WiFi SSID: CoWrks

Password: @C0Wrks@!@#

Blockchain Hands on Meetup

Shubhamoy Chakrabarty

How many of you wish to create a decentralized application?

Problem Statement

Pranjal, Tushar, and Mehak started a venture together. Customers pay them in cryptos for their services but they’ve challenges in transferring the funds based on everyone’s share. So they’ve approached us to create a smart contract which would receive all the funds and automatically transfer each share to their individual wallets. They also want us to provide a web interface to modify the wallet address and percentage share.

Let’s quickly fire up the environment

What all do I need?

1. Node.JS
2. Truffle Framework
3. Ganache (optional)

How to Install Node.JS?

1. Volunteers

How do I install Truffle Framework?

```
$> npm i -g truffle
```

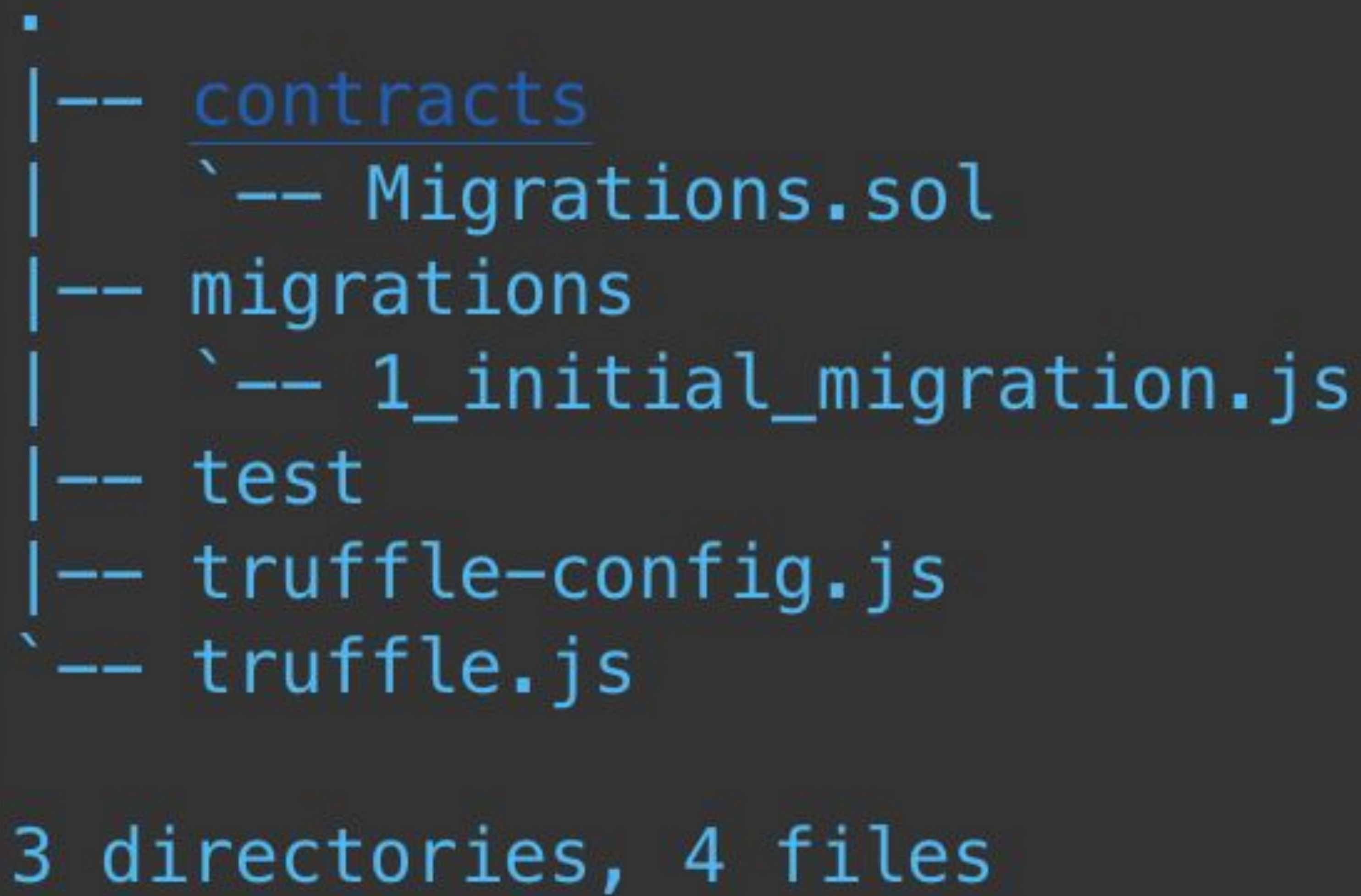
How do I install Ganache? (optional)

1. Download from https://truffleframework.com/ganache
2. If you face an issue, then we need to use `` `truffle develop` `` inside our project directory

Time for Project Setup

How to create a new project?

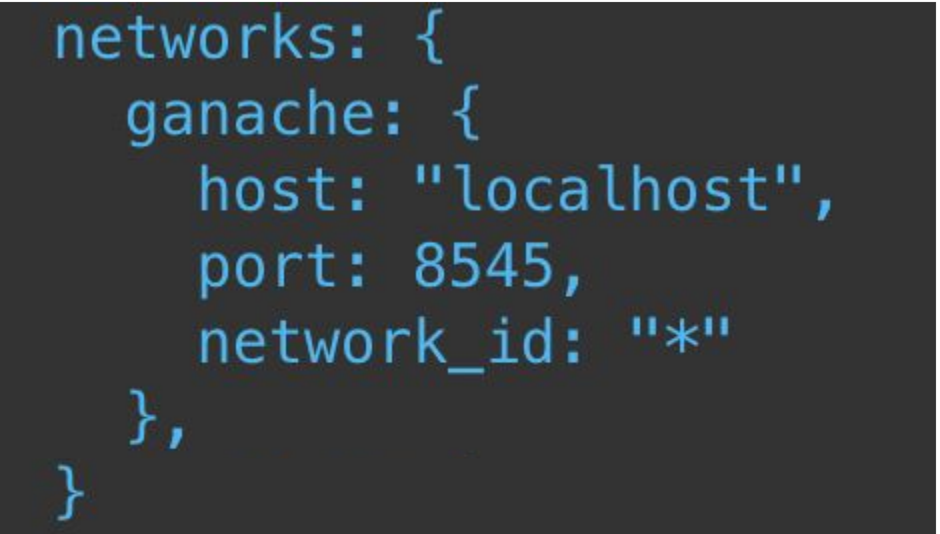
1. Create a new directory
2. Run ’truffle init‘ and `npm init` inside the directory
3. Our project is ready, let’s understand the structure



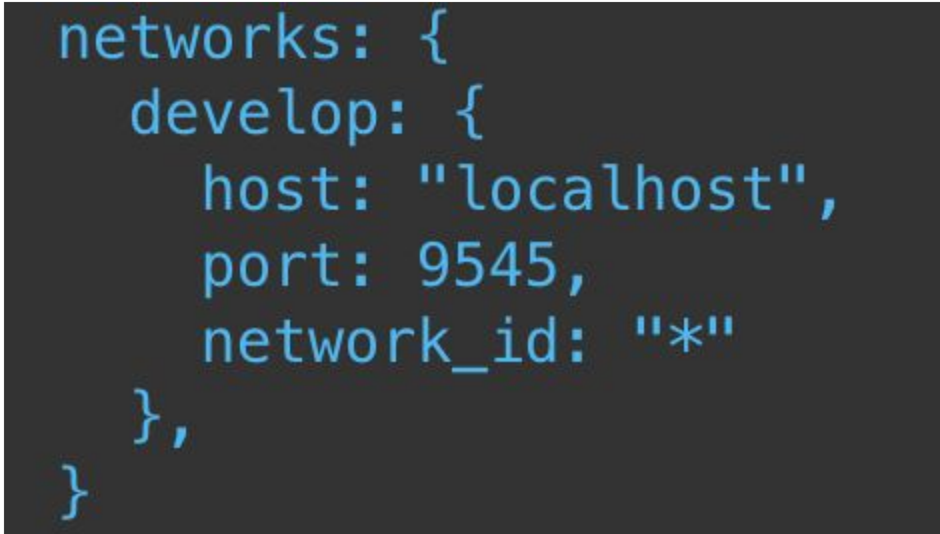
```
.
|-- contracts
|   `-- Migrations.sol
|-- migrations
|   `-- 1_initial_migration.js
|-- test
|-- truffle-config.js
`-- truffle.js

3 directories, 4 files
```

Connect truffle to use ethereum

Open `truffle.js` in the project directory and create a network object.



```
networks: {
  ganache: {
    host: "localhost",
    port: 8545,
    network_id: "*"
  },
}
```



```
networks: {
  develop: {
    host: "localhost",
    port: 9545,
    network_id: "*"
  },
}
```

Let’s test

Enter the project root and enter the following command

```
$> truffle console --network ganache
```

Console Commands

1. web3.eth
2. web3.eth.accounts
3. web3.eth.coinbase
4. web3.eth.getBalance(web3.eth.accounts[0])
5. web3.fromWei(web3.eth.getBalance(web3.eth.accounts[0]), “ether”).toNumber()
6. .exit

Let’s start with the contract...

Before we begin, once more we will read the problem statement.

Problem Statement

Pranjal, Tushar, and Mehak started a venture together. Customers pay them in cryptos for their services but they’ve challenges in transferring the funds based on everyone’s share. So they’ve approached us to create a smart contract which would receive all the funds and automatically transfer each share to their individual wallets. They also want us to provide a web interface to modify the wallet address and percentage share.

Let’s begin…

Process

1. Write the contract
2. Deploy
3. Test
4. Write down the test cases (this should come after first step)
5. Linting

Pseudocode

Anybody would like to volunteer?

Hints:

1. Arrays to store wallet address and percentages
2. Method to configure
3. Method to receive funds and transfer as per configuration

Download the barebone contract

http://uri.im/sw1

Check this ppt online at https://uri.im/ethppt

Alert, Warning, Notice...

The DAO and SafeMath

```
$> npm i openzeppelin-solidity
```

Final Version

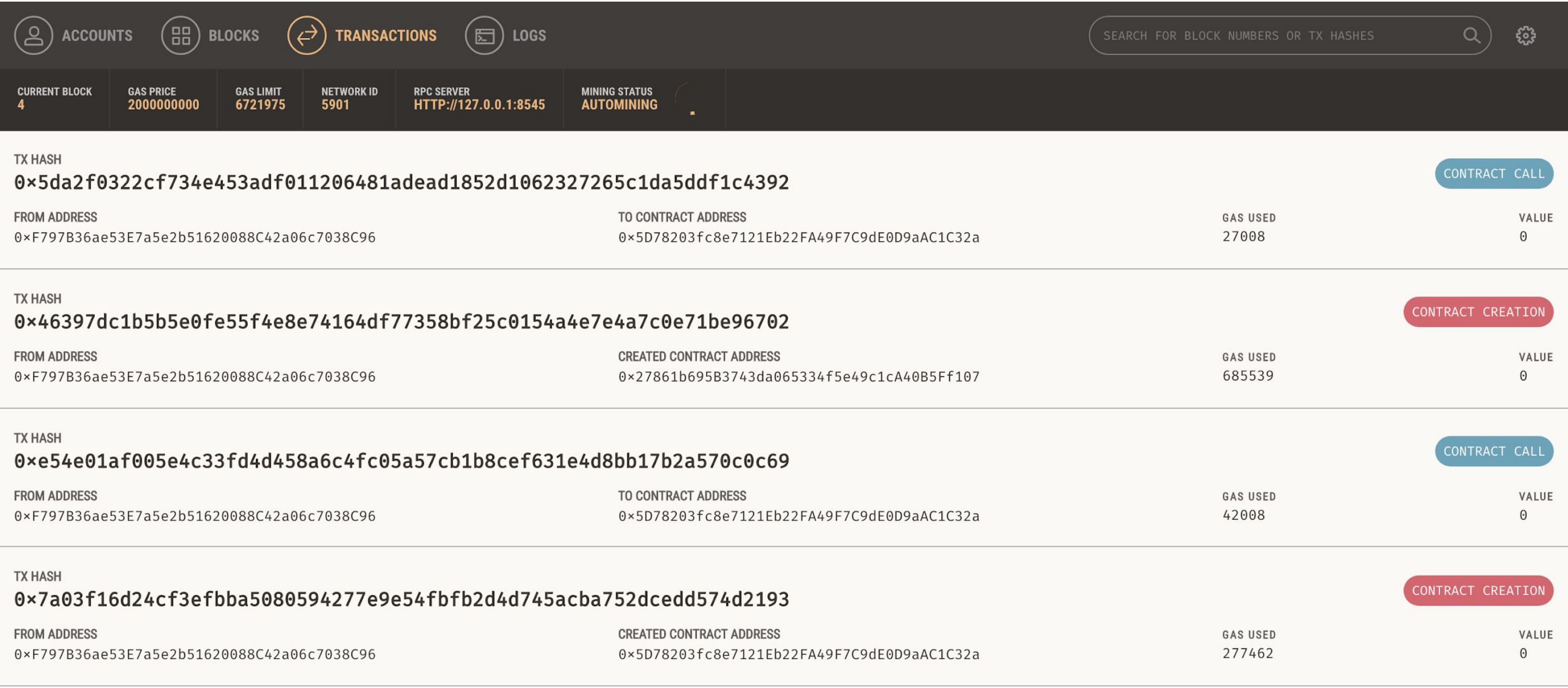
http://uri.im/sw2

Migration

https://uri.im/sw3

Migrate the contract

```
$> truffle migrate --network ganache
```



CURRENT BLOCK
4

GAS PRICE
2000000000

GAS LIMIT
6721975

NETWORK ID
5901

RPC SERVER
HTTP://127.0.0.1:8545

MINING STATUS
AUTOMINING

TX HASH
0x5da2f0322cf734e453adf011206481adead1852d1062327265c1da5ddf1c4392
CONTRACT CALL

FROM ADDRESS
0xF797B36ae53E7a5e2b51620088C42a06c7038C96

TO CONTRACT ADDRESS
0x5D78203fc8e7121Eb22FA49F7C9dE0D9aAC1C32a

GAS USED
27008

VALUE
0

TX HASH
0x46397dc1b5b5e0fe55f4e8e74164df77358bf25c0154a4e7e4a7c0e71be96702
CONTRACT CREATION

FROM ADDRESS
0xF797B36ae53E7a5e2b51620088C42a06c7038C96

CREATED CONTRACT ADDRESS
0x27861b695B3743da065334f5e49c1cA40B5Ff107

GAS USED
685539

VALUE
0

TX HASH
0xe54e01af005e4c33fd4d458a6c4fc05a57cb1b8cef631e4d8bb17b2a570c0c69
CONTRACT CALL

FROM ADDRESS
0xF797B36ae53E7a5e2b51620088C42a06c7038C96

TO CONTRACT ADDRESS
0x5D78203fc8e7121Eb22FA49F7C9dE0D9aAC1C32a

GAS USED
42008

VALUE
0

TX HASH
0x7a03f16d24cf3efbba5080594277e9e54fbfb2d4d745acba752dcedd574d2193
CONTRACT CREATION

FROM ADDRESS
0xF797B36ae53E7a5e2b51620088C42a06c7038C96

CREATED CONTRACT ADDRESS
0x5D78203fc8e7121Eb22FA49F7C9dE0D9aAC1C32a

GAS USED
277462

VALUE
0

How do I write a test case?

1. Enter the project directory and create a new file inside the test directory
2. https://uri.im/sw4

Let’s do linting

```
$> npm i -g solium

$> solium --init

$> solium --file contracts/SmartWallet.sol
```

End of Section 1

In this section, we started with setting up the development environment followed by creating our barebone project using truffle framework. We configured our project to either use ganache or truffle develop as a ethereum instance. After that we interacted with truffle console followed it we understood the process involved. Smart contract started taking shape from the barebone code. We touched topics of DAO and SafeMath.

We migrated our contract to the ethereum instance using the migration script. This was followed by writing test cases and linting.

Frontend Development

Process

1. Setup webserver
2. Install MetaMask
3. Setup a webform
4. Send request to the ethereum instance using web3 library

Setup the web server

```
$> npm i --save-dev lite-server
```

```
$> npm i -g lite-server
```

(Windows users only)

Configure the web server

Create a **`bs-config.json`** file in the project root

```
{"server":{"baseDir": ["./src", "./build/contracts"]}}
```

Add `"dev": "lite-server",` in the `scripts` section of `package.json`

Install MetaMask

https://metamask.io

Download the files

https://uri.im/sw5

Let’s review distributed systems!

Challenges of Distributed Systems

1. Data **C**onsistency Issue
2. **A**vailability
3. **P**artition Tolerance

CAP Theorem: A distributed system isn’t free from network partition. In presence of a network partition, we’re left with either availability or consistency.

Byzantine Generals’ Problem

1. An agreement problem
2. A node may appear functioning to one node and faulty to another
3. Byzantine Fault Tolerance

Consensus Algorithms

1. Paxos
2. Raft
3. PBFT

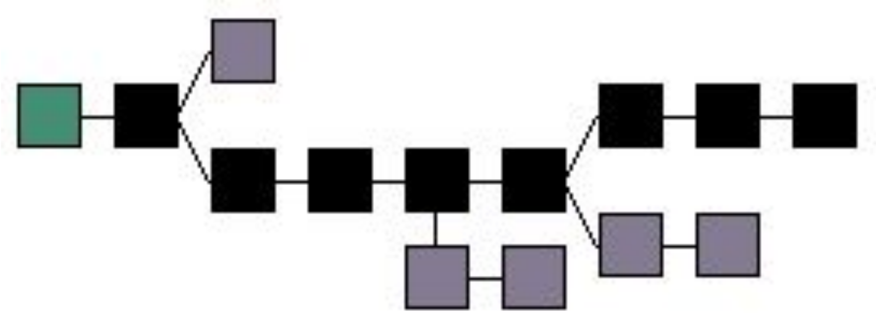
Take Away

1. Blockchains are distributed systems
2. CAP Theorem
3. Algorithms for achieving consensus in a distributed network
4. Read: Vector Clocks

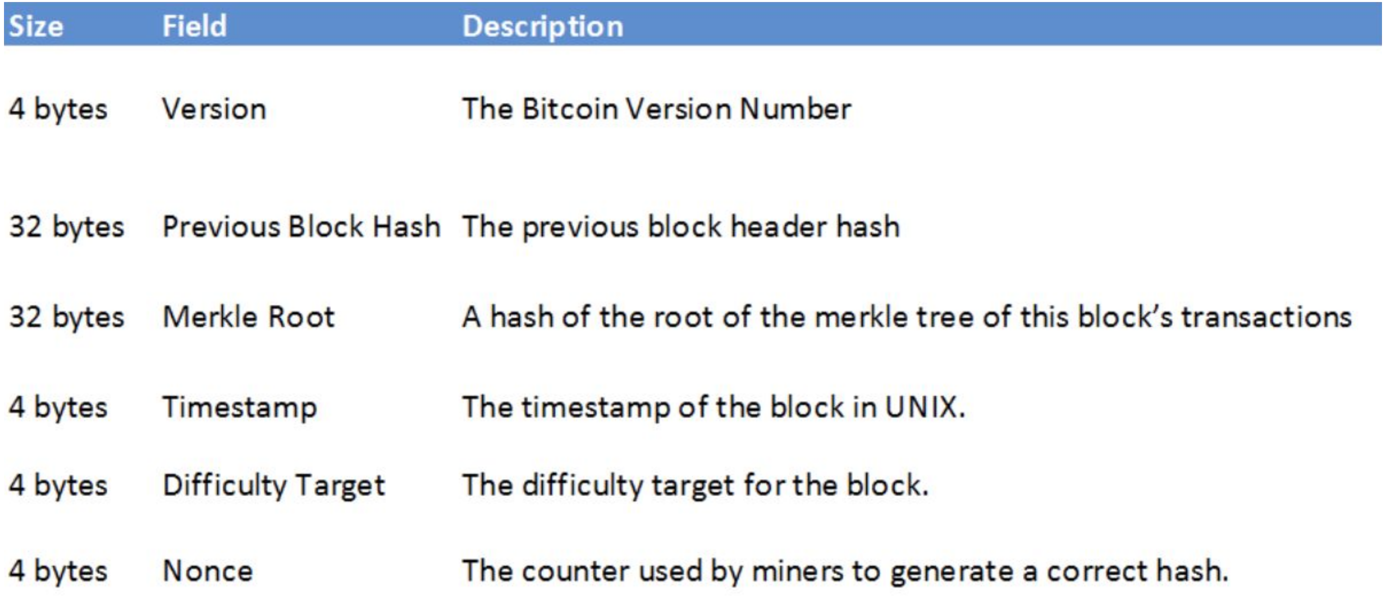
Blockchain and related terms

What is blockchain?

Blocks have data and the address to the previous block. Blocks are linked using cryptography.



Block Header (bitcoin)



Size	Field	Description
4 bytes	Version	The Bitcoin Version Number
32 bytes	Previous Block Hash	The previous block header hash
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The timestamp of the block in UNIX.
4 bytes	Difficulty Target	The difficulty target for the block.
4 bytes	Nonce	The counter used by miners to generate a correct hash.

Merkle Tree

1. txn1, txn2, txn3, and txn4
2. hashTxn1, hashTxn2, hashTxn3, and hashTxn4
3. hashTxn12, hashTxn34
4. hashTxn1234 <= Merkle Root
5. Ethereum uses Patricia Tree

Proof of Work

1. Consensus Protocol
2. Solving mathematical puzzle (largest 6 digit even number divisible by 2)
3. Cannibalistic Arms Race (fastest computer wins)
4. Maintain consistency of the blockchain
5. Ethereum plans to switch to Proof of Stake

Ever downloaded a file using uTorrent?

1. Pretty relevant for understanding blockchains
2. File downloaded isn’t stored at a single location

I’ve a question…

https://uri.im/ethq