Smart Contracts on an Ethereum Blockchain EnCo, Proximus

Dhondt Maarten[1], Mottiat Frederic[2], and Marlair Julien[2]

[1]Blockchain Developer, Realdolmen
[2]Business Innovation & Development Manager, Proximus

February 2019

1 Smart Contracts

Proximus has developed smart contracts to monitor shipping containers. The smart contracts are written in *Solidity*, the most popular smart contract language that compiles to bytecode for the Ethereum Virtual Machine.

The idea is that a sensor (pressure, motion, ...) notices a change in the environment of a container and registers that occurrence on a blockchain. Each time a sensor is made to monitor a container, a new smart contract will be created that contains the state the of the container. The state of all containers will be visible by everyone with access to the blockchain; but changing the blockchain for a specific container will only be possible by the exact sensor that created the smart contract for that container.

The main contract is the `ContainerHub` contract. It allows to create new `Container` smart contracts. The `ContainerHub` also keeps track of which containers are linked to which sensors by using a container id. So one sensor can be reused to be linked to multiple container ids and thus multiple `Container` smart contracts.

```
pragma solidity ^0.4.24;

contract Container {

    uint256 _id;
    bool    _doorsOpen = false;
    bool    _movement = false;
    address _owner;

    event DoorsOpened();
    event DoorsClosed();

    constructor(uint256 id) public {
        _id = id;
        _owner = tx.origin;
    }

    modifier onlyOwner() {
        require(tx.origin == _owner);
        _;
    }

    function getId() public view returns (uint256 id) {
        return _id;
    }

    function areDoorsOpen() public view returns (bool doorsOpen) {
        return _doorsOpen;
    }

    function setDoorsOpen(bool doorsOpen) public onlyOwner {
        _doorsOpen = doorsOpen;
        if (doorsOpen) {
            emit DoorsOpened();
        } else {
            emit DoorsClosed();
        }
    }
    function movementDetected() public onlyOwner {
        _movement = true;
    }

    function hasMovementBeenDetected() public view returns (bool movementDetected) {
        return _movement;
    }
}
```

The address/wallet that calls the constructor of the `Container` is the owner of it. That's the only address that will be able to make changes to the state of the container. To make it easier to create and manage a `Container`, the `ContainerHub` can be used. That way containers can be identified by an integer instead of its Ethereum address.

```
pragma solidity ^0.4.24;

import "./Container.sol";

contract ContainerHub {

    mapping(address => uint256[]) containerIds;
    mapping(address => mapping (uint256 => Container)) containers;

    event ContainerCreated(address indexed owner, uint256 indexed id, address containerAddress);

    function createContainer(uint256 id) public {
        Container container = new Container(id);
        containers[msg.sender][id] = container;
        containerIds[msg.sender].push(id);
        emit ContainerCreated(msg.sender, id, container);
    }

    function getMyContainers() public view returns (uint256[] myIds, address[] myAddresses) {
        return getContainersFor(msg.sender);
    }

    function getContainersFor(address owner) public view returns (uint256[] myIds, address[] myAddresses) {
        uint256[] memory ids = containerIds[owner];
        address[] memory addresses = new address[](ids.length);
        for (uint i = 0; i < ids.length; i++) {
            addresses[i] = containers[owner][ids[i]];
        }
        return (ids, addresses);
    }

    function getMyContainer(uint256 id) public view returns (address myContainer) {
        return containers[msg.sender][id];
    }

    function setDoorsOpen(uint256 id, bool doors) public {
        Container(containers[msg.sender][id]).setDoorsOpen(doors);
    }

    function setDoorsOpen(address containerContract, bool doors) public {
        Container(containerContract).setDoorsOpen(doors);
    }

    function areDoorsOpen(uint256 id) public view returns (bool doorsOpen) {
        return Container(containers[msg.sender][id]).areDoorsOpen();
    }

    function areDoorsOpen(address containerContract) public view returns (bool doorsOpen) {
        return Container(containerContract).areDoorsOpen();
    }
}
```

2 EnCo Blockchain Management API

The `ContainerHub` which will create the `Container` contracts for you, has already been deployed to the EnCo blockchain called `enco_shared`. What's left is creating a wallet/address for a sensor to be able to invoke the functions of the `ContainerHub` contract. Because Ethereum blockchains need funds to operate, we will also need to request funds.

1. Creating a wallet

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –X POST https://
      api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/wallets?name=mySensor

{"name":"mySensor","address":"0x0c92784C35c52Cf278A36a2DF26D968f15D7f041"}
```

   This created a new wallet for you with name “mySensor”. It can be found on the blockchain at address `0x0c92784C35c52Cf278A36a2DF26D968f15D7f041`. In the next API calls you can use the name “mySensor” and the address `0x0c92784C35c52Cf278A36a2DF26D968f15D7f041` interchangeably.

2. Requesting funds for the wallet (can only be done once every 5 minutes, but the requested funds are more than enough for a while)

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –X GET https://
      api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/faucet/mySensor?amount=5

{"transactionHash":"0x5f22c08171f309c7b7b4577019d5178c7f490cc237cc0c3da330447b9b0777a3","fromAddress":"0
      x3Bd9400C0a27c2cB6D5cF32e376Af7Ef5Fe9C929","toAddress":"0x0c92784C35c52Cf278A36a2DF26D968f15D7f041","value
      ":5000000000000000000,"nonce":12,"gasLimit":21000,"gasPrice":22000000000}
```

   This shows a transaction of 5 000 000 000 000 000 000 WEI (= 5 ETHER) made from the `enco_shared` faucet towards your (new) wallet.

3 Invoking the Smart Contracts

1. Invoke the function `getMyContainers` on the `ContainerHub` contract to see all the containers linked to our (new) wallet.

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d "{}" –X POST
      https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/getMyContainers?walletIdentifier=
      mySensor

{"myIds":[],"myAddresses":[]}
```

   This (correctly) shows that our sensor does not have any linked containers yet.

2. Creating a container smart contract based on an integer id. For large transactions (such as the creation of a new `Container` smart contract), a higher gas limit is needed than the EnCo default of 200 000. That’s why we provide it with as a query param.

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d '{"id" :
      1991}' –X POST 'https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/createContainer?
      walletIdentifier=mySensor&gasLimit=500000'

{"transactionHash":"0x38211bd14ee2d6f9a70ad81a96bc280e87ba6abf6bc68034d25ac6f146586513"}
```

   The hash of the transaction which wrote (the new `Container` smart contract) to the blockchain is returned.

3. Reinvoking the `getMyContainers` function should now return the id and the respectively corresponding address of the container contract that was created.

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d "{}" –X POST
      https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/getMyContainers?walletIdentifier=
      mySensor

{"myIds":[1991],"myAddresses":["0x1984564c678a5ec0ee0c70b4d04a906dc9bf46e3"]}
```

Now we see that we have a `Container` contract at address `0x1984564c678a5ec0ee0c70b4d04a906dc9bf46e3` for container with id 1991. After creating a few more containers, the output could look like:

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d "{}" –X POST
      https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/getMyContainers?walletIdentifier=
      mySensor

{"myIds":[1991,5684,2563],"myAddresses":["0x1984564c678a5ec0ee0c70b4d04a906dc9bf46e3","0
      x311b0ca1c53d19406527a3190ef59f4fbdd8c8e8","0x4672d379e72158887f7592a072b1d6eb8f23ccc7"]}
```

4. Marking the doors of a container as opened:

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d '{"id": 1991,
       "doors": true}' –X POST https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/setDoorsOpen?
      walletIdentifier=mySensor

{"transactionHash":"0x8f2dd8e44d006edd09d8c1450ba84c016c6ff3cc18cdd74aa07e27c8cf21703c"}
```

5. Checking if the doors of a container are open:

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d '{"id":
      1991}' –X POST https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/areDoorsOpen?
      walletIdentifier=mySensor

{"doorsOpen":true}
```

6. Marking a container with a detected movement:

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d '{"id":
      1991}' –X POST https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/movementDetected?
      walletIdentifier=mySensor

{"transactionHash":"0xdc68b7a5e9bf42770125a8355776a7aafb88116d3e2ab6c7413b0635b6ea565c"}
```

7. Checking if a container was moved:

```
$ curl –sH "Authorization: Bearer aed6fdf1c14a94f86c12a8a55ec4351d" –H "Content–Type: application/json" –d '{"id":
      1991}' –X POST https://api.enco.io/bc–mgmt/1.0.0/chains/enco_shared/contracts/ContainerHub/hasMovementBeenDetected
      ?walletIdentifier=mySensor

{"movementDetected":true}
```

4 CloudEngine Script to Interact with Blockchain

```
import Debug;
object blockchain = create("Blockchain", "rinkeby");

function run(object data, object tags, string asset){

  int containerId = 1991;

  boolean createContainer = false;
  boolean getContainers = false;
  boolean checkDoorsOpen = true;
  boolean changeDoors = false;

  blockchain.loadWallet("sensorOwner");
  blockchain.loadContract("ContainerHub");

  if (createContainer) {
      string txHash = blockchain.invokeCustom("createContainer", [ containerId ], null, 500000, null);
      Debug.log("A container contract is being created with transaction hash " + txHash);
  }

  if (getContainers) {
      object containers = blockchain.invoke("getMyContainers", []);
      object ids = containers["myIds"];
      object addresses = containers["myAddresses"];

      foreach (id in ids) {
          Debug.log("A container contract for container " + ids[id] + " is deployed at " + addresses[id]);
```

```
        }
    }

    if (checkDoorsOpen) {
        object containers = blockchain.invoke("getMyContainers", []);
        object ids = containers["myIds"];
        boolean doorsOpen;
        foreach (id in ids) {
            doorsOpen = blockchain.invoke("areDoorsOpen", [ ids[id] ]);
            if (doorsOpen) {
                Debug.log("Doors for container " + ids[id] + " are open");
            } else {
                Debug.log("Doors for container " + ids[id] + " are closed");
            }
        }
    }

    if (changeDoors) {
        boolean doorsOpen = blockchain.invoke("areDoorsOpen", [ containerId ]);
        if (doorsOpen) { # close doors
            string txHash = blockchain.invoke("setDoorsOpen", [ containerId, false ]);
            Debug.log("Closing doors for container " + containerId + " in transaction " + txHash);
        } else { # open doors
            string txHash = blockchain.invoke("setDoorsOpen", [ containerId, true ]);
            Debug.log("Opening doors for container " + containerId + " in transaction " + txHash);
        }
    }

}
```