# **Meeting Agenda**

**Facilitator:** Fortiss

**Attendees:** Arnd, Cristian

**Objective: Tech presentations**

**Date:** May 10, 2019

## **MEETING AGENDA ITEMS**

## **Hyperledger Fabrics**

* **Description:**  Tech presentation on Hyperledger
* **Notes:**
* Hyperledger used for parties which don't trust each other
* Only Endorsers run chain code but always also have a ledger
* Chaincode -> only way to interact with blockchain
* Channel -> private connection between organisations
* Not every peer needs the code; enough if code is in the channel
* Instantiate chaincode "-p" -> policy
* **Consensus:**
* Client application -> Endosser -> confirm
* Client application -> Ordering Service -> Order -> Create Block
* Ordering Service -> broadcastet block to all peers -> Check if Interactions is signed by Endorsers
* State fine -> update state; State not fine -> captured in lock but no state update
* Notify client application with result
* Same channel => same state => same chain code
* One peer could be in different channels -> two states, two ledgers, etc.
* usually chain code in one file

Build consensus, you need to have a procedure, how the data is added to the blockchain;

### **Hyperledger Projects**

* Indy: ID management
* Explorer: Get info from Blockchain easier
* The one used for our projects is Fabric, the others are more or less in alpha state;

### **MSP (Membership Service Provider)**

* Take the keys and create identity for you
* Consortium network: each org has his MSP which identifies the peers of its organization

### **Ordering Service**

* One instance of the service with multiple peers;
* Solo: e
* nough for the project;
* Kafka: Apache, Public-Subscribe Broadcast Library: Peers subscribe to transaction, the protocol updates the peer with information; (nice-to -have);

### **Peers**

* Endorsers: executes code and simulate transaction, (In Etherum: all nodes executes smart contracts);
* Committing peer: receive transaction from Endorsers, validate transactions and update the chain ;

### **RW: Read-Write Set**

* all values that needs to be read (contains versions of the values);
* the version number must be the same for a transaction to be validated;

### **Chaincode**

* A program like a smart contract;
* Only way to interact with the blockchain (state);
* Run the code in endorsers container; code isolated from other peers;
* Endorsment policy: who is endorser;
* Every chain code has an endorsement policy
* Org1.member : endorsers of org1 are allowed to use the code;

### **E -> RW**

* Read SetEndorser receive a(5, version 100) increase 5 to 10
* Write Set: Endorser: a(10, version 101), version not necessary incremental!?!
* If the version is 100 but has different value => Manipulation => Validation fails;

### **Transaction Flow**

* The simulation of RW sets are sent back the application being sent and all results are sent by the application to the ordering service;
* Ordering service sends block to the peers, these validate the transaction and verify if the endorsement policy is respected;
* If transaction are invalid: transaction are logged, but not submitted to the chain.
* Transaction, states cannot be accessed from peers that are not part of the channel;

### **Private data collection**

* Only some of the members should be able to access it: see policy;
* If some of those peers fails, there should be a way to recover data. From other authorized peers: define requiredPeerCount to ensure the data redundancy;
* blockToLive: how many blocks does the info remains available until deleted;

### **Choose how to store private data**

* More private data exchanged, channels are more suitable
* More public data and some private data, private data collections are more suitable;

### **Development tips**

* Fabric initially supported only GO => Genuine first language supported;
* Chain code is usually in only one file : not usually to split chain code in different files;

## **BigchainDB**

* Description: Tech presentation on BigchainDB
* Notes:
* Transaction: Asset (could not be changed), Metadata (can be changed);
* Use Case: Shared DB between authorities to retrieve some kind of data or documents between them;

### **Architecture:**

* each Node has MongoDB instance -> stores the data;
* BigChainDB-Server: manages the mongoDB connection;
* each Node has Tendermint for Broadcasting the information, protocol and synchronize to all the peers in the network;

## **IPFS**

* encyption necessary, because every one who has the Hash can see the content
* Hash is based on content, not on Filename, etc.
* Same content different Filename -> same Hash
* "ipfs pin"-Command to ensure that the file won't be removed from the network
* IPFS API => JS Library for accessing IPFS;

### **Usual Commands**

ipfs init

* ipfs add *filename* provides hash;
* ifs get *hash* => gets the file with hash from ipfs;
* ipfs cat *hash* => calls cat for file content with the given hash;
* ipfs pin add *hash*: keep file available in the network, the file will not be removed from ipfs file system;
* ipfs pin rm *hash*: unpin file with hash *hash*;
* ipfs repo gc: removes temporary files;

Filename is irrelevant, only content is important and gets hashed => two identical files have same hash;

Ipfs daemon => Joins a network => provides also bootstrapping nodes => local files are now available for other nodes;

## **Ethereum**

### **General**

* public permission less network
* everybody can join as a miner -> permission less
* public -> every can see everything
* each miner is running Ethereum VM;
* Metamask: Gateway to Ethereum Blockchain;
* **Drizzle**: Retrieve smart contract data in front-end; Tell name of contract and abi and where is deployed; Drizzle can be used easily with React;
* **Truffle**: Deploy Smart Contracts;

### **Smart Contracts**

* remix.ethereum.org to test contracts; Can not connect to blockchain -> needs Metamask;
* Hub and spoke topology: you have store smart contract and spoke smart contracts; a way of adapting smart contracts;

### **MidTerm**

* **No Coding** until than;
* It’s important to have a clear defined process with use cases for Blockchain usage and Stakeholder Maps;
* Come with team to develop @ Fortiss to one of the appointments afterwards;