A hands-on tutorial: Working with Smart Contracts in Ethereum

by Roman Vitenberg

Joint work with Mohammad H. Tabatabaei from the University of Oslo

Different tools provide different functionality

<table>
<thead>
<tr>
<th>Activities</th>
<th>Tools</th>
<th>Remix</th>
<th>Ganache</th>
<th>MyEtherWallet</th>
<th>Geth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Configure the Blockchain</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>2 Deploy the Blockchain</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>3 Develop the contract</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>4 Compile the contract</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>5 Create user account</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>6 Deploy the contract</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>7 Create the UI for interacting</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>8 Run the client</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>9 Interact with the contract &amp; have fun</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>10 Monitor the execution</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

https://remix.ethereum.org/
http://truffleframework.com/ganache/
https://github.com/kvhnuke/etherwallet/releases/tag/v3.21.06
Use which tool for what purpose? (1/2)

• Use Geth for everything?
  • Powerful but command-line only

• What should I use?
  • As a starting point for developing contracts – mostly Remix

• What cannot Remix do?
  • Configure the blockchain
  • Create real (non-test) user accounts and transfer funds between user accounts
  • Monitor the execution
  • Other advanced operations

Use which tool for what purpose? (2/2)

• Why use Ganache?
  • To inspect and monitor the execution
  • To visualize certain elements in a better way

• Why use MyEtherWallet?
  • To create a personal wallet (real user account) and transfer funds between user accounts
Smart Contracts

1. Developing a simple contract
2. Compiling the contract
3. Deploying the contract
4. Interacting with the contract
5. Adding more functions to our code to make it more practical

Open Remix: remix.ethereum.org

• An open source tool for writing, compiling and testing Solidity contracts
Start Coding

• Setter and Getter: Set and get the information.

![Code snippet showing Setter and Getter functions](image)

Compile the Contract

• Compile tab: Start to compile button

![Compilation interface](image)
Set Environment (1/2)

• Run tab: Environment = JavaScript VM

Set Environment (2/2)

• JavaScript VM: All the transactions will be executed in a sandbox blockchain in the browser. Nothing will be persisted and a page reload will restart a new blockchain from scratch, the old one will not be saved.

• Injected Provider: Remix will connect to an injected web3 provider. Mist and Metamask are example of providers that inject web3, thus they can be used with this option.

• Web3 Provider: Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.

• Gas Limit: The maximum amount of gas that can be set for all the transactions of a contract.

• Value: The amount of value for the next created transaction (wei = 10^{-18} of ether).
Types of Blockchain Deployment

- **Private**: e.g., Ganache sets a personal Ethereum blockchain for running tests, executing commands, and inspecting the state while controlling how the chain operates.
- **Public Test**: Like Ropsten, Kovan and Rinkeby which are existing public blockchains used for testing and which do not use real funds.
- **Public Real**: Like Bitcoin and Ethereum which are used for real and which available for everybody to join.

---

Deploy the Contract on the Private Blockchain of Remix

- **Run tab**: Deploy button
Interact with the Contract

• Setter = Red Button: Creates transaction
• Getter = Blue Button: Just gives information

Press `getValue` to see the initial amount

Press `setValue` button to create and confirm the transaction
Press `getValue` again to see the result

Additional features

• Saving the address of the contract creator
• Limiting the users’ access to functions
• Transferring funds from an account to the contract
• Withdrawing funds from the contract to an account
Constructor
- A function with the name of the contract
- Will be called at the creation of the instance of the contract

```solidity
pragma solidity ^0.4.0;

contract financialContract{
    uint amount;
    address issuer;

    function financialContract(){
        issuer = msg.sender;
    }

    function getValue() constant returns(uint){
        return amount;
    }

    function setValue(uint newValue) {
        amount = newValue;
    }
}
```

We want to save the address of the contract creator.

Modifier
- Conditions you want to test in other functions
- First the modifier will execute, then the invoked function

```solidity
pragma solidity ^0.4.0;

contract financialContract{
    uint amount;
    address issuer;

    function financialContract(){
        issuer = msg.sender;
    }

    function getValue() constant returns(uint){
        return amount;
    }

    modifier ifIssuer(){
        if(issuer != msg.sender){
            throw;
        }else{
            return;
        }
    }

    function setValue(uint newValue) ifIssuer {
        amount = newValue;
    }
}
```

Only the contract creator is permitted to set value.
Receive ether (1/2)

- Transfer money to the contract

Payable keyword allows receiving ether

We can get the balance of the contract

Receive ether (2/2)

1. Input the value as wei (10^{-18} of ether)
2. Click the receiveFunds button to transfer the money to the contract
Withdraw funds

- Transfer ether from the contract to the user account

Transfer some money from the contract to the mentioned account

Solidity version

More details in the source code: https://github.com/ramyhardan/proof-of-existence

An event can be listened to by any client

Getters are automatically generated for public vars

The modifier code can be prepended to any function

Constructor is executed when the contract is created

ProofCreated event is emitted to the blockchain

View/constant functions are read-only and do not cost gas
Now deploying a smart contract on an external blockchain

<table>
<thead>
<tr>
<th>Activities</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Remix</td>
</tr>
<tr>
<td>1 Configuring the Blockchain</td>
<td>-</td>
</tr>
<tr>
<td>2 Deploying the Blockchain</td>
<td>Not Persistent</td>
</tr>
<tr>
<td>3 Developing the contract</td>
<td>+</td>
</tr>
<tr>
<td>4 Compiling the contract</td>
<td>+</td>
</tr>
<tr>
<td>5 Creating user account</td>
<td>+</td>
</tr>
<tr>
<td>6 Deploying the contract</td>
<td>+</td>
</tr>
<tr>
<td>7 Creating the UI for interacting</td>
<td>+</td>
</tr>
<tr>
<td>8 Run the client</td>
<td>+</td>
</tr>
<tr>
<td>9 Interact with the contract &amp; have fun</td>
<td>+</td>
</tr>
<tr>
<td>10 Monitoring the execution</td>
<td>-</td>
</tr>
</tbody>
</table>

Run Ganache
MyEtherWallet

- add your custom network that you want to test your contracts on

Import your RPC server address and the port number from Ganache to MyEtherWallet
MyEtherWallet

- Contracts tab: Deploy Contract

![MyEtherWallet interface with Deploy Contract option highlighted]

Remix

- Type your contract and compile it

![Remix interface with code snippet and compile button highlighted]
Remix
Click on Details Button: access ByteCode to import it to MyEtherWallet

Ganache
Access your private key for signing your contract in MyEtherWallet.
1. Paste the contract’s ByteCode from Remix

2. Gas Limit will automatically be calculated

3. Paste your private key from Ganache

4. Click Unlock

5. Now you have access to your wallet

MyEtherWallet

Click on Sign Transaction button to deploy your contract
Ganache
You can see now you have one transaction for your address and your balance has been changed because of the amount of gas you paid for creating the contract.

Interacting with the smart contract

- Extract the contract address from Ganache
- Extract the ABI (Application Binary Interface) of the code from Remix
- Interact with the contract in MyEtherWallet (Import the contract address and the ABI into the MyEtherWallet)
- Select a function
- Write
- Pay some gas
- Generate the transaction
- Receive the result
Ganache
Transactions tab: Copy the created contract address

Remix
Click on Details button: Copy the ABI
(ABI is the interface that tells MyEtherWallet how to interact with the contract)
MyEtherWallet
Contracts tab:
Interact with Contract = Paste the contract address from Ganache and the ABI from Remix

MyEtherWallet
You now can interact with the contract by selecting a function and invoking it
MyEtherWallet

If you select the getValue function you will receive the value without paying any gas
(There is no operation cost for getting information)

MyEtherWallet

If you choose a function that updates the state of the contract, you will need to pay gas for it in a transaction.
MyEtherWallet

Now if you try getValue function again, you will see the change.

Create your own Ethereum Blockchain

• Instead of using Ganache with its default properties for private blockchain you can run your own blockchain
• Install Geth: One of the implementations of Ethereum written in Go
• Create the genesis block
• Create storage of the blockchain
• Deploy blockchain nodes
• Connect MyEtherWallet to your blockchain to interact with it
Homebrew (package manager for mac)

- Install homebrew with the command from its website: https://brew.sh/

Geth

- An Ethereum program written in Go

```
# Install Ethereum/ethereum
1. mohamht~ $ brew tap ethereum/ethereum
2. mohamht~ $ brew install ethereum
```
Geth help

Genesis block

- The first block in the chain and a json file that stores the configuration of the chain

- Create and store the file as genesis.json
Create the storage of the blockchain

• Go to the directory of the genesis.json file
• Specify directory of your blockchain
• Create the storage from the genesis block

Inside the Blockchain Folder

• `geth` folder: Store your database
• `keystore`: Store your Ethereum accounts
Start the Ethereum peer node

- Start the blockchain

```bash
geth --datadir fistBC --networkid 100 console
```

- Networkid provides privacy for your network.
- Other peers joining your network must use the same networkid.

Blockchain started

- Type `admin.nodeInfo` to get the information about your current node
Create an account

• Type `personal.newAccount` to create as many accounts as you need

```
> personal.newAccount('Type your password here')
"8xa7eb41a0f096d4d8c4c9ca5196427aaa3fbd33"
```

• See the created account(s)

```
> eth.accounts
["0xa7eb41a0f096d4d8c4c9ca5196427aaa3fbd33", "0x354d952e40fc35a47562d479c86e41f6623e5f8c"]
```

---

Mining

• Type `miner.start()` to start mining

```
> miner.start()
INFO [05-10] [12:07:54] Updated mining threads: threads=4
INFO [05-10] [12:07:54] Transaction pool price threshold: updated price: 0.0000000800
INFO [05-10] [12:07:54] Starting mining operation
INFO [05-10] [12:07:54] Commit new mining work
8.827µs
INFO [05-10] [12:07:57] Generating DAG in progress
INFO [05-10] [12:07:59] Generating DAG in progress
INFO [05-10] [12:08:03] Generating DAG in progress
INFO [05-10] [12:08:06] Generating DAG in progress
INFO [05-10] [12:08:08] Generating DAG in progress
INFO [05-10] [12:08:10] Generating DAG in progress
INFO [05-10] [12:08:13] Generating DAG in progress
INFO [05-10] [12:08:16] Generating DAG in progress
INFO [05-10] [12:08:18] Generating DAG in progress
INFO [05-10] [12:08:20] Generating DAG in progress
```

Thank you