Integrate Ethereum and IPFS

With Javascript (2020)

Andrea Lisi, andrealisi.12lj@gmail.com

Decentralized data storage

Smart contracts cannot store big portions of data

- Each transaction costs a fee to the user
- The transaction size cannot exceed the gas limit of a block
  - 8M units of gas

The application is not anymore decentralized if it relies of centralized data centers

IPFS

IPFS (InterPlanetary File System) is a P2P network for the storage of the data in a decentralized way

- Each stored content is hashed and that hash is used to fetch it
  - Content-based addressing Vs Location-based addressing
- If at least a peer with a content C is online, every peer in the network is able to fetch it
  - Otherwise that content is not available

Part 5
Connect IPFS

Put together smart contracts and IPFS

Walkthrough

1. Init a Truffle project
    a. Create a smart contract and compile it
2. Install all the NodeJs requirements
    a. Truffle-contract
    b. Web3
    c. Ipfs-http-client
3. Combine smart contracts and IPFS

Example contract

```
pragma solidity 0.6.6;

contract IPFSContract {

   string public cid = "None";

   function setCid(string memory _cid) public {
       cid = _cid;
   }
}
```

Init a Truffle project

Initialize a Truffle project, code the contract and compile it so that its ABI and bytecode is in the *build/* folder

Install NodeJs requirements

We are going to install:

- truffle-contract
- web3
- ipfs-http-client

```
$ npm init
$ npm install --save @truffle/contract
$ npm install --save web3
$ npm install --save ipfs-http-client
```

Install NodeJs requirements

The file package.json should have these dependencies

```
"dependencies": {
  "@truffle/contract": "^4.2.1",
  "ipfs-http-client": "^44.0.0",
  "web3": "^1.2.6"
}
```

Coding

Create a file *interaction.js* (or any other name). This file is going to:

1. Import all the libraries
2. Instantiate the smart contract
    a. If the smart contract has already been deployed, fetch it
3. Connect to an IPFS node
    a. We are going to use a public gateway provided by Infura
4. Code the logic
    a. Post a text file onto IPFS, store the CID in the contract
    b. Use the stored CID to retrieve the file and print it

Coding, 1 - libraries

```
// Smart contracts
const contract = require("@truffle/contract");
const Web3 = require("web3");
const contract_data = require("./build/contracts/IPFSContract.json");

// IPFS
const ipfsClient = require("ipfs-http-client") ;
const {globSource} = ipfsClient;

const localhost = "http://127.0.0.1:8545";
const web3 = new Web3(localhost);
const provider = new Web3.providers.HttpProvider(localhost);
```

Coding, 2 - create smart contract

```
// Libraries
// ...

const IPFSContract = contract(contract_data);
IPFSContract.setProvider(provider);

async function main() {
   // Do everything
   // Terminate process
   process.exit();
}
main();
```

Coding, 2 - create smart contract

```
const IPFSContract = contract(contract_data);
IPFSContract.setProvider(provider);

async function main() {
   const accounts = await web3.eth.getAccounts();
   const alice = accounts[0];
   // Create the contract
   const instance = await IPFSContract.new({from: alice});
   console.log("Cid: " + await instance.cid());

}
main();
```

Coding, 2 - get a smart contract

```
const IPFSContract = contract(contract_data);
IPFSContract.setProvider(provider);

async function main() {
   const accounts = await web3.eth.getAccounts();
   const alice = accounts[0];
   // Get last deployed instance
   const instance = await IPFSContract.deployed();
   // Get the instance at the address contract_address
   const instance = await IPFSContract.at(contract_address);
}
main();
```

Coding, 3 - connect to IPFS

```
async function main() {
   // Create instance smart contract …
   // …

   // Connect IPFS
   // Connect to local node
   const ipfs = ipfsClient("localhost", "5001", {protocol: 'http'});

   // Connect to Infura with public gateway
   const ipfs = ipfsClient({host: "ipfs.infura.io", port: "5001", protocol: 'https'});
}
main();
```

Coding, 4.a - Store file on IPFS

```
async function main() {
   // ...
   // Read filename from command line
   const args = process.argv.slice(2);
   const file_name = args[0];
   let file_ipfs_data;
   for await (const f of ipfs.add(globSource(file_name))) {
       file_ipfs_data = f;
       console.log(file_ipfs_data);
   }
}
main();
```

Coding, 4.a - Store CID on ETH

```
async function main() {
   // ...
   // Store the CID in the smart contract
   const tx = await instance.setCid(file_ipfs_data.cid.toString(), {from: alice});
}
main();
```

Coding, 4.b - Get the CID from ETH

```
async function main() {
   // ...
   // Get the CID from the smart contract
   const cid = await instance.cid();
   console.log("Cid: " + cid);

}
main();
```

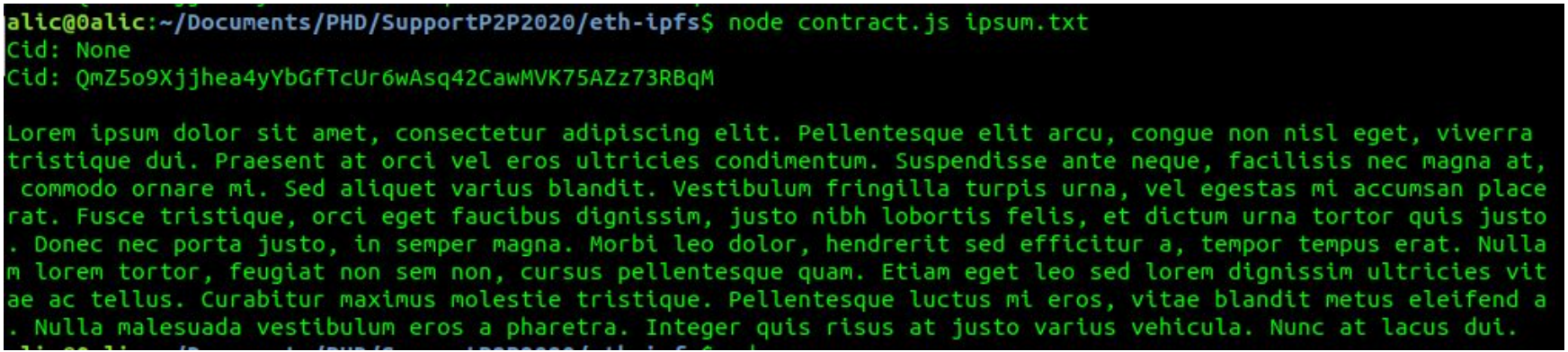
Coding, 4.b - Get the file from IPFS

```
async function main() {
   // ...
   // Retrieve the file
   const bytes = [];
   for await (const chunk of ipfs.cat(cid)) {
       bytes.push(chunk);
   }
   console.log(Buffer.concat(bytes).toString());
   // Terminate process
   process.exit();
}
main();
```

Result

Execute with

- `$ node interaction.js file_name`



Resources

IPFS: https://docs.ipfs.io/

Tutorial: https://www.youtube.com/watch?v=pTZVoqBUjvI

- Warning: the version of ipfs-http-client in that tutorial is older

for await … of in Javascript:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for-await...of