But the best option for beginning is just to run it in development mode by setting --dev parameter on Docker container running command.

Here’s the command that starts Docker container in development mode and exposes Ethereum RPC API on port 8545.

The one really good message when running that container in development mode is that you have plenty of Ethers on your default, test account.

$ docker run -d --name ethereum -p 8545:8545 -p 30303:30303 ethereum/client-go:v1.8.6 --rpc --rpcaddr "0.0.0.0" --rpcapi="db,eth,net,web3,personal" --rpccorsdomain "\*" --dev --allow-insecure-unlock

To run the docker in a network

docker run -d --name ethereum --network catenabend -p 8545:8545 ethereum/client-go:v1.8.6 --rpc --rpcaddr "0.0.0.0" --rpcapi="db,eth,net,web3,personal" --rpccorsdomain "\*" --dev

let’s create some other test accounts and also check out some things. To achieve it we need to run Geth’s interactive JavaScript console inside Docker container.

$ docker exec -it ethereum geth attach ipc:/tmp/geth.ipc

In the console

1. To get the default account type

> eth.coinbase

2. To get the list of available accounts

> eth.accounts

3. TO get their balance for first account

>eth.getBalance(eth.accounts[0])

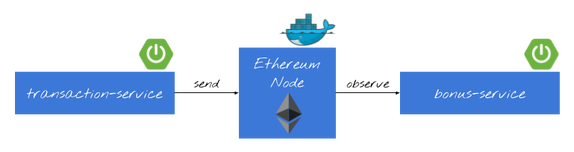
4. To create some test accounts

>personal.newAccount("123456")

5. To transfer some fund from base account to newly created account

> eth.sendTransaction({from:eth.coinbase,to:eth.accounts[1],value:1000})

**SYSTEM ARCHITECTURE**

The architecture of our sample system is very simple. I don’t want to complicate anything, but just show you how to send transaction to Geth node and receive notifications. While transaction-service sends new transaction to Ethereum node, bonus-service observe node and listening for incoming transactions. Then it send bonus to the sender’s account once per 10 transactions received from his account. Here’s the diagram that illustrates an architecture of our sample system.  


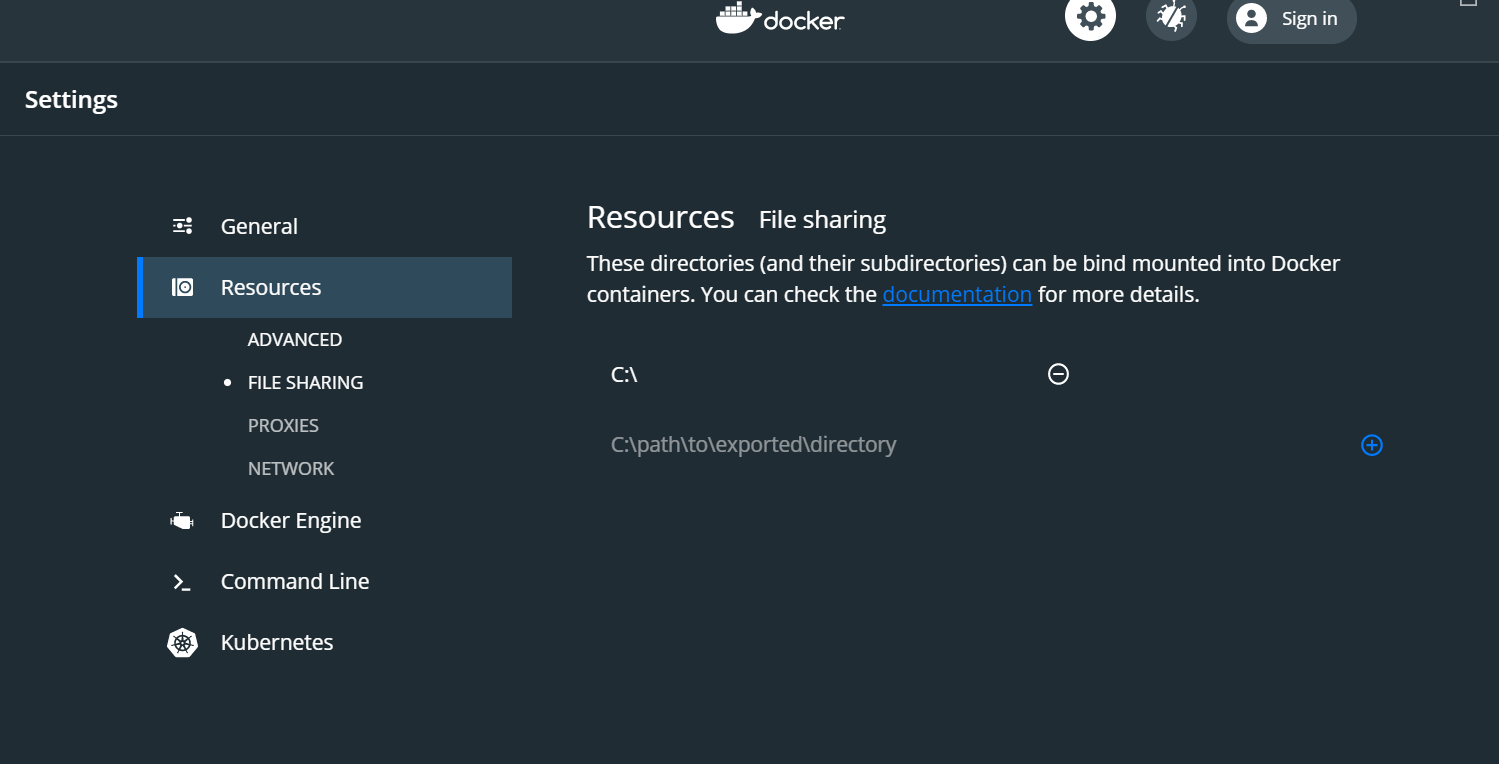
Before sending any transactions you should also unlock sender account.

>personal.unlockAccount(eth.accounts[1])

Once we have created solidity contract (.sol file) we have to compile it and generate source code that can be used inside our application to deploy contract and call its functions. For just a quick check, you can use the Solidity compiler available online on site [https://remix.ethereum.org](https://remix.ethereum.org/)

To generate the abi and bin files use the below docker command

Before executing this command first the drive in windows machine i.e) C:/ to be mounted in docker as given below



docker run -v C:/Innovation/blockchain/Chai4SoWell:/source ethereum/solc:stable -o /source/output --abi --bin /source/CentralContract.sol

in the above command the local directory path “C:/Innovation/blockchain/Chai4SoWell” is mounted as “/source” in the docker. Subsequently all the folders inside the path is referenced by the mounted reference “/source”

1. Generate the abi and bin file from the above link and copy the files into specific location

web3j solidity generate -a=CentralContract.abi -b=CentralContract.bin -o=/src -p=pl.piomin.services.transaction.model

web3j solidity generate -a=StateContract.abi -b=StateContract.bin -o=/src -p=pl.piomin.services.transaction.model

web3j solidity generate -a= DisbursementContract.abi -b= DisbursementContract.bin -o=/src -p=pl.piomin.services.transaction.model

web3j solidity generate -a= DonationContract.abi -b= DonationContract.bin -o=/src -p=pl.piomin.services.transaction.model

Web3J can be downloaded from

https://github.com/web3j/web3j/releases/tag/v4.5.5

In the above example abi and bin files are kept in the same path where the above command was run and it generates the src folder under c:/

Generating abi and bin files using Solc

**solc --optimize --overwrite  -o output --abi --bin CentralContract.sol**

**solc --optimize --overwrite  -o output --abi --bin StateContract.sol**

**solc --optimize --overwrite  -o output --abi --bin DonationContract.sol**

**Learning**

When you make a view only call following error was always thrown

org.web3j.tx.exceptions.ContractCallException: Empty value (0x) returned from contract. The reason for the error is due to solc version. After updating to 0.5.4+ the issue got resolved

Solc can be downloaded from <https://github.com/ethereum/solidity/releases?after=v0.5.10>

**Ethereum Browser**

<https://github.com/etherparty/explorer>

Install Git

Install geth

Install Node JS and npm

Type the below command

>git clone <https://github.com/etherparty/explorer>

Go inside the installed folder i.e) c:\Innovation\tools\ethExplorer\explorer

Type

>npm start

To access the Ethereum browser <http://localhost:8000>

<https://www.codota.com/code/java/classes/org.web3j.protocol.admin.Admin>

<https://github.com/ethjava/web3j-sample/blob/master/src/main/java/com/ethjava/TokenClient.java>

<https://piotrminkowski.wordpress.com/2018/06/22/introduction-to-blockchain-with-java-using-ethereum-web3j-and-spring-boot/>

<https://dzone.com/articles/intro-to-blockchain-with-ethereum-web3j-and-spring>

<https://www.mobilefish.com/developer/blockchain/blockchain_quickguide_using_solc.html>

Base Address: 0x4c9c0dddfa17819a1317b559725b582ad470cbc3

Contract Address: 0xc4af8d58661da199a6116d07cd5f2c5b6acec578

<https://www.baeldung.com/learn-spring-security-course#table>

<https://github.com/web3j/sample-project-gradle>

<https://sgitario.github.io/blockchain-using-springboot/>

<https://github.com/ethereum/solidity/releases>

**Points to explore**

1. Main network uses “proof of work” as consensus algorithm. Ethereum suggests to use “clique – Proof of authority consensus algorithm” for private networks. It is less resource intensive than proof of work.

Clique consensus is a proof-of-authority system where new blocks can be created by authorized ‘signers’ only. The clique consenus protocol is specified in [EIP-225](https://geth.ethereum.org/docs/interface/clique-eip). The initial set of authorized signers is configured in the genesis block. Signers can be authorized and de-authorized using a voting mechanism, thus allowing the set of signers to change while the blockchain operates. Clique can be configured to target any block time (within reasonable limits) since it isn’t tied to the difficulty adjustment.

Steps to create Private network

1. Creating the Genesis block using genesis.json
2. Creating initial signer
3. Define difficulty
4. Create a blockchain using the genesis block
5. Setting up peer-to-peer network. Setting up bootstrap node which act as the entry point through which other nodes join
6. If private network to connect across internet bootnode and other nodes have public IP address assigned and both TCP and UDP traffic are allowed in firewall
7. Member nodes need to be initialized with the same genesis file used by bootstrap node
8. Unlocking the signer account for mining

When we create an account, all the details of the account is stored in a **UTC file** in the directory mentioned during account creation (path: **./datadir/keystore**). To import the accounts, we need to copy these files and paste in the **keystore** directory under the Data Directory (path: **./myDataDir/keystore**)

[**https://piotrminkowski.com/2018/06/22/introduction-to-blockchain-with-java-using-ethereum-web3j-and-spring-boot/**](https://piotrminkowski.com/2018/06/22/introduction-to-blockchain-with-java-using-ethereum-web3j-and-spring-boot/)

[**https://medium.com/edureka/docker-networking-1a7d65e89013**](https://medium.com/edureka/docker-networking-1a7d65e89013)

**To check ping between containers**

[**https://www.digitalocean.com/community/questions/how-to-ping-docker-container-from-another-container-by-name**](https://www.digitalocean.com/community/questions/how-to-ping-docker-container-from-another-container-by-name)

To see list of container

Docker container ls

To login into a container

Docker exec -ti <container id> bash

To install curl

Apt install curl

**Docker Swarm**

Step 1:

docker swarm init --advertise-addr 192.168.56.101

Swarm initialized: current node (d2wppfz8yj1pqnblqke8jp1ks) is now a manager.

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

docker swarm join --token SWMTKN-1-4cx2648tywnh4sisx3z7u0nahkr8qsgpi9ud6tb4b0fjszo3kh-c693z585r5znm98y3ei02o208 192.168.56.101:2377

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

Step 2:

docker network create -d overlay myoverlaynetwork

ywcfg3fb0a5a17rxmf1l0xs8f

Step 3:

Mongodb1 as service

docker service create --name mongodb1 -d --network myoverlaynetwork -p 27017:27017 mongo:latest

o2jjtttw4iug04elvz8kae65r

Step 4:

Ethereum as service

docker service create --name ethereum -d --network myoverlaynetwork -p 8545:8545 ethereum/client-go:v1.8.6 --rpc --rpcaddr "0.0.0.0" --rpcapi="db,eth,net,web3,personal" --rpccorsdomain "\*" --dev

7lu9li37wekg7s9jvrbks8m5t

Step 5: Check the services are created

docker service ls

Step 6: Installing the spring boot as service in docker storm

docker service create -p 8091:8091 --name catenaService --network myoverlaynetwork catena

docker run --rm -p 8091:8091 --name catena --network myoverlaynetwork catena

y4lk3pjx08fpd6dn4rugax04r