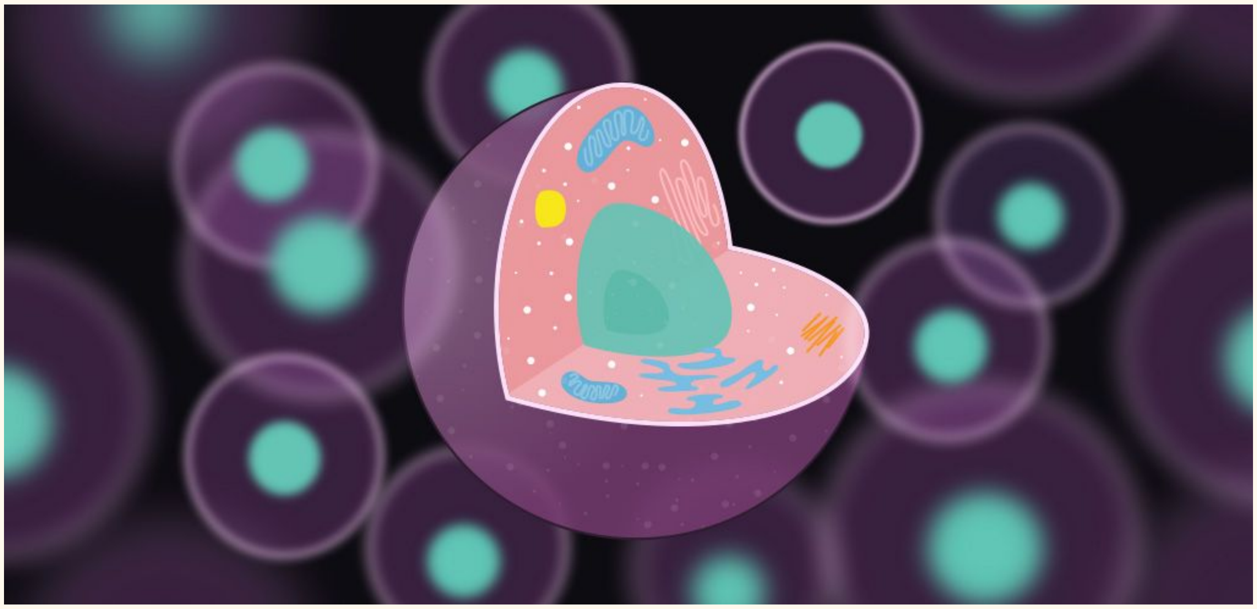
SECOND EDITION
ETHEREUM

PRIVATE BLOCKCHAIN & SMART CONTRACTS

By Andy Danov



INTRODUCTION

Proof-of-Authority is a newer concept in the blockchain world where you have a number of **pre-approved authority nodes** (called sealers, think of these as mining nodes). Any new node that you want to add has to be voted on by the currently approved set of authority nodes, this **gives you full control over which nodes can seal blocks** (mine) on your network. To make sure a malicious signer cannot do too much harm to the network any signer can sign at most one of a number of consecutive blocks
`(SIGNER_COUNT / 2) + 1`.
The same consensus is applied when an authority node is removed from the network.

The Ethereum Proof-of-Authority protocol is called Clique and is well described in the Clique Github issue. Ethereum currently uses this algorithm for the Rinkeby test network.

Proof-of-Authority is a near perfect fit for private networks but not at all suited for public networks where the trust should be as distributed as possible.

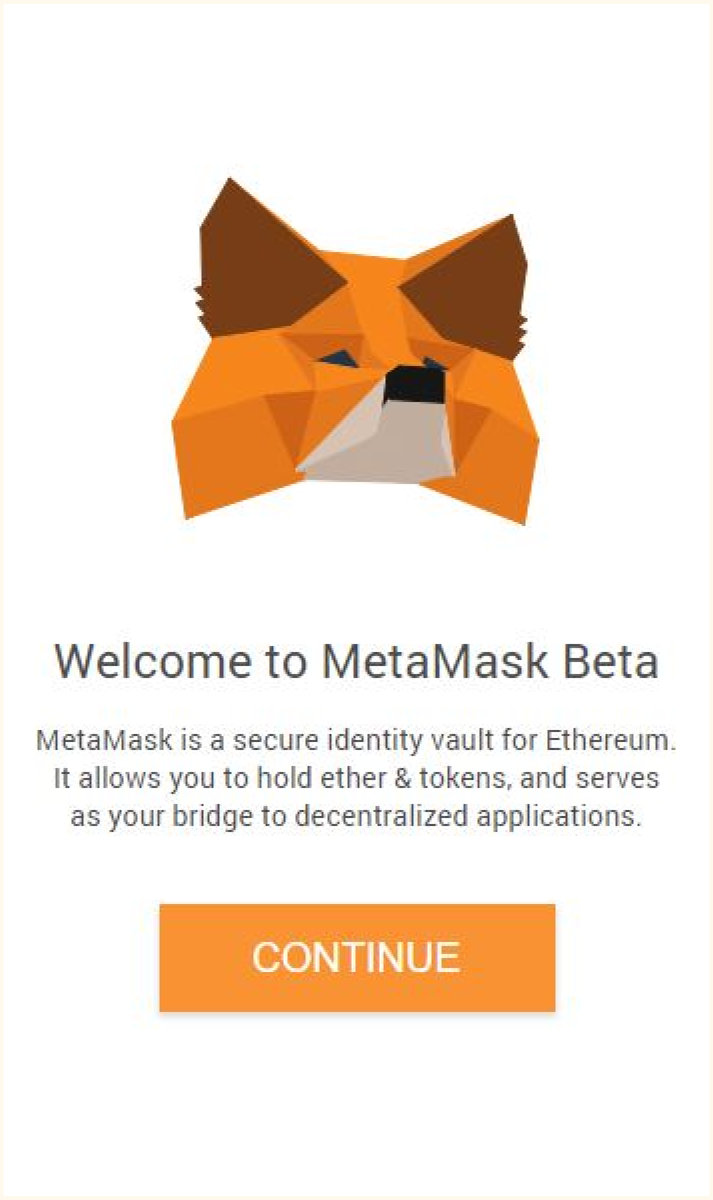
Your First Contract

You don't need to install anything except a Chrome / Firefox / Opera / Brave extension called Metamask. So head to **https://metamask.io/** and check the installing instructions

MetaMask **is a bridge** that allows you to visit the distributed web of tomorrow in your browser today. It allows you to run Ethereum dApps right in your browser without running a full Ethereum node.

MetaMask includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.

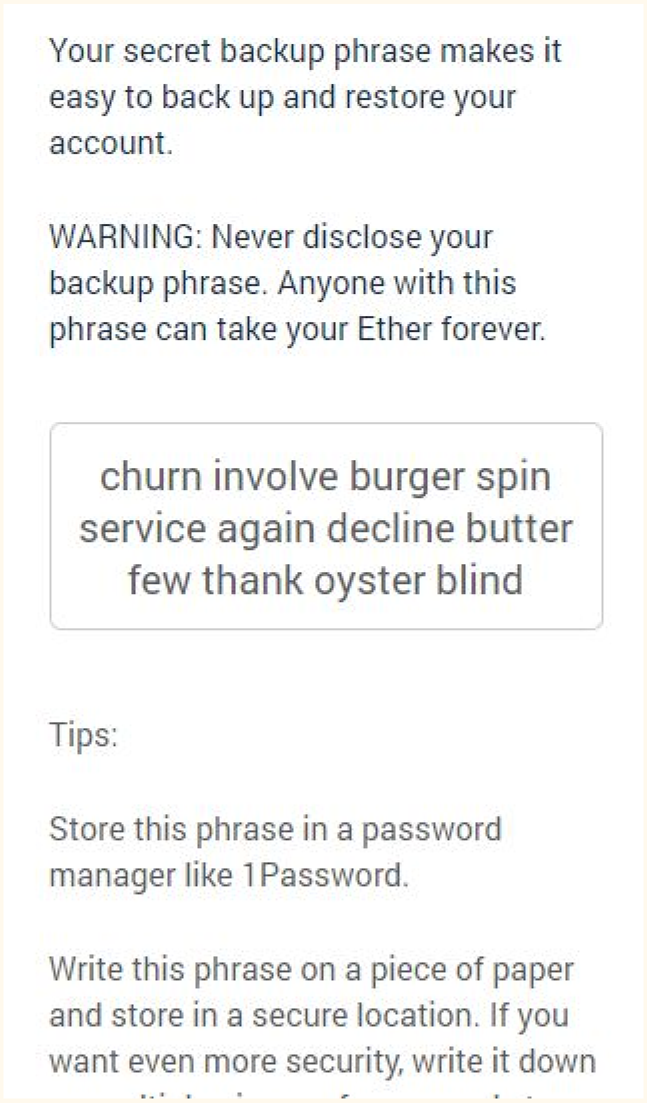
Metamask connects through Infura (https://infura.io/) for the mainnet and testnet setting. Infura maintains those nodes.It also allows you to connect to a custom RPC by clicking the network name for the dropdown.



Welcome to MetaMask Beta

MetaMask is a secure identity vault for Ethereum. It allows you to hold ether & tokens, and serves as your bridge to decentralized applications.

After you install it and go though the license acceptance stuff, you'll be presented with a seed phrase.



Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

churn involve burger spin service again decline butter few thank oyster blind

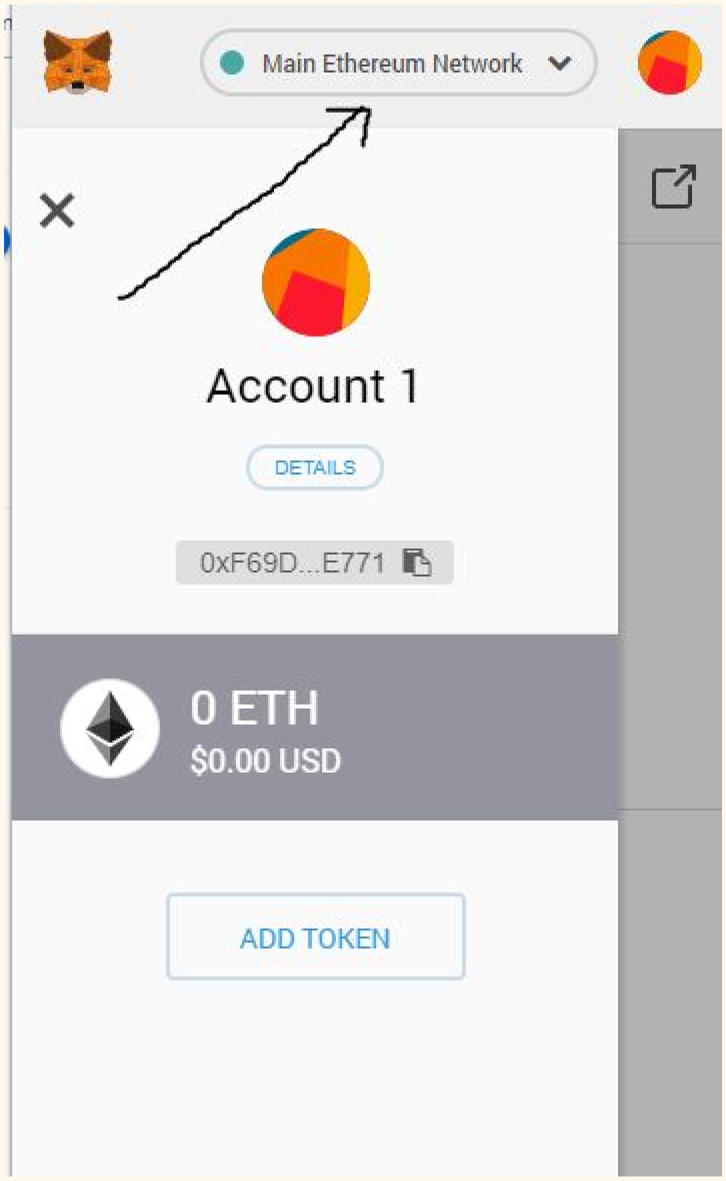
Tips:

Store this phrase in a password manager like 1Password.

Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down

Write it down in a txt file.

From the above menu let’s switch to Rinkeby



And copy our brand new address.

In order to get some test ETH in our account we'll use Rinkeby Faucet

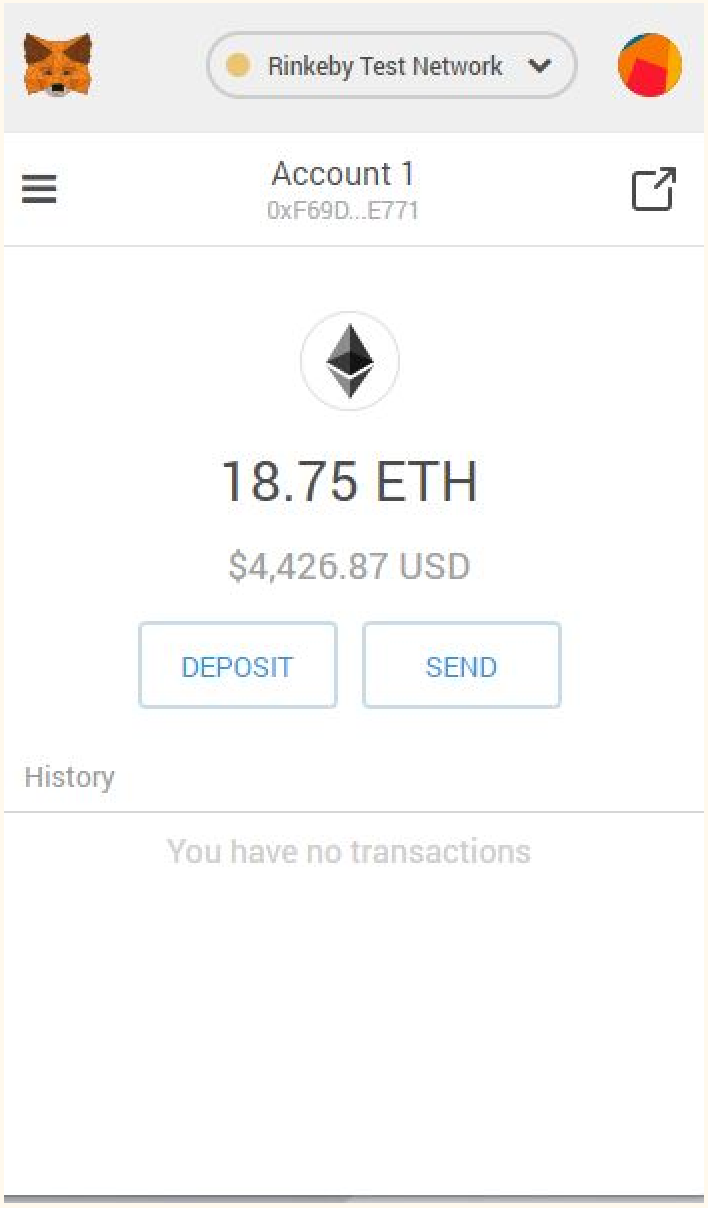
Why Rinkeby ?

Rinkeby is a Proof-of-Authority network, so uses a different consensus mechanism to the main net. The Ropsten testnet is Proof-of-Work, so more similar to the public main net.

Head over to:

https://faucet.rinkeby.io/

You need to publish a post in twitter or google plus and write in it your address.



Yey! 18.75 ETH in our account with just few clicks.

1. Your first contract

We needed some ETH in order to deploy our contract on Rinkeby Network.

Here’s the first code that we will publish together

```
pragma solidity ^0.4.25;

contract Store {

    string public version;

    function setVersion(string ver) public {

        version = ver;

    }

    function getVersion() public constant  returns (string) {

        return version;

    }

}
```

We have to compile it, so open up

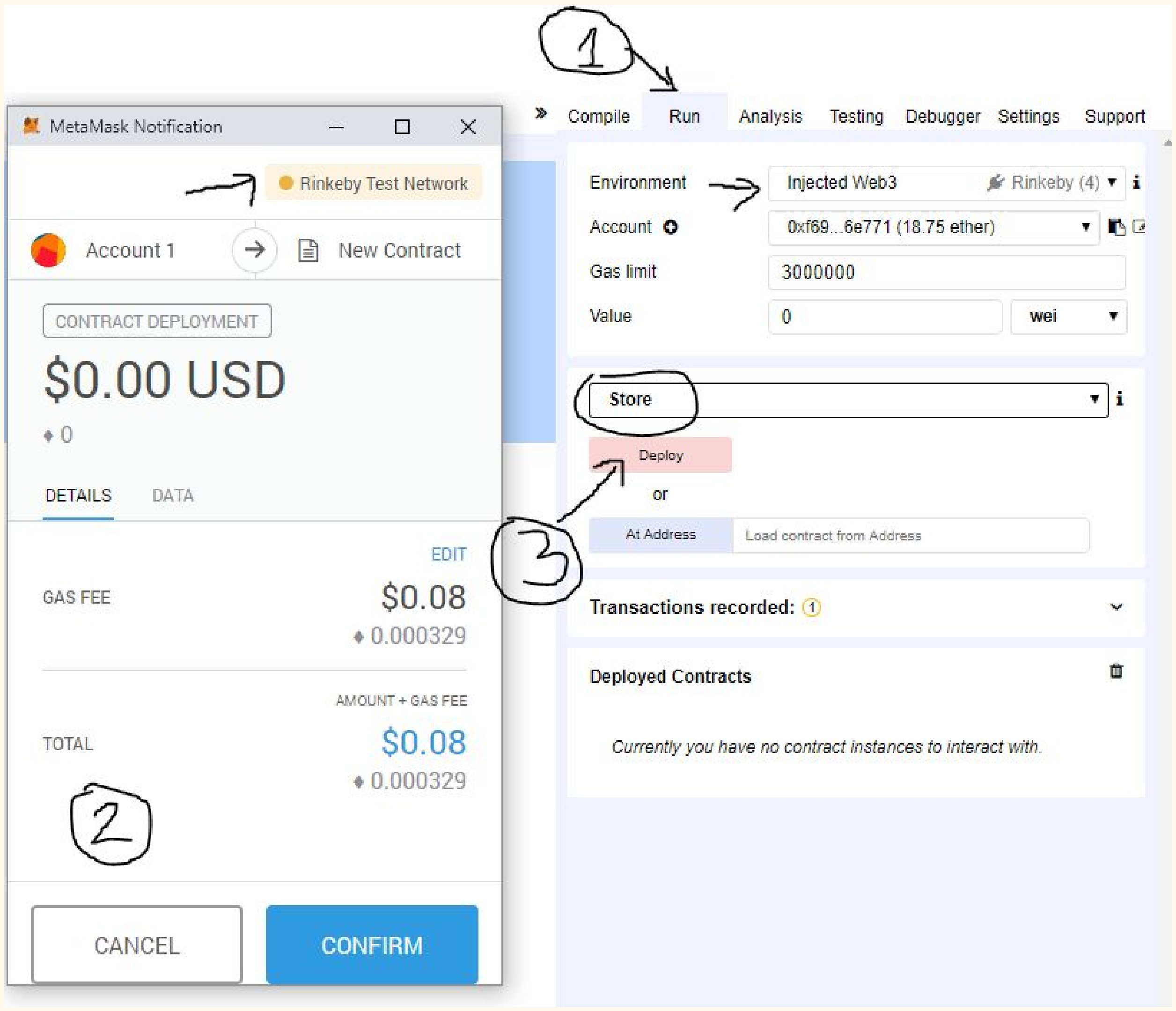
https://remix.ethereum.org

Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in Javascript, Remix supports both usage in the browser or locally.

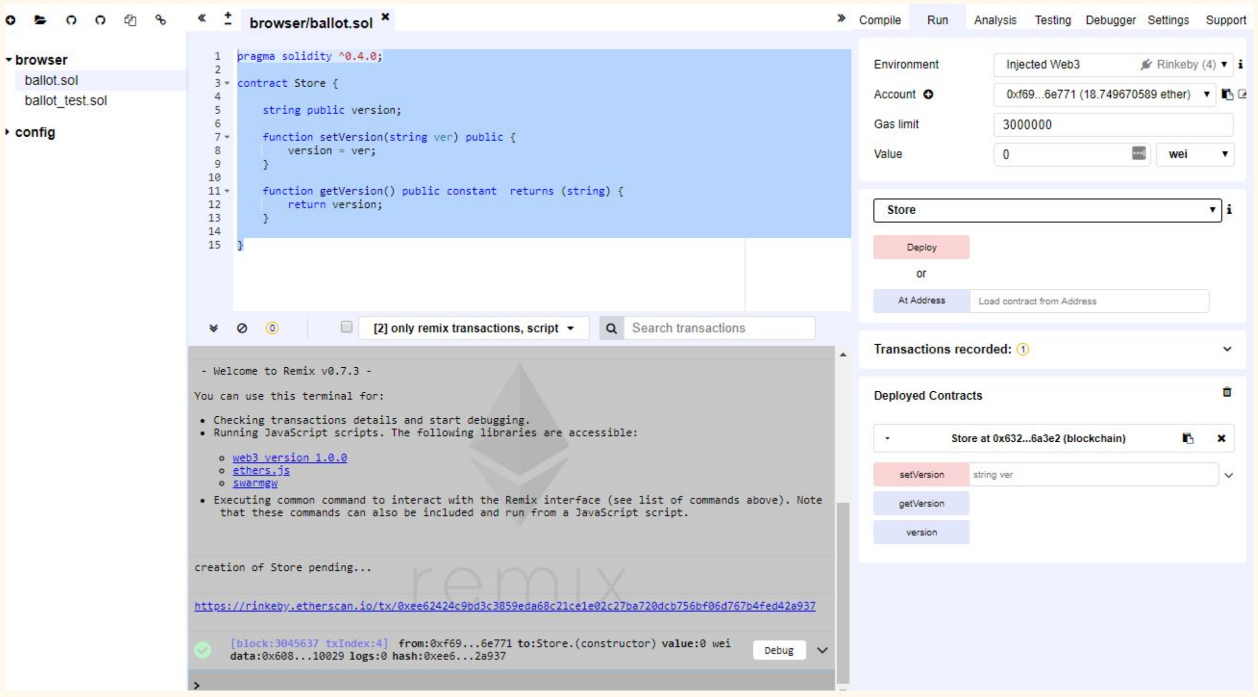
Remix also supports testing, debugging and deploying of smart contracts and much more.

The Remix editor recompiles the code each time the current file is changed or another file is selected. It also provides syntax highlighting mapped to solidity keywords.

Paste there the above code. Then go to the Run tab and click deploy



Environment: Injected Web3 — Rinkeby (4)
Account: 0xf69...6e771 (18.75 ether)
Gas limit: 3000000
Value: 0 wei
Store

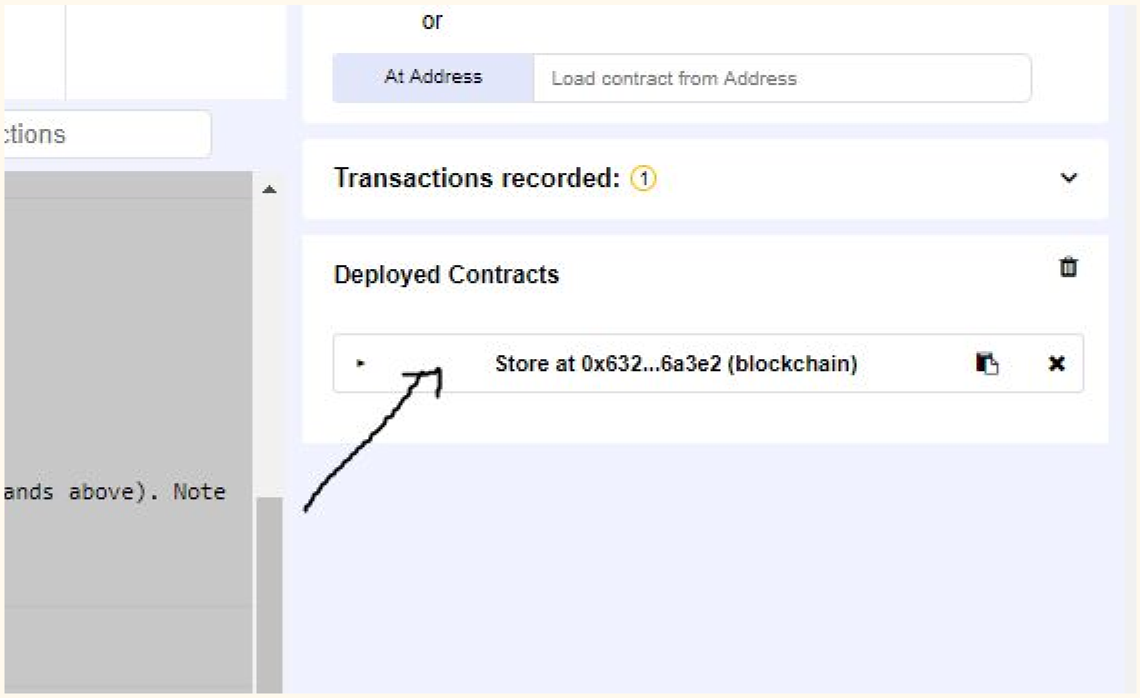


```
pragma solidity ^0.4.0;

contract Store {

    string public version;

    function setVersion(string ver) public {
        version = ver;
    }

    function getVersion() public constant  returns (string) {
        return version;
    }

}
```

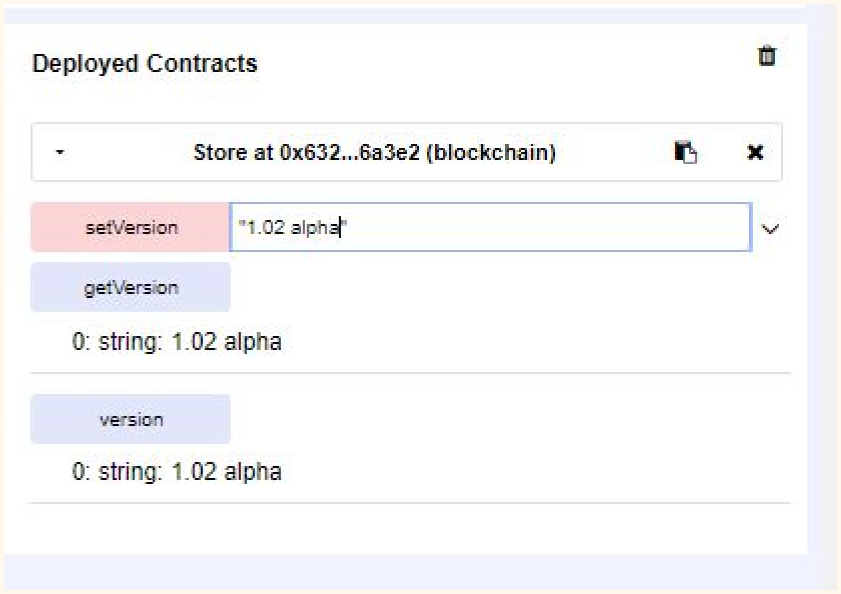
```
 - Welcome to Remix v0.7.3 -

You can use this terminal for:

 • Checking transactions details and start debugging.
 • Running JavaScript scripts. The following libraries are accessible:

    o web3 version 1.0.0
    o ethers.js
    o swarmgw

 • Executing common command to interact with the Remix interface (see list of commands above). Note
   that these commands can also be included and run from a JavaScript script.

creation of Store pending...

https://rinkeby.etherscan.io/tx/0xee62424c9bd3c3859eda68c21ce1e02c27ba720dcb756bf06d767b4fed42a937

[block:3045637 txIndex:4] from:0xf69...6e771 to:Store.(constructor) value:0 wei
data:0x608...10029 logs:0 hash:0xee6...2a937
```

Environment: Injected Web3 — Rinkeby (4)
Account: 0xf69...6e771 (18.749670589 ether)
Gas limit: 3000000
Value: 0 wei

Congratulations, you deployed your first contract on the Etherem Test Network Rinkeby.
If you click on the Store at 0x…. Under the deployed contracts



You can now interact with your contract



```
0: string: 1.02 alpha
```

```
0: string: 1.02 alpha
```

Try to put your own version like “abc” (note the “ quotes, since the value there is a string (text).

A popup from metamask will ask you to sign the transaction. After about 15 seconds, when you click on the getVersion, you'll see your version got written in the blockchain. You can click as many times as you want on getVersion or version, without any annoying popups from metamask.

But when you try to write a new version, you'll be presented with popup asking you to pay

AMOUNT + GAS FEE
$0.01
♦ 0.000033

Meaning -> writing on the blockchain costs money!

Reading from the blockchain is free.

Lessons Learned

1. You can test your contracts on the Ethereum Test Net before you publish them into the main-net.
2. Metamask is very easy to use
3. Writing to the blockchain costs, reading is free

Chapter 2: Solidity Quick Intro

Solidity is a programming language that is **typed statically**, meaning that every variable type must be specified at compile time.
Types can **interact with** other **expressions** containing operators.
Value types refer to the **types**, that are **always passed by a value**, meaning, for example, that they are always copied when used in assignments or for arguments.

Before you start reading on the official documentation of Solidity, let's talk about coding conventions.

Solidity borrowed it's style from Python https://www.python.org/dev/peps/pep-0008/

Code Layout

Indentation

Use 4 spaces per indentation level.

Tabs or Spaces

Spaces are the preferred indentation method.

Mixing tabs and spaces should be avoided.

Blank Lines

Surround top level declarations in solidity source with two blank lines.

Order of Functions

Ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier.

Functions should be grouped according to their visibility and ordered:

- constructor
- fallback function (if exists)
- external
- public
- internal
- private

Within a grouping, place the view and pure functions last.

More about it here. https://solidity.readthedocs.io/en/v0.4.24/style-guide.html

Now it’s time to spend reading ->

https://solidity.readthedocs.io/en/v0.4.24/solidity-in-depth.html

You don’t need to learn it by heart, just read it once and move to the next chapter, where we will build our first token.

Chapter 3: Your First Token

Let’s code now your very own token. Call it “yourname coin” or however you’d like to call it.
ERC-20 tokens are tokens designed and used solely on the Ethereum platform.

What is ERC-20 ?
They follow a **list of standards** so that they can be shared, exchanged for other tokens, or transferred to a crypto-wallet.

The Ethereum community created these standards with three optional rules, and six mandatory.

Optional Rules

- Token Name
- Symbol
- Decimal (up to 18)

Mandatory Rules

2. totalSupply
3. balanceOf
4. transfer
5. transferFrom
6. approve
7. allowance

You can code however you want, but if you go towards an ERC20 compliant token you must respect the above “rules”

Why do you need ERC-20 ?

- Makes it much easier for other developer to interact with your token
- Makes it easier for exchanges to integrate it
- Makes it easy for iOS / Android wallets to integrate it

Let’s go over each of the some of the rules:

totalSupply identifies the total number of ERC-20 tokens created. Usually people make it 1 billion or more

transfer allows a certain number of tokens to be transferred from the total supply to a user account.

balanceOf function returns the number of tokens a given address has in its account.

transferFrom is the function that allows a user to transfer tokens to another user.

approve checks a transaction against the total supply of tokens.

As of July 26 2018, there were more than 103,621 ERC-20 token contracts. Among the most successful ERC20 token sales are EOS, Filecoin, Bancor, Qash, and Bankex, raising over $80 million each.

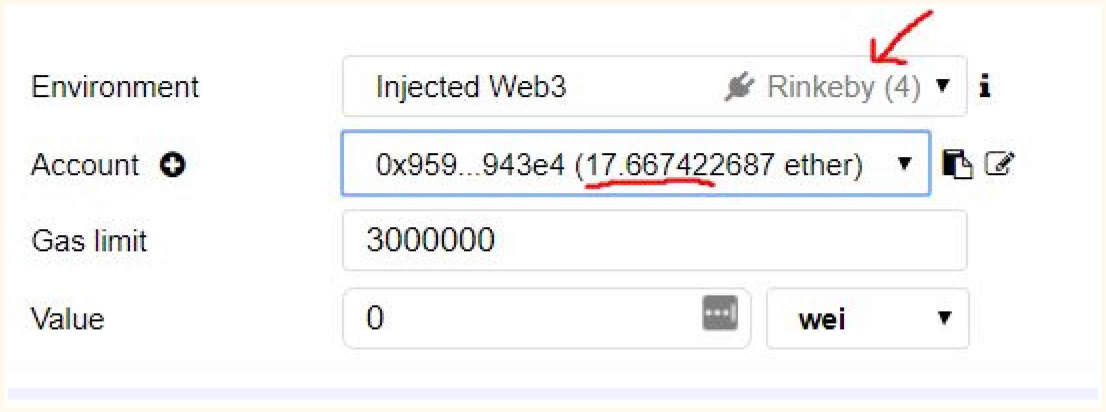
Starting your first token, step by step

1. **Minimum Know-How:**

Read this: https://theethereum.wiki/w/index.php/ERC20_Token_Standard

2. Make sure you have some ETH in your rinkeby test account and you’re logged in into metamask.

In remix ide https://remix.ethereum.org select from Environment -> Injected Web3. You should see with grey written “rinkeby”



Environment: Injected Web3 — Rinkeby (4)

Account: 0x959...943e4 (17.667422687 ether)

Gas limit: 3000000

Value: 0 wei

In the browser, remove the ballot_test.sol if it's present and rename ballot.sol to Token.sol

Copy paste the code you find here: https://theethereum.wiki/w/index.php/ERC20_Token_Standard and modify it's name and symbol to your wish.

```
pragma solidity 0.4.25;


// ----------------------------------------------------------------------------

// 'FIXED' 'Example Fixed Supply Token' token contract

//

// Symbol      : FIXED

// Name        : Example Fixed Supply Token

// Total supply: 1,000,000.000000000000000000

// Decimals    : 18

//

// Enjoy.
```

```
//

// (c) BokkyPooBah / Bok Consulting Pty Ltd 2018. The MIT Licence.

// ----------------------------------------------------------------------------



// ----------------------------------------------------------------------------

// Safe maths

// ----------------------------------------------------------------------------

library SafeMath {

    function add(uint a, uint b) internal pure returns (uint c) {

        c = a + b;

        require(c >= a);

    }

    function sub(uint a, uint b) internal pure returns (uint c) {

        require(b <= a);

        c = a - b;

    }

    function mul(uint a, uint b) internal pure returns (uint c) {

        c = a * b;

        require(a == 0 || c / a == b);

    }

    function div(uint a, uint b) internal pure returns (uint c) {

        require(b > 0);
        c = a / b;
    }
}



// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint
remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

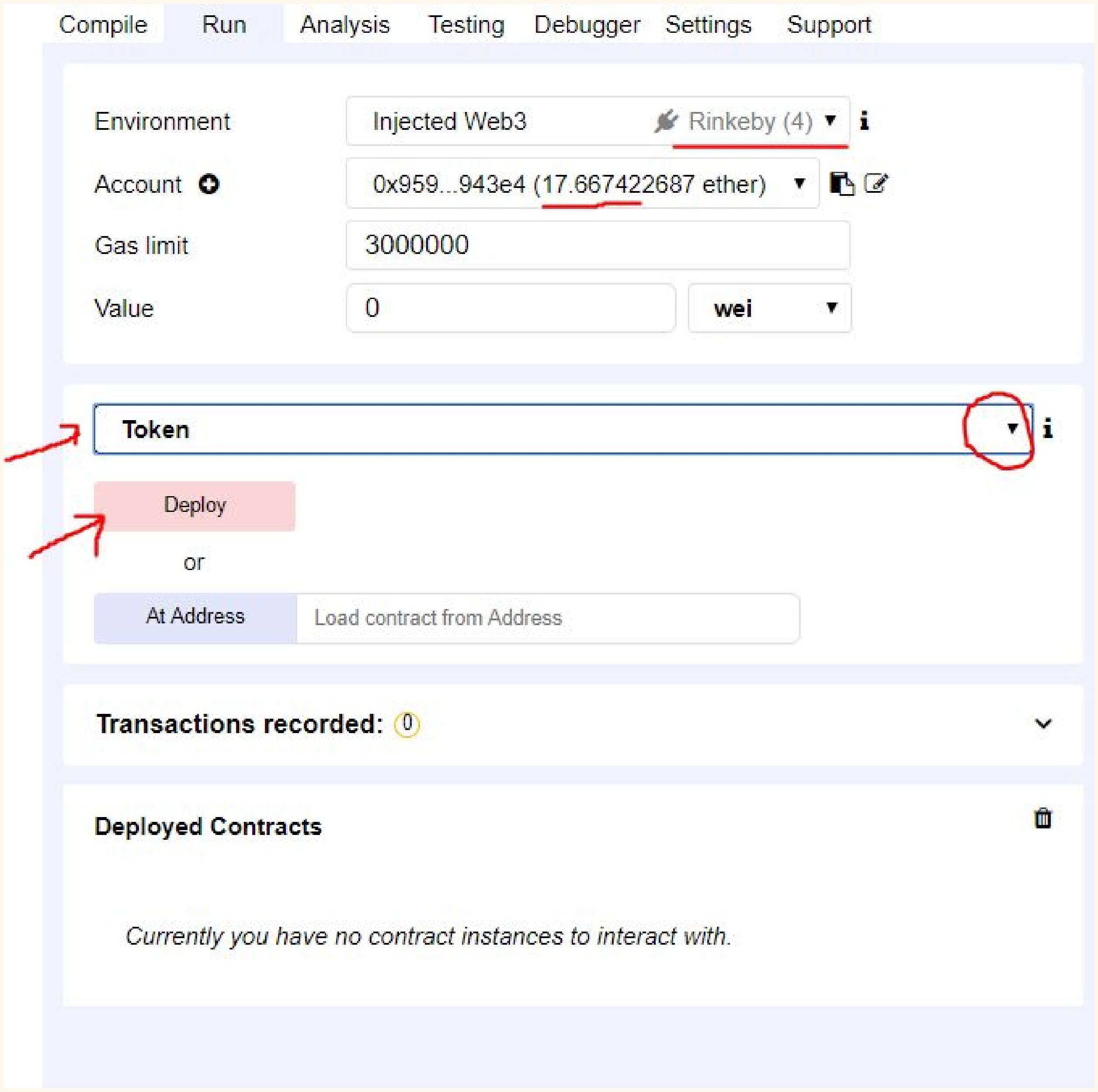
```
// ----------------------------------------------------------------------------

// Contract function to receive approval and execute function in one call

//

// Borrowed from MiniMeToken

// ----------------------------------------------------------------------------

contract ApproveAndCallFallBack {

    function receiveApproval(address from, uint256 tokens, address token, bytes data) public;

}



// ----------------------------------------------------------------------------

// Owned contract

// ----------------------------------------------------------------------------

contract Owned {

    address public owner;

    address public newOwner;


    event OwnershipTransferred(address indexed _from, address indexed _to);


    constructor() public {

        owner = msg.sender;

    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}


// ----------------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and a
// fixed supply
// ----------------------------------------------------------------------------
contract FixedSupplyToken is ERC20Interface, Owned {
    using SafeMath for uint;
```

```
string public symbol;

string public  name;

uint8 public decimals;

uint _totalSupply;


mapping(address => uint) balances;

mapping(address => mapping(address => uint)) allowed;




// ------------------------------------------------------------------------

// Constructor

// ------------------------------------------------------------------------

constructor() public {

    symbol = "ANDY";

    name = "Andy Token";

    decimals = 18;

    _totalSupply = 1000000 * 10**uint(decimals);

    balances[owner] = _totalSupply;

    emit Transfer(address(0), owner, _totalSupply);

}
```

```
// ----------------------------------------------------------------------------

// Total supply

// ----------------------------------------------------------------------------

function totalSupply() public view returns (uint) {

    return _totalSupply.sub(balances[address(0)]);

}



// ----------------------------------------------------------------------------

// Get the token balance for account `tokenOwner`

// ----------------------------------------------------------------------------

function balanceOf(address tokenOwner) public view returns (uint balance) {

    return balances[tokenOwner];

}



// ----------------------------------------------------------------------------

// Transfer the balance from token owner's account to `to` account

// - Owner's account must have sufficient balance to transfer

// - 0 value transfers are allowed

// ----------------------------------------------------------------------------

function transfer(address to, uint tokens) public returns (bool success) {

    balances[msg.sender] = balances[msg.sender].sub(tokens);
    balances[to] = balances[to].add(tokens);

    emit Transfer(msg.sender, to, tokens);

    return true;

}



// ------------------------------------------------------------------------

// Token owner can approve for `spender` to transferFrom(...) `tokens`

// from the token owner's account

//

// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md

// recommends that there are no checks for the approval double-spend attack

// as this should be implemented in user interfaces

// ------------------------------------------------------------------------

function approve(address spender, uint tokens) public returns (bool success) {

    allowed[msg.sender][spender] = tokens;

    emit Approval(msg.sender, spender, tokens);

    return true;

}



// ------------------------------------------------------------------------

// Transfer `tokens` from the `from` account to the `to` account
```

```
    //
    // The calling account must already have sufficient tokens approve(...)-d
    // for spending from the `from` account and
    // - From account must have sufficient balance to transfer
    // - Spender must have sufficient allowance to transfer
    // - 0 value transfers are allowed
    // ------------------------------------------------------------------------
    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = balances[from].sub(tokens);
        allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        emit Transfer(from, to, tokens);
        return true;
    }


    // ------------------------------------------------------------------------
    // Returns the amount of tokens approved by the owner that can be
    // transferred to the spender's account
    // ------------------------------------------------------------------------
    function allowance(address tokenOwner, address spender) public view returns (uint
remaining) {
        return allowed[tokenOwner][spender];
    }



    // ------------------------------------------------------------------------

    // Token owner can approve for `spender` to transferFrom(...) `tokens`

    // from the token owner's account. The `spender` contract function

    // `receiveApproval(...)` is then executed

    // ------------------------------------------------------------------------

    function approveAndCall(address spender, uint tokens, bytes data) public returns (bool
success) {

        allowed[msg.sender][spender] = tokens;

        emit Approval(msg.sender, spender, tokens);

        ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);

        return true;

    }



    // ------------------------------------------------------------------------

    // Don't accept ETH

    // ------------------------------------------------------------------------

    function () public payable {

        revert();

    }
    // ------------------------------------------------------------------------
    // Owner can transfer out any accidentally sent ERC20 tokens
    // ------------------------------------------------------------------------
    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner
returns (bool success) {
        return ERC20Interface(tokenAddress).transfer(owner, tokens);
    }
}
```

The code is well commented and you shouldn’t have any problems understanding it.



Environment: Injected Web3 Rinkeby (4)
Account: 0x959...943e4 (17.667422687 ether)
Gas limit: 3000000
Value: 0 wei

Token

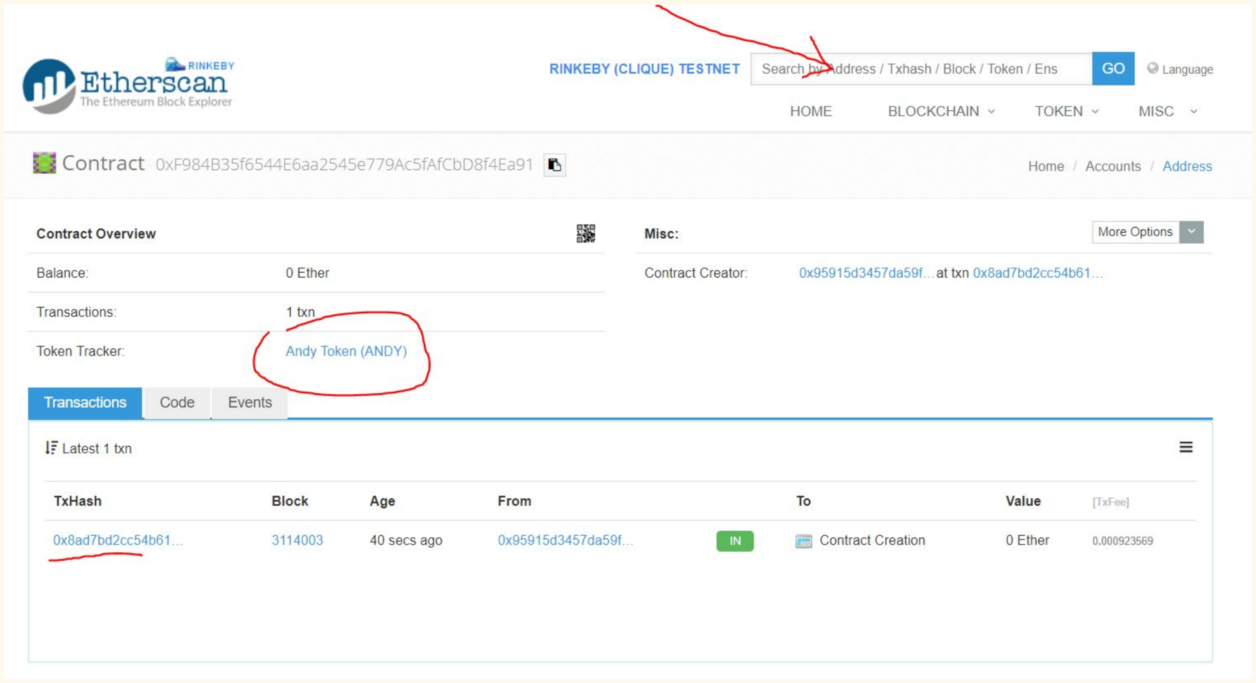
Transactions recorded: 0

Deployed Contracts

Currently you have no contract instances to interact with.

Select the Token from the dropdown and click deploy.
Metamask will prompt you to confirm the transaction. After ~10 seconds your contract will appear.

If you click the copy to clipboard on the contract address and paste it in the block explorer for Rinkeby you should see it



Contract 0xF984B35f6544E6aa2545e779Ac5fAfCbD8f4Ea91

Contract Overview

Balance:	0 Ether
Transactions:	1 txn
Token Tracker:	Andy Token (ANDY)

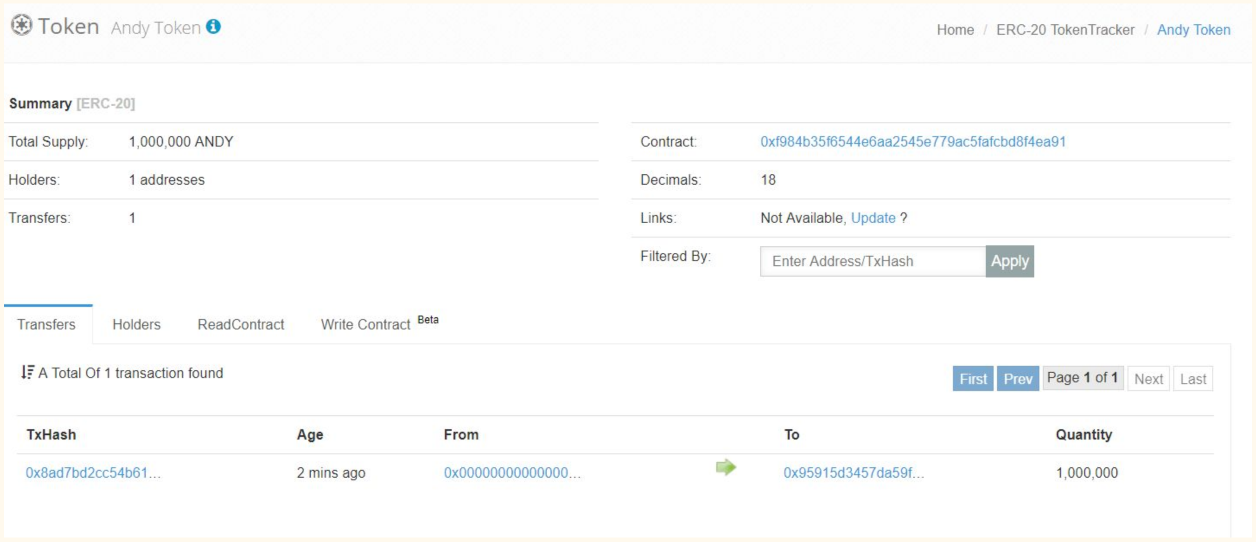
Misc:

Contract Creator: 0x95915d3457da59f... at txn 0x8ad7bd2cc54b61...

Latest 1 txn

TxHash	Block	Age	From		To	Value	[TxFee]
0x8ad7bd2cc54b61...	3114003	40 secs ago	0x95915d3457da59f...	IN	Contract Creation	0 Ether	0.000923569

Click on your token name and you should see it detected as an ERC20 token



Token Andy Token

Summary [ERC-20]

Total Supply:	1,000,000 ANDY
Holders:	1 addresses
Transfers:	1

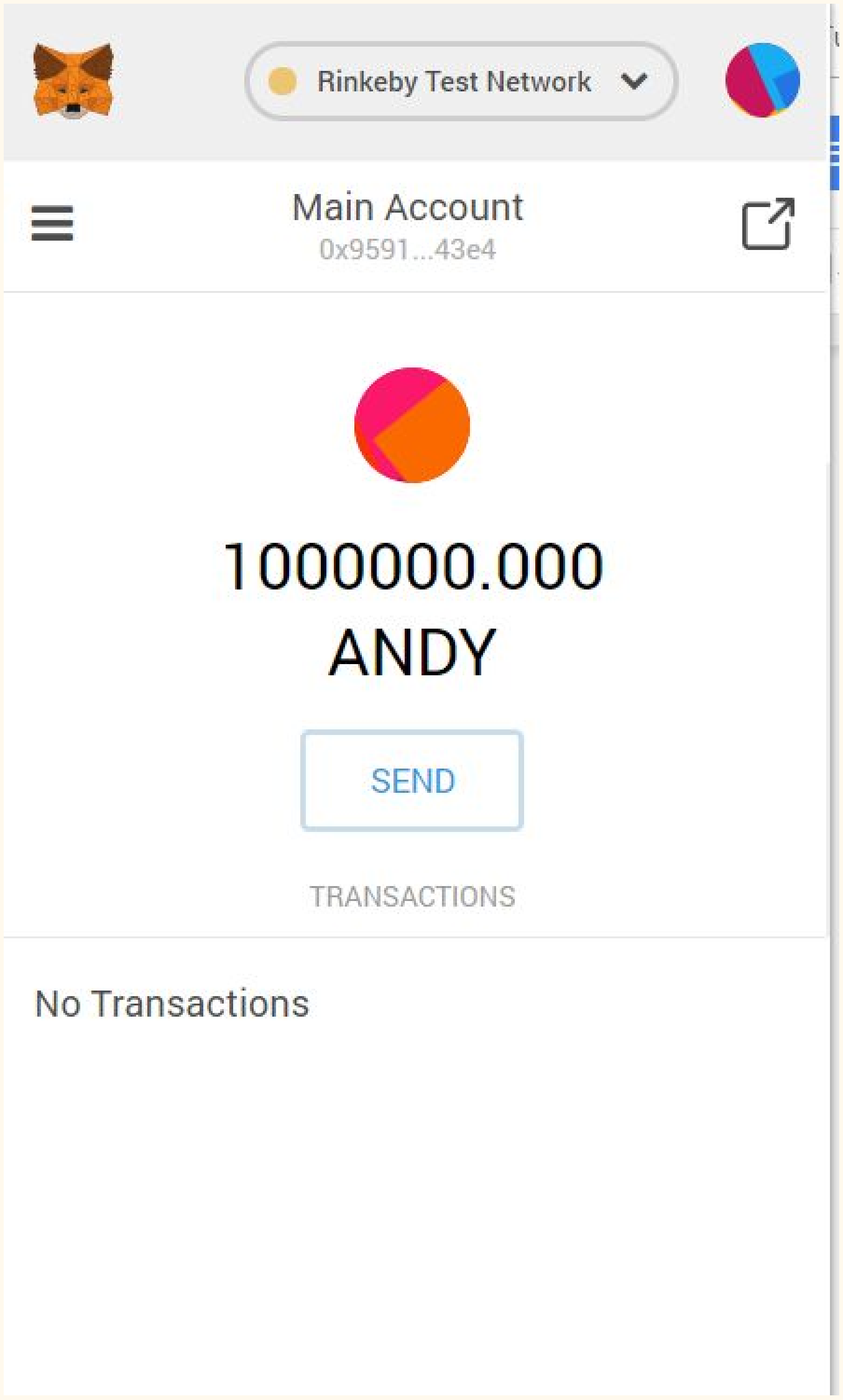
Contract:	0xf984b35f6544e6aa2545e779ac5fafcbd8f4ea91
Decimals:	18
Links:	Not Available, Update ?
Filtered By:	Enter Address/TxHash

A Total Of 1 transaction found

TxHash	Age	From		To	Quantity
0x8ad7bd2cc54b61...	2 mins ago	0x00000000000000...	➡	0x95915d3457da59f...	1,000,000

With our contract published we can now add the tokens to our metamask account.

Select Menu -> Add Token -> Custom Token and just paste the token address. It should automatically detect it's name and symbol.



1000000.000 ANDY

No Transactions

Congratulations!

Now that everything works perfectly we just have to verify our smart contract so that everyone on the blockchain can read and understand it. It's always a good practice to verify since it helps establish trust.

Now go to your contract address and click on Contract Code tab. Select your compiler version, in our case 0.4.25, optimization no. and paste the code.

Now that everything's working, if you want, it's time to deploy it on the MainNet and let other people use it.

This part is simple.

Just do steps 3 & 4, but instead of being connected to the Rinkeby Test Network, you want to be connected to the MainNet. Make sure your MetaMask account is in Mainnet mode.

Chapter X: Moving from remix IDE to ...

Remix IDE is an awesome editor, but usually solidity developers don't just write solidity, they also write Javascript or they need to generate other stuff like bindings etc. The typical setup of a blockchain developer is:

1. 90% use VSCode https://code.visualstudio.com/
   Why ? It's fast, has lots of extensions, easy to customize and it's free
   a. Others worth mentioning: Intellij IDEA, Sublime Text
2. 2. Ganache running for tests. Usually ganache-cli, but you can start with ganache gui for now
3. 3. Truffle Framework to build, publish and test contracts.

Setup an Ethereum Private Blockchain using Geth

In a nutshell: we will setup two nodes on the same machine, creating a peer-to-peer network on our localhost. In addition to the two nodes, a bootnode (discovery service) will also be setup.

It took me quite some time and extensive research and googling to finally have a solid ethereum development environment for testing my smart contracts and my DApps.

1. Initial Setup

You need 3 clean Ubuntu 16 machines (vps, virtual machine, aws, digital ocean or anywhere you'd like). I'll be using virtual machines. You can start them with virtualbox or vmware player (both free).

1 - micro -> called **controller**, 2 - medium called **nodes**

Open a txt file on your computer called poageth.txt and write their IPs:

Controller: 192.168.121.128

Node0: 192.168.121.129

Node1: 192.168.121.130

Setup the controller:

$sudo apt-get install openssh-server
$ssh-keygen (enter 3 times)

Setup the nodes:

$sudo apt-get install openssh-server

$ssh-keygen (enter 3 times)

1. Set the right permissions:
2. chmod 700 ~/.ssh
3. Create the authorized_keys file:
4. touch ~/.ssh/authorized_keys
5. Set the right permissions:
6. chmod 600 ~/.ssh/authorized_keys

Back to the controller

$ssh-copy-id username@node_ip

Enter the password for the node

If you use AWS:

$cat ~/.ssh/id_rsa.pub | ssh -i idchain.pem ubuntu@1.2.3.4 “mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys”

In the end you should be able to login from the controller to the node instances only using

$ssh user@node_ip

This will enable the controller (puppeth) to perform it's setup on the nodes.

Still stuck ? check this nice tutorial -> **https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys-on-ubuntu-1604**

Setup the nodes (part 2):

$sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce docker-compose
sudo usermod -aG docker $USER
```

Relog and then run **"docker ps"** to make sure permissions are correct.

Setup the controller (part 2):

sudo apt-get install build-essential bison git

Now since Geth is written in Go, we need to install Go

```
cd /tmp
```

```
wget -q https://storage.googleapis.com/golang/getgo/installer_linux
chmod +x installer_linux
./installer_linux
source $HOME/.bash_profile
echo 'export GOPATH=$HOME/go' >> ~/.bashrc
echo 'export PATH=${PATH}:${GOPATH}/bin' >> ~/.bashrc
source ~/.bashrc
```

Now it's time to install Geth

```
cd ~
git clone https://github.com/ethereum/go-ethereum.git
cd go-ethereum
make all
```

Check if Puppeth is working

```
cd build/bin
$./puppeth
```

You should see:

```
+-----------------------------------------------------------+

| Welcome to puppeth, your Ethereum private network manager |

|                                                           |

| This tool lets you create a new Ethereum network down to  |
| the genesis block, bootnodes, miners and ethstats servers |
| without the hassle that it would normally entail.         |
|                                                           |
| Puppeth uses SSH to dial in to remote servers, and builds |
| its network components out of Docker containers using the |
| docker-compose toolset.                                   |
+-----------------------------------------------------------+

Please specify a network name to administer (no spaces or hyphens, please)
> ^C
```

Close it for now.

We need some some accounts that we'll use for different things like the nodes, faucet etc.

Create a password file called

password.txt

Write a password inside and close it

```
cd ~
nano password.txt
sudo perl -pi -e 'chomp if eof' password.txt
chmod 700 password.txt
```

Now it's time to get 10 accounts

```
for ((n=0;n<10;n++)); do ~/go-ethereum/build/bin/geth account new --password
~/password.txt; done
```

Where are those accounts created ?

Geth stores them ~/.ethereum/keystore

```
ubuntu@ubuntu:~/.ethereum/keystore$ ll ~/.ethereum/keystore/

total 48

drwx------ 2 ubuntu ubuntu 4096 Sep 15 03:47 ./

drwx------ 3 ubuntu ubuntu 4096 Sep 15 03:47 ../

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-44.766619742Z--7bed420acdf37e61231407de006b50e6d22ada0d
```

```
-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-45.339734780Z--65f26d8f840344c23e7f83f5a62ffaa5e5ba525c

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-45.906787731Z--68e4a38765a14410b238557c193a3ccc10379d87

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-46.478851086Z--e327c665d53b10cdbd3b594fb3d7d58aaf003b4f

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-47.081282958Z--f17cfc697f9975491774696370d60a02df0024a9

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-47.671313954Z--cd45c0f5a6960a3b2b69f26caa940d2c1c06b520

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-48.259993821Z--bb9ec4b97daef7e8f0c66174e385579d47614aa4

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-48.836033927Z--0b38820039b2adbd8f3396da21895abfba5e66a6

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-49.428882968Z--e8f98b579565e63b2fd5edb259c0eb5099e8d2eb

-rw------- 1 ubuntu ubuntu  491 Sep 15 03:47
UTC--2018-09-15T10-47-50.034061077Z--5d06b383b6504bded962712b98e68a075f7edc41
```

If you notice, the last part of the file is the wallet address of the account

We need to copy some of them to our poageth.txt file

The first one will be for sealer0 (our node0), then sealer1 (node1), and the faucet.

We also need what’s inside the file. So Just run

cat UTC--2018-09-15T10-47-44.766619742Z--7bed420acdf37e61231407de006b50e6d22ada0d and copy it.

In the end we have something like this

Controller: 192.168.121.128

Node0: 192.168.121.129

7bed420acdf37e61231407de006b50e6d22ada0d

{"address":"7bed420acdf37e61231407de006b50e6d22ada0d","crypto":{"cipher":"aes-128-ctr","ciphertext":"85973620b1acfbad33b419f418784ffaf31850646ed6352410bb2f2a5e41d32f","cipherparams":{"iv":"2e823fb4070aa734be726e829784ab68"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"6a63bae517b9233aca43cdb0b2d0fcfe6fe156b055e58ebd9ceb318c71b5d65c"},"mac":"8b2761b4c62225328384878b55b494ebfd53b6c5c6a95a93f0ee0f8cbd3171f1"},"id":"8b6a8bc0-6b81-4a2e-815d-0919448ce028","version":3}

Node1: 192.168.121.130

65f26d8f840344c23e7f83f5a62ffaa5e5ba525c

{"address":"65f26d8f840344c23e7f83f5a62ffaa5e5ba525c","crypto":{"cipher":"aes-128-ctr","ciphertext":"2ec64761ea01a2593739e1cc57a45eeebcd44955253d5482247d7d6d5bebf8e5","cipherparams":{"iv":"2abc973fcc87b8a75b11f31389ea4712"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"71883b09644ec015dcabec8bc3366bf503db1a11dfa4a204ab0fe86e70bd49a8"},"mac":"40f801a53519d93cb7f2e14a966faeacc16dbaaece3553a6d052b2c9dfdc0c8d"},"id":"dd8200b2-8a12-4210-a1fc-52e84fc7e838","version":3}

Faucet:

68e4a38765a14410b238557c193a3ccc10379d87

{"address":"68e4a38765a14410b238557c193a3ccc10379d87","crypto":{"cipher":"aes-128-ctr","ciphertext":"a12280684d0935bdd9846ec44dc54b2e3ac570c35a043d03eb1e6ccd4118d50b","cipherparams":{"iv":"17e08314699f7c579b870b913fab8a42"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"55c2b337cd92eb8bd56cfc754e8e7a27e01f6014a2940ea53bc737df76c07d73"},"mac":"bd17b7035501fb58ec37851f5da318f3be388d9451827b412a20bc1885312956"},"id":"89a56e1b-bff8-41e4-8a8f-8b90765abde0","version":3}

e327c665d53b10cdbd3b594fb3d7d58aaf003b4f

f17cfc697f9975491774696370d60a02df0024a9

cd45c0f5a6960a3b2b69f26caa940d2c1c06b520

bb9ec4b97daef7e8f0c66174e385579d47614aa4

0b38820039b2adbd8f3396da21895abfba5e66a6

e8f98b579565e63b2fd5edb259c0eb5099e8d2eb

5d06b383b6504bded962712b98e68a075f7edc41

In Proof of Authority, there’s particular nodes which are assigned to validate blocks. These are known as “sealer” nodes. Each one needs its own account. You’re going to reserve three of our accounts for that purpose. New authority nodes can be added but they need to be elected by the rest of the network through their geth console. For now, you’re only concerned with setting up the initial authority network, so you’ll stick with the two sealer nodes.

There will also need to be an account which will handle the “faucet” you’ll install later. This will allow non-sealer nodes to request some eth for operational purposes. This faucet will come pre-loaded with some Ether on your network.

Creating the Genesis

It is important that you follow the next steps as they are, later when you learn how to do it, fell free to play around with settings

Every blockchain has to start somewhere, so there's what's called a genesis block at the beginning. This is the first block, and in it the creators of Ethereum were at liberty to say "To start, the following accounts all have X units of my cryptocurrency." Any transfer of that ether on the blockchain will have originated from one of these initial accounts (or from mining).

We'll use Puppeth to generate it

On the controller, start puppeth with

./go-ethereum/build/bin/puppeth

"Please specify a network name to administer (no spaces or hyphens, please)"

Type your network name: I'll call mine "space"

The next prompt is:

What would you like to do? (default = stats)

1. Show network stats

2. Configure new genesis

3. Track new remote server

4. Deploy network components

>

Choose 2), then again 2 for Clique - proof-of-authority

How many seconds should blocks take? (default = 15)

I'll choose 5

Which accounts are allowed to seal? (mandatory at least one)

> 0x

Here paste the accounts of node0 and node1 from the poageth.txt file you have, then just hit Enter on the last one

Next question is:

Which accounts should be pre-funded? (advisable at least one)

> 0x

Here paste the node0, node1 and faucet accounts + couple of other accounts. I'll choose 3. At the end just hit Enter to continue to the next step.

Specify your chain/network ID if you want an explicit one (default = random)

>

I'll choose 5555

Next step is:

What would you like to do? (default = stats)

1. Show network stats

```
 2. Manage existing genesis

 3. Track new remote server

 4. Deploy network components

>
```

Choose 2.

```
 1. Modify existing fork rules

 2. Export genesis configuration

 3. Remove genesis configuration

>
```

Choose 2 to export genesis configuration

```
Which file to save the genesis into? (default = space.json)

>
```

Usually it's called "genesis.json" so type it.

You should see now

```
INFO [09-15|10:37:56.609] Exported existing genesis block
```

Quit puppeth with CTRL+C

If you check what files you have in your current directory, you'll see a new file appeared called "genesis.json".

Take some time to inspect it.

Do the network in the following order: netstats (or ethstats), bootnode, sealers, wallet, faucet, dashboard

Deploy EthStats

You need to install Ethstats first for it to monitor the network. Ethstats shows you the status of your ethereum network

Start puppeth and when asked

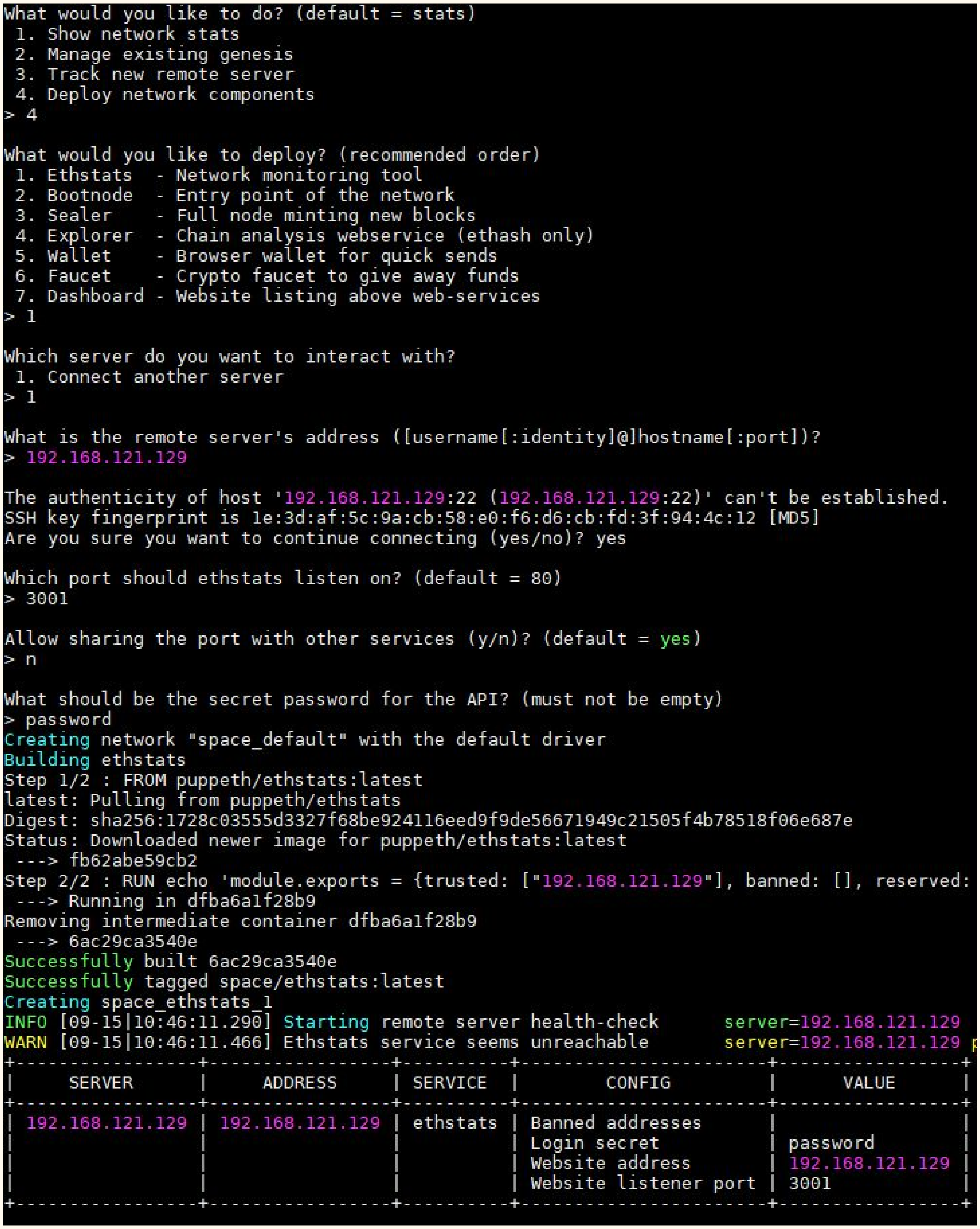
Please specify a network name to administer (no spaces or hyphens, please)

Type “space” (or whatever name you gave it)

You’re going to install this (and other applications) on “node0”

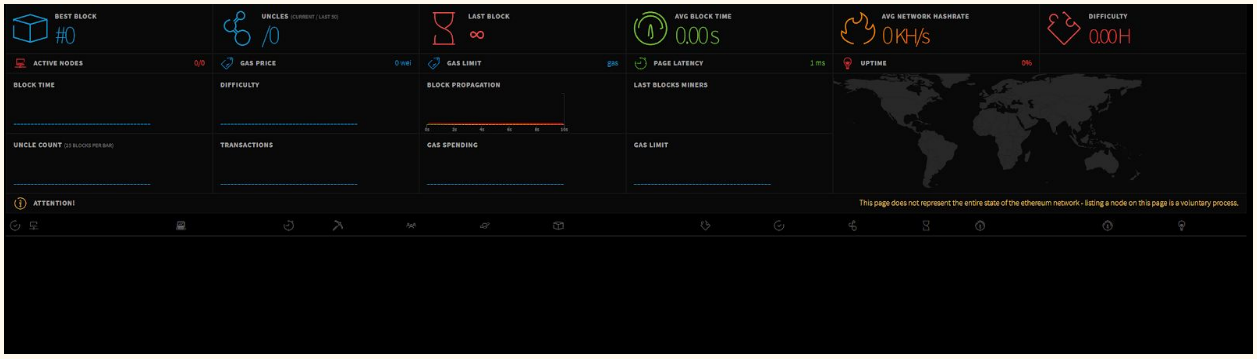
1. Type “4” for “4. Deploy network components”
2. Type “1” for “1. Ethstats — Network monitoring tool”
3. Type “1” for “1. Connect another server”
4. Paste in the IP address for “node0” (then accept the certificate)
5. Type in “3001” for the port Ethstats listens on
6. Type “n” to prevent port sharing with other services.
7. Type out a secret password for the API (put it in poageth.txt file)

Puppeth will connect to that server, pull the ethstats from docker and start it. At the end you should see something like this:



```
What would you like to do? (default = stats)
 1. Show network stats
 2. Manage existing genesis
 3. Track new remote server
 4. Deploy network components
> 4

What would you like to deploy? (recommended order)
 1. Ethstats  - Network monitoring tool
 2. Bootnode  - Entry point of the network
 3. Sealer    - Full node minting new blocks
 4. Explorer  - Chain analysis webservice (ethash only)
 5. Wallet    - Browser wallet for quick sends
 6. Faucet    - Crypto faucet to give away funds
 7. Dashboard - Website listing above web-services
> 1

Which server do you want to interact with?
 1. Connect another server
> 1

What is the remote server's address ([username[:identity]@]hostname[:port])?
> 192.168.121.129

The authenticity of host '192.168.121.129:22 (192.168.121.129:22)' can't be established.
SSH key fingerprint is 1e:3d:af:5c:9a:cb:58:e0:f6:d6:cb:fd:3f:94:4c:12 [MD5]
Are you sure you want to continue connecting (yes/no)? yes

Which port should ethstats listen on? (default = 80)
> 3001

Allow sharing the port with other services (y/n)? (default = yes)
> n

What should be the secret password for the API? (must not be empty)
> password
Creating network "space_default" with the default driver
Building ethstats
Step 1/2 : FROM puppeth/ethstats:latest
latest: Pulling from puppeth/ethstats
Digest: sha256:1728c03555d3327f68be924116eed9f9de56671949c21505f4b78518f06e687e
Status: Downloaded newer image for puppeth/ethstats:latest
 ---> fb62abe59cb2
Step 2/2 : RUN echo 'module.exports = {trusted: ["192.168.121.129"], banned: [], reserved:
 ---> Running in dfba6a1f28b9
Removing intermediate container dfba6a1f28b9
 ---> 6ac29ca3540e
Successfully built 6ac29ca3540e
Successfully tagged space/ethstats:latest
Creating space_ethstats_1
INFO [09-15|10:46:11.290] Starting remote server health-check      server=192.168.121.129
WARN [09-15|10:46:11.466] Ethstats service seems unreachable       server=192.168.121.129 p
+-----------------+-----------------+----------+-----------------------+-----------------+
|     SERVER      |     ADDRESS     | SERVICE  |        CONFIG         |      VALUE      |
+-----------------+-----------------+----------+-----------------------+-----------------+
| 192.168.121.129 | 192.168.121.129 | ethstats | Banned addresses      |                 |
|                 |                 |          | Login secret          | password        |
|                 |                 |          | Website address       | 192.168.121.129 |
|                 |                 |          | Website listener port | 3001            |
+-----------------+-----------------+----------+-----------------------+-----------------+
```

If we go to the node0 address in the browser and the port 3001 we should see



This page does not represent the entire state of the ethereum network - listing a node on this page is a voluntary process.

Create A Bootnode

A bootnode is a stripped down version of our Ethereum client implementation that only takes part in the network node discovery protocol, but does not run any of the higher level application protocols. It can be used as a lightweight bootstrap node to aid in finding peers in private networks.

My choices are in bold!

What would you like to do? (default = stats)

1. Show network stats

2. Manage existing genesis

3. Manage tracked machines

4. Manage network components

> 4

1. Tear down Ethstats on 192.168.121.129

2. Deploy new network component

```
> 2

What would you like to deploy? (recommended order)
 1. Ethstats  - Network monitoring tool
 2. Bootnode  - Entry point of the network
 3. Sealer    - Full node minting new blocks
 4. Explorer  - Chain analysis webservice (ethash only)
 5. Wallet    - Browser wallet for quick sends
 6. Faucet    - Crypto faucet to give away funds
 7. Dashboard - Website listing above web-services
> 2

Which server do you want to interact with?
 1. 192.168.121.129
 2. Connect another server
> 1

Where should data be stored on the remote machine?
> /home/ubuntu/bootnode

Which TCP/UDP port to listen on? (default = 30303)
>30310
```

How many peers to allow connecting? (default = 512)

> leave it default, just press enter here

How many light peers to allow connecting? (default = 256)

>leave it default, just press enter here

What should the node be called on the stats page?

> bootnode

Found orphan containers (space_ethstats_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.

Building bootnode

Step 1/4 : FROM ethereum/client-go:latest

latest: Pulling from ethereum/client-go

Create Sealer Nodes

In a PoA System you got Sealers and Signer - Nodes. Sealer are predefined in the genesis Block. So A Sealer Node is without a vote of the network allowed to mine/generate new blocks. If you want after a couple of time add new "Sealer"-Nodes, you need to add signers-Node.

A Signers Node is only allowed after n+1 (51%) Sealer-Nodes accept it. If in a Network are more signers, also the votes of the signer count.

So a Signer Node is practical the same like a sealer node, just with the different, that sealer node are defined in the genesis block and per-se allowed to mine/signing new Blocks and signers node first need a positive voting to mine new blocks.

BTW: only signers node can also be disabled to mine new block by a voting.

Create a sealer node for “node0”

```
What would you like to do? (default = stats)
 1. Show network stats
 2. Manage existing genesis
 3. Manage tracked machines
 4. Manage network components
> 4

 1. Tear down Bootnode on 192.168.121.129
 2. Tear down Ethstats on 192.168.121.129
 3. Deploy new network component
> 3

What would you like to deploy? (recommended order)
 1. Ethstats  - Network monitoring tool
 2. Bootnode  - Entry point of the network
 3. Sealer    - Full node minting new blocks
 4. Explorer  - Chain analysis webservice (ethash only)
 5. Wallet    - Browser wallet for quick sends
 6. Faucet    - Crypto faucet to give away funds
```

7. Dashboard - Website listing above web-services

> 3

Which server do you want to interact with?

1. 192.168.121.129

2. Connect another server

> 1

Where should data be stored on the remote machine?

> /home/ubuntu/sealer

Which TCP/UDP port to listen on? (default = 30303)

> **hit enter**

How many peers to allow connecting? (default = 50)

> **hit enter**

How many light peers to allow connecting? (default = 0)

> **hit enter**

What should the node be called on the stats page?

> sealer0

```
Please paste the signer's key JSON:

>  [here you would page the content of the generated account file.
{"address":"7bed420acdf37e61231407de006b50e6d22ada0d","crypto":{"cipher":"aes-128-ctr","ciphertext":"85973620b1acfbad33b419f418784ffaf31850646ed6352410bb2f2a5e41d32f","cipherparams":{"iv":"2e823fb4070aa734be726e829784ab68"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"6a63bae517b9233aca43cdb0b2d0fcfe6fe156b055e58ebd9ceb318c71b5d65c"},"mac":"8b2761b4c62225328384878b55b494ebfd53b6c5c6a95a93f0ee0f8cbd3171f1"},"id":"8b6a8bc0-6b81-4a2e-815d-0919448ce028","version":3}

What's the unlock password for the account? (won't be echoed)

> [enter the password]

What gas limit should empty blocks target (MGas)? (default = 7.500)

> hit enter

What gas limit should full blocks target (MGas)? (default = 10.000)

>hit enter

What gas price should the signer require (GWei)? (default = 1.000)

> 0.001

Found orphan containers (space_bootnode_1, space_ethstats_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the
--remove-orphans flag to clean it up.

Building sealnode

Step 1/6 : FROM ethereum/client-go:latest
```

```
 ---> 301f57eade08

Step 2/6 : ADD genesis.json /genesis.json

 ---> Using cache

 ---> 84c5f6761a95

Step 3/6 : ADD signer.json /signer.json

 ---> 7dfc4aed4e99

Step 4/6 : ADD signer.pass /signer.pass

 ---> e7f5dfa6cfd9

Step 5/6 : RUN   echo 'geth --cache 512 init /genesis.json' > geth.sh &&        echo 'mkdir -p
/root/.ethereum/keystore/ && cp /signer.json /root/.ethereum/keystore/' >> geth.sh &&    echo
$'exec geth --networkid 5555 --cache 512 --port 30303 --maxpeers 50  --ethstats
\'sealer0:password@192.168.121.129:3001\' --bootnodes
enode://27831d362aa2bd46fd926c01a5ae447c5a9468e60951ea12d0adcbeb86eff6613c4b0180
145fc1c2740bbef45991f3ce5633a83efdb9a7090921675d90dad5bc@192.168.121.129:30310
--unlock 0 --password /signer.pass --mine --miner.gastarget 7500000 --miner.gaslimit 10000000
--miner.gasprice 1000000' >> geth.sh

 ---> Running in fd545326d778

Removing intermediate container fd545326d778

 ---> ced522f516bf

Step 6/6 : ENTRYPOINT ["/bin/sh", "geth.sh"]

 ---> Running in db60fce0405e

Removing intermediate container db60fce0405e

 ---> 6b258396413b

Successfully built 6b258396413b

Successfully tagged space/sealnode:latest

Creating space_sealnode_1
```

```
INFO [09-15|11:01:44.115] Waiting for node to finish booting
INFO [09-15|11:01:47.116] Starting remote server health-check      server=192.168.121.129
```

I hope you got the idea how Puppeth is creating things. In case you're not careful and select the wrong ip, you can always tear-down the installed component.

Repeat the same procedure for node1 (remember at the beginning to select the node1 ip)

Start Mining

Login into node0

```
$docker ps
```

You should see a bunch of container ids.

Look for the container id for the sealer node

```
docker exec -it [CONTAINER_ID] geth attach ipc:/root/.ethereum/geth.ipc
```

```
> admin.nodeInfo.enode
"enode://2de0c70703e0b5483f60f69acb6ed60fb8a675af24f32e2c4d0d341121de86db45e067b7
7e41917d39515dc3b701305e9482e12062cb9f4f87b3b532e55dbc61@[::]:30303"
```

You can read more about enode here https://github.com/ethereum/wiki/wiki/enode-url-format

replace the [::] with the IP address of this server.

```
"enode://2de0c70703e0b5483f60f69acb6ed60fb8a675af24f32e2c4d0d341121de86db45e067b7
7e41917d39515dc3b701305e9482e12062cb9f4f87b3b532e55dbc61@192.168.121.129:30303"
```

And store it in poageth.txt file

Do the same for the other sealer node.

Now in the geth instance on the node0 add the enode of node1

```
admin.addPeer("<enode from other server with IP inserted>")
```

And the same for node1 using the enode for node0

[If we would have more sealer nodes, we would add all the other nodes “enodes” too)

Remains wallet, faucet and a dashboard

Wallet

```
What would you like to do? (default = stats)

 1. Show network stats

 2. Manage existing genesis

 3. Manage tracked machines

 4. Manage network components

> 4


 1. Tear down Sealnode on 192.168.121.129

 2. Tear down Faucet on 192.168.121.129

 3. Tear down Ethstats on 192.168.121.129

 4. Tear down Bootnode on 192.168.121.129

 5. Tear down Sealnode on 192.168.121.130
 6. Deploy new network component
> 6

What would you like to deploy? (recommended order)
 1. Ethstats  - Network monitoring tool
 2. Bootnode  - Entry point of the network
 3. Sealer    - Full node minting new blocks
 4. Explorer  - Chain analysis webservice (ethash only)
 5. Wallet    - Browser wallet for quick sends
 6. Faucet    - Crypto faucet to give away funds
 7. Dashboard - Website listing above web-services
> 5

Which server do you want to interact with?
 1. 192.168.121.129
 2. 192.168.121.130
 3. Connect another server
> 1

Which port should the wallet listen on? (default = 80)
> 3003

Allow sharing the port with other services (y/n)? (default = yes)
```

```
> n

Where should data be stored on the remote machine?

> /home/ubuntu/wallet

Which TCP/UDP port should the backing node listen on? (default = 30303)

> 30301

Which port should the backing RPC API listen on? (default = 8545)

>

What should the wallet be called on the stats page?

> wallet

Found orphan containers (space_faucet_1, space_sealnode_1, space_bootnode_1,
space_ethstats_1) for this project. If you removed or renamed this service in your compose file,
you can run this command with the --remove-orphans flag to clean it up.

Building wallet

Step 1/5 : FROM puppeth/wallet:latest

latest: Pulling from puppeth/wallet
```

Faucet

```
What would you like to deploy? (recommended order)

 1. Ethstats  - Network monitoring tool

 2. Bootnode  - Entry point of the network

 3. Sealer    - Full node minting new blocks

 4. Explorer  - Chain analysis webservice (ethash only)

 5. Wallet    - Browser wallet for quick sends

 6. Faucet    - Crypto faucet to give away funds

 7. Dashboard - Website listing above web-services

> 6


Which server do you want to interact with?

 1. 192.168.121.129

 2. 192.168.121.130

 3. Connect another server

> 1


Which port should the faucet listen on? (default = 80)

> 3002


Allow sharing the port with other services (y/n)? (default = yes)

> n


How many Ethers to release per request? (default = 1)
```

```
> 1

How many minutes to enforce between requests? (default = 1440)
>

How many funding tiers to feature (x2.5 amounts, x3 timeout)? (default = 3)
>

Enable reCaptcha protection against robots (y/n)? (default = no)
>
WARN [09-15|11:27:15.297] Users will be able to requests funds via automated scripts

Where should data be stored on the remote machine?
> /home/ubuntu/faucet

Which TCP/UDP port should the light client listen on? (default = 30303)
> 30307

What should the node be called on the stats page?
> faucet

Please paste the faucet's funding account key JSON:
```

>

{"address":"68e4a38765a14410b238557c193a3ccc10379d87","crypto":{"cipher":"aes-128-ctr","ciphertext":"a12280684d0935bdd9846ec44dc54b2e3ac570c35a043d03eb1e6ccd4118d50b","cipherparams":{"iv":"17e08314699f7c579b870b913fab8a42"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"55c2b337cd92eb8bd56cfc754e8e7a27e01f6014a2940ea53bc737df76c07d73"},"mac":"bd17b7035501fb58ec37851f5da318f3be388d9451827b412a20bc1885312956"},"id":"89a56e1b-bff8-41e4-8a8f-8b90765abde0","version":3}

What's the unlock password for the account? (won't be echoed)

>

Permit non-authenticated funding requests (y/n)? (default = false)

> y

Found orphan containers (space_sealnode_1, space_bootnode_1, space_ethstats_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.

Building faucet

Step 1/6 : FROM ethereum/client-go:alltools-latest

alltools-latest: Pulling from ethereum/client-go

And finally, the Dashboard

1. “4. Deploy network components”
2. “8. Deploy new network component”

3. “7. Dashboard — Website listing above web-services”
4. Select option #1 for “node0”
5. Type in “**3000**” for the port the dashboard listens on
6. Type “n” to prevent port sharing with other services.
7. Type “1” to list the local Ethstats service
8. Type “2” to opt-out of an explorer service
9. Type “1” to list the local wallet service
10. Type “1” to list the local faucet service
11. Go with the default for including the Ethstats secret on the dashboard

Optional:

Let’s configure nginx to serve the dashboard on port 80. You can also add all the goodies that nginx has, certificates etc. etc.

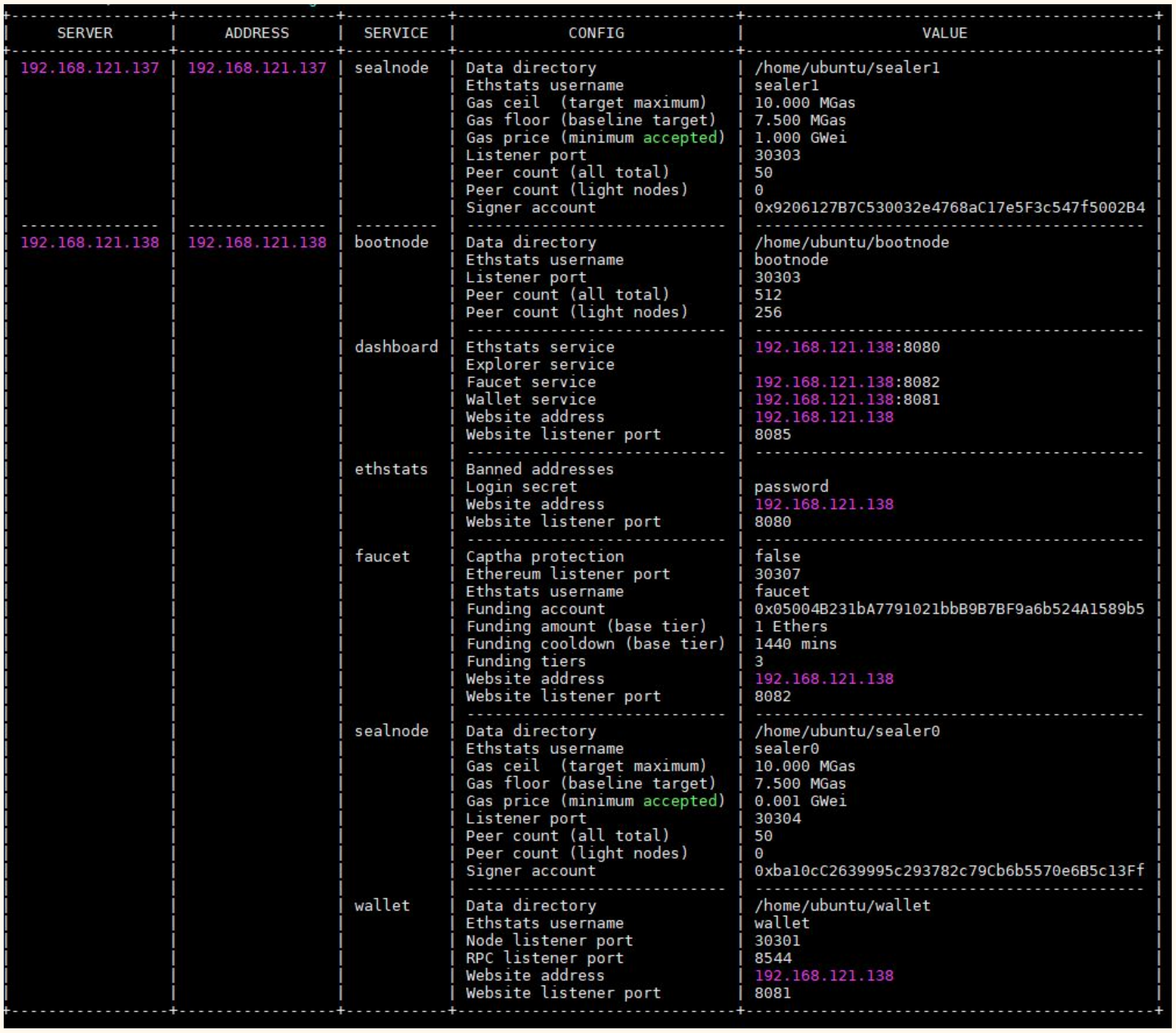
On the server containing the dashboard edit the sites-enabled

sudo nano /etc/nginx/sites-enabled/default

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        location / {
          proxy_pass http://localhost:8085;
          proxy_buffering off;
```

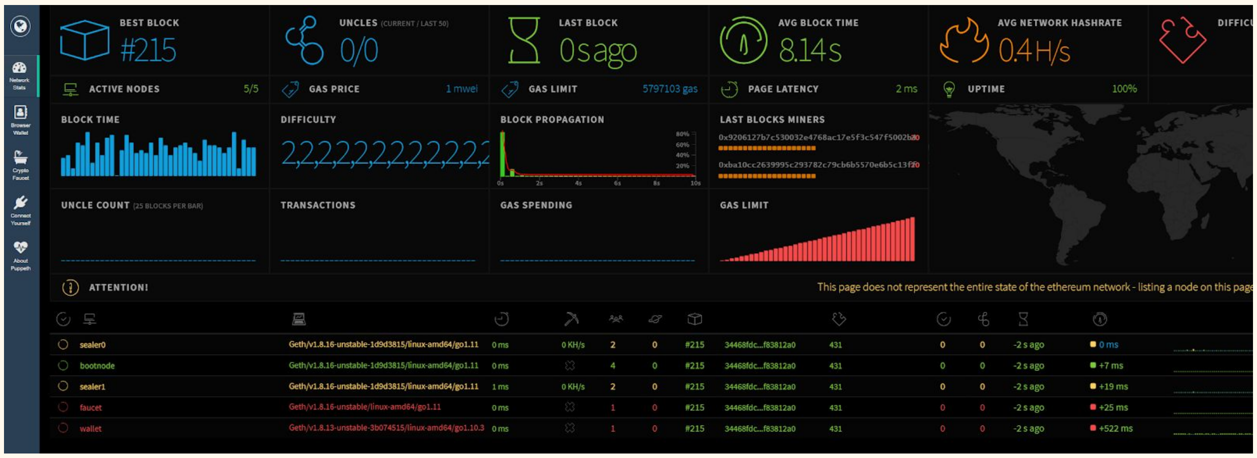
```
    }

}

sudo nginx -t

sudo service nginx restart
```

Finally you should have something like this:



```
+-----------------+-----------------+-----------+------------------------------+---------------------------------------------+
|     SERVER      |     ADDRESS     |  SERVICE  |            CONFIG            |                    VALUE                    |
+-----------------+-----------------+-----------+------------------------------+---------------------------------------------+
| 192.168.121.137 | 192.168.121.137 | sealnode  | Data directory               | /home/ubuntu/sealer1                        |
|                 |                 |           | Ethstats username            | sealer1                                     |
|                 |                 |           | Gas ceil  (target maximum)   | 10.000 MGas                                 |
|                 |                 |           | Gas floor (baseline target)  | 7.500 MGas                                  |
|                 |                 |           | Gas price (minimum accepted) | 1.000 GWei                                  |
|                 |                 |           | Listener port                | 30303                                       |
|                 |                 |           | Peer count (all total)       | 50                                          |
|                 |                 |           | Peer count (light nodes)     | 0                                           |
|                 |                 |           | Signer account               | 0x9206127B7C530032e4768aC17e5F3c547f5002B4  |
| --------------- | --------------- | --------- | ---------------------------- | ------------------------------------------- |
| 192.168.121.138 | 192.168.121.138 | bootnode  | Data directory               | /home/ubuntu/bootnode                       |
|                 |                 |           | Ethstats username            | bootnode                                    |
|                 |                 |           | Listener port                | 30303                                       |
|                 |                 |           | Peer count (all total)       | 512                                         |
|                 |                 |           | Peer count (light nodes)     | 256                                         |
|                 |                 | --------- | ---------------------------- | ------------------------------------------- |
|                 |                 | dashboard | Ethstats service             | 192.168.121.138:8080                        |
|                 |                 |           | Explorer service             |                                             |
|                 |                 |           | Faucet service               | 192.168.121.138:8082                        |
|                 |                 |           | Wallet service               | 192.168.121.138:8081                        |
|                 |                 |           | Website address              | 192.168.121.138                             |
|                 |                 |           | Website listener port        | 8085                                        |
|                 |                 | --------- | ---------------------------- | ------------------------------------------- |
|                 |                 | ethstats  | Banned addresses             |                                             |
|                 |                 |           | Login secret                 | password                                    |
|                 |                 |           | Website address              | 192.168.121.138                             |
|                 |                 |           | Website listener port        | 8080                                        |
|                 |                 | --------- | ---------------------------- | ------------------------------------------- |
|                 |                 | faucet    | Captha protection            | false                                       |
|                 |                 |           | Ethereum listener port       | 30307                                       |
|                 |                 |           | Ethstats username            | faucet                                      |
|                 |                 |           | Funding account              | 0x05004B231bA7791021bbB9B7BF9a6b524A1589b5  |
|                 |                 |           | Funding amount (base tier)   | 1 Ethers                                    |
|                 |                 |           | Funding cooldown (base tier) | 1440 mins                                   |
|                 |                 |           | Funding tiers                | 3                                           |
|                 |                 |           | Website address              | 192.168.121.138                             |
|                 |                 |           | Website listener port        | 8082                                        |
|                 |                 | --------- | ---------------------------- | ------------------------------------------- |
|                 |                 | sealnode  | Data directory               | /home/ubuntu/sealer0                        |
|                 |                 |           | Ethstats username            | sealer0                                     |
|                 |                 |           | Gas ceil  (target maximum)   | 10.000 MGas                                 |
|                 |                 |           | Gas floor (baseline target)  | 7.500 MGas                                  |
|                 |                 |           | Gas price (minimum accepted) | 0.001 GWei                                  |
|                 |                 |           | Listener port                | 30304                                       |
|                 |                 |           | Peer count (all total)       | 50                                          |
|                 |                 |           | Peer count (light nodes)     | 0                                           |
|                 |                 |           | Signer account               | 0xba10cC2639995c293782c79Cb6b5570e6B5c13Ff  |
|                 |                 | --------- | ---------------------------- | ------------------------------------------- |
|                 |                 | wallet    | Data directory               | /home/ubuntu/wallet                         |
|                 |                 |           | Ethstats username            | wallet                                      |
|                 |                 |           | Node listener port           | 30301                                       |
|                 |                 |           | RPC listener port            | 8544                                        |
|                 |                 |           | Website address              | 192.168.121.138                             |
|                 |                 |           | Website listener port        | 8081                                        |
+-----------------+-----------------+-----------+------------------------------+---------------------------------------------+
```

And the dashboard



BEST BLOCK #215
UNCLES (CURRENT / LAST 50) 0/0
LAST BLOCK 0s ago
AVG BLOCK TIME 8.14s
AVG NETWORK HASHRATE 0.4 H/s
DIFFICU

ACTIVE NODES 5/5
GAS PRICE 1 mwei
GAS LIMIT 5797103 gas
PAGE LATENCY 2 ms
UPTIME 100%

BLOCK TIME
DIFFICULTY 2,2,2,2,2,2,2,2,2,2,2,2
BLOCK PROPAGATION
LAST BLOCKS MINERS
0x9206127b7c530032e4768ac17e5f3c547f5002b20
0xba10cc2639995c293782c79cb6b5570e6b5c13f20

UNCLE COUNT (25 BLOCKS PER BAR)
TRANSACTIONS
GAS SPENDING
GAS LIMIT

ATTENTION! This page does not represent the entire state of the ethereum network - listing a node on this page

sealer0	Geth/v1.8.16-unstable-1d9d3815/linux-amd64/go1.11	0 ms	0 KH/s	2	0	#215	34468fdc...f83812a0	431	0	0	-2 s ago	0 ms
bootnode	Geth/v1.8.16-unstable-1d9d3815/linux-amd64/go1.11	0 ms		4	0	#215	34468fdc...f83812a0	431	0	0	-2 s ago	+7 ms
sealer1	Geth/v1.8.16-unstable-1d9d3815/linux-amd64/go1.11	1 ms	0 KH/s	2	0	#215	34468fdc...f83812a0	431	0	0	-2 s ago	+19 ms
faucet	Geth/v1.8.16-unstable/linux-amd64/go1.11	0 ms		1	0	#215	34468fdc...f83812a0	431	0	0	-2 s ago	+25 ms
wallet	Geth/v1.8.13-unstable-3b074515/linux-amd64/go1.10.3	0 ms		1	0	#215	34468fdc...f83812a0	431	0	0	-2 s ago	+522 ms

Using the Private Blockchain

With all the hard work put into deploying all the above stuff, let's now see how we can put our brand new shiny blockchain to work.

On your local pc (laptop) copy the genesis.json file that you have on the controller instance.

We want to connect using a full node

A **full node** synchronizes the blockchain by downloading the full chain **from the genesis block to the current head block**, but does not execute the transactions. Instead, it downloads all the transactions receipts along with the entire recent state. **As the node downloads the recent state directly, historical data can only be queried from that block onward.**

Initial processing required to synchronize is more bandwidth intensive, but is light on the CPU and has significantly reduced disk requirements. Mid range machines with HDD storage, decent CPUs and 4GB+ RAM should be enough.

```
geth --identity "mylocalnode" --datadir=$HOME/.space init genesis.json
```

replace the network id, the enodes and what kind of apis you want to use

When you open the dashboard you can see the command you have to run to connect to your nodes. We're going to add two things to it

1) increase the number of peers by adding our sealer nodes enodes.
2) add rpc apis, websocket apis, shh api

Mine looks something like this.
Largest part of this command are the 3 nodes.

```
geth --datadir ~\.space --networkid 1337
--bootnodes=enode://328b36c2897e2ddb36f1091d0622efc47e80f124a867b93faa4c0dfedecaf5c
0d0aed08d75704b6dbae582e20d2d79cffaa4b11d06a88968727ee89d237ad4cb@192.168.121.
138:30303,enode://8bb144e304bd25123a522593f4540c64b7d42ebbad7f42f743e1313026a681
ec40560c903ff90a4f70b6526be443cc9f4749d6ae816586e26a7623cd238b1a03@192.168.121.
137:30303,enode://42aa0f3f526ed6a9032e0c1b3526e15b20df4dc0229f9a6f991e6f31d86561dfb
2d5d14d36dcaa29867a31709ab0d7e0fc868eb6d7dde344116318386aeb87d5@192.168.121.13
8:30304 --rpc --rpcapi admin,db,eth,net,web3,personal,txpool --shh --ws --rpcaddr localhost
--rpcport 8545 --rpccorsdomain "*"
```

Now you can connect to it, publish your contracts and do whatever you want.

Troubleshooting:

1.
   ```
   WARN [09-23|19:33:10.881] Synchronisation failed, dropping peer
   peer=8bb144e304bd2512 err="retrieved hash chain is invalid"
   ```

Run geth removedb --datadir ~/.space and resync