



Private Ethereum Blockchain on Docker

Aymen Jemmali | Mohamed Bouzidi

Summary

| | |
|---------------------------------|----|
| Introduction..... | 1 |
| Requirements..... | 1 |
| Docker..... | 1 |
| Installation..... | 1 |
| Creating Ethereum Accounts..... | 4 |
| Creating Docker Images..... | 6 |
| Monitoring with EthStats..... | 13 |
| The Attack..... | 15 |

Introduction

The Ethereum platform allows for the creation of private blockchain networks which are intended for testing purposes. In the current document, we will give a step-by-step description for setting up a private Ethereum Blockchain network running on Docker.

Requirements

This setup process was performed on a Linux machine running Ubuntu 17.04 stable and nodes using the Ethereum Go client Geth version 1.8.8-stable all running in containerized environment using Docker version 18.03.1-ce.

Docker

Docker is an operating-system-level virtualization software, that creates isolated environments to execute user code. It uses Dockerfiles, which describe a process and are used to generate an Image which is a blueprint from which we can instantiate processes. These processes run inside containers which are an isolated environment based on a given Image. Docker reduces overhead found in traditional virtual machines by letting the host operating system manage resources for each container.

Installation

Before we install the required software, we first prepare our workspace as follows:

```
~/Blockchain  
→ ls  
Backend bootnode node  
~/Blockchain  
→
```

Next step is to install Docker community edition

```
~/Blockchain  
→ sudo apt update && sudo apt install docker-ce
```

Next Geth should be installed as follows

```
~/Blockchain  
→ sudo apt install software-properties-common  
~/Blockchain  
→ sudo add-apt-repository -y ppa:ethereum/ethereum  
~/Blockchain  
→ sudo apt update && sudo apt install ethereum
```

Creating Ethereum Accounts

Before we add any nodes to our network we need to generate Ethereum accounts to use.

```
~/Blockchain  
→ geth account new  
INFO [05-24|22:20:05] Maximum peer count      ETH=25 LES=0  
total=25  
Your new account is locked with a password. Please give a  
password. Do not forget this password.  
Passphrase: _
```

We generate a few more accounts to work with

```
~/Blockchain  
→ geth account list  
INFO [05-25|05:12:33] Maximum peer count  
ETH=25 LES=0 total=25  
Account #0: {dc7d0d1dc3e824ae4a7a4abaf84cb5839530d52a}  
keystore:///home/banana/.ethereum/keystore/UTC--2018-04-01T09-  
59-49.826226021Z--dc7d0d1dc3e824ae4a7a4abaf84cb5839530d52a  
Account #1: {9e17817d69f26d61ef10320615a8b1ce2bd339b7}  
keystore:///home/banana/.ethereum/keystore/UTC--2018-04-03T01-  
31-37.463881305Z--9e17817d69f26d61ef10320615a8b1ce2bd339b7  
Account #2: {22c0db175c0f70f64996253c1d758b1cc5cfd608}  
keystore:///home/banana/.ethereum/keystore/UTC--2018-04-19T16-  
02-10.478318627Z--22c0db175c0f70f64996253c1d758b1cc5cfd608  
Account #3: {8038b22abea2ce78f9af1f296e6b7ee7d01263a3}  
keystore:///home/banana/.ethereum/keystore/UTC--2018-04-19T16-  
25-32.333013335Z--8038b22abea2ce78f9af1f296e6b7ee7d01263a3
```

Creating Client Docker Image

Now that we have setup our accounts, we need a genesis block description using a JSON file in order to create our Blockchain.

```
~/Blockchain
→ cat node/scripts/genesis.json
{
  "config": {
    "chainId": 33,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "nonce": "0x00000000000000033",
  "timestamp": "0x0",
  "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000
0000",
  "gasLimit": "0x8000000",
  "difficulty": "0x100",
  "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000
0000",
  "alloc": {}
}
```

We want our accounts to contain funds when we create the blockchain, so, we will use the following Python script to alter the “alloc” field in the genesis block description which is used to allocate initial funds to accounts.

```

import re
import json
import code
import argparse
from os import listdir, getcwd, mkdir
from shutil import copyfile
from os.path import isfile, join, exists

def init_parser():
    parser = argparse.ArgumentParser(description="Create
genesis file which allocates Ether for existing accounts")
    parser.add_argument("--genesis", type=str,
default="genesis.json", help="Genesis file to update with new
Accounts")
    parser.add_argument("keystore", type=str, help="Path to
keystore containing accounts")

    return parser

def copy_keystore(path, files):
    dirname = join(getcwd(), 'keystore')
    if not exists(dirname):
        mkdir(dirname)
    for file in files:
        copyfile(join(path, file), join(dirname, file))

def read_keystore_addresses(keystore_path):
    addresses = []
    files = [f for f in listdir(keystore_path) if
isfile(join(keystore_path, f)) and re.match('^UTC.*', f)]
    for file in files:
        with open(join(keystore_path, file)) as f:
            data = json.load(f)
            addresses.append(data['address'])
    # copy_keystore(keystore_path, files)
    return addresses

```

```
def read_genesis_file(genesis_file):
    with open(genesis_file) as file:
        genesis_dict = json.load(file)
    return genesis_dict


def fill_genesis_file(genesis, addresses):
    for address in addresses:
        genesis.get('alloc')[address] = {'balance':
'1000000000000000000000000000000000000000'}
    with open("_genesis.json", "w") as f:
        f.write(json.dumps(genesis))


def main():
    parser = init_parser()
    args = parser.parse_args()
    if hasattr(args, 'genesis') and hasattr(args, 'keystore'):
        genesis = read_genesis_file(args.genesis)
        addresses = read_keystore_addresses(args.keystore)
        fill_genesis_file(genesis, addresses)


if __name__ == '__main__':
    main()
```

Then we run the script to obtain the modified genesis block description

```
~/Blockchain/node/scripts
→ python3 generate_genesis.py ~/.ethereum/keystore/
~/Blockchain/node/scripts
→
```

The final script is `start_node.sh` which starts the Geth node once the container launches. The startup script is based on a specific bootnode which is used for auto discovery of neighbors when the Ethereum node launches.

```
~/Blockchain/node/scripts
→ cat start_console.sh
geth --datadir /geth_node/datadir --bootnodes
"enode://30f8e12c60eed7f27a6a82fca8ab3aaf0de8fcf064377927addf69
67b12eb9cea1fd71cc680697150f6456a22904e675bf379ecb5d0983000f626
749534c0ed7@172.19.0.2:30303" --rpc --rpccorsdomain "*"
--rpcvhosts "*" --ethstats="$HOSTNAME:secret@backend:3000"
console 2> /geth node/log
```

Next we write a Dockerfile specifying the steps needed to generate an image for our Geth node.

```
FROM ethereum/client-go:alpine

RUN mkdir /geth_node/
WORKDIR /geth_node/

COPY ./scripts/_genesis.json /geth_node/

ENV PATH="/:${PATH}"
RUN geth init --datadir /geth_node/datadir
/geth_node/_genesis.json 2> /geth_node/init.log
ADD ./scripts/keystore /geth_node/datadir/keystore/
ADD ./scripts/start_console.sh /geth_node/start_console.sh
RUN chmod u+x /geth_node/start_console.sh

EXPOSE 30303
EXPOSE 8545

ENTRYPOINT sh
```

Our node is based on the official Ethereum/client-go running on Linux alpine. First, we copy the modified genesis JSON file in (line 6), then we use it to generate a blockchain (line 9) and finally copy the accounts and startup script inside the container (lines 10, 11).

```
→ docker images
```

| REPOSITORY | TAG | IMAGE ID | |
|--------------------|--------|--------------|----|
| node | latest | 7ee6047ac4bd | 2 |
| days ago | 49.3MB | | |
| bootnode | latest | d1979d867dd2 | 2 |
| days ago | 49.3MB | | |
| ethereum/client-go | alpine | dcc94a9616b3 | 10 |
| days ago | 49.3MB | | |
| backend | latest | 6ed3e90db90d | 5 |
| weeks ago | 295MB | | |

We can see our newly create image by inspecting the list of images using docker images.

```
~/Blockchain/
→ docker run -it -d --name client1 --hostname client1
--network ETH node
```


Now we can instantiate the image to create a container running a full node.

```
~/Blockchain/  
→ docker attach client1  
/geth_node # ./start_console.sh  
Welcome to the Geth JavaScript console!  
  
instance: Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4  
coinbase: 0xdc7d0d1dc3e824ae4a7a4abaf84cb5839530d52a  
at block: 16 (Tue, 22 May 2018 23:00:01 UTC)  
  datadir: /geth_node/datadir  
  modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0  
personal:1.0 rpc:1.0 txpool:1.0 web3:1.0  
  
>
```

Monitoring the network with EthStats

This is a visual interface for tracking ethereum network status. It uses WebSockets to receive stats from running nodes and output them through an angular interface. It is the front-end implementation for eth-net-intelligence-api.

```
FROM alpine:latest  
  
RUN mkdir /geth_monitor/  
WORKDIR /geth_monitor/  
  
RUN apk update && apk add nodejs  
RUN apk add git  
  
RUN git clone https://github.com/cubedro/eth-netstats  
RUN cd eth-netstats && npm install  
RUN npm install -g grunt-cli  
RUN cd eth-netstats && grunt  
RUN cd /geth_monitor && echo "cd eth-netstats &&  
WS_SECRET=secret npm start" > start.sh && chmod u+x start.sh  
  
ENTRYPOINT sh
```

Next, we create a second Dockerfile for EthStats which allows us to monitor the state of the blockchain network using an Angular web application.

```

→ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
963f5e3dee9b       node               "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
zombie3
221fc831e5af       node               "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
zombie2
10f7e60eb09d       node               "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
zombie1
de26e5893274       node               "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
client2
aee757febb59       node               "/bin/sh -c sh"    2 days ago         Up 4 minutes       8545/tcp,
30303/tcp    client1
80a426f5ba11       node               "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
victim
dc4fa3aa513a       bootnode:latest    "/bin/sh -c sh"    2 days ago         Exited (137) 2 hours ago
bootnode
a11b7a1891ff       backend:latest      "/bin/sh -c sh"    3 days ago         Exited (137) 2 hours ago
backend

```

The final list of containers looks like this, with 1 victim, 2 legitime clients and 3 infected zombie machines.

```

→ docker start backend
backend
→ docker attach backend
/geth_monitor # ./start.sh

> eth-netstats@0.0.9 start /geth_monitor/eth-netstats
> node ./bin/www

2018-05-25 04:41:18.976 [API] [CON] Connected client1
2018-05-25 04:41:19.015 [API] [BLK] Block: 16 from: client1
2018-05-25 04:41:19.023 [API] [STA] Stats from: client1
2018-05-25 04:41:19.031 [API] [HIS] Got history from: client1
2018-05-25 04:41:34.009 [API] [BLK] Block: 16 from: client1

```

The screenshot displays the 'Ethereum Network Status' page in Google Chrome. The page is divided into several sections:

- Top Metrics:**
 - BEST BLOCK:** #16
 - UNCLES:** 0/0 (CURRENT / LAST 50)
 - LAST BLOCK:** 4 min ago
 - AVG BLOCK TIME:** 33.00s
 - AVG NETWORK HASHRATE:** 4 KH/s
 - DIFFICULTY:** 131.71 KH
- Active Nodes:** 4/4
- Gas Price:** 18 gwei
- Gas Limit:** 132135519 gas
- Page Latency:** 6 ms
- Uptime:** 100%

Below these metrics are several charts and graphs:

- Block Time:** A line graph showing block times over time.
- Difficulty:** A line graph showing difficulty over time.
- Block Propagation:** A line graph showing block propagation over time.
- Gas Spending:** A line graph showing gas spending over time.
- Gas Limit:** A line graph showing gas limit over time.
- World Map:** A world map showing the distribution of nodes.

At the bottom, there is a table of active nodes:

| Node Name | Client | Version | OS | Arch | Block | Hashrate | Latency | Uptime | Difficulty | | | | | | | |
|-----------|--------------------|---------|-------|-------|---------|----------|---------|--------|------------|---------------------|-----------|---|---|-----------|--------|------|
| client1 | Geth/v1.8.8-stable | 2088a04 | linux | amd64 | gp1.8.4 | 0 ms | 1 | 0 | #16 | 742ae79b...bd52e42b | 2,101,184 | 0 | 0 | 4 min ago | 0 ms | 100% |
| client2 | Geth/v1.8.8-stable | 2088a04 | linux | amd64 | gp1.8.4 | 0 ms | 1 | 0 | #16 | 742ae79b...bd52e42b | 2,101,184 | 0 | 0 | 1 min ago | 11.5 s | 100% |
| bootnode | Geth/v1.8.8-stable | 2088a04 | linux | amd64 | gp1.8.4 | 0 ms | 3 | 0 | #16 | 742ae79b...bd52e42b | 2,101,184 | 0 | 0 | 32 s ago | 14 min | 100% |
| victor | Geth/v1.8.8-stable | 2088a04 | linux | amd64 | gp1.8.4 | 1 ms | 1 | 0 | #16 | 742ae79b...bd52e42b | 2,101,184 | 0 | 0 | 12 s ago | 14 min | 100% |





Our attack is a simple variant of the Eclipse attack, which takes advantage of the random selection of peers performed by a node after a reboot. Our approach consists of maximizing the number of malicious nodes in the network.

We check the peers of the victim before a reboot takes place.

```
> var peers = admin.peers
> peers.map(function(p) { console.log(p.id,
p.network.remoteAddress); });
30f8e12c60eed7f27a6a82fca8ab3aaf0de8fcf064377927addf6967b12eb9c
ea1fd71cc680697150f6456a22904e675bf379ecb5d0983000f626749534c0e
d7 172.19.0.4:46708
502fd47926208ffc8732984bcd2f2c9e9d2a93c834c60950a7febd1ed172ec7
0f95a105457f2a7014427ec4f0e1d727bbbc317d8f7840438fdf809d32d250c
e5 172.19.0.6:30303
7389b368a40d17e25cbafd8445716f95d2d732550d7de77a113b1cb31f9b1ed
0655f4198a93e4700a25dbc8bb0cae688fc463df383de84f972196ad9a503f2
23 172.19.0.2:57216
```

After introducing malicious nodes into the network and attempting to force the victim to reboot.

```
> var peers = admin.peers
> peers.map(function(p) { console.log(p.id,
p.network.remoteAddress); });
de5b425478e131480dbe896a5ce0998b41961ede96ed0eafbe6dd838e866af4
7cd43c052530459ec5aa0af7588208c3b42f5a66145eb7a72e4ddf919d66707
fe 172.19.0.7:30303
e1bf90e0e2c68672516450c811f7a2567487f4708ba978034940f94f342bb4d
c36bcf0d05cc5b2c2123193c54d02adcc2bb90970015417b3a64e351c319d1a
82 172.19.0.5:36258
```

|   |  |  |  |  |  |  |  | |
|---|---|---|---|---|---|--|---|-----------|
|  client1 | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 1 ms |  | 6 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  client2 | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 0 ms |  | 6 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  victim | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 0 ms |  | 2 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  bootnode | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 0 ms |  | 5 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  zombie1 | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 0 ms |  | 6 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  zombie2 | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 1 ms |  | 6 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |
|  zombie3 | Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.9.4 | 0 ms |  | 5 | 0 | #16 | 742ed79b...bd52e42b | 2,101,184 |