Ciber Experis BlockChain

A search for distributed wisdom

2017

Contents

[1. Preliminary 2](#_Toc479322939)

[2. Ethereum 3](#_Toc479322940)

[Architecture 3](#_Toc479322941)

[Tooling 4](#_Toc479322942)

[Networks 5](#_Toc479322943)

[Contract development 5](#_Toc479322944)

[Example contract 8](#_Toc479322945)

[Frontend development 8](#_Toc479322946)

[3. Hyperledger Fabric 10](#_Toc479322947)

# Preliminary

The focus of this project is on building decentralized applications on top of existing application platforms. It is not the intention to design and build core Blockchain protocols from the ground-up, but instead focus on developing software on top of this fundament.

Blockchain development platforms (and deployments in general) come in different variants. Either you deploy in public or in a consortium-private[[1]](#footnote-1) context. The prominent Blockchain application platforms available are Ethereum and Hyplerledger Fabric.

**Ethereum** is a public, open source (GPL) Blockchain application platform, allowing for the deployment of smart contracts which can be interacted with via native wallets and web-based applications and plugins. We will describe Ethereum in more detail in the chapter *Ethereum*.

**Hyperledger Fabric** is a consortium, open source (Apache 2.0) Blockchain application platform, allowing for the deployment of chaincode and the custom implementation of authorization concepts (certification). Hyperledger is explained in chapter *Hyperledger Fabric*.

In this document we will not describe how exactly these platforms work, neither will we explain the general concept of Blockchain principles.

# Ethereum

Ethereum is an open sourced Blockchain application platform. The advantage of using Ethereum is that the infrastructure requirements are covered, meaning the supporting P2P network is available for direct use, offering both a test and live/production network. Additionally, Ethereum benefits from a vibrant online community and a multiplicity of development tools which are freely available. Therefore, Ethereum is a good starting point for understanding Blockchain basics, allowing for a steep learning-curve due to network effects (contract code itself is ‘public’ as well). Downside to public deployments are transaction costs, and fixed capacity constraints in smart contract processing.

Smart contract related code is executed via the Ethereum Virtual Machine (EVM) running on network nodes, at a limited price (i.e. ‘gas’) which is paid for in ‘Wei’, the smallest unit of measure for the Ether cryptocurrency—see section Units.

Microsoft is sponsoring Ethereum.

## Architecture

The architecture of any Ethereum application is split over at least two layers: a contract that runs on the Blockchain, and a website that is able to communicate with the contract. The vast majority of Ethereum apps (named Dapps: distributed apps) also rely on the functionality of the Chrome plugin *Metamask*. A breakdown of a common Dapp architecture is shown in Figure 1.

architecture-physical (1)

Figure 1 Common Ethereum Dapp architecture

## Tooling

To develop Dapps on Ethereum, different tools and networks are available. Note that what is described here is limited to the tools used during this project.

#### Ethereum Studio

A web-based IDE provided by Cloud9 (Microsoft Azure) and hosted by ether.camp. If offers a virtual Linux Ubuntu workspace for developing complete Dapps. It features a sandbox environment for testing smart contracts, a local HTTP server for testing web-based applications, a Solidity debugger, an EVM compiler, network integration via RPC URL to directly deploy contracts to either test or live networks and a CLI supporting GIT[[2]](#footnote-2) integration. Because it is free and comprehensive, Ethereum Studio was our initial development environment of choice.

Signup to ether.camp is required and can be done [here](https://live.ether.camp/).

After starting the sandbox, you can click ‘contract’ to determine the contract ABI. The ABI should be used for the application and must be updated every time variables are declared ‘public’ or when a function is adapted—see section Contract development to see what this actually means. The local http server can be found at [https://[username].by.ether.camp:8080/.](https://[username].by.ether.camp:8080/)

#### Remix

A web-based IDE with more limited functionality than Ethereum Studio, but has the advantage of integrating with the Metamask Plugin and by directly interacting with smart contracts on test or live networks.

For example, when specifying a contract address, you can call ‘public’ variables to have their value returned from the IDE itself. When performing a function on a contract, Metamask notifications are automatically triggered.

Remix was used when Ethereum Studio could no longer be used for sending contracts to the test network.

#### Mist Wallet and Ethereum Wallet

Both wallets are software for connecting to the Ethereum networks, manage accounts, transmitting Ether via public keys, creating contracts and calling functions in contracts. The Mist Wallet additionally features a browser supporting web3 services, allowing for online Ether payments for web applications enabled for web3. This is an alternative to the Metamask plugin. The remix IDE is embedded into the Mist wallet.

#### Metamask

A freely available chrome plugin that injects web3 services into the browser, allowing for transactions to be signed from Chrome. Whenever a web3 transaction is caught, a Metamask notification is prompted requiring your approval.

#### Etherscan.io

A website for tracking transactions, contracts and their statuses on the differently available networks. It provides a log of transactions per contract. live.ether.camp allows for the same, but limited to the main network and proves less reliable. Pro for live.ether.camp is that you can request free Ether, which will then send 5 Ether to your account so that this may be used in testing.

#### https://www.4byte.directory/signatures/

Aa useful website for determining contract function ID’s and data types. When invoking contract functions, a function ID should be specified (e.g.: 0xdce293a7) along with the payload.

## DTAP

C:\Users\stijn.koopal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\otap.png

Figure 2 DTAP for Dapps

Any application that is worth mentioning has something called a DTAP street: Development-Test-Acceptance-Production. For DApps something similar is in place—see Figure 2.

Development and testing can be done on a Sandbox environment. This fully configurable environment can be started and stopped on request. A sandbox is provided by Ethereum Camp (<https://live.ether.camp/eth-studio>). Any exchange with the Sandbox is instant. That is, every action you do a contract is executed immediately, in contrast to BlockChain where actions are mined.

Multiple Ethereum test environments are available. On test environments it is possible to acquire as much Ether as you want, but the environments are not configurable. They look a lot like the Main network, but it does not operate on real Ether.

## Networks

Test networks (RPC URL):

* https://ropsten.infura.io/
* <https://ropsten.infura.io/metamask>

## Contract development

Smart contracts executed on the Ethereum Blockchain are written in Solidity: a contract-oriented programming language.

Solidity documentation: <http://solidity.readthedocs.io/en/develop/index.html>

Solidity can be used to program a smart contract. This contract can be deployed to the Ethereum blockchain and can then be interacted with. The deployment of a contract will cost gas. Interacting with a contract can also cost gas. Gas is an internal pricing mechanism used to prevent spam on the network and as an incentive to miners. Ether is the “value token” of the Ethereum network; it is a cryptocurrency and can be transferred between participants and contracts. Contract files have a “.sol” extension.

### Functions

Solidity is a function based programming language. Functions programmed in a contract can be called. The scope of these functions can be specified as external, public (default), internal or private. Everything inside a contract is always visible for external observers. Defining a function as private for example does not change this.

The *constructor* function has to be declared with the same name as the contract name. This function is only executed on deployment of the contract to the Ethereum network (can be useful for example to set the owner of the contract).

The *payable* keyword in a function definition enables this function to send/receive ether.

Function can have *modifiers* in their function definition to easily change the behavior of functions, for example to check a condition before executing the function (preconditions).

### Storage

Solidity is a strongly typed programming language. This means variables have to be declared with a type. Variables in solidity can also have different scopes. Declaring a variable inside a function will make then local for the function. Declaring variables at the top of your contract will make them globally available in your contract. When adding the *public* addition to these global variables, it will make them readable from the outside.

### Types

When variable type “var” is used, the type of the variable will be chosen based on the content given to it. Available types in solidity are:

* Int/Uint
* Bytes32
* Address
* Bool

All the normal arithmetic operators can be used when working with integer types.

All the normal comparison operators can be used when working with variables.

A structure is declared using “struct” and an array can be created using a mapping of this structure.

### Events

Events can be triggered in a contract and allow the front-end to react on this. They have to be declared at the top of the contract and can be triggered in a function.

### Global variables

A smart contract has a few global variables always available. These variables are:

* Block (information about the current block)
* This (information about the contract)
* Msg (information about the current message)
* Now (current block timestamp)
* Tx (information about the transaction)

### Units

Ether is the main currency of the Ethereum network, but currency inside contracts is measured in Wei. This is a much smaller unit and converts to ether in the following way:

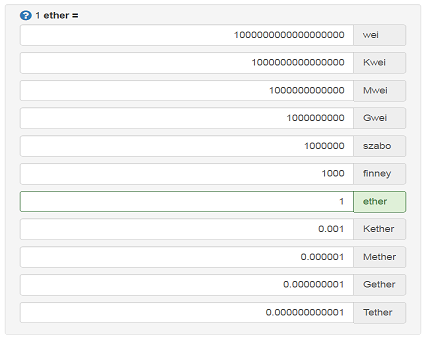


Figure 3 Conversion of Ether to it's subcurrencies

### Oracles

Smart contracts live like in a walled garden they cannot fetch external data on their own. Oraclize is there to help since it acts as a data carrier, a reliable connection between Web APIs and the contract. Oraclize main goal as a company is to provide a way for smart contracts to break free of their constrains and provide them the ability to access all the data they need from the web, without compromising their trustless nature. Currently, the following data sources can be called using Oraclize:

* URL (any public web API or webpage on the internet)
* WolframAlpha
* Blockchain (blockchain related data like bitcoin difficulty, litecoin hashrate)

<http://www.oraclize.it>

### Constraints

Now that we have looked at the basics of the solidity programming language and some example code, we have also experienced some constraints and technical difficulties along the way.

Deterministic approach

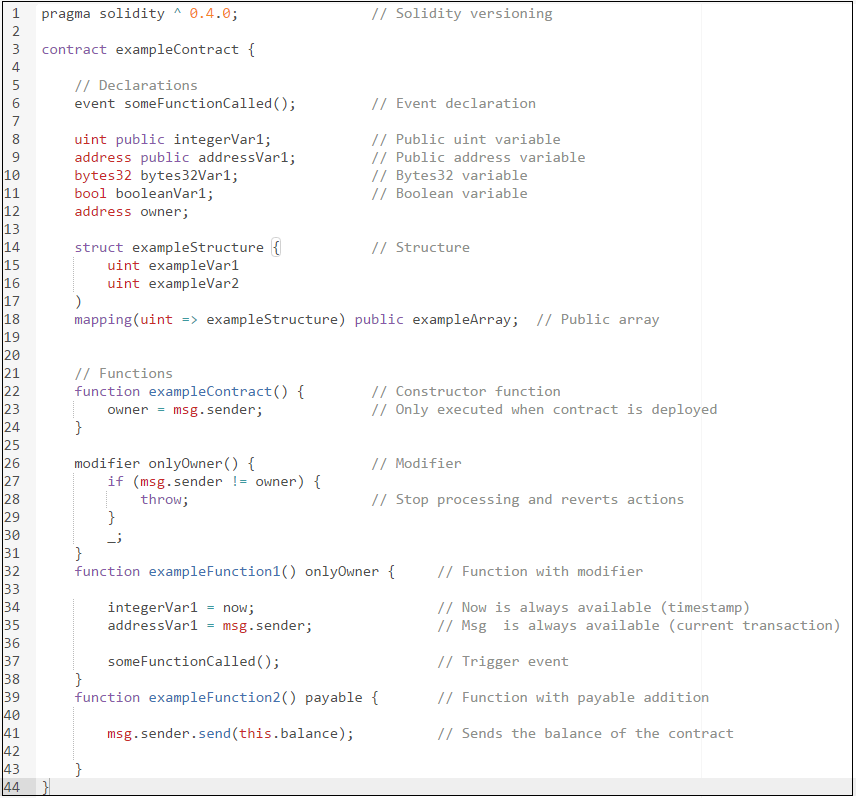
A solidity contract should always be deterministic. It should always give the same result based on the same input. This is needed because when a transaction is validated by different nodes, it should always return the same result. If this is not the case, consensus cannot be possible and the transaction will not be processed.

A random number is easy in any other environment, but when dealing with the blockchain this becomes a notable problem. For any real randomization, the use of an oracle is probably

Gas costs

Based on the actions executed in a contract, gas costs will become greater. This forces contracts to be as simple as possible. A developer should be very careful around loops since these can make gas costs explode.

### Example contract



## 

## Frontend development

Due to the technical nature of the Ethereum BlockChain, interacting with a contract directly is cumbersome, user unfriendly, time consuming and error prone. Hence, every contract should have a web application in place that is able to translate the technicalities of the chain and contract into a user friendly experience.

The interaction with any node in the Ethereum network is based on Json-RPC over HTTP (<http://www.jsonrpc.org/specification>). This protocol is based on the lightweight JSON format and it’s used to remotely invoke procedures on the exposing node, hence the name Remote Procedure Call (RPC). The underlying JSON format is of little importance, however the JavaScript Ethereum implementation of JSON-RPC, named *web3*, is. In the following section we will describe *web3* in more detail.

Metamask is a plugin that is able to communicate with any website that uses *web3*. How this communication is facilitated is explain the following sections.

### Web3

This is the Ethereum compatible JavaScript API which implements the JSON-RPC spec. It comprises library functions to get Ethereum node specifications, network details and contract details. Furthermore, it provides means to invoke contract functions and utility functions.

The web3 library is packaged into the web application, and it is therefore served with every request to the website.

### Metamask

Metamask (<https://metamask.io/>) is a Chrome plugin that injects Ethereum authentication into the browser. That is, for every website that uses web3, Metamask will inject details of the authenticated Ethereum user to the website.

The injection principle provided by Metamask is very simple. Once installed, it will provide a *web3* object into the *window* object of the browser. Something similar to this will be seen in the source code of the website:

if (typeof web3 !== 'undefined') {   
 window.web3 = new Web3(web3.currentProvider);  
 window.web3.eth.defaultAccount = web3.eth.accounts[0];  
}

### Javascript design principles

#### ES2016

Vanilla JavaScript currently supported by all major browsers currently is ECMAScript 5. In order to work with Promises, lambda’s are essential. For this reason, our JavaScript is transpiled from ECMAScript 2016.

#### Promises

The *web3* API allows for two programming models: synchronously, and asynchronously. While the former one provides easier access to the Ethereum Network, it will freeze any application due to the asynchronous nature of the blockchain. Hence, the asynchronous programming model provides the best fit for dealing with the blockchain.

The asynchronous API provided by *web3* is based on callbacks and therefore provides a suboptimal solution. (Why you say? <http://callbackhell.com/>). We chose to add some syntactic sugar to the API in order to support Promises.

#### Solidity Events

Any state change to a contract can be distributed to connected applications by means of Events. The contract can throw events that can be catched by the application. We use this principle in order to refresh frontends whenever the contract state is changed.

# Hyperledger Fabric

**Hyperledger Fabric** is a consortium, open source (Apache 2.0) Blockchain application platform, allowing for the deployment of chaincode, custom implementation of authorization concepts (certification) and consists of a modular architecture allowing for pluggable components (e.g.: consensus, validation protocols). Considering that the currently available consensus protocols are foremost based on PBFT, this application platforms is predominantly oriented towards small-sized network deployments (i.e. low node scalability).

Since that Fabric is private, no infrastructure is readily available. This means that nodes must be setup by yourself. Guides are available for this purpose. IBM also offers production-ready Blockchain services via Bluemix, providing a Blockchain network of nodes for test purposes.

Chaincode is executed in virtual Docker containers via Docker images created for a given operating system running on a Blockchain node. Since it is private, Fabric does not feature a native cryptocurrency nor does it expose hard restrictions on operations. A consideration for private deployments is that, unlike public deployment, anonymity is not feasible since some form of administration is required to participate. Currently, performance (tps, latency) in consortium chains is far better than public, due to its expensive PoW consensus algorithm.

IBM and Intel are sponsoring the Hyperledger foundation and its projects.

1. *Although completely privately owned, intra-organizational Blockchain deployments are theoretically conceivable, we do not consider this a practical deployment scenario which appreciates the key Blockchain capabilities. Corsortium deployment refers to an interacting network of participating entities. Therefore any reference to private applications in this document refers to consortium deployments.* [↑](#footnote-ref-1)
2. We used Github as our hosting provider. Github is a collaborative, distributed version control system for projects. Advantage of using Github in combination with Ethereum Studio is that you can perform pull and push requests form the bash CLI. The Github repository used for this project can be found [here](https://github.com/ExperisCiber/CiberBlockchain). [↑](#footnote-ref-2)