**Hyperledger Fabric Multi Node Network Setup on AWS using Docker Swarm**

**konda.kalyan@gmail.com**

Github Repo: [**https://github.com/konda-kalyan/HLF-Multi-Host-Docker-Swarm**](https://github.com/konda-kalyan/HLF-Multi-Host-Docker-Swarm)

Except ‘with\_kafka’ branch, RAFT is used as Orderer type in all other branches.

I have used AWS as hosting environment. If you are using other cloud or physical machines, you need to update AWS references accordingly.

Reference site: [**https://www.skcript.com/svr/setting-up-a-multi-node-hyperledger-fabric-network-with-docker-swarm/**](https://www.skcript.com/svr/setting-up-a-multi-node-hyperledger-fabric-network-with-docker-swarm/)

~~Another site in which docker-swarm has been used as well. There are some slight differences:~~

* 1. ~~All nodes are configured as Manager Nodes~~
  2. ~~Used docker-compose up to bring up containers, (rather docker stack deploy)~~

[~~https://medium.com/@kctheservant/multi-host-setup-with-raft-based-ordering-service-29730788b171~~](https://medium.com/@kctheservant/multi-host-setup-with-raft-based-ordering-service-29730788b171)

**Network topology and different components:**

* Network name is ‘**hlf\_multi\_host\_network**’
* Fabric network with **3 organizations** installed on VMs
* **Kafka/Raft (based on the git branch you have chosen)** based ordering service with 3 Orderers (one per each Organization)
* **One Fabric CA** per Organization
* **CLI** is on Org1 – peer0 (only one CLI)
* **Couchdb** as world state on every Peer.
* **One Channel** called ‘mychannel’
* Based on the branch you chose,

**One Chaincode** named ‘simple’ (written in ‘go’ language), or

**One Chaincode** named ‘couchdb\_java\_simple’ (written in ‘java’ language)

Used ‘**Docker Swarm**’ as container orchestration tool.

**One Manager Node** and **two Worker Nodes**.

**Org1** components are deployed on **ManagerNode**

**Org2** components are deployed on **WorkerNode1**

**Org3** components are deployed on **WorkerNode2**.

#### **STEP 1 - Initial preparations on VMs**

1. **Create 3 VMs**

**Do all steps (mentioned in this section) on all 3 VMs**

**If your deployment environment is AWS, then it is recommended that VM instance type should be minimum ‘t2.medium’. Also, make sure that required ports are opened.**

Note: I have tested on Ubuntu VMs with 16.04 version.

**Caution**: Better set PS1 (in .bashrc) accordingly to make sure that we are doing steps on right machine. It avoids confusion. Example:

*export PS1="[ Org1 - :- \u@\h:\w]$ "*

*sudo apt update*

1. **Install Docker and Docker Compose**

**Do all steps (mentioned in this section) on all 3 VMs**

To install Docker

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04>

*sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu* ***xenial*** *stable"*

Based on Ubuntu version (18.04 or 16.04), change bionic or xenial. Currently, I am setting up on 16.04 machine and hence I have used **xenial.**

**Add ‘User’ to the Docker group so that you no need to run docker commands with ‘sudo’**

To install docker-compose

<https://www.digitalocean.com/community/tutorials/how-to-install-docker-compose-on-ubuntu-16-04>

Install docker-compose Command-line completion (Note that command-line completion is available for docker when you install docker)

*sudo curl -L https://raw.githubusercontent.com/docker/compose/1.25.5/contrib/completion/bash/docker-compose -o /etc/bash\_completion.d/docker-compose*

Checkpoint on installations:

*docker version*

*docker-compose version*

1. **Make sure Manager Node can SSH and SCP to Worker Nodes**

MANAGER NODE is the one where we run most of the commands and operate

**Run this step only on Manager node**

Ref link: <https://www.youtube.com/watch?v=OwptfrdgzDA>

In ‘~/.ssh’ directory, create a file called ‘id\_rsa’ and copy the contents of .**pem** file (basically private key) which you have downloaded while you launched AWS VM.

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/.ssh$] vim id\_rsa*

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/.ssh$] chmod 600 id\_rsa*

Checkpoint: Just check whether you can ssh and scp to Worker nodes

1. **‘AWS specific’ docker daemon configuration**

If you are setting up this environment on AWS, then do below step. Skip this step otherwise.

**Do all steps (mentioned in this section) on all 3 VMs**

**Follow steps explained in this link:** <https://success.docker.com/article/why-do-my-services-stay-pending-when-trying-to-schedule-them-with-placement-contraints>

1. Add labels to the daemon by editing the *‘/etc/docker/daemon.json*’ file. Of course, the region and zone need to update accordingly (based on VPC region/zone your VMs are running).

**Note: VMs might have created on different regions and zones. Just watch and configure accordingly.**

*sudo vim /etc/docker/daemon.json*

{

"labels": [

"aws.region=us-east-2",

"aws.zone=a"

]

}

1. Then restart the docker daemon

***sudo service docker restart***

#### **STEP 2 - Create Fabric Environment**

1. **Clone the repository**

**Do below step on all 3 VMs**

Clone Main project repository as well

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/$] git clone* [<https://github.com/konda-kalyan/HLF-Multi-Host-Docker-Swarm>*.git*](https://github.com/konda-kalyan/hlf-multi-node-setup-docker-swarm.git)

Clone Chaincode repository as well

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/$] cd ~/HLF-Multi-Host-Docker-Swarm/chaincodes; rm -rf HLF-Chaincode-Java-CouchDB-Rich-Queries; git clone* <https://github.com/konda-kalyan/HLF-Chaincode-Java-CouchDB-Rich-Queries>

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/$] cd ~/HLF-Multi-Host-Docker-Swarm/network/*

1. **Configure Fabric versions and VM hostnames/IP-addresses**
2. **Update Versions and VM’s hostnames in environment file**

In ‘~/HLF-Multi-Host-Docker-Swarm/network/.env’ file, update following variables:

* VM’s ip addresses
* Fabric version number
* AWS region info

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *vim .env\_*

*OS\_ARCH=amd64*

*FABRIC\_JUST\_VERSION=****1.4.4***

*FABRIC\_VERSION=$OS\_ARCH-1.4.4*

*FABRIC\_CA\_VERSION=$OS\_ARCH-1.4.4*

*COUCHDB\_KAFKA\_ZOOKEEPER\_IMAGE\_VERSION=$OS\_ARCH-0.4.18*

# VERY IMPORTANT NOTE: If deployment environment is AWS then these IP addresses should be hostnames (as mentioned in below, mention ONLY private addresses and that too in the same format mentioned below.

TO DO: Need to check why public addresses and private addresses in *172.31.13.137* is not working.

*MANAGER\_NODE\_HOSTNAME=ip-172-31-13-137*

*WORKER\_NODE1\_HOSTNAME=ip-172-31-12-5*

*WORKER\_NODE2\_HOSTNAME=ip-172-31-20-131*

*AWS\_REGION=us-east-2*

FROM HERE ONWARDS, everything can be run by SCRIPTS (no manual modification/update of any files). There is a ‘**MASTER\_SCRIPT\_TO\_RUN\_ALL.sh**’ script that does **ALL** tasks from STEP2 (except .env changes that are just right above) to STEP5. In fact, if you just run this one master script, you are DONE. It does EVERYTHING for you (starting from fabric environment creation to Chaincode invocation/execution).

Again, there is one MAIN script (**STEP….sh** scripts) at/for each step (Step2 to Step5). Individual step scripts do all tasks mentioned in that whole step.

However… I would **strongly** recommend you go one task at time (not even at STEP level), run individual task level script (example: ./scp\_env\_file\_to\_worker\_nodes.sh) and understand what is going and see outputs/work doing by that task.

**ALL** (yes, you read it correctly 😉) the scripts need to be run JUST **ONLY** on **Manager node** and **NOT** required to run on any of the Worker nodes. Those commands which need to be run on Worker nodes are included in the scripts which run from Manager Node itself (by ssh to Worker nodes in **helper** scripts).

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*

***./MASTER\_SCRIPT\_TO\_RUN\_ALL.sh***

(for the first time, it asks for confirmation (yes/no) when connecting to other 2 systems, so say ‘yes’)

**If you see Chaincodes are executed properly and you see results, then ALL GOOD.**

-------------------------------------------------------------------------

\*\*\*\* Just run below one script which does rest of the tasks mentioned in **step 2**. If you want to do/refer step by step, go one task at time so that you understand what is going on

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]* **./*STEP2\_pull\_fabric\_images\_binaries\_on\_all\_machines.sh***

---------------------------------------------------------

1. **Move ‘.env’ file to Worker/Other nodes**

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*

*./scp\_env\_file\_to\_worker\_nodes.sh*

1. **Pull fabric images & binaries (on all 3 machines)**

\*\* Below script take quite amount of time

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  ./*pull\_fabric\_images\_binaries\_on\_all\_machines.sh*

1. **Clean the environment (from previous runs, if any) (just safety step)**

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  ./*bring\_down\_whole\_network\_on\_all\_machines.sh*

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  ./*clean\_artifacts\_on\_all\_machines.sh*

#### **STEP 3 - Generate Artifacts (Crypto (Certs & Keys) and Channel related) and move them to other machines as well**

\*\*\*\* Just run below one script which does all the tasks mentioned in **step 3**. If you want to do/refer step by step, go one step at time so that you understand what is going on.

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]* **./*STEP3\_generate\_artifacts\_and\_copy\_to\_worker\_nodes.sh***

---------------------------------------------------------

1. **Generate *artifacts (Crypto (Certs & Keys) and Channel)***

*[Org1 - ubuntu@ip-172-31-18-197 ~/hlf-multi-node-setup-docker-swarm/network$]*  *./generate.sh*

1. **Update/Modify/Replace VM’s hostnames and fabric image versions in all docker-compose files.**

*[Org1 - ubuntu@ip-172-31-18-197 ~/hlf-multi-node-setup-docker-swarm/network$]*  *./populate\_hostname.sh*

1. **Copy all artifact files to one common directory *(/var/mynetwork*)**

*[Org1 - ubuntu@ip-172-31-18-197 ~/hlf-multi-node-setup-docker-swarm/network$]*   *./copy\_crypto.sh*

1. **Copy artifacts to other nodes as well**

*[Org1 - ubuntu@ip-172-31-18-197 ~/hlf-multi-node-setup-docker-swarm/network$]*   *./scp\_artifacts\_to\_worker\_nodes\_and\_copy\_to\_common\_dir.sh*

#### **STEP 4 - Create Docker-Swarm Network and Bring up ALL containers**

\*\*\*\* Just run below one script which does all the tasks mentioned in **step 4**. If you want to do/refer step by step, go one step at time so that you understand what is going on.

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]* ***./STEP4\_init\_docker\_swarm\_and\_bringup\_fabric\_network.sh***

---------------------------------------------------------

1. **Setup Swarm Network**
   * + - 1. **Create a swarm network**

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$] docker swarm init*

Output from this command is…

Swarm initialized: current node (**ce67vmcbzz7x3bqbgk70lorz4**) is now a manager.

To add a worker to this swarm, run the following command:

**docker swarm join --token SWMTKN-1-2vijhxziu6oxy4n3013oxo7krlngz9mh4gdqqcdha0zxcz3u7y-aglktsqm6g1s92uk0zav04lbu 172.31.18.197:2377**

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions*.*

* + - * 1. **Join Worker nodes to Swarm network**

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]* *ssh ubuntu@<WORKER\_NODE****1****\_HOSTNAME> docker swarm join --token SWMTKN-1-2vijhxziu6oxy4n3013oxo7krlngz9mh4gdqqcdha0zxcz3u7y-aglktsqm6g1s92uk0zav04lbu 172.31.18.197:2377*

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$] ssh ubuntu@<WORKER\_NODE****2****\_HOSTNAME> ‘docker swarm join --token SWMTKN-1-2vijhxziu6oxy4n3013oxo7krlngz9mh4gdqqcdha0zxcz3u7y-aglktsqm6g1s92uk0zav04lbu 172.31.18.197:2377’*

Make sure that both nodes are joined Swarm network now. If you run below command on **Manager node** machine, it should show 3 nodes.

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *docker node ls*

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

ce67vmcbzz7x3bqbgk70lorz4 **\*** ip-172-31-18-197 Ready Active **Leader** 19.03.8

j7x9gtge075114a4m1v9d62pp ip-172-31-20-131 Ready Active 19.03.8

cemkwh2rmd6nj2yc8dio8mi09 ip-172-31-22-238 Ready Active 19.03.8

1. **Create Overlay Network**

Note that network name is ‘***hlf\_multi\_host\_network***’

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *docker network create --driver overlay --subnet=10.200.1.0/24 –attachable hlf\_multi\_host\_network*

Make sure that ‘***hlf\_multi\_host\_network***’ swarm network is created. ‘*docker network ls*’ should show below output.

NETWORK ID NAME DRIVER SCOPE

abei8ujurmtz hlf\_multi\_host\_network overlay swarm

1. **Bring up the network - Deploy the containers**

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  ***./start\_all.sh***

Just in case, you want to start one org containers at a time and see whether containers are getting up or not, then, run one org script at a time. Below is sample for Org1

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *./scripts/network/deploy\_services\_org1.sh*

Do same for other 2 orgs

Cross verify whether all the services are up and running.

***docker service ls | grep "0/1"***

The above command basically checks if there are any failed containers. If in case you find any failed containers, run the following commands to debug what went wrong.

*docker service ps --no-trunc <service id>*

*or*

*docker inspect <service id>*

Docker container logs gets stored at

*/var/lib/docker/containers/<container id>/<container id>-json.log*

**MAKE SURE THAT NETWORK IS UP AND RUNNING ON ALL 3 NODES. CHECK DOCKER NETWORK, NODES, SERVICES AND CONTAINERS ARE RUNNING AS EXPECTED.**

#### **STEP 5 - Create Channel; Join Peers; Install & Instantiate Chaincode; Invoke CC**

\*\*\*\* Just run below one script which does all the tasks mentioned in **step 5**. If you want to do/refer step by step, go one step at time so that you understand what is going on.

*[Org1 - ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]* ***./STEP5\_create\_channel\_then\_do\_chaincode\_operations.sh***

---------------------------------------------------------

* + 1. **Create Channel and get Peers join the Channel**

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *./scripts/create\_join\_channel.sh*

* + 1. **Install, instantiate and invoke Chaincode**

*[****Org1*** *- ubuntu@ip-172-31-18-197 ~/HLF-Multi-Host-Docker-Swarm/network$]*  *./scripts/install\_instantiate\_invoke\_chaincode.sh*

**If you see Chaincodes are executed properly and you see results, then ALL GOOD.**

**Congratulations….**