $host;

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

proxy\_connect\_timeout 30;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection $connection\_upgrade;

proxy\_pass http://{{index $app.Hosts 0}};

}

{{- end}}

}

}

This is your Nginx template. If you deploy an app named as "hello-world" in Marathon, Nixy will reload your Nginx and you will be able to access it in: <http://proxy-n/hello-world/>, where N is 1, 2 or 3.   
I don't think it is needed to say, but, thanks do Nginx, there's no downtime when Nixy is reloading it.

**Configure Nixy to start as a service**

Since Nixy has no stop/start script, we will create ours.

Add the following lines to **/etc/systemd/system/nixy.service**:

[Unit]

Description=Nixy Service

After=nginx.service

[Service]

Type=simple

ExecStart=/bin/sh -c '/opt/nixy/nixy -f /opt/nixy/nixy.toml &> /var/log/nixy.log'

Restart=always

[Install]

WantedBy=multi-user.target

And then, enable it to start when our machine is up:

sudo systemctl enable nixy

Lastly but not least, start Nixy: (be sure that both Nginx and Marathon are running)

sudo systemctl start nixy

**Validating**

You can check if Nixy is running checking its status:

sudo systemctl status nixy

Or its log:

tail -f /var/log/nixy.log

Or even its URL:

curl -X GET <http://localhost:8000/v1/health>

**Other posts about REST and Microservices**