cpb

business as a service

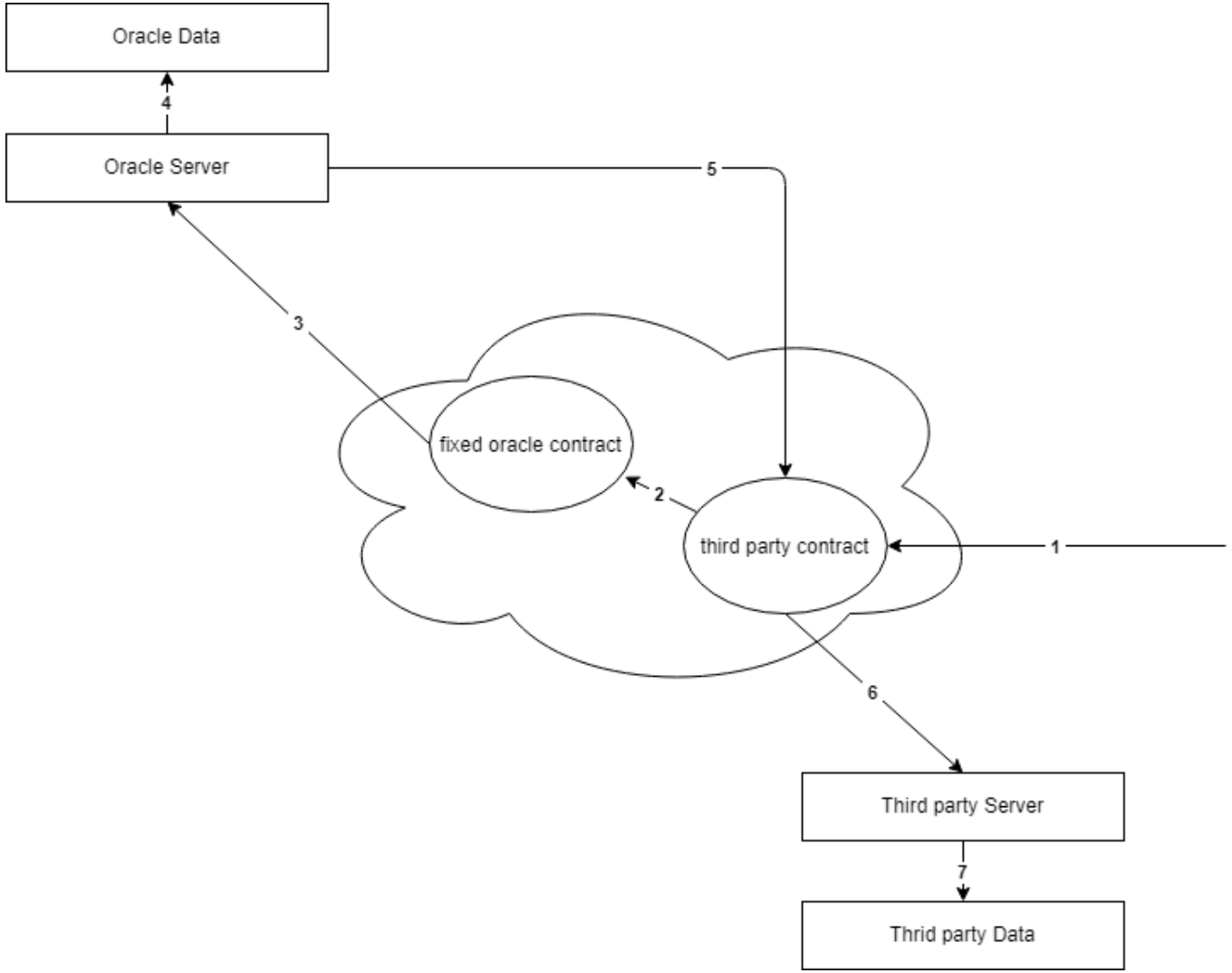
Smartcontract-Oracle

04.09.2019

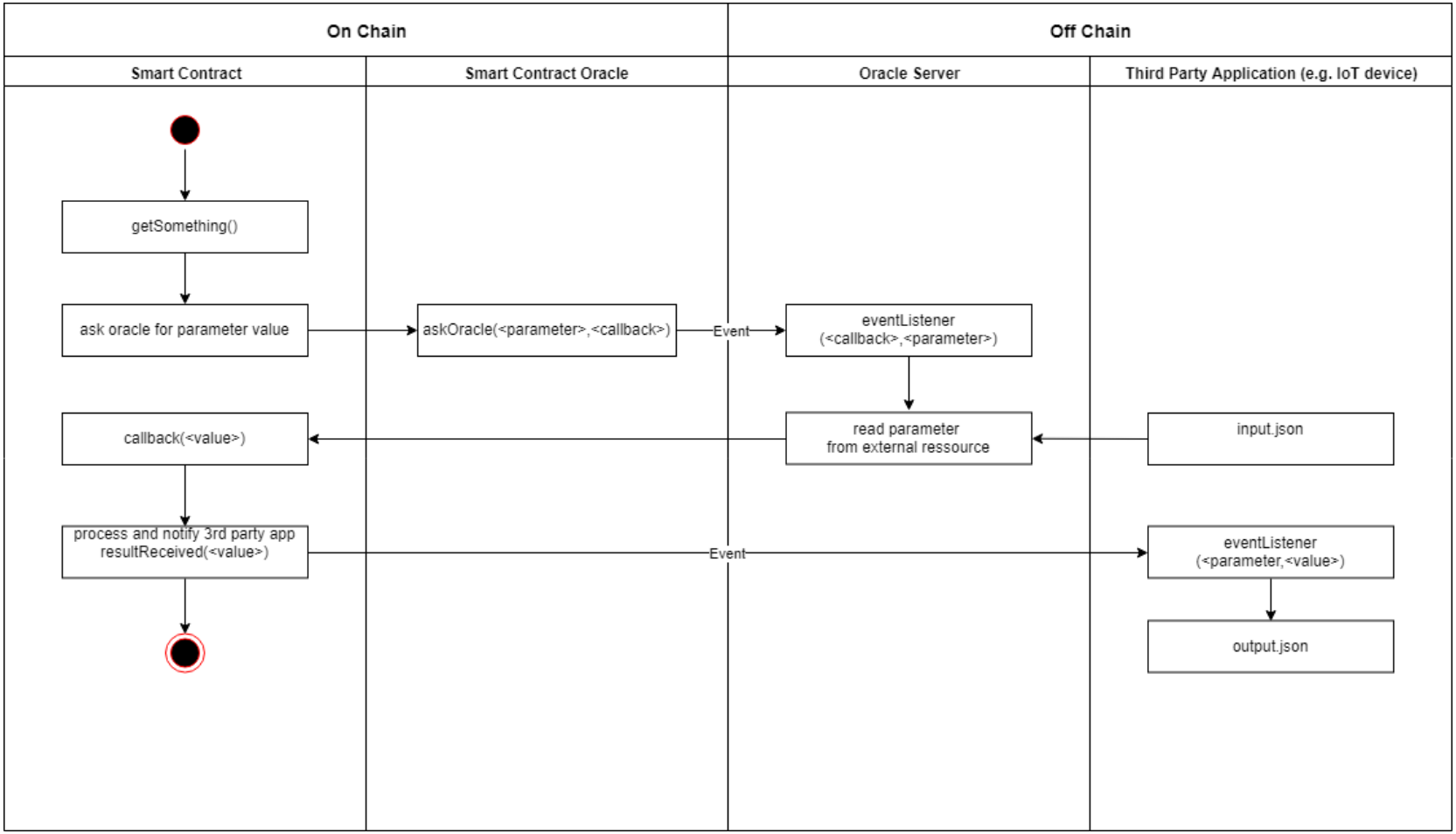
2 Aufgabenstellung

- „Smartcontract-Oracle“
  - Oracles, bidirektional
  - Daten in/aus Json Files
    - Sensor liefert Daten
    - Oracle indiziert Daten in Blockchain
    - Smartcontract regiert auf Daten
    - Output Json File

3 Lösungskonzept



4 Lösungskonzept



5 Lösungskonzept- Voraussetzung

- *OpenJDK 11* – Das Java Development Kit, das für den Server benötigt wird.
- *Eclipse IDE* – Eine Java Entwicklungsumgebung.
- *Maven* – Ein Paketmanager für Java Libraries.
- *Spring* – Ein Java Framework das einen Server beinhaltet und Schnittstellen zur Kommunikation mit Clients anbietet.
- *Web3j* – Eine Java Library die das Kommunizieren mit der Ethereum Chain ermöglicht.
- *Web3j CLI* – Ein Command Line Interface mit dem aus Smart Contracts Java Code generiert werden kann. Außerdem kann mit diesem Tool auch eine Wallet generiert werden.
- *Solidity Compiler* – Der Compiler für in Solidity geschriebene Smart Contract.
- *Parity Ethereum Client* – Ein Client mit dem eine private Ethereum Chain aufgesetzt werden kann.

6 Lösungskonzept- Setup Parity

- **Parity Installation**
  `bash <(curl https://get.parity.io -L) -r stable`
- Nach der Installation eine private Proof of Authority (POA) Chain aufsetzen analog zu Tutorial
- **Zusammengefasst:**

1. Chain Spezifikation erstellen
2. Config für die beiden Nodes (node0.toml und node1.toml)
3. Via RPC Authority Accounts erstellen
4. Nodes miteinander verbinden

7 Lösungskonzept- Setup Parity

- Beispiel der Config von Node0 AuthNode0.toml:

```
[parity]
chain = "oracle-spec.json"
base_path = "/myOracle/parity0"
[network]
port = 30300
[rpc]
port = 8540
apis = ["web3", "eth", "net", "personal", "parity", "parity_set", "traces", "rpc", "parity_accounts"]
[websockets]
port = 8450
```

- Beispiel der Config von Node0 AuthNode0.toml:

```
[parity]
chain = "oracle-spec.json"
base_path = "/myOracle/parity1"
[network]
port = 30301
[rpc]
port = 8541
apis = ["web3", "eth", "net", "personal", "parity", "parity_set", "traces", "rpc", "parity_accounts"]
[websockets]
port = 8451
[ipc]
disable = true
```

- Als nächstes wird die erste Node gestartet um User zu erstellen:
- parity --config authNode0.toml

- Ein engine Signer für das POA Netzwerk kann mit folgendem Befehl erstellt werden:
- curl --data '{"jsonrpc":"2.0","method":"parity_newAccountFromPhrase","params":["node0", "test"],"id":0}' -H "Content-Type: application/json" -X POST localhost:8540
- Response: {"jsonrpc":"2.0","result":"0x00bd138abd70e2f00903268f3db08f2d25677c9e","id":0}
- Anschließend werden zwei Test User erstellt mit:
- curl --data '{"jsonrpc":"2.0","method":"parity_newAccountFromPhrase","params":["user1", "test"],"id":0}' -H "Content-Type: application/json" -X POST localhost:8540
-
- Response: {"jsonrpc":"2.0","result":"0x00d695cd9b0ff4edc8ce55b493aec495b597e235","id":0}
- curl --data '{"jsonrpc":"2.0","method":"parity_newAccountFromPhrase","params":["user2", "test"],"id":0}' -H "Content-Type: application/json" -X POST localhost:8540
- Response: {"jsonrpc":"2.0","result":"0x001ca0bb54fcc1d736ccd820f14316dedaafd772","id":0}
- Danach wird Node0 gestoppt und Node1 gestartet um auch dort einen Engine Signer zu erstellen:
- parity --config authNode1.toml
- curl --data '{"jsonrpc":"2.0","method":"parity_newAccountFromPhrase","params":["node1", "test"],"id":0}' -H "Content-Type: application/json" -X POST localhost:8541
- Response: {"jsonrpc":"2.0","result":"0x00aa39d30f0d20ff03a22ccfc30b7efbfca597c2","id":0}

Lösungskonzept- Wallet erstellen

- **Web3j Installation**
  via brew:

```
brew tap web3j/web3j
brew install web3j
```

  via zip File:

```
unzip web3j-<version>.zip
```

- **Wallet erstellen**

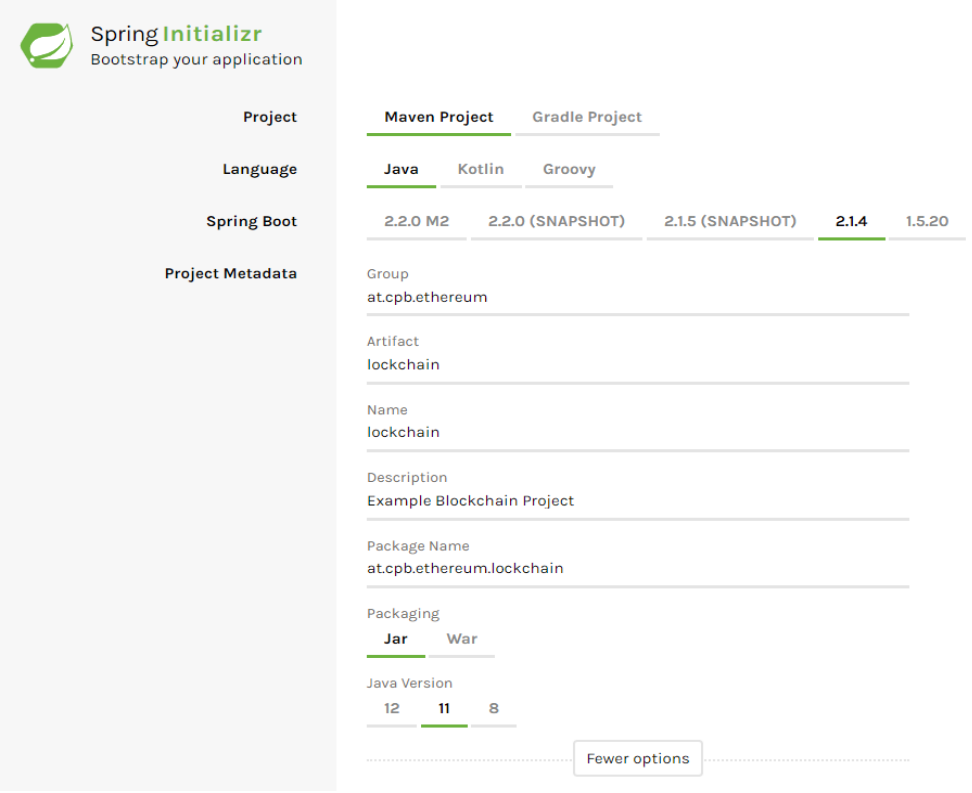
```
web3j wallet create
```

- **Wallet zu Chain hinzufügen**

```
curl --data
'{"method":"parity_newAccountFromWallet",
     "params":["{
          \"id\": \"9c62e86b-3cf9...\",...}",
          "PASSWORD"]
     ,"id":1,"jsonrpc":"2.0"}'
-H "Content-Type: application/json" -X POST
localhost:8540
```

Lösungskonzept- Server

- **Installation von Java**
  Im Terminals mittels dem Befehl: sudo apt install openjdk-11-jre-headless
- **Installation von Eclipse**
  Eclipse Installer ausführen
- **Spring Projekt**
  Das Projekt kann via https://start.spring.io/ generiert werden
- Das erstellte Projekt in Eclipse als Maven Projekt importieren



Lösungskonzept- Server

- Die benötigten Libraries im POM File hinzufügen

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
    </dependency>
    <dependency>
        <groupId>io.projectreactor</groupId>
        <artifactId>reactor-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.web3j</groupId>
        <artifactId>core</artifactId>
        <version>4.2.0</version>
    </dependency>
</dependencies>
```

Lösungskonzept- Smart Contract

```
pragma solidity 0.5.1;


contract MyOracle {
    //address of owner
    address payable owner;

    //whoever deploys contract is the owner
    constructor() public{
        owner = msg.sender;
    }

    //check if caller is owner -> only owner can access function
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    event QueryParamEvent(address indexed caller, bytes32 indexed queryParam);
    event SetParamEvent(address indexed caller, bytes32 indexed param,  bytes32 indexed val);

    //function to ask Oracle for status of a param
    function queryOracle(bytes32 __queryParam, address __caller) external payable  {
        emit QueryParamEvent(__caller, __queryParam);
    }

    function setParamInOracle(bytes32 __param, bytes32 __val, address __caller) external payable  {
            emit SetParamEvent(__caller, __param, __val);
    }

}
```

Lösungskonzept- Smart Contract – Oracle Teil1

```
pragma solidity 0.5.1;

contract OracleInterface {

    function queryOracle(bytes32 __queryParam, address __caller) public;
    function setParamInOracle(bytes32 __param, bytes32 __val, address __caller) public;

}

contract usingMyOracle {

    //function to ask Oracle for status of a param
    function askOracle(bytes32 _queryParam, address _caller)  internal {
        //TODO set the address of the oracle
        OracleInterface oracle = OracleInterface(0x20055c789636d220DfE78B958d0F4cCfbaA161A9);
        oracle.queryOracle(_queryParam, _caller);
    }

    function setParamInOracle(bytes32 _param, bytes32 _val, address _caller) internal {
        //TODO set the address of the oracle
        OracleInterface oracle = OracleInterface(0x20055c789636d220DfE78B958d0F4cCfbaA161A9);
        oracle.setParamInOracle(_param, _val, _caller);
    }
}
```

Lösungskonzept- Smart Contract – Oracle Teil2

```
contract ExampleContract is usingMyOracle {

    //address of owner
    address payable owner;

    bytes32 hot = 0x484f540000000000000000000000000000000000000000000000000000000000;
    bytes32 cold = 0x434f4c4400000000000000000000000000000000000000000000000000000000;

    //whoever deploys contract is the owner
    constructor() public{
        owner = msg.sender;
    }

    event ResultReceived(bytes32  resp);

    function __callback(bytes32 _result) external {
        if(_result == hot){
            emit ResultReceived(cold);
        } else if (_result == cold){
            emit ResultReceived(hot);
        } else {
            emit ResultReceived( _result);
        }
    }

    function getSomething(bytes32 query) external {
        askOracle(query, address(this));
    }

    function setSomething(bytes32 param, bytes32 val) external {
        setParamInOracle(param, val, address(this));
    }

}
```

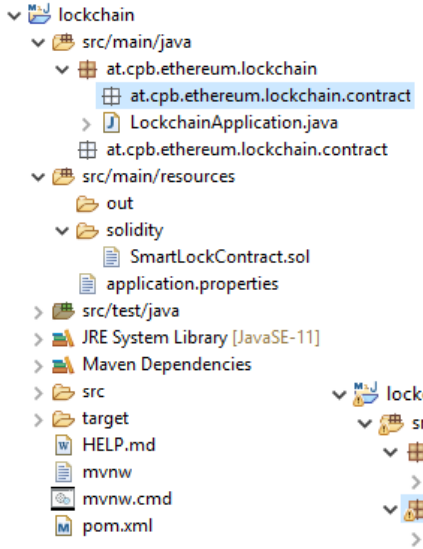
Lösungskonzept- Server

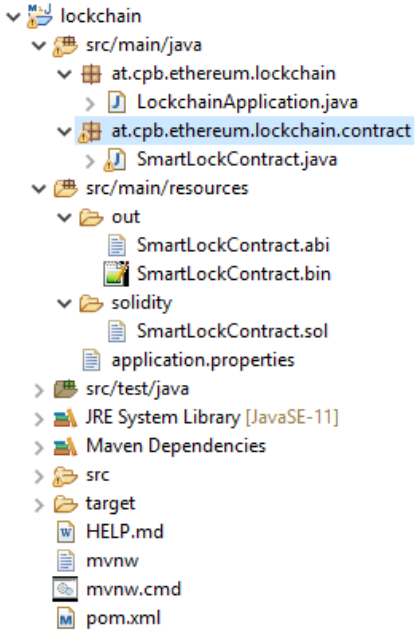
- Projektstruktur anpassen und Smart Contract hinzufügen
- Via Web3j aus dem Contract eine abi und bin Datei generien

```
solc <Pfad zu Contract> --bin
--abi –optimize –o <Output Pfad>
```

- Daraus kann nun eine Java Datei generiert werden

```
web3j solidity generate –b <Pfad
zu binary File> –a <Pfad zu abi
File> –o <Pfad zu Java Ordner> –p
<Package name>
```





Lösungskonzept- Server

- Verbinden von Server mit Parity Chain via Web3j Library
  - Verbindung aufbauen

```
Web3j web3j = Admin.build(new
            HttpService("http://localhost:8540/"));
```

  - Verbindung testen

```
Web3ClientVersion version = web3j.web3ClientVersion().send();
```

Vielen Dank für Ihr Interesse.

CPB SOFTWARE AG
Campus Viertel Zwei
Vorgartenstraße 206c,
1020 Wien, Österreich
T: +43 1 42701 0
E: office@cpb-software.com

CPB SOFTWARE (AUSTRIA) GMBH
Campus Viertel Zwei
Vorgartenstraße 206c,
1020 Wien, Österreich
T: +43 1 42701 0
E: office@cpb-software.com

CPB SOFTWARE (GERMANY) GMBH
Im Bruch 3,
63897 Miltenberg, Deutschland
T: +49 9371 9786 0
E: germany@cpb-software.com