Ethereum DApp development

With Javascript (2019)

Andrea Lisi, andrealisi.12lj@gmail.com
Web3

Web3 is an interface to the Ethereum network.

It talks to the network by means of the JSON-RPC protocol, Remote Procedure Call.

En fact, communications with Ethereum network are asynchronous.
Web3
Web3 implementations

web3Js: JavaScript
web3J: Java
web3py: Python
hs-web3: Haskell
web3.php
Tools overview: Smart contract Dev

Solidity

Write Ethereum
Smart contracts
Tools overview: Smart contract Dev

- Solidity
  - Write Ethereum Smart contracts

- Truffle
  - Compile, Deploy
  - Test locally

- Ganache
Tools overview: DApp Development

- Solidity: Write Ethereum Smart contracts
- Truffle: Compile, Deploy Test locally
- Ganache: Test locally
- Javascript
- Metamask: Develop Web App
Tools overview: DApp Development

- **Solidity**: Write Ethereum Smart contracts
- **Truffle**: Compile, Deploy Test locally
- **Ganache**: Compile, Deploy Test locally
- **Metamask**: Develop Web App
- **Javascript**: Develop Web App
- **Infura**: Deploy to real Ethereum network
Tools overview: NodeJs

Some of these tools and libraries need to be installed by means of NodeJs and Npm (Node Package Manager)
Part 1

Truffle framework

Play with Truffle. Compile, migrate and test the contracts.
Walkthrough

- Introduce the first set of tools
- Init a new Truffle project
  - Project structure
  - Configuration file
- Development
  - Coding
  - Compiling
  - Migrating
  - Testing
Tools: the Truffle Framework

Truffle is a framework providing developers sweet tools for Ethereum smart contracts. In particular:

- the Solidity **compiler**
- **migrate** contracts to the Ethereum network;
- a **testing** environment;
- DApp code **boilerplates**;
- NodeJs **console** to interact with the migrated contracts;
- **execute** NodeJs scripts within the Truffle environment to automate commands inserted in the console.
Tools: Ganache

Ganache is a **blockchain** running *locally*, useful during the development phase. It provides fake Ethereum accounts that can be used for testing.
## Ganache, how it looks like

<table>
<thead>
<tr>
<th>Address</th>
<th>Balance</th>
<th>TX Count</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x627306090abaB3A6e1400e9345bC60c78a8BEf57</td>
<td>100.00 ETH</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0xf17f52151EbEF6C7334FAD080c5704D77216b732</td>
<td>100.00 ETH</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0xC5df4076b8F3A5357c5E395ab970B5B54098Fef</td>
<td>100.00 ETH</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0x821aEa9a577a9b44299B9c15c88cf3087F3b5544</td>
<td>100.00 ETH</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0x69e352B4A5e59005baE58C577f2B39377b2C9C9A5537</td>
<td>100.00 ETH</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Installation: Truffle

Requirements: NodeJs (v8.9.4 or later) and Node Packet Manager (npm)

Installation:

```
$ (sudo) npm install -g truffle
```

Installs truffle globally in your system
Truffle: create a project

$ truffle init

Initialize an empty Ethereum project in the current folder structured as:
- contracts/
- migrations/
- test/
- truffle-config.js
Truffle: contracts/  

This folder is supposed to contain the Solidity smart contracts. It includes a Migration contract useful for Truffle. Don’t delete it. I did it once and nothing was working anymore.
Truffle: migrations/

This folder is supposed to contain Javascript sources that Truffle will execute during the migration phase, i.e. when the contracts will be migrated / deployed to the blockchain.

Basically, such scripts should perform an initialization phase creating (deploying) the contracts.
The scripts are executed following a lexicographic order, so the convention is to name them 1_***.js, 2_+++.js
Truffle: test/

This folder is supposed to contain Javascript sources that should test the correctness of the smart contracts.

Truffle uses the Mocha testing framework.
Truffle: truffle-config.js

This is Truffle configuration file. With it, is possible to configure target networks (Eth mainnet, test net, local net, private net if you have one etc...), the Solidity compiler Solc and settings for Mocha.
Truffle: truffle-config.js, networks

```javascript
networks: {
  development: {
    host: "127.0.0.1", // Localhost (default: none)
    port: 7545,       // Standard Ethereum port (default: none)
    network_id: "*", // Any network (default: none)
  },
  ropsten: {
    provider: () => new HDWalletProvider(mnemonic,
      `https://ropsten.infura.io/${infuraKey}`),
    network_id: 3,    // Ropsten's id
    gas: 5500000,    // Ropsten has a lower block limit than mainnet
  },
}
```
Example contract

```solidity
contract MyContract {

    uint public value;
    event click();

    constructor() public {
        value = 1;
    }

    function myFunction() public view returns (uint) {
        return value + 7;
    }

    function pressClick() public {
        emit click();
    }
}
```
Truffle: dev phase, compilation

Write a Solidity smart contract;

Compile it with the command

$ truffle compile

It compiles all the contracts in contract/. It creates the build/ folder with the results of the compilation.
Write the migration file. We can continue the 1_initial_migrations.js file.

```javascript
const Migrations = artifacts.require("Migrations");
// load the json file MyContract.json from the /build folder
const MyContract = artifacts.require("MyContract");

module.exports = function(deployer) {
    deployer.deploy(Migrations);

    deployer.deploy(MyContract); // Deploy a new instance of MyContract
};
```
Run Ganache, and migrate the contracts with

```
$ truffle migrate --reset --network development
```

`development` is the network defined in `truffle-config.js`. Actually, this is the default network, so `--network development` option can be omitted. `-- reset` is an option that says “replace all previously migrated contracts”. Without it, Truffle may not migrate contracts if it sees the “network up to date”
const MyContract = artifacts.require("MyContract");
contract("MyContract", accounts => {
  it("should test the correctness of the functions", () => {
    // MyContract is a contract artifact (ABI), not the instance itself.
    // We cannot call the functions on the artifact.
    // We need to retrieve the deployed instance first
    MyContract.deployed() // Retrieve the last instance of MyContract
      .then(instance => {
        // instance is the instance of MyContract
        instance.myFunction().then(result => {
          // result is the result of myFunction(): solidity's uint are BigNumber objects
          assert.equal(result.toNumber(), 8, "Result should be 8");
        });
      });
});
Truffle: dev phase, testing

Create mytest.js in test/ folder.

Execute all the tests inside test/ folder with:

```sh
git truffle test --network development
```

Execute one test inside test/:

```sh
git truffle test test/mytest.js --network development
```

(--network development can be omitted)
Truffle: dev phase, testing

Web3 calls do not return the result itself, but a Javascript Promise object. They are asynchronous calls to the Ethereum network. For this reason the code goes on until the Promise gets the result and executes the `.then(() => {// code})` statement.

This leads to long Promise chains. We can use the async / await syntax to lighten the code.
Truffle: dev phase, testing

```javascript
const MyContract = artifacts.require("MyContract");

contract("MyContract", accounts => {
  it("test the correctness of the functions", async () => {
    // async function
    // Retrieve the last instance of MyContract
    const instance = await MyContract.deployed(); // await
    const result = await instance.myFunction(); // await
    console.log(result);
  });
});
```
Smart contract return values

Functions which are labeled as view / pure return the value they are supposed from their return statement. They won’t be put in a block (mined) and they do not cost gas to the caller.

Functions which are not are so called transactions, meaning that they will be mined, they will be put in a block and they do cost gas. When called by the web3 their return value IS NOT the value in their return statement, but a transaction receipt.
Smart contract return values

How can I get the return value of a transaction?

Ideas

● Emit an event inside, with the result a parameter of the event
● Call a view function right after the transaction (gas free)

More information on

https://truffleframework.com/docs/truffle/getting-started/interacting-with-your-contracts
Truffle: more stuff

Truffle console

£ truffle console --network development

Truffle execute scripts

$ truffle exec ./path/script.js --network development
Truffle: docs

Config file: https://truffleframework.com/docs/truffle/reference/configuration

Compilation: https://truffleframework.com/docs/truffle/getting-started/compiling-contracts

Migration: https://truffleframework.com/docs/truffle/getting-started/running-migrations

Testing with Js: https://truffleframework.com/docs/truffle/testing/writing-tests-in-javascript
Part 2
Web DApp

A simple web application with help of Metamask. Deploy to the Ropsten network with Infura.
Walkthrough

- Introduce the second set of tools
- Setup the environment
  - Create a server for the DApp
  - Link the web3 libraries
- DApp initialization process
- Run the DApp
  - Connect it to the local network
  - Connect it to Ropsten
    - Setup Infura
Tools: Metamask

Metamask is an Ethereum wallet implemented as a browser extension.

It is possible to create accounts for free for different target networks: Ethereum main network or test networks such as Ropsten and Rinkeby.

It is also possible to import Ganache accounts if we test our DApp with a local blockchain.
Tools: Infura

Infura is a hosted Ethereum node cluster that lets users run your application without requiring them to set up their own Ethereum node or wallet.

Infura can be used to migrate a DApp to a supported Ethereum network.

It is necessary to register to the service and create a project which generates an ID and an API key together with the endpoint URL.
Create HTML project

Create a `src/` folder inside the Truffle folder. Inside create all the necessary folders such as `js/`, `css/` and `index.html`.

The web page needs to get the contract abstractions we have in our `build/` folder.

To do this, we need a running local server, serving our app. We can install it with NodeJs.
Lite server

$ npm install lite-server

Create its the configuration file inside the Truffle project root directory. bs-config.json

```
{
  "server": {
    "baseDir": ["./src", ".build/contracts"]
  }
}
```
There should be a `package.json` file for NodeJs. If not, create it with

```
$ npm init .
```

Insert “dev”: “lite-server” in “scripts”. `package.json` should look like this:

```json
"scripts": {
  "dev": "lite-server",
},
"dependencies": {
  "lite-server": "^2.4.0"
}
```
NodeJs

That's how we roll

DEPENDENCIES

DEPENDENCIES EVERYWHERE
Web3 and truffle-contract

As browser application, we need to import inside our index.html the scripts. We need the libraries web3 and truffle-contract

Latest version of Web3 is 1.0.0-beta.55, but I could find this minified source (v -beta.34). Be aware, many online tutorials use the old 0.2.x version, which has a different syntax from the version 1.x

Truffle-contract is an higher abstraction of web3, github repo
Web3 and truffle-contract

index.html

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<script src="path_to/web3.min.js"></script>
<script src="path_to/truffle-contract.js"></script>
<!-- Our App script -->
<script src="js/app.js"></script>
Example contract

```solidity
contract MyContract {

    uint public value;
    event click();

    constructor() public {
        value = 1;
    }

    function myFunction() public view returns (uint) {
        return value + 7;
    }

    function pressClick() public {
        emit click();
    }
}
```
DApp initialization

app.js script file

1. Init web3
2. Init smart contracts (read json files)
3. Activate event listeners
4. Render page (call smart contract functions useful for initialization)
App = {
    // Attributes
    init: function() { return App.initWeb3(); },
    // Functions
}

// Call init whenever the window loads
$(function() {
    $(window).on('load', function () {
        App.init();
    });
});
App = {
    contracts: {}, // Store contract abstractions
    web3Provider: null, // Web3 provider
    init: function() { return App.initWeb3(); },

    initWeb3: function() { /* initialize Web3 */ return App.initContract(); },

    initContract: function() { /* Upload the contract's */ return App.listenForEvents(); },

    listenForEvents: function() { /* Activate event listeners */ return App.render(); },

    render: function() { /* Render page */ }
}
app.js, 1. Init web3

```javascript
initWeb3: function() {
    if(typeof web3 !== 'undefined') { // Check whether exists a provider, e.g Metamask
        App.web3Provider = window.ethereum; // !! new standard, since 2/11/18
        web3 = new Web3(App.web3Provider);
        try { // Permission popup
            ethereum.enable().then(async() => { console.log("DApp connected"); });
        } catch(error) { console.log(error); }
    } else { // Otherwise, create a new local instance of Web3
        App.web3Provider = new Web3.providers.HttpProvider(App.url); // <=
        web3 = new Web3(App.web3Provider);
    }
    return App.initContract();
},
```
initContract: function() {

    // Load content's abstractions
    $.getJSON("MyContract.json").done(function(c) {
        App.contracts["MyContract"] = TruffleContract(c);
        App.contracts["MyContract"].setProvider(App.web3Provider);

        return App.listenForEvents();
    });
};
昕. Activate event listeners

```javascript
listenForEvents: function() {
    App.contracts["MyContract"].deployed().then(async (instance) => {
        web3.eth.getBlockNumber(function (error, block) {
            // click is the Solidity event
            instance.click().on('data', function (event) {
                $("#eventId").html("Event catched!");
                console.log("Event catched");
                console.log(event);
                // If event has parameters: event.returnValue.valueName
            });
        });
    });
    return App.render();
},
```
app.js, 3. Activate event listeners

Usually online we find solutions involving `event().watch(callback)`, but the `watch` function is not anymore supported by the most recent versions of web3Js.

Source, Github Issue

More on events:
https://web3js.readthedocs.io/en/1.0/web3-eth-contract.html#contract-events

https://ethereum.stackexchange.com/questions/64872/truffle-how-to-get-event
render: function() {

    // Retrieve contract instance
    App.contracts['MyContract'].deployed().then(async(instance) => {

        // Call the value function (value is a public attribute)
        const v = await instance.value();
        console.log(v);
        $('#valueId').html("" + v);
    });
},

app.js, 4. Render
DApp, test it

- Start lite server with
  
  $ npm run dev

- Don’t forget to run Ganache and to migrate the contracts
- Open browser, open Metamask and select “Private network”
- Import an account from Ganache by copying the private key and pasting it into Metamask
- If the accounts are already imported and previously used, it may need to reset them, otherwise their nonce conflicts with the new Ganache instance
(Development goes on...)
Migrate to real network

Migration to the Ropsten test network

We need to go back to truffle and look at the configuration file

```javascript
networks: {
  ropsten: {
    provider: () => new HDWalletProvider(mnemonic,
    `https://ropsten.infura.io/${infuraKey}`),
    network_id: 3,     // Ropsten's id
    gas: 5500000,     // Ropsten has a lower block limit than mainnet
  },
}
```
Setup Infura

- Install `truffle-hdwallet-provider` with npm
- Metamask mnemonic, 12 words phrase
  - Go to Metamask and copy them in variable in `truffle-config.js`
  - Or better, in a file that you can keep private. Such words are the seed to create all of your private keys, and thus anyone with them has access to all of your Eth
- An Infura API key
  - Go to Infura, register with email (for free) and create a Project. From the project, get an endpoint URL for Ropsten network. Also this URL should be kept secret.
Infura and truffle-config

How it should look like

```javascript
const HDWalletProvider = require('truffle-hdwallet-provider'); // install it with npm
const infuraKey = "fj4jll3k....."; // Keep it secret
const mnemonic = "bla1 bla2 bla3 ... bla11 bla12"; // Keep it secret

networks: {
  ropsten: {
    provider: () => new HDWalletProvider(mnemonic,
    'https://ropsten.infura.io/v3/${infuraKey}'),
    network_id: 3, // Ropsten's id
    gas: 5500000, // Ropsten has a lower block limit than mainnet
  },
}
```
Truffle, migrate to Ropsten

Migrate the contracts with

```
$ truffle migrate --network ropsten
```

*ropsten* is the network defined in *truffle-config.js*. Since Ropsten network is a real network (but using fake Ether, for testing), transactions here have to be properly mined. So the migration phase may take a few seconds (or a couple of minutes)
DApp, test it

- Start lite server with
  
  ```
  $ npm run dev
  ```

- Open browser, open Metamask and select “Ropsten network”
- In this case there is no need to import accounts or to reset them.
- But we need Ropsten Ether. We can get them from
  
  https://faucet.metamask.io/. It may take a minute to arrive (the transaction needs to be mined)

- Interact with the Ui
Docs and materials

Truffle and Infura: https://truffleframework.com/tutorials/using-infura-custom-provider


Dapp University
Dapp, full tutorial (it concentrates more on Solidity though, it goes fast on Javascript), 2018: https://www.youtube.com/watch?v=3681ZYbDSSk

Dapp, tutorial updates for 2019: https://www.youtube.com/watch?v=X6DzzeoRTS0
Code boilerplates

As said at the very beginning, Truffle provides DApp code boilerplates. Create a base DApp with Truffle boxes:

```
$ truffle unbox pet-shop
```

More boxes at: https://truffleframework.com/boxes

My personal DApp skeleton project, with HTML, Js:
https://github.com/0Alic/DAppSkeleton
ReactJs and Drizzle

Create a DApp with modern Web App frameworks

Truffle provides another tool, Drizzle, for frontend development with ReactJs [https://truffleframework.com/drizzle](https://truffleframework.com/drizzle)

I keep using truffle-contract also developing with ReactJs
Popular DApp: Cryptokitties