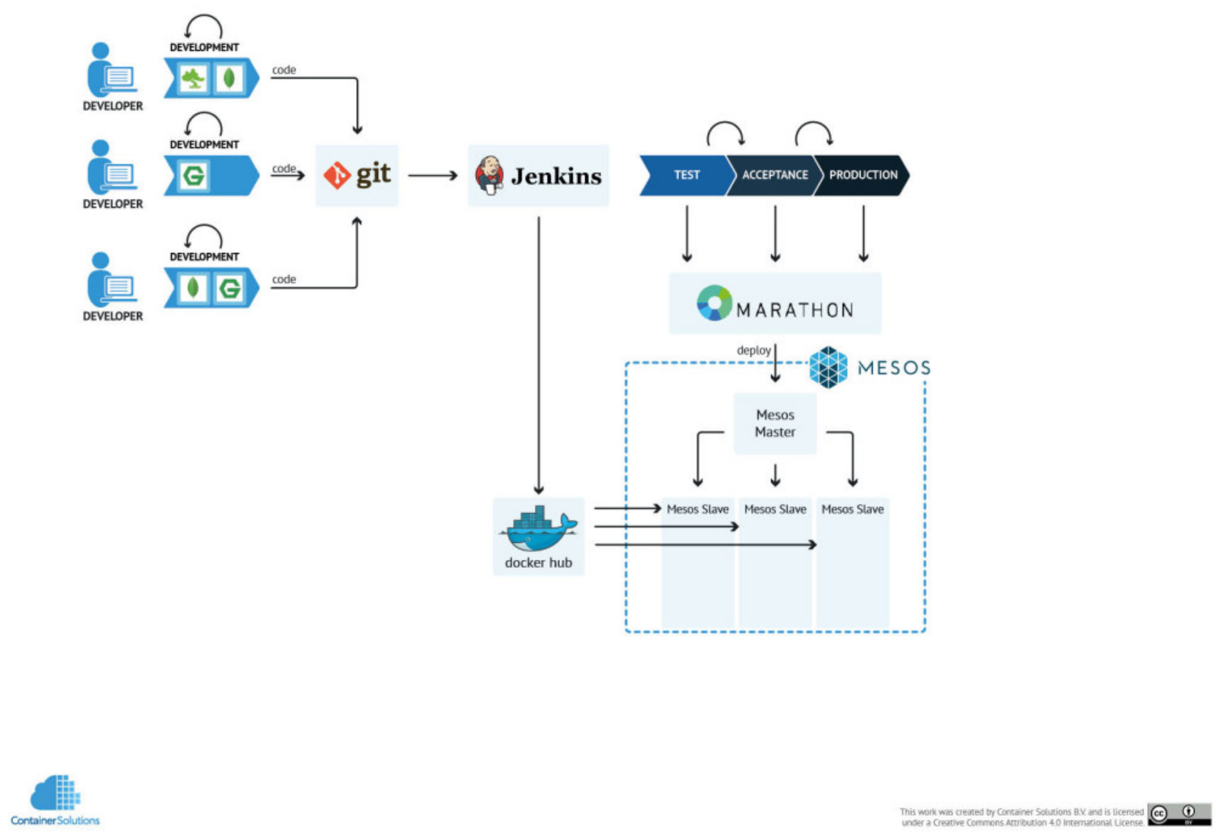
CI/CD tutorial - Part 2

In the Part 1 I showed how to dockerize a node.js application on the development machine and later deploy Jenkins and Docker registry using Docker Compose and use them for continuous integration of the node.js app.

In Part 2 I continue the setup for Mesos and Marathon and complete the Continuous Delivery cycle.

Cloud



If you have never heard about Mesos or Marathon, at this point it will be useful to read a bit about them: here, here or here.

Now, when we have functional development and continuous integration environments, we can start building the Mesos cluster.

Here is the full *docker-compose.yml* file that includes all parts of the system. In addition to previously configured Jenkins and Docker registry we have Mesos master, single Mesos slave, Mesosphere Marathon and Zookeeper for internal Mesos communication.

```
# Zookeeper: -p 2181:2181 -p 2888:2888 -p 3888:3888
zookeeper:
  image: jplock/zookeeper:3.4.5
  ports:
    - "2181"
master:
  image: mesoscloud/mesos-master
  hostname: master
  links:
    - zookeeper:zookeeper
  environment:
    - MESOS_ZK=zk://zookeeper:2181/mesos
    - MESOS_QUORUM=1
    - MESOS_WORK_DIR=/var/lib/mesos
    - MESOS_LOG_DIR=/var/log
  ports:
    - "5050:5050"
marathon:
  #image: garland/mesosphere-docker-marathon
  image: mesosphere/marathon
  links:
    - zookeeper:zookeeper
  ports:
    - "8080:8080"
  # this image does not respect MARATHON_ env variables, so adding the params via
command
  command: --master zk://zookeeper:2181/mesos --zk zk://zookeeper:2181/marathon
slave:
  image: mesoscloud/mesos-slave
  links:
    - zookeeper:zookeeper
    - master:master
  environment:
    - MESOS_MASTER=zk://zookeeper:2181/mesos
    - MESOS_EXECUTOR_REGISTRATION_TIMEOUT=5mins
    - MESOS_CONTAINERIZERS=docker,mesos
    - MESOS_ISOLATOR=cgroups/cpu,cgroups/mem
    - MESOS_LOG_DIR=/var/log
  volumes:
    - /var/run/docker.sock:/run/docker.sock
    - /usr/bin/docker:/usr/bin/docker
    - /sys:/sys:ro
    - mesosslace-stuff:/var/log
  expose:
    - "5051"
jenkins:
  image: containersol/jenkins_with_docker
  links:
    - marathon:marathon
  volumes:
    - jenkins-stuff:/var/jenkins_home
    - .:/var/jenkins_data
    - /var/run/docker.sock:/var/run/docker.sock
    - /usr/bin/docker:/usr/bin/docker
  ports:
    - "8081:8080"
registry:
  image: registry
  environment:
    - STORAGE_PATH=/registry
  volumes:
    - registry-stuff:/registry
  ports:
    - "5000:5000"
```

There is not much to explain in the *docker-compose.yml*. All of the environment parameters are taken from the usage instructions for the images on the Docker Hub.

The Mesos slave container also using the trick with the mounted socket, but nothing needs to be fixed here since slave is running the root user that has permissions to access the socket.

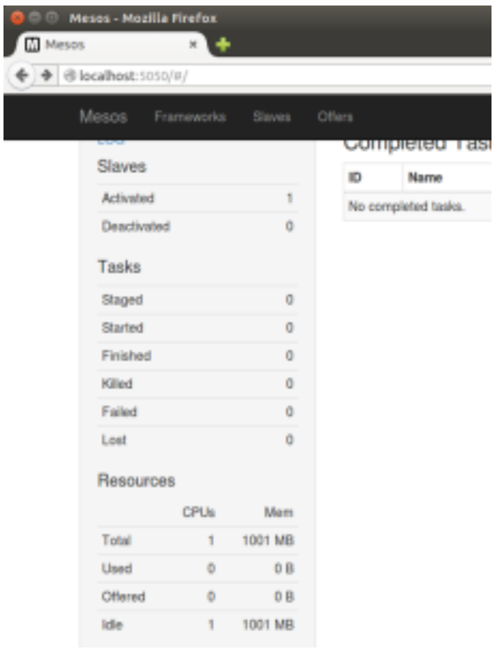
It is important to note that the Jenkins container now includes a link to Marathon. This is required to be able to post the requests from the Jenkins container to the Marathon container. We will see it in the next part about deployment.

Now we can restart the system and see it all up and running:

```
$ docker-compose up
Creating cddemo_registry_1...
Recreating cddemo_zookeeper_1...
Creating cddemo_master_1...
Creating cddemo_slave_1...
Recreating cddemo_marathon_1...
Creating cddemo_jenkins_1...
```

The containers will start very quickly (as they always do), but it will take about 30 seconds until all the services are online (on an Ubuntu VM running on MacBook Air).

Mesos will start on . You can also see one active slave in the following screenshot. The slave has no publicly exposed port in this setup.



Slaves

Activated	1
Deactivated	0

Tasks

Staged	0
Started	0
Finished	0
Killed	0
Failed	0
Lost	0

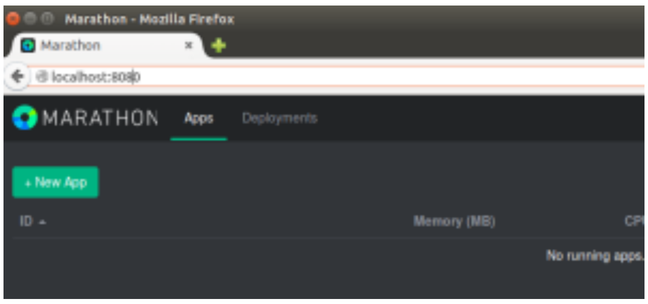
Resources

	CPUs	Mem
Total	1	1001 MB
Used	0	0 B
Offered	0	0 B
Idle	1	1001 MB

Completed Tas

ID	Name
No completed tasks.	

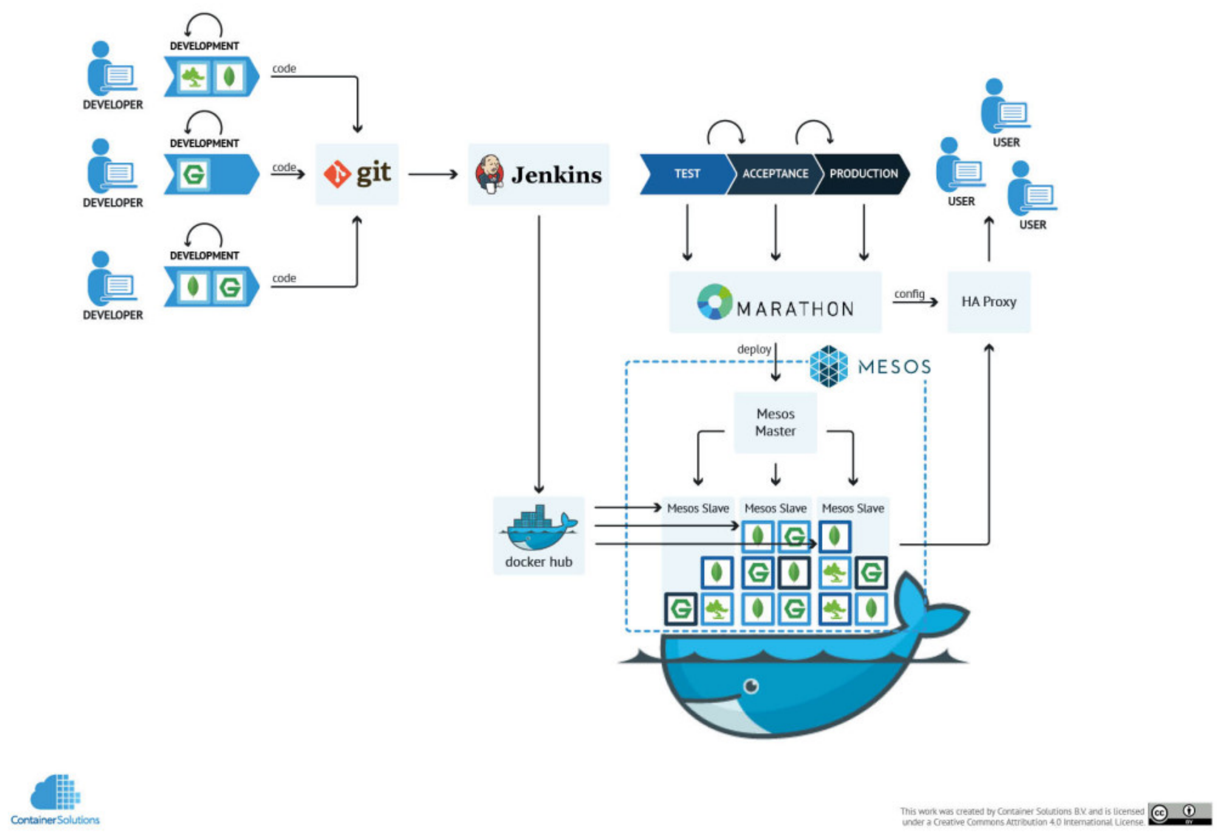
Marathon will be accessible at



No running apps.

Deployment

The last part of the journey is to deploy our freshly built docker image on Mesos using Marathon.



First we need to create the configuration file for scheduling an application on Marathon, let's call it *app_marathon.json*:

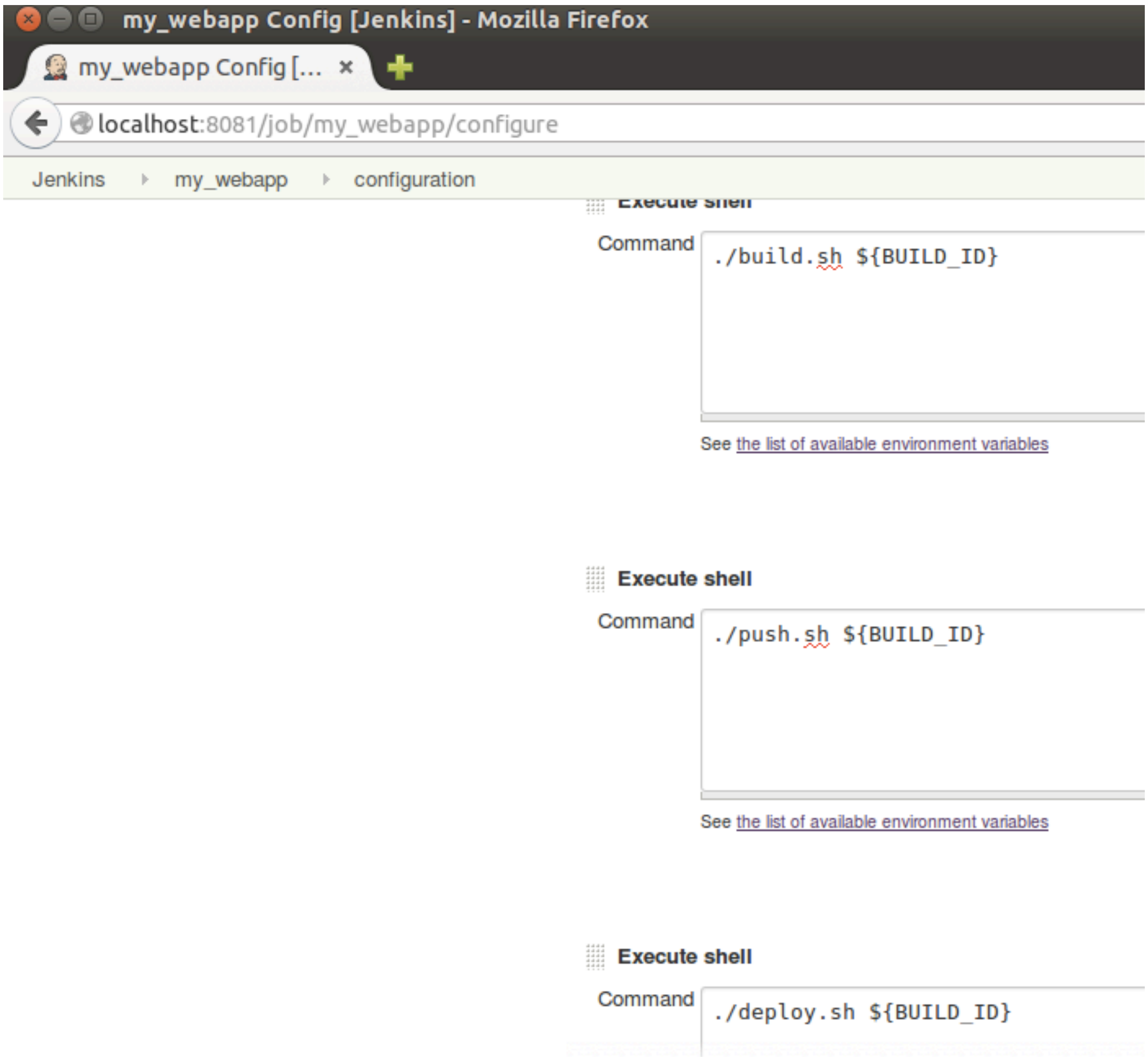
```
{
    "id": "app",
    "container": {
      "docker": {
        "image": "localhost:5000/containersol/nodejs_app:latest",
        "network": "BRIDGE",
        "portMappings": [
          {"containerPort": 8000, "servicePort": 8000}
        ]
      }
    },
    "cpus": 0.2,
    "mem": 512.0,
    "instances": 1
}
```

Here again there are some shortcuts. For example, an important missing piece is a health-check that should tell Marathon when the application is running and when it isn't.

Once we have the json file to publish we can add the last script *deploy.sh* that will remove the currently running application and redeploy it using the new image. There are better upgrade strategies but, again, I won't discuss them here.

```
#!/bin/bash

if [ -z "${1}" ]; then
   version="latest"
   marathon="localhost"
else
   version="${1}"
   marathon=${MARATHON_PORT_8080_TCP_ADDR}
fi

# destroy old application
curl -X DELETE -H "Content-Type: application/json"
http://${marathon}:8080/v2/apps/app

# I know this one is ugly. But it works for now.
sleep 1

# these lines will create a copy of app_marathon.json and update the image version
cp -f app_marathon.json app_marathon.json.tmp
sed -i "s/latest/${version}/g" app_marathon.json.tmp

# post the application to Marathon
curl -X POST -H "Content-Type: application/json" http://${marathon}:8080/v2/apps -
d@app_marathon.json.tmp
```

The last step would be to add *deploy.sh* to Jenkins configuration and run the build.


my_webapp Config [Jenkins] - Mozilla Firefox

localhost:8081/job/my_webapp/configure

Jenkins › my_webapp › configuration

Execute shell

Command
```
./build.sh ${BUILD_ID}
```
See the list of available environment variables

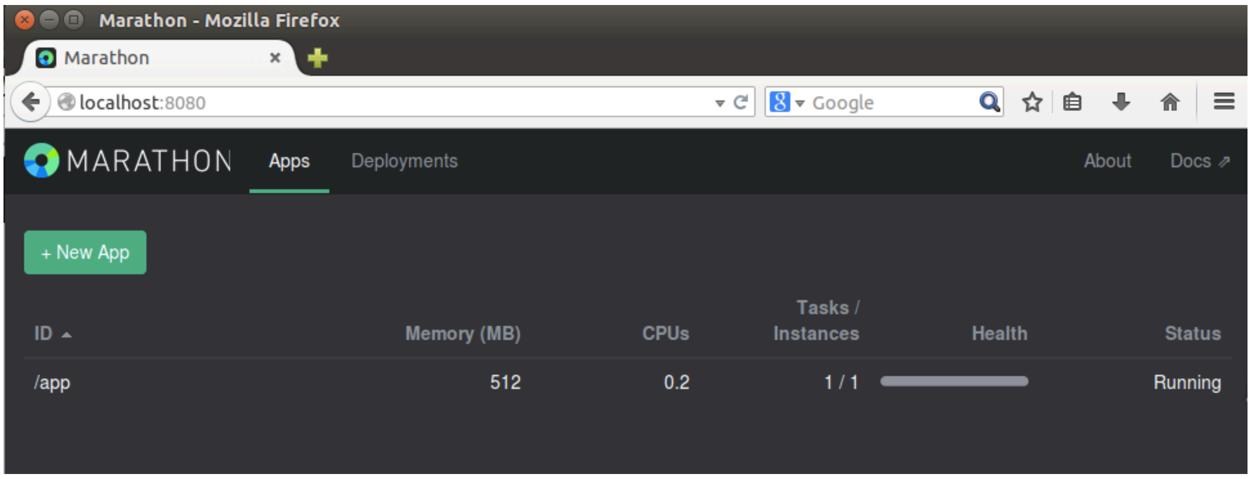
Execute shell

Command
```
./push.sh ${BUILD_ID}
```
See the list of available environment variables

Execute shell

Command
```
./deploy.sh ${BUILD_ID}
```

And after the build has successfully finished we can see the application running on Marathon:


Marathon - Mozilla Firefox

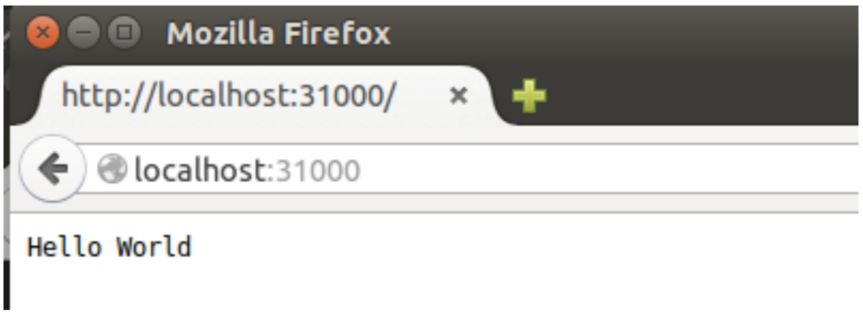
localhost:8080

MARATHON Apps Deployments About Docs

+ New App

ID	Memory (MB)	CPUs	Tasks / Instances	Health	Status
/app	512	0.2	1 / 1		Running

And we can see our app on :



```
Hello World
```

Now you can try changing your application and triggering the build in Jenkins. In just few seconds it will find it's way through Jenkins, the Docker Hub and Marathon to Mesos!

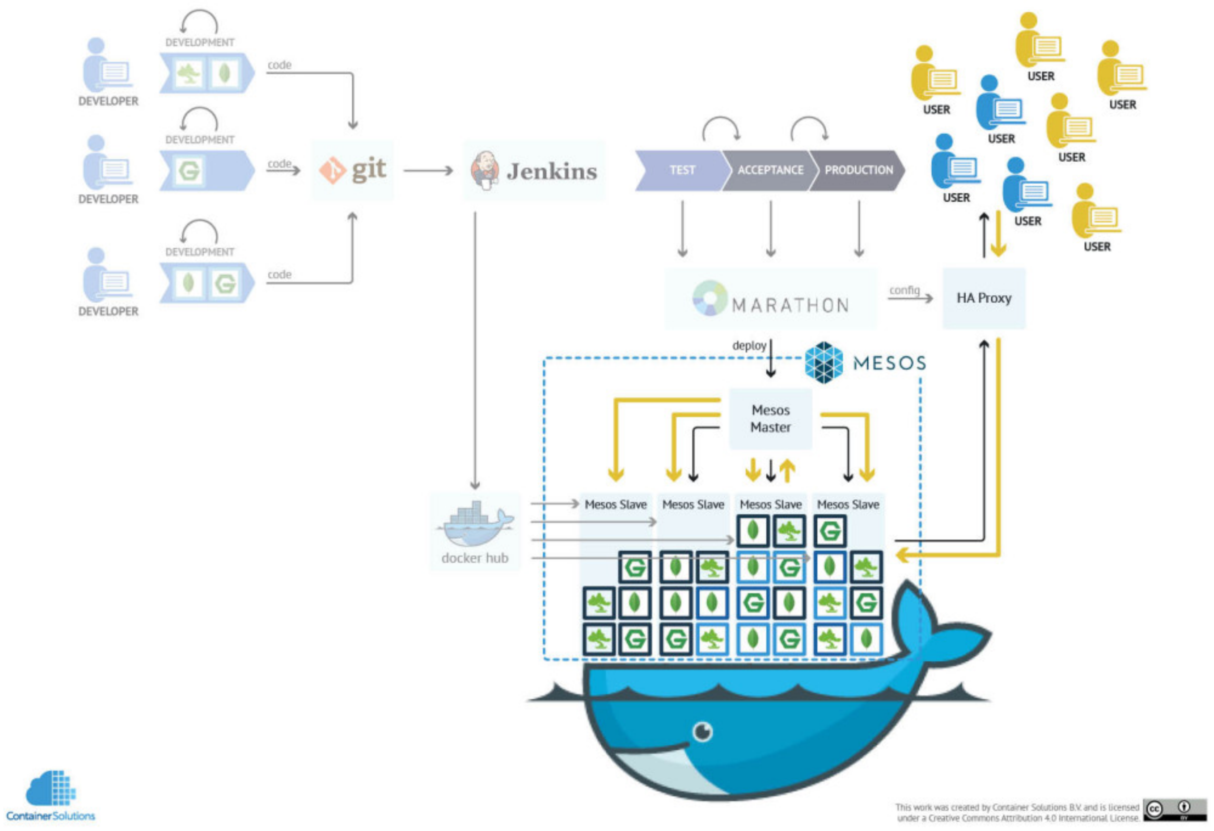
Future Improvements

There are two main direction to improve this system – adding more functionality and deepening the quality of the setup.

The list of possible extensions is very long. Here are some examples:

- Extending HelloWorld example to be a proper web application
- Adding multiple languages
- Using a multi-container setup deployed on Mesos
- Adding automated tests on multiple levels (unit tests, systems tests, performance tests, etc.
- Triggering Jenkins builds automatically from a Git hook
- Deploying to public clouds such as GCE, AWS etc
- Running on multiple hosts.
- HAProxy setup
- Auto-scaling with simulation of load using jmeter
- Deploying a microservices based system
- Using Flocker for persistent storage
- Using Weave for networking containers
- Using Consul for automatic service discovery
- Adding system monitoring
- Adding centralised logging

My preferred next step would be to focus of the part of the system facing external users and add HAProxy and auto-scaling capabilities as demonstrated in the next diagram:


DEVELOPER
DEVELOPMENT
code
DEVELOPER
DEVELOPMENT
code
DEVELOPER
DEVELOPMENT
code
git
Jenkins
TEST
ACCEPTANCE
PRODUCTION
USER
MARATHON
config
HA Proxy
deploy
MESOS
Mesos Master
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
docker hub
ContainerSolutions
This work was created by Container Solutions B.V. and is licensed under a Creative Commons Attribution 4.0 International License.