Baue deine eigene

BLOCKCHAIN

Erstelle deine eigene Blockchain und einen Bot zum minen für Kryptowährungen. Schritt für Schritt Anleitung, für Anfänger geeignet!

Magelan Cyber Security

EIGENE Blockchain und Smart Contract's erstellen

VORWORT
INHALTSVERZEICHNIS
1. Basisinstallationen: Geth, Solc, private Test-Ethereum-Blockchain
2. Erster Smart Contract: Hallo-Welt-Demo
3. Mini Token-Smart-Contract: Mein Token Demo
4. Remix Online Solidity Compiler als grafische Entwicklungsumgebung (GUI-IDE)
5. Einen MeinToken-Transfer mit dem GUI-Tool Mist ausführen
6. DApp-Webseite für den Smart Contract mit Node.js
7. Transfer von Ether mit Web3j und Java
8. Transfer von eigenen Smart-Contract-Tokens mit Web3j und Java
9. DApp-Webseite für den Smart Contract mit Java
10. Öffentliche Rinkeby-Test-Ethereum-Blockchain statt privater Blockchain
11. Smart Contract in die öffentliche Rinkeby-Blockchain deployen
12. DApp-Webseite für den Smart Contract in der Rinkeby-Blockchain
13. Truffle für einfache Entwicklung und schnelle Tests
14. Embark für einfache Entwicklung und schnelle Tests
15. DApp-Webseite für den Smart Contract sowohl mit Truffle als auch mit Embark
16. Verwendung des Oraclize-Dienstes für externe Informationsabfragen
17. Analyse der Blöcke und Transaktionen
18. Solidity-Plugin für JetBrains IntelliJ IDEA
Impressum

VORWORT

Blockchain bezeichnet eine kontinuierlich erweiterbare Liste von Datensätzen, welche mittels kryptographischer Verfahren miteinander verkettet sind, so dass die Kette der Datensätze unveränderlich und fälschungssicher ist. Neue Transaktionen werden validiert und in Blöcke verpackt, und die Blöcke werden nach Durchlaufen eines Konsensus-Algorithmus an die Blockchain angehängt und an alle anderen Blockchain-Server gesendet.

Ethereum basiert auf einer öffentlichen Blockchain, die in einem dezentralen Peer-to-Peer-Netz auf vielen Ethereum-Servern als DLT betrieben wird. Ethereum beinhaltet die Kryptowährung Ether und ermöglicht darüber hinaus "Smart Contracts". Damit können Verträge programmiert werden, die elektronisch ausgeführt und überprüft werden. Dazu werden Skripte erstellt (meistens in der Programmiersprache Solidity) und in der Ethereum Virtual Machine (EVM) ausgeführt.

Dies eröffnet sehr vielfältige Möglichkeiten, beispielsweise basieren viele ICOs darauf.
Allerdings muss bei der Programmierung von Smart Contracts besondere Vorsicht gelten: Einprogrammierte Sicherheitslücken können fatale Folgen haben, wie beim "The DAO Hack", der zum Ethereum-Hard-Fork "Ethereum Classic" führte.
Grundsätzliche Erläuterungen gibt es unter: Kryptowährungen, Bitcoin, Ethereum, Blockchain. Im

Folgenden werden einige einfache Programmierbeispiele für Smart Contracts gezeigt.

INHALTSVERZEICHNIS

1. **Basisinstallationen: Geth, Solc, private Test-Ethereum-Blockchain**
2. **Erster Smart Contract: Hallo-Welt-Demo**
3. **Mini-Token-Smart-Contract: MeinToken-Demo**
4. **Remix Online Solidity Compiler als grafische Entwicklungsumgebung (GUI-IDE)**
5. **Einen MeinToken-Transfer mit dem GUI-Tool Mist ausführen**
6. **DApp-Webseite für den Smart Contract mit Node.js**
7. **Transfer von Ether mit Web3j und Java**
8. **Transfer von eigenen Smart-Contract-Tokens mit Web3j und Java**
9. **DApp-Webseite für den Smart Contract mit Java**
10. **Öffentliche Rinkeby-Test-Ethereum-Blockchain statt privater Blockchain**
11. **Smart Contract in die öffentliche Rinkeby-Blockchain deployen**
12. **DApp-Webseite für den Smart Contract in der Rinkeby-Blockchain**
13. **Truffle für einfache Entwicklung und schnelle Tests**
14. **Embark für einfache Entwicklung und schnelle Tests**
15. **DApp-Webseite für den Smart Contract sowohl mit Truffle als auch mit Embark**
16. **Verwendung des Oraclize-Dienstes für externe Informationsabfragen**
17. **Analyse der Blöcke und Transaktionen**
18. **Solidity-Plugin für JetBrains IntelliJ IDEA**

1. Basisinstallationen: Geth, Solc, private Test-Ethereum-Blockchain

Diese Demo zeigt:
 Wie " **geth**" (Go Ethereum) installiert wird.
 Wie " **solc**" (Solidity Compiler) installiert wird.
 Wie eine eigene private **Test-Ethereum-Blockchain** eingerichtet und gestartet wird.
 Wie ein **Account** angelegt und **Mining** gestartet wird.

Verwendet werden folgende Versionen:
 geth 1.8.2
 solc 0.4.19
 web3 0.20.1
 Windows 10

Diese Demo zeigt:

- Wie "geth" (Go Ethereum) installiert wird.
- Wie "solc" (Solidity Compiler) installiert wird.
- Wie eine eigene private Test-Ethereum-Blockchain eingerichtet und gestartet wird.
- Wie ein Account angelegt und Mining gestartet wird.

Verwendet werden folgende Versionen:

- geth 1.8.2
- solc 0.4.19
- web3 0.20.1
- Windows 10

Der folgende Text legt den Fokus auf Einfachheit und gute Nachvollziehbarkeit. Vorläufig werden nur Kommandozeilen-Tools eingesetzt. Auf grafische Tools wird weiter unten eingegangen.
In diesem Beispiel wird eine private nur auf dem eigenen PC existierende Ethereum-Blockchain eingerichtet. Wie eine öffentliche Ethereum-Blockchain verwendet werden kann, wird weiter unten gezeigt.
Die Kommandos sind für Windows dargestellt. Bei Verwendung von Linux oder Mac OS X genügt es häufig, in Pfadangaben "\" durch "/", in PATH-Angaben ";" durch ":" und bei Platzhaltern %MEINE_VARIABLE% durch $MEINE_VARIABLE zu ersetzen.
Führen Sie die im Folgenden beschriebenen Schritte aus.

1. Wechseln Sie in Ihr bevorzugtes Workspace-Verzeichnis (z.B. \MeinWorkspace) und führen Sie folgende Kommandos aus:
   cd \MeinWorkspace
   mkdir EthereumDemo
   cd EthereumDemo
   mkdir solc
   mkdir src
   tree /F
2. Installieren Sie solc (Solidity Compiler):
   Downloaden Sie von https://github.com/ethereum/solidity/releases die für Ihr Betriebssystem geeignete Installationsdatei, beispielsweise für Windows: solidity-windows.zip.
   Unter Windows entzippen Sie diese Zip-Datei in das Verzeichnis: \MeinWorkspace\EthereumDemo\solc.
   Für andere Betriebssysteme verfahren Sie entweder analog oder wie beschrieben unter: Installing the Solidity Compiler.
3. Führen Sie im Kommandozeilenfenster aus:
   cd \MeinWorkspace\EthereumDemo

solc\solc.exe --version
solc\solc.exe --help
Beide Kommandos müssen plausible Ergebnisse zeigen.

4. Installieren Sie geth (Go Ethereum):
   Downloaden Sie von https://geth.ethereum.org/downloads/ eine für Ihr Betriebssystem geeignete Geth-Version, beispielsweise für Windows "Geth 1.8.2 for Windows".
   Verwenden Sie nicht die Geth-Versionen 1.8.0 und 1.8.1, weil es damit zum Web3j Issue 318 kommt.
   Für Windows erhalten Sie die Datei "geth-windows-amd64-1.8.2-b8b9f7f4.exe". Führen Sie diese Datei aus. Die Installation erweitert den Windows-Such-PATH um das Geth-Verzeichnis.
   Für andere Betriebssysteme verfahren Sie wie beschrieben unter: https://www.ethereum.org/cli oder https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum.
5. Öffnen Sie ein neues Kommandozeilenfenster, damit Konfigurationsänderungen und PATH-Erweiterungen wirksam werden. Führen Sie darin aus:
   cd \MeinWorkspace\EthereumDemo
   geth version
   geth help
   Beide Kommandos müssen plausible Ergebnisse zeigen. Zu den Kommandozeilenoptionen von geth finden Sie auch eine Auflistung unter: Command Line Options.
6. Genesis-Block:
   Um eine eigene private Test-Ethereum-Blockchain zu starten, wird eine manuelle Initialisierung des ersten Genesis-Blocks benötigt. Erstellen Sie im src-Unterverzeichnis die JSON-Datei:
   genesis-block.json

   ```
   {
   ```

```
    "alloc": {},
    "coinbase":
"0x0000000000000000000000000000000000000000",
    "config": {},
    "difficulty": "0x4000",
    "extraData": "0x4711",
    "gasLimit": "0xffffffff",
    "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000042",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "timestamp": "0x00"
}
```

Achten Sie beim Speichern darauf, dass die Datei mit Unix-Zeilenendezeichen und als ASCII-Datei ohne UTF-BOM gespeichert werden muss.
Einige Hinweise zum Genesis-Block finden Sie unter: Creating The Genesis Block.

7. Erstellen Sie den Genesis-Block, indem Sie ausführen:
   cd \MeinWorkspace\EthereumDemo
   geth --datadir "Private-Blockchain" init src/genesis-block.json
   Sie erhalten u.a.:

   ```
   Successfully wrote genesis state
   ```

8. Start der eigenen Test-Ethereum-Blockchain (vorerst ohne Mining):
   Führen Sie aus (in einer Zeile):
   geth --networkid 77 --identity "MeineDevChain" --datadir "Private-Blockchain" --nodiscover --rpc --rpcapi

"db,eth,net,web3,personal,txpool" --rpccorsdomain "*" console
Sie erhalten u.a.:

```
Starting peer-to-peer node
HTTP endpoint opened: http://127.0.0.1:8545
Welcome to the Geth JavaScript console!
```

Und Sie erhalten die Geth-JavaScript-Console, in der Sie Geth-JavaScript-Kommandos eingeben können.

9. Sehen Sie sich die Doku zu den web3.eth-Kommandos an: web3.eth 1.0 bzw. Web3 0.20.x JavaScript API. Fragen Sie die von Ihnen verwendete Web3.js-Version ab:
   web3.version.api
10. Testen Sie, ob bereits Accounts existieren. Falls das Einfügen von Kommandos per Strg+V nicht funktioniert, versuchen Sie die rechte Maustaste, "Einfügen" und Return:
    web3.eth.accounts
    Sie erhalten eine leere Menge:

```
[]
```

11. Account anlegen:
    Legen Sie einen Account an (denken Sie sich ein schwierigeres Passwort aus und merken Sie es sich gut):
    personal.newAccount("Meine Ethereum-Test-Passphrase")
    Sie erhalten eine 40-stellige Hex-Zahl als Account-Adresse, beispielsweise:

```
"0x2f94831a57a96041064d9d0c24583b12f807f2a5"
```