

# Private / permissioned Blockchains und Azure

Artsiom Kaliaha

# Agenda



- Blockchain für Kryptowährungen (recap)
- Vertrauen zwischen Geschäftspartnern
- RAFT
- Microsoft Confidential Consortium Framework (CCF)
- RAFT vs CCF
- Azure Confidential Ledger (ACL)
- Private / permissioned Blockchains auf k8s: BESU
- Live Demo

# Blockchain für Kryptowährungen



- Public/Private Schlüssel Kryptographie (e.g. ECC und RSA)
- P2P Netzwerke
- Protokolle (Proof Of XYZ, Blockgröße, Rewards fürs Mining)
- Verteilte Architektur
- Immutabilität

# Vertrauen zwischen Geschäftspartnern



## Herausforderungen:

- Mangelndes Vertrauen
- Unterstützung und Automatisierung in solchen Fragen ist wichtig

## Anforderungen an solch eine Lösung:

- Skalierbarkeit
- Datenschutz
- Kooperation in Konsortien

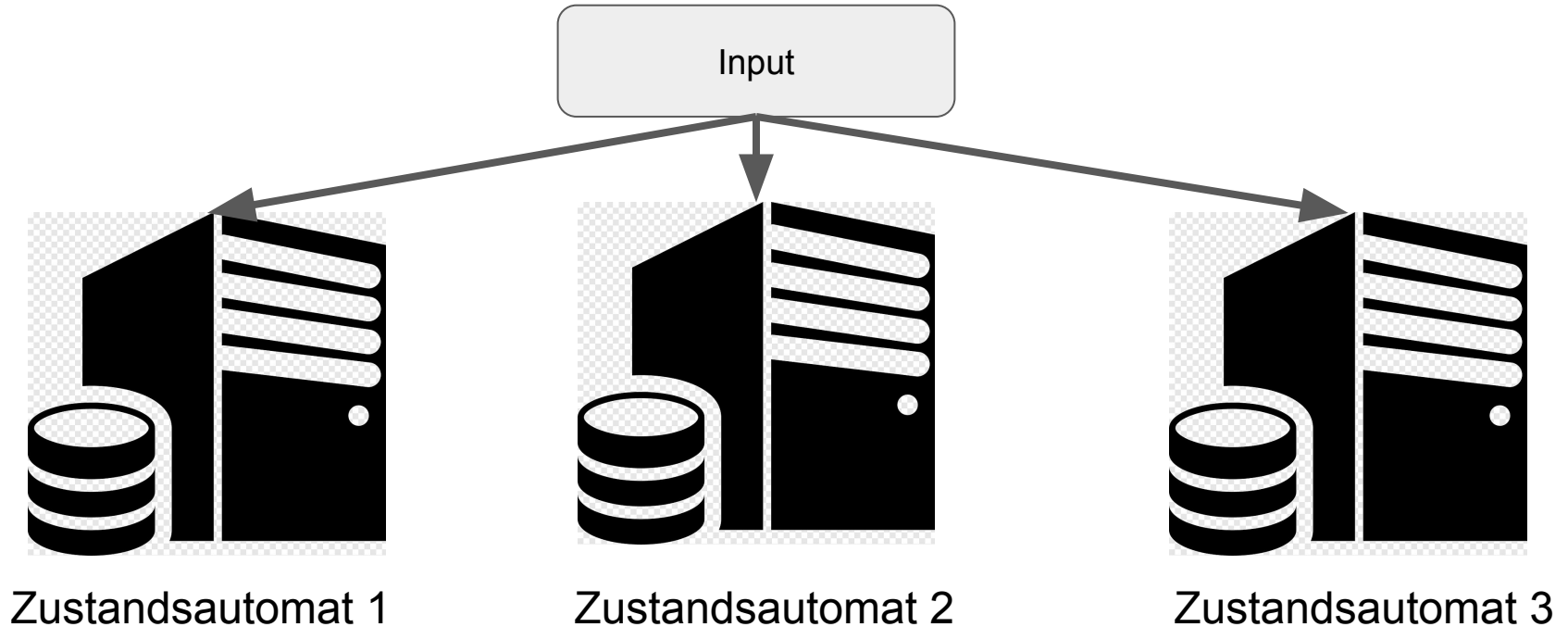
# The RAFT Consensus Algorithm



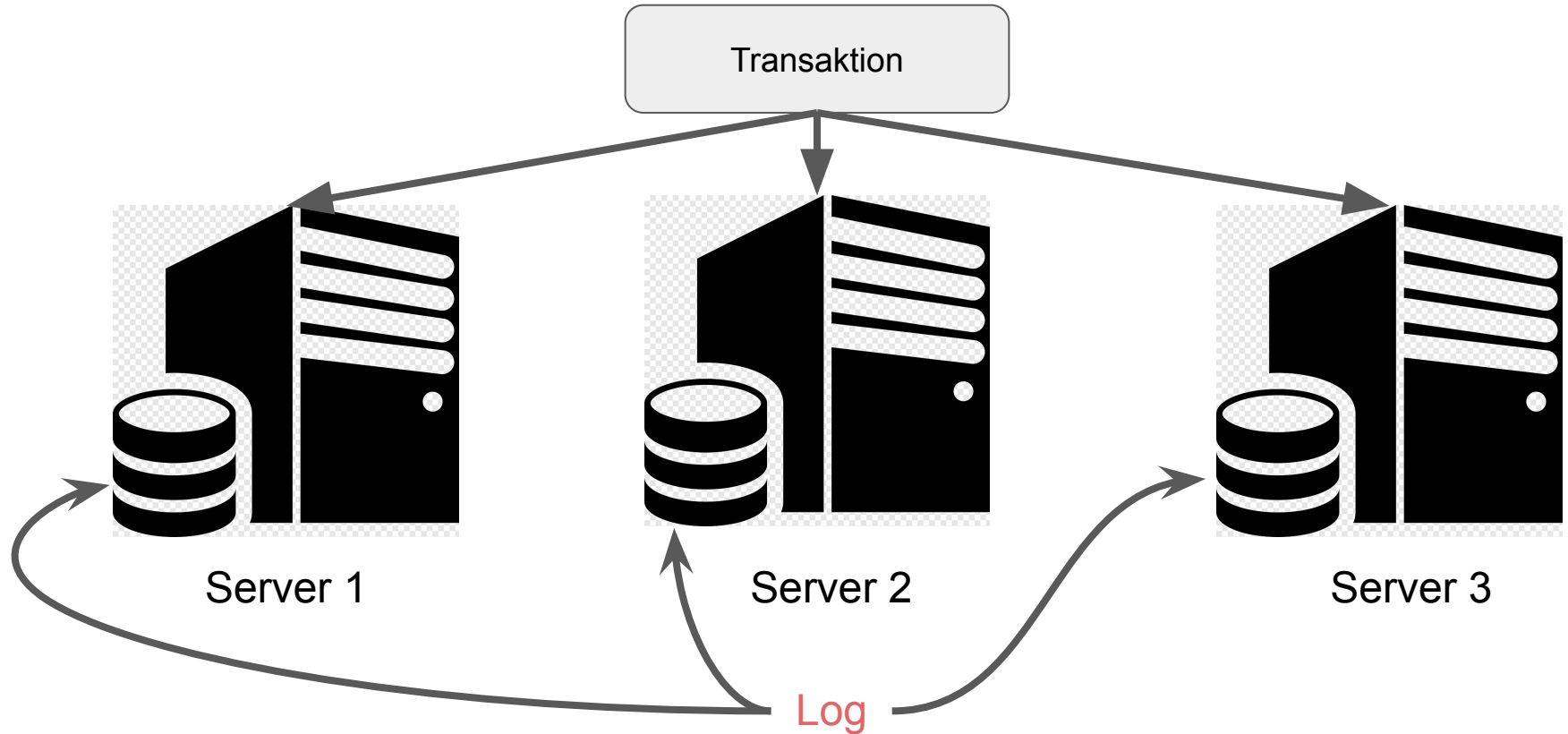
## Key Facts:

- Replikation eines Zustandsautomaten (Verwaltung eines replizierten Logs)
- Sehr stark inspiriert durch PAXOS
- Basis für **ALLE** hochverfügbare Services (Cloud, Datenbanken, Microservices)

# Replikation eines Zustandsautomaten (Theorie)

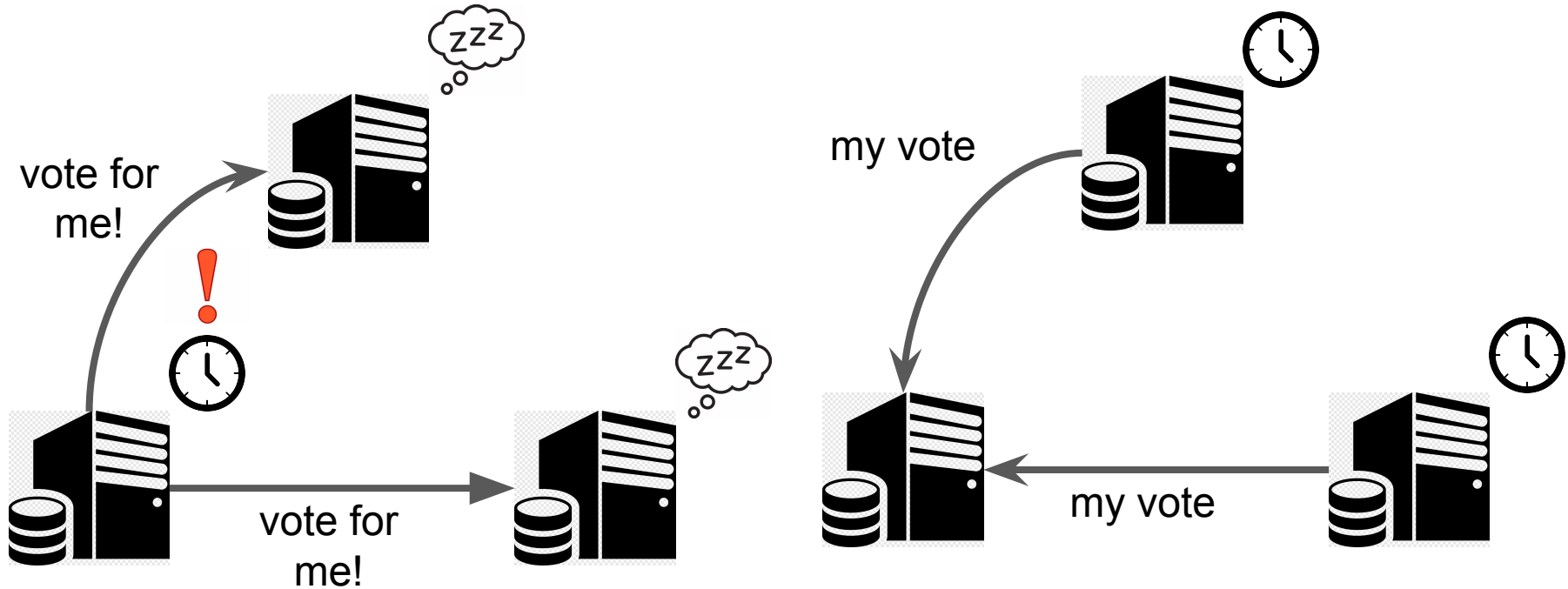


# Replikation eines Zustandsautomaten (Praxis)



# Term 0 (Amtsperiode)

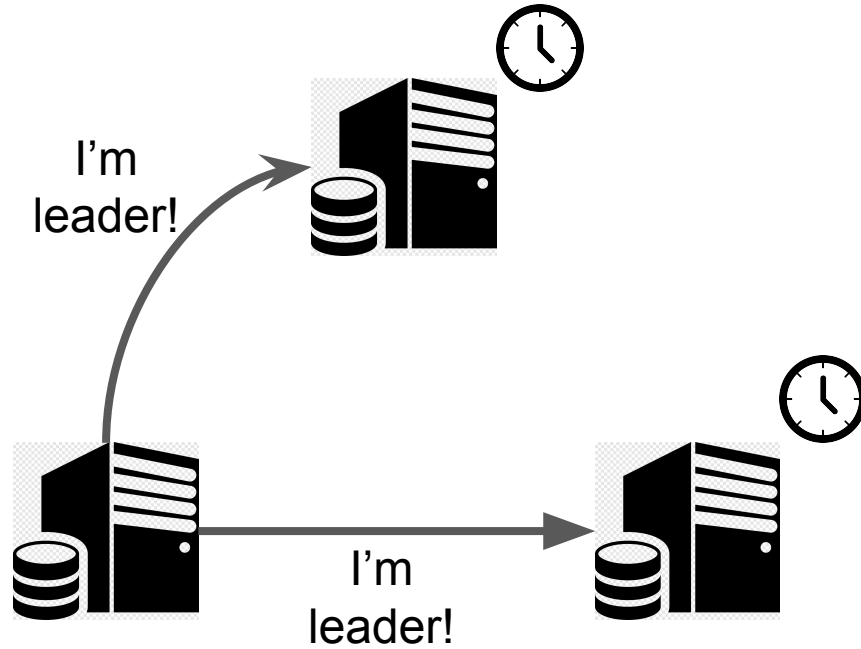
0 Term





# Term 1

1 Term



# Single Master

xyz Term

User eines  
hochverfügbaren  
Services



replicate  
it in your  
log!



replicate  
it in your  
log!



save it!



# Single Master

xyz Term

User eines  
hochverfügbaren  
Services



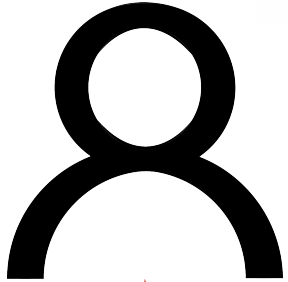
replicated



replicated



done!

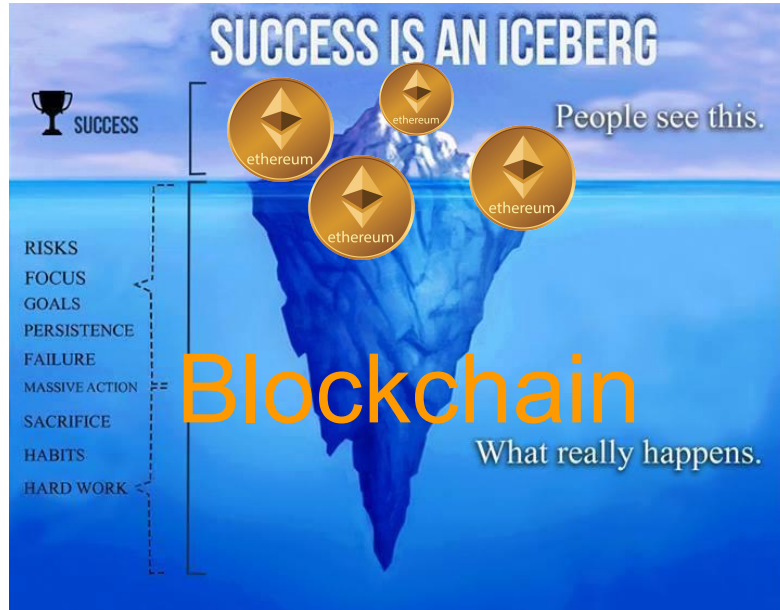


# Microsoft Confidential Consortium Framework (CCF)

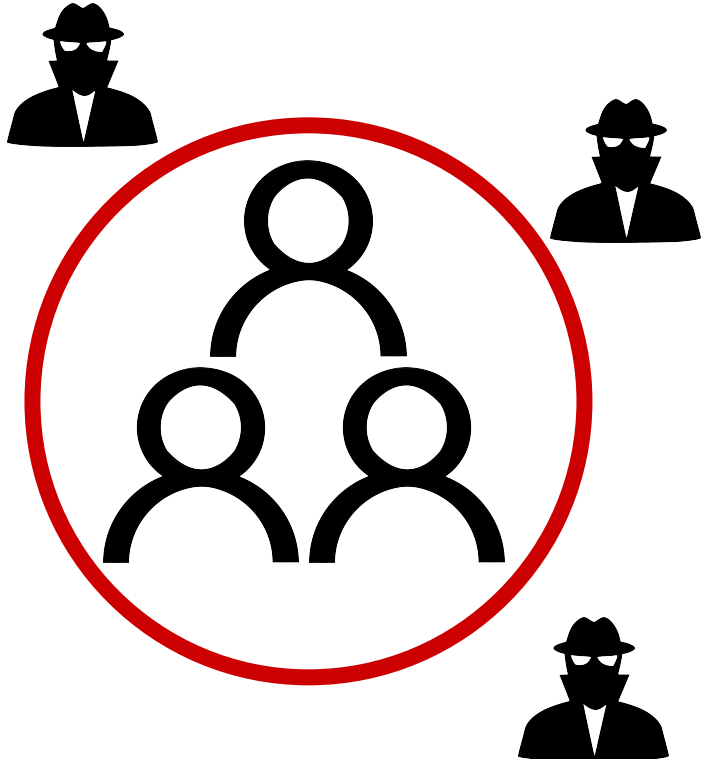


- Im April 2019 erschienen
- Zusammenarbeit Azure + Microsoft
- Sehr stark inspiriert durch RAFT
- Knoten ↔ Knoten über TLS
- Intel SGX TEEs
- Transaktionen auf Basis Ethereum (hahaha)
- Knoten in separaten Datenzentren oder On-Premise
- Open Source auf Github

# Blockchain aus Community und Microsoft Sicht



# Konsortium



- Ein stabiles Konsortium der Mitglieder
- Nutzer führen ihre Transaktionen über die Knoten des Konsortiums durch

Mitglieder müssen sich einigung sein über:

- Code des Services
- wie wird das Governance umgesetzt
- initiale Konfiguration des Services

# User-Typen



User. Users können Transaktionen innerhalb der vorgegebenen Zugangsrechten durchführen.



Member. Durch das Governance Modul können Members das Konsortium verwalten und seine Konfiguration modifizieren.

# Tabellen



User. Speichert Zertifikate und Zugangsrechte jeweiliger User



Member. Bewahrt Zertifikate, Keyshare Encryption Keys und den aktuellen Status jedes Mitglieds.



# Tabellen und Transaktionen

CCF Server



- Log
- Tabellen
- Ledger

RAFT Server



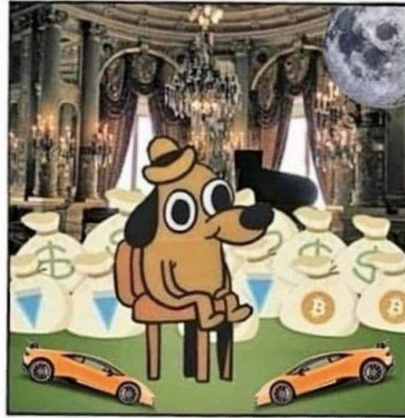
- Log

# Leistungsoptimierung

## HOW *bitcoin* WORKS



 IT'S OKAY,  
I'LL JUST HODL  
& BUY MORE



 MONTHS  
LATER

- Merkle Tree statt eine Kette der Blöcke. Tree wird nur mit neuen Transaktionen erweitert (nicht erneut aufgebaut)
- Jeder Knoten entscheidet für sich, ob er nur Merkle Tree oder den ganzen Ledger speichert
- Checkpoints

# RAFT vs CCF

	RAFT	CCF
Anwendungsfall	Replication / Fehlertoleranz in verteilten Systemen	Mangelndes Vertrauen in Verteilten Systemen
Multimaster	-	-
Zustand	Log	Log, Tabellen, Ledger
Integrität des Logs	-	Merkle Tree, Checkpoints
Master	Leader	Primary
Anhänger	Follower	Backup
Amtsperiode	Term	View

# RAFT vs CCF

	RAFT	CCF
Lese-Operationen	alle Knoten	alle Knoten
Schreib-Operationen	nur Leader	nur Primary
Verschlüsselung	-	+
Verwaltung der Zugangsrechte	-	+
Technologien, die auf diesem Algorithmus basieren	Cassandra (nicht ganz) Neo4j (nicht ganz) Zookeeper (nicht ganz)  HashiCorp Consul (100%) CockroachDB (100%) etcd (100%)	CCF Open Source Azure Confidential Ledger

# Azure Confidential Ledger



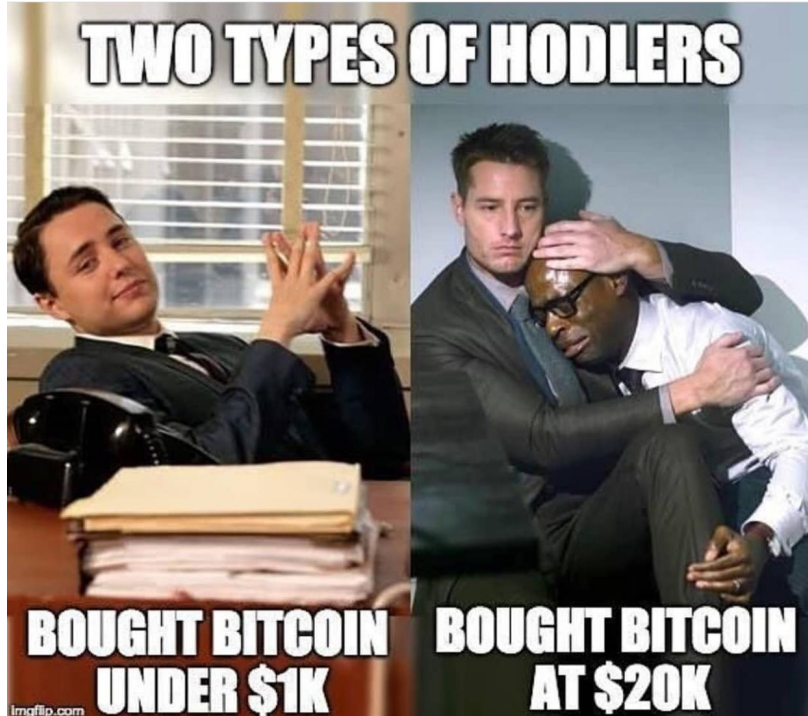
- Kommerzielle Implementierung von CCF
- Service zur Verwaltung sensibler Daten / Metadaten

## Anwendungsfälle:

- Geschäftsrelevante Transaktionen
- Digitale Assets, wie z.B. Verträge
- Log der Änderung der Zugangsrechte für ein System

Eine mögliche Re-Implementierung von CCF?

# Private / permissioned Blockchains auf k8s: BESU



- Open Source Ethereum Client (Java)
- Veröffentlicht in 2018 von ConsenSys
- Enterprise Anwendungen mit der Option Zugriffsrechte einzuschränken
- API für Performance Metriken (APM)
- Konsensprotokolle: Proof of Work, Authority und Stake
- kein Schlüsselmanagement innerhalb des Clients

## Kurze Zwischenfrage



<https://www.tributech.io/>

<https://oceanprotocol.com/>



# Live Demo



## BESU auf k8s. Pods

NAME	READY	STATUS	RESTARTS	AGE
bootnode1-0	1/1	Running	0	17m
bootnode2-0	1/1	Running	0	17m
minernode-0	1/1	Running	0	17m
node-0	1/1	Running	0	17m
node-1	1/1	Running	0	13m

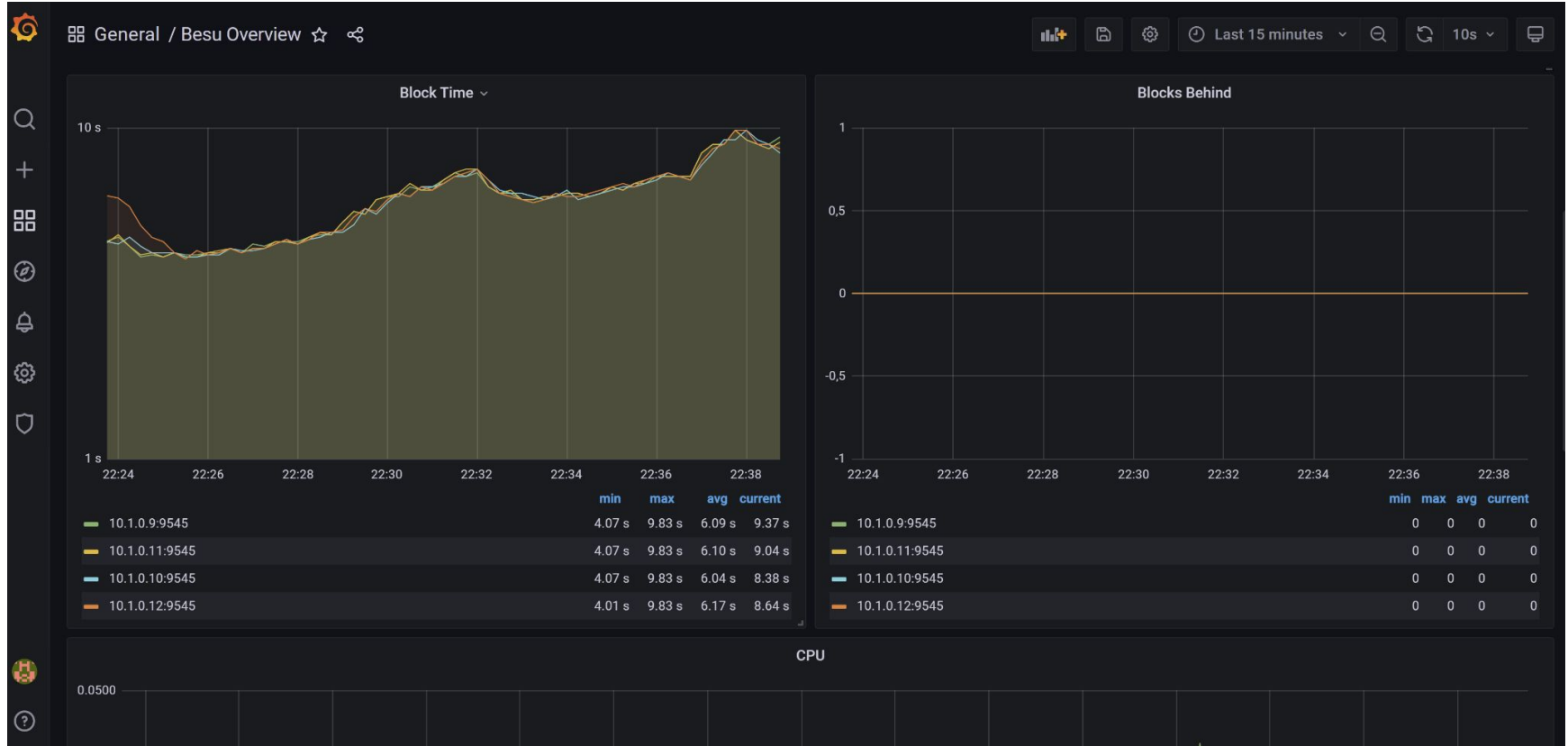
# BESU auf k8s. genesis.kson

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: besu-genesis-configmap
5    labels:
6      app: besu-genesis-configmap
7      namespace: besu
8  data:
9    genesis.json: |-
10     {
11       "config": {
12         "constantinoplefixblock": 0,
13         "ethash": {
14           "fixeddifficulty": 1000
15         },
16         "chainID": 1981
17       },
18       "nonce": "0x42",
19       "gasLimit": "0x1000000",
20       "difficulty": "0x10000",
21       "alloc": {
22         "fe3b557e8fb62b89f4916b721be55ceb828dbd73": {
23           "privateKey": "8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefbfd1542c692be63",
24           "comment": "private key and this comment are ignored. In a real chain, the private key should NOT be stored",
25           "balance": "0xad78ebc5ac6200000"
```

# BESU auf k8s. Monitoring



# BESU auf k8s. Monitoring



# Truffle Tests auf Ganache Netz

```
PS C:\Users\stakk\Documents\fh\master\2\block\BlockchainFH\truffle-tmp> truffle test --network development
Using network 'development'.
```

```
Compiling your contracts...
```

```
=====
```

```
> Compiling .\contracts\Notary.sol
> Compiling .\test\NotaryTest.sol
> Compiling truffle\Assert.sol
> Compiling truffle\AssertAddress.sol
> Compiling truffle\AssertAddressArray.sol
> Compiling truffle\AssertBalance.sol
> Compiling truffle\AssertBool.sol
> Compiling truffle\AssertBytes32.sol
> Compiling truffle\AssertBytes32Array.sol
> Compiling truffle\AssertGeneral.sol
> Compiling truffle\AssertInt.sol
> Compiling truffle\AssertIntArray.sol
> Compiling truffle\AssertString.sol
> Compiling truffle\AssertUint.sol
> Compiling truffle\AssertUintArray.sol
> Compiling truffle\DeployedAddresses.sol
> Artifacts written to C:\Users\stakk\AppData\Local\Temp\test--5520-VXU7TaCMVHuy
> Compiled successfully using:
  - solc: 0.4.24+commit.e67f0147.Emscripten.clang
```

# Truffle Tests auf Ganache Netz

```
[  
  '0xaE890bD0FbC3b456D0dcCDaC6b71083Fc8c8A95F',  
  '0xCE135B49ffc08F1FABC648b7E6a69d5A6Cd56883',  
  '0x13c1938ea6c236C0D21B707128f427265d366c5f',  
  '0x154bC49AEa29F7E418e6ED4Bb0fc53C8C91C4F6d',  
  '0xCc8E90e3080af62D077F9eDf3C6494d50d2f3E35',  
  '0xf5373EA89d4903ab4F32D4Cf03DF20A4379F1160',  
  '0xec6Cc7eFB1ffc8934698d944A66a5FF5F4ED1eA6',  
  '0x9F8a2FC0d3F616A553FCdb8F3f772f03B73273Dc',  
  '0x9F75a3e16E2A7323FcDF9104179895B773735937',  
  '0x4255f28F9BAa5241a86485cFFF26dF323a22D88B',  
]
```

## NotaryTest

✓ testAddAndRead (1396ms)

## Contract: Notary

✓ should not have an entry for an unknown hash (540ms)

✓ should have an entry for a known hash (1020ms)

# Deployment eines Contracts auf k8s BESU

```
PS C:\Users\stakk\Documents\fh\master\2\block\BlockchainFH\truffle-tmp> truffle migrate --network k8s

Compiling your contracts...
=====
> Compiling .\contracts\Notary.sol
> Artifacts written to C:\Users\stakk\Documents\fh\master\2\block\BlockchainFH\truffle-tmp\build\contracts
> Compiled successfully using:
   - solc: 0.4.24+commit.e67f0147.Emscripten.clang

Starting migrations...
=====
> Network name:      'k8s'
> Network id:       1981
> Block gas limit: 16777216 (0x1000000)
```



# Deployment eines Contracts auf k8s BESU

```
1_notary.js
```

```
=====
```

```
Replacing 'Notary'
```

```
-----
```

```
> transaction hash: 0x7535774875b1a4005ac817220c8e296fb9e699fafd03eca29da0bed6e3b0548d
> Blocks: 1        Seconds: 4
> contract address: 0x9a3DBCa554e9f6b9257aAa24010DA8377C57c17e
> block number:    241
> block timestamp: 1654116075
> account:         0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73
> balance:         484
> gas used:        563588 (0x89984)
> gas price:       0.000001 gwei
> value sent:      0 ETH
> total cost:      0.000000000563588 ETH
```

```
> Saving artifacts
```

```
-----
```

```
> Total cost:      0.000000000563588 ETH
```

```
Summary
```

```
=====
```

```
> Total deployments: 1
> Final cost:        0.000000000563588 ETH
```

# Dokument im Contract speichern

```
### Publish notary entry
```

```
Send Request
```

```
POST http://localhost:3000/chain/entry
```

```
Content-Type: application/json
```

```
{  
  "hash": "0x9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a02",  
  "filename": "testFileName 2",  
  "comment": "test comment 2"  
}
```

# Dokument im Contract speichern (Nodejs Server)

[illegible]

# Dokument abfragen (Nodejs Server)

### Request notary entry

Send Request

GET http://localhost:3000/chain/entry/0x9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a02


Response(21ms) X


```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 133
5 ETag: W/"85-se3T4eUEw17BB1H5GEHwwoyNOSA"
6 Date: Wed, 01 Jun 2022 20:49:20 GMT
7 Connection: close
8
9 {
10   "filename": "testFileName 2",
11   "timestamp": "1654116378",
12   "comment": "test comment 2",
13   "hash": "0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73"
14 }
```




# Dokument abfragen (Remix)

OR


At Address

Transactions recorded 0 


Deployed Contracts 

 NOTARY AT 0X9A3...7C17E (BLOCKCHA  

addEntry



getEntry




0: string: testFileName 2

1: uint256: 1654116378

2: string: test comment 2

3: address: 0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73

Low level interactions 

CALLDATA

Transact