Docker orchestration, service discovery, and Vagrant

**Objective**:

Create a small web application which on request retrieves several rows of data  
from the database.  
Dockerizing the web app and the database. The application should resolve database  
configuration using the service discovery tool of your choice.  
Web application and database should run on different virtual nodes, configured by  
Vagrant.

**Prerquisites**

Install Vagrant - <https://www.vagrantup.com/downloads.html>

Download/git clone the repo which contains the necessary setup - <https://github.com/venkrao/docker-vagrant-consul.git>

**Configuring database node**

Go to the command prompt/equivalent place depending on the operating system(run->cmd, or a terminal), and run the following commands from the directory where the GIt repo is checked out to/downloaded to.

1. cd database
2. vagrant up
3. vagrant ssh
4. Get the ip address of docker0 interface, and update the /etc/default/docker(DOCKER\_OPTS variable) to use docker interface as the DNS server(which can be read by using “ifconfig docker0”) using the following:

sudo echo ‘DOCKER\_OPTS="--dns <ip address of the0> --dns 8.8.8.8 --dns 8.8.4.4"’ >> /etc/default/docker

This is needed for service discovery applications to use the consul service name in the .consul domain as host name, for example, consul.service.consul, and the DNS server from consul will resolve it to the IP address of the nodes that have this service running. Consul returns a random IP of the host running the given service, on a round-robin basis, and this serves as load balancing too.

1. Restart docker –

sudo restart docker

1. Spin up the consul, and registrator containers using the following command:

cd /opt/scripts;

node\_ip=<private ip of this node> bridge\_ip=<ip address of docker0> ./consul.sh

This should start the consul service, and the registrator service, that will watch for all the containers that publish the ports, and registers them with Consul.

From the browser, you should now, if you go to the address: http://<private\_ip\_of\_webapp\_node>:8500/

see the web UI offered by Consul with only one node registered, showing a few ports under monitoring.

1. Now, start the mysql docker container using the following commands:

docker run --name mysql\_server -p **<private\_ip\_of\_this\_host>**:3306:3306 -e MYSQL\_ROOT\_PASSWORD=Gnleln\_2 -d mysql:latest && sleep 5

docker run -it -v /opt/config:/opt/config -v /opt/scripts:/opt/scripts --link mysql\_server:mysql --rm mysql sh -c '/opt/scripts/startdb.sh'

**Configuring the webapp node**

Go to the command prompt/equivalent place depending on the operating system(run->cmd, or a terminal), and run the following commands from the directory where the Git repo is checked out to/downloaded to.

1. cd webapp
2. vagrant up
3. vagrant ssh
4. Get the ip address of docker0 interface, and update the /etc/default/docker(DOCKER\_OPTS variable) to use docker interface as the DNS server(which can be read by using “ifconfig docker0”) using the following:

sudo echo ‘DOCKER\_OPTS="--dns <ip address of the0> --dns 8.8.8.8 --dns 8.8.4.4"’ >> /etc/default/docker

This is needed for service discovery applications to use the consul service name in the .consul domain as host name, for example, consul.service.consul, and the DNS server from consul will resolve it to the IP address of the nodes that have this service running. Consul returns a random IP of the host running the given service, on a round-robin basis, and this serves as load balancing too.

1. Restart docker –

sudo restart docker

1. Spin up the consul, and registrator containers using the following command:

cd /opt/scripts;

node\_ip=<private ip of this node> bridge\_ip=<ip address of docker0> **join\_ip=<Private IP address of database node>** ./consul.sh

OBSERVE THAT we are adding a second node to the Consul cluster, bootstrapped on the database node.(i.e the first node in the Consul cluster is the DB node)

This should start the consul service, and the registrator service, that will watch for all the containers that publish the ports, and registers them with Consul.

From the browser, you should now, if you go to the address: http://<private\_ip\_of\_webapp\_node>:8500/

you should see the web UI offered by Consul.

1. Now, start the nginx container using the following command

docker run --name nginx -p 8080:80 -v /opt/webapp:/usr/share/nginx/html richarvey/nginx-php-fpm:latest

**Result**

At this stage, at the URL http://<private\_ip\_of\_webapp\_node>:8080, you should see a simple listing of 4 countries, and their dialing codes,

**Testing service discovery/automatic fallback.**

So far as we just had one node running webserver, and one node running the database, if the database goes down, then the webapp is unusable.

In a realworld scenario, perhaps the database runs on a cluster of machines, and the service discovery should identify the faulty nodes, ad not route the requests to them, but to the ones that are healthy.

In this simulated setup, to prove the failover/automatic discovery if the primary node goes down, please follow the same set of steps listed under “database node setup”, to spin up **a third host(or backup database node)**, and run all the steps mentioned in there, but set the “join\_ip=private\_ip\_of\_first\_database\_node> while starting the Consul service.

When the backup node is a member of the Consul cluster, stop the mysql container on the primary db node using “docker stop <container>” and you should see that the webapp will still work as Consul will take care of routing the requests to the backup db node automatically. This setup is backed by the Consul DNS interface, and it is faster than the traditional DNS synchronization in routing the requests to the backup node of the Consul’s design backed by the Gossip protocol, that checks when services/nodes are started/stopped.

And with both primary, and backup node running, the request is rounted on a round-robin basis to both and hence the load on the Consul cluster is uniform.

**Remarks**

* Preparing this setup involves some manual intervention because making assumptions like “Assigning a static private IP to the nodes” involves a bit of risk as it just might be the one already used in the users’s network. Hence it is left to DHCP to assign a private IP, and then, it has to be noted by the user configuring Consul.
* Consul and Docker maybe used together in 2 ways.
  + - 1. **Running Consul inside Docker container**.(This is how we used Consul so far)

Though this works just fine for a simple use case like ours, it doesn’t have the health check facility running

* + - 1. **Running Consul on the host running Docker(or container)**

With this, health checks may be run more easily than the last case.