Private und permissioned Blockchains

Artsiom Kaliaha
Department for Mobility and Energy
University of Applied Sciences Upper Austria
Hagenberg, Austria
S2110455009@students.fh-hagenberg.at

***Zusammenfassung*—Blockchain ist eine Technologie mit sehr viel Potenzial und Kryptowährungen sind nur eine von vielen Ausprägungen. Das vorliegende Paper soll den Einsatz der Blockchain bei dem Aufbau einer Enterprise Architektur in der Cloud untersuchen.**

***Index Terms*—Blockchain, Azure, Microsoft, BESU**

I. Einleitung

Blockchain als Technologie zeichnen folgende Aspekte aus:

- **Public/Private Schlüssel Kryptographie** (e.g. ECC und RSA). Bei der asymmetrischen Kryptographie benötigt ein Nutzer einen privaten Schlüssel, mit dem Transaktionen unterschrieben werden, und einen public Schlüssel, der (nach dem Hashen) mit der Kontoadresse vergleichbar ist. Asymmetrische Kryptographie ist ein Garant dafür, dass die Transaktionen auf der Blockchain nicht manipuliert werden können.
- **P2P Netzwerke** (es gibt keinen zentralen Server, alle Parteien kommunizieren direkt miteinander)
- **Protokolle** (Proof Of XYZ, Blockgröße, Verteilung des Rewards fürs Mining)
- **Verteilte Architektur.** Jeder Knoten im Netzwerk wendet jede Transaktion bei sich lokal an. Somit wird der gleiche globale Zustand auf allen Knoten erreicht.
- **Immutabilität.** Blöcke in der Kette werden einmal und für immer gespeichert. Die einzige Möglichkeit das Blockchain Netz zu hacken, ist N / 2 + 1 Knoten im Netzwerk zu bestechen.

Bei der Enterprise Architektur kommen nicht alle diese Eigenschaften zum Einsatz, oder zumindest nicht alle im gleichen Maße. Das kommt vor allem aus den Anforderungen an diese Technologie, die sich von denen der Kryptowährungen unterscheiden, und Anwendungsfällen, wo Blockchain ihre Stärken zeigen kann.

Große Firmen im Bereich Logistik, Immobilien und Bau müssen ihren Partnern vertrauen, zumal es oft eine zeitliche Verzögerungen zwischen der Dienstleistungserbringung und der Zahlung gibt. Oft erfolgt die Zahlung nur dann, wenn alle im Vertrag aufgelisteten Bedingungen eingehalten werden. Implizit hat das sehr viel mit dem Vertrauen in einen konkreten Geschäftspartner zu tun. Aus diesem Grund suchen Firmen aus den vorher genannten Branchen nach Lösungen, die ihre Prozesse aus technologischer Sicht unterstützen und automatisieren könnten. Eine Lösung, die helfen soll, Vertrauen zwischen Partnern aufzubauen, soll folgende Kriterien erfüllen:

- **Skalierbarkeit** (Transaktionendurchsatz wie bei Datenbanken). Bei der gewährleisteten Immutabilität muss der Datenspeicher auch performant genug sein. Wenn eine Firma sehr wenige Transaktionen mit nur ein paar Partnern durchführen müsste, dann würde kein Bedarf für so eine Lösung bestehen.
- **Datenschutz.** Nur berechtigte Mitglieder dürfen auf die Transaktionen zugreifen können, geschweige denn Transaktionen auf einer öffentlichen Blockchain unverschlüsselt zu veröffentlichen.
- **Kooperation in Konsortien.** Änderungen im Netz sind nur dann möglich, wenn seine Mitglieder sich darauf einigen. Die Daten, die auf der Blockchain gespeichert werden, machen das Geschäftsgeheimnis aus und der Leak von solchen Daten kann z.B. den Aktienkurs der betroffenen Firmen negativ beeinflussen.

II. Replikation eines Zustandsautomaten mittels RAFT

Um den Weg zu modernen Ledger Technologien verinnerlichen zu können, muss man sich zuerst an die Geschichte der Verteilten Systeme wenden und an die Algorithmen / Protokolle die für diese Zwecke zum Einsatz kommen.

Raft ist ein Konsens-Algorithmus zum Verwalten eines replizierten Logs. Er erzeugt ein Ergebnis, das dem (multi-)Paxos entspricht. Er ist ebenfalls genau so effizient wie Paxos, aber seine Struktur unterscheidet sich von ihm. Es ist wichtig zu beachten, dass Raft kein neuer Algorithmus ist, sondern eine Optimierung und eine inkrementelle Verbesserung eines bereits vorhandenen Algorithmus namens Paxos.

Konsens-Algorithmen entstehen typischerweise im Kontext replizierter Zustandsautomaten. Diese werden normalerweise mithilfe eines replizierten Logs implementiert, wie in Abbildung **??** gezeigt wird. Jeder Knoten speichert ein Log mit einer Reihe von Befehlen, die von seinem Zustandsautomaten bearbeitet wird. Jedes Log enthält die gleichen Befehle in der gleichen Reihenfolge, sodass jeder Zustandsautomat die gleiche Folge von Befehlen verarbeitet. Da die Zustandsautomaten deterministisch sind, berechnet jeder den gleichen Zustand und die gleiche Folge von Ausgaben.

Das replizierte Log konsistent zu halten ist die Aufgabe des Konsensus-Algorithmus. Das Konsensmodul an einem Knoten empfängt Befehle von Clients und fügt sie seinem Log hinzu. Es kommuniziert mit den Konsensmodulen an anderen Knoten, um sicherzustellen, dass jedes Log schließlich

dieselben Werte in derselben Reihenfolge enthält, auch wenn einige Knoten fehlschlagen (siehe Abbildung 1).

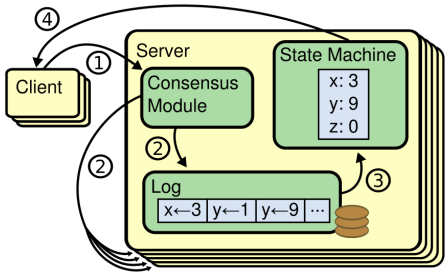

Abbildung 1. Architektur eines replizierten Zustandsautomaten.

Zu jedem Zeitpunkt ist jeder Knoten in einem von drei möglichen Zuständen sein:

- **Leader.** Ein Leader verarbeitet alle Client-Anfragen. Wenn ein Client einen Follower kontaktiert, wird er von diesem Follower an den Leader weitergeleitet.
- **Follower** (Anhänger). Followers sind passiv: sie selber schicken keine Anfragen aus und antworten nur auf mögliche Anfragen des Leader oder eines Kandidaten.
- **Candidate** (Kandidat). Wenn ein Follower zum Kandidaten wird, darf er sich zum Leader vorschlagen, wobei er vom Quorum aller Knoten unterstützt werden muss.

Im Normalbetrieb es gibt genau einen Leader und alle anderen Knoten sind Followers.

Raft teilt die Zeit in Terms (Amtsperioden) beliebiger Länge. Terms werden mit aufeinanderfolgenden ganzen Zahlen nummeriert. Jeder Term beginnt mit einer Wahl, bei der ein oder mehrere Kandidaten versuchen, Leader zu werden. Wenn ein Kandidat die Wahl gewinnt, fungiert er für den Rest des Terms als Leader. Verschiedene Knoten können die Übergänge zwischen Terms zu verschiedenen Zeiten beobachten. In einigen Situationen übersieht ein Knoten möglicherweise eine Wahl oder sogar ganze Terms. Terms agieren in Raft als eine "logische Uhr" und ermöglichen es den Knoten, veraltete Informationen wie z.B. veraltete Leaders zu erkennen. Jeder Knoten speichert eine aktuelle Term-Nummer, die mit der Zeit monoton zunimmt. Der aktuelle Term wird ausgetauscht, wenn Knoten kommunizieren. Ein Term wird gleich nach dem Absturz des aktuellen Leaders beendet.

A. Wahlprozess des Leaders

Jeder neu gestartete Knoten wird standardmäßig zum Follower. Ein Knoten bleibt im Follower-Zustand, solange er einen Heartbeat von einem Leader erhält. Um eine Wahl zu beginnen, erhöht ein Follower den zuletzt gesehenen Term und wechselt in den Kandidatenzustand. Er stimmt dann für sich selbst und schickt RequestVote-Anfragen parallel an alle anderen Knoten im Cluster. Ein Kandidat bleibt in diesem Zustand, bis eine von drei Optionen vorkommt:

- der Kandidat gewinnt die Wahl
- ein anderer Knoten etabliert sich als Leader
- ein Zeitraum vergeht ohne Gewinner
- ein Kandidat gewinnt eine Wahl, wenn er Stimmen von einem Quorum für denselben Term erhält

B. Replikation des Logs

Sobald ein Leader gewählt wurde, beginnt er mit der Bearbeitung der Client -Anfragen. Jede Client-Anfrage enthält einen Befehl, der von den replizierten Zustandsautomaten ausgeführt werden soll. Der Leader hängt den Befehl als einen neuen Eintrag an sein Log an und schickt AppendEntries-Anfragen zu jedem anderen Knoten, um den Eintrag zu replizieren. Wenn der Eintrag sicher repliziert wurde (d.h. auf einem Quorum aller Knoten), wendet der Leader den Eintrag auf seinen Zustandsautomaten an und gibt das Ergebnis dieser Ausführung an den Client zurück. Jeder Logeintrag enthielt einen Befehl inklusive einer Term-Nummer.

III. Microsoft Confidential Consortium Framework

A. CCF

Mit den Anforderungen der Firmen im Hinterkopf hat Microsoft Research zusammen mit Azure im April 2019 CCF oder Coco Framework (Confidential Consortium Framework) vorgeschlagen [3]. Die Grundidee von CCF besteht darin, einen verteilten und sicheren Datenspeicher auf Basis von Blockchain betreiben zu können (eine typische permissioned Blockchain) [3]. Dieses Framework hat keine darunter liegende Technologie, die dessen Leistung bestimmen oder technisch einschränken könnte [2]. Kurzer Überblick über die Eigenschaften von Coco Framework:

- Knoten kommunizieren miteinander über RAFT Protokoll (zumal RAFT verständlicher als PAXOS ist)
- Die Kommunikation zwischen Knoten erfolgt über TLS 1.2
- Die Sicherheit der Knoten ist durch Intel SGX TEE (trusted execution environments) gewährleistet
- Das Konsortium der vertrauenswürdigen Mitgliedern ist konfigurierbar
- Transaktionen werden auf Basis von Ethereum durchgeführt (statt Ethereum kann rein theoretisch jede beliebige Chain verwendet werde, so kann CCF in beliebige Blockchain Netzwerke integriert werden)
- Mitglieder oder Knoten können in separaten Datenzentren oder auf On-Premise Server laufen
- Eine Open Source Implementierung ist auf Github abrufbar

Permissioned Blockchains erreichen ihre Effizienz durch ein stabiles Konsortium aus Knoten. Eine größere Menge von Nutzern können ihre Transaktionen über die Knoten des Konsortiums einreichen. So würde z.B. ein Bankensystem auf Basis von permissioned Blockchains funktionieren. Einige Implementierungen der permissioned Blockchain zeigen gute Leistungen, aber keine Confidentiality und andere eine durchschnittliche Leistung mit hoher Confidentiality [2]. Das CCF, in deren Kern Intel SGX TEE liegt, bietet diese beiden Eigenschaften. In CCF müssen Mitglieder des Konsortiums

nicht unbedingt einander vertrauen, um zu kooperieren und zusammen einen Service zur Verfügung zu stellen. Diese Mitglieder müssen sich nur über folgende Punkte einig sein:

- der Code des Services
- wie wird das Governance umgesetzt
- initiale Konfiguration des Services (Zugangsdaten und die Anzahl der TEE-fähigen Hosts)

Für die Realisierung der Anwendung stellen die Knoten im CCF einen Key-Value-Speicher bereit. Der Code der Anwendung (auch Smart Contract genannt) kann in unterschiedlichen Programmiersprachen realisiert werden [2]. Zusätzlich zu den Daten der Anwendung werden im Speicher Informationen über die Mitglieder und Governance Regel aufbewahrt. Zwecks Schutz gegen unterschiedlicher Angriffe wurden in CCF folgende Mechanismen vorgesehen:

- TEEs selber bieten die erste Sicherheitsschicht
- Die Schlüsselrotierung soll gegen Kompromittierung wirken
- Die Änderungen in der Verwaltung werden auf dem Ledger abgespeichert
- Merkle Tree für den ganzen Inhalt des Ledgers wird kontinuierlich berechnet und gespeichert

CCF nutzt eine Art Replizierung eines Zustandsautomaten. Der sichtbare Zustand eines Replicas (eines Knoten) besteht aus dem Log der Transaktionen und dem Commit-Index [2]. Stabil im Log sind diejenigen Transaktionen, die unter dem Commit-Index stehen. Jeder Knoten kann eine neue Transaktion vorschlagen und zwischen Knoten verbreiten. Gleich nach der Initialisierung eines Konsortiums ist der Log leer und der Commit-Index Null.

Der angebotene Service wird durch seinen Code (Smart Contract) definiert und wie dieser Code den Key-Value-Speicher manipuliert. Clients, wenn sie erfolgreich authentifiziert wurden, können vordefinierte Befehle ausführen, die als Transaktionen auf dem Key-Value-Speicher durchgeführt werden. CCF definiert zwei Client-Typen:

- **User.** User können Transaktionen innerhalb der vorgegebenen Zugangsrechten durchführen.
- **Member.** Durch das Governance Modul können Members das Konsortium verwalten und die Konfiguration modifizieren. Die Änderungen im Konsortium erfolgen erst nach einer Abstimmung zwischen den Member-Mitgliedern.

Der Speicher existiert auf jedem Knoten und wird durch eine Menge von Tabellen dargestellt [2]. Es existieren zwei Tabellentypen, und zwar Application und Governance Tabellen. Application Tabellen bewahren anwendungsspezifische Daten (diese Tabellen bleiben öfter privat). Governance Tabellen (normalerweise für alle lesbar) verwalten folgende Informationen:

- **User Tabelle.** Speichert Zertifikate und Zugangsrechte jeweiliger User (d.h. die Befehle, die sie Ausführen dürfen).
- **Member Tabelle.** Bewahrt Zertifikate, Keyshare Encryption Keys und den aktuellen Status (aktiv, nicht aktiv, accepted) jedes Mitglieds.

Zusätzlich zu den zwei Tabellen sieht CCF weitere drei Tabellen für die Verwaltung der Public-Key Infrastruktur vor. All diese Tabellen spiegeln nur den aktuellen Zustand des Clusters und der Anwendung wieder. Um die Zustandsänderungen später abspielen zu können, wird jede Transaktion auf einem Ledger gespeichert [2] (für Sicherheit und Auditing Zwecke). Eine Transaktion kann mehrere Änderungen in mehreren Tabellen umfassen. Die Änderungen in privaten Tabellen werden verschlüsselt, für public Tabellen wird nur die Integrität der Änderungen garantiert.

Governance erfolgt über JSON-RPC Protokoll und hat folgende Operationen:

- **Read.** Mitglieder mit mindestens Status “accepted” dürfen Governance Tabellen durchschauen.
- **Propose.** Ein Proposal wird von einem Governance Member vorgeschlagen und enthält Actions zur Änderung der Governance Tabellen. Solch eine Aktion kann z.B. die Erstellung eines neuen Eintrags in der Member Tabelle sein. Ein neuer Proposal wird immer in die entsprechende Tabelle mit dem Status “open” geschrieben.
- **Vote.** Hiermit können Member ihre Stimmen zu einem Proposal abgeben.
- **Complete.** Jedes Mitglied kann ein Complete durchführen, um die Abstimmung abzuschließen und Votes zu zählen. Wenn die Anzahl der Stimmen ausreichend ist (mindestens ein Quorum aller Governance Mitglieder) und ein Proposal den Status “open” hat, dann wird er angenommen.
- **Ack.** Diese Nachricht wird von jedem Mitglied regelmäßig ausgeschickt, um zu zeigen, dass er korrekt funktioniert.

CCF hat einige Mechanismen eingebaut, um eine im Vergleich mit anderen Blockchains, überdurchschnittliche Leistung zu erreichen:

- Die erste Leistungsoptimierung in CCF besteht darin, dass ein Merkle Tree statt einer Blockchain verwendet wird. Dadurch kann eine logarithmische Laufzeit O(log(n)) für den Zugriff auf die alten Blöcke gewährleistet werden.
- Zweitens wird der Baum nie erneut aufgebaut, sondern nur mit neuen Transaktionen erweitert. Die Darstellung in Form eines Baumes ist auch dadurch günstig, dass nicht alle Knoten alle Blöcke speichern müssen. Infolgedessen kann jeder Knoten für sich entscheiden, mit welcher Abstraktionsebene er arbeiten will (nur Hashes oder ganze Blöcke) und welche Transaktionen lokal aufbewahrt werden.
- Die dritte Leistungsoptimierung sind Checkpoints. Um den aktuellen Zustand schneller abfragen oder den verlorenen Zustand wiederherstellen zu können, werden zusätzlich zum Ledger, der alle Transaktionen aufbewahrt, Checkpoints gespeichert. Ein Checkpoint wird

ebenfalls in Form einer Transaktion auf dem Ledger gesichert. Dieser Mechanismus ähnelt sehr stark CQRS.

Eins-zu-eins wie im RAFT Protokoll hat CCF einen Master, der bei CCF allerdings nicht “Leader”, sondern “Primary” genannt wird. “Followers” (oder Replica) heißen bei CCF “Backups” und RAFT’s “Terms” (Amtsperioden eines Leaders) wurden in “Views” umbenannt. Die Durchführung einer Transaktion erfordert (mit wenigen Ausnahmen) dieselben Schritte wie bei RAFT:

- Primary schreibt eine Transaktion in eigenes Log
- Primary schickt diese Transaktion an alle Backups
- Ein Quorum aller Knoten schreibt die Transaktion in ein eigenes Log. Jeder dieser Knoten schickt ein ACK an den Primary.
- Primary erhöht Commit-Index und aktualisiert den Merkle Tree. Ab diesem Zeitpunkt gilt eine Transaktion als sicher gespeichert. Die Aktualisierung des Merkle Tree ist der einzige Schritt, der in RAFT nicht vorgesehen ist.

Die Wahl eines Primary erfolgt ebenfalls eins-zu-eins wie im RAFT Algorithmus. Die Schritte nach der Wahl eines Primarys sind wie folgt definiert:

- Primary schickt View-Change Transaktionen an Backups
- Genauso wie bei allen anderen Transaktionen, müssen diese auch durch ein Quorum aller Knoten unterstützt werden (durch ACKs)
- Ab dem Zeitpunkt, ab dem die Transaktion unterstützt ist, gelten alle Transaktionen in der bisherigen View als gültig

Der Zustand jedes Knoten in CCF besteht aus folgenden Komponenten:

- Aktuelle View
- Last Seen Zeit des aktuellen Primarys
- Ledger
- Log
- Commit-Index

Zusammenfassend kann man sagen, dass CCF eine kommerziell ausgerichtete Mischung von RAFT, CQRS und TLS für Geschäftsprozesse mit mangelndem Vertrauen zwischen involvierten Partnern ist. Der Vergleich von CCF und RAFT ist in der Tabelle I zu finden.

B. Azure Confidential Ledger

Eine kommerzielle Implementierung des CCF ist Azure Confidential Ledger (ACL) [5]. Laut Azure Dokumentation ist ACL ein Service zur Verwaltung sensibler Daten auf Basis von permissioned Blockchain [5]. Besonderheiten des ACLs sind Immutabilität und Append-Only Ledger Operationen (Abbildung 2). Außerdem behauptet Microsoft, dass keiner (nicht einmal Microsoft) die Kontrolle über das Netzwerk hat [5]. ACL beruht auf TCBs (Trusted Computing Base) [4]. TCB ist eine Menge sicherheitskritischer Hardware-, Firmware- und/oder Software-Komponenten und wird in modernen Betriebssystemen so klein wie möglich gehalten. John Rushby definiert TCB z.B. als die Zusammenschmelzung eines sicheren Betriebssystems und sicherer Prozesse. ACL ist für Anwendungsfälle attraktiv, wo kritische Metadaten nicht modifiziert werden sollen, inklusive Archivierung und Einhaltung gesetzlicher Vorschriften [5]. Hier ist eine kurze Liste der Sachen, die man auf ACL speichern kann:

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

Ledger Daten werden auf dem Azure Blob Storage verschlüsselt oder unverschlüsselt (nach Bedarf) gespeichert. Zurzeit ist ACL nur im US East Region verfügbar.

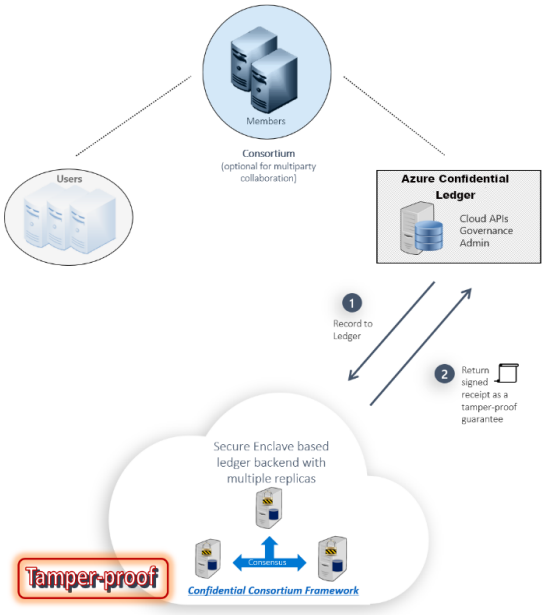

Abbildung 2. Architektur des ACL.

IV. Private oder permissioned Blockchains auf Basis Kubernetes

Kubernetes ist ein Open-Source-System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von Container-Anwendungen, das ursprünglich von Google entworfen und an die Cloud Native Computing Foundation (CNCF) gespendet wurde [6]. Es zielt darauf ab, eine „Plattform für das automatisierte Bespielen, Skalieren und Warten von Anwendungscontainern auf verteilten Hosts“ zu liefern [6]. Es unterstützt eine Reihe von Container-Tools, einschließlich Docker. Die Orchestrierung mittels Kubernetes wird von führenden Cloud-Plattformen wie Microsofts Azure, IBM Cloud, Red Hats OpenShift, Amazons EKS, Googles Kubernetes Engine und Oracles OCI unterstützt [6].

Auf Basis von Kubernetes können nur private Blockchain Netze aufgestellt werden, da die Verwaltung eines Clusters normalerweise zentralisiert ist [6]. Obgleich das Teilen eines Clusters zwischen mehreren Teilnehmern innerhalb einer Organisation eine gängige Praxis ist, geschieht das zwischen mehreren Organisationen eher selten und wenn ja, dann hauptsächlich bei Forschungsprojekten [6].

	Raft	**CCF**
Multimaster	-	-
Anwendungsfall	Replication / Fehlertoleranz in verteilten Systemen	Mangelndes Vertrauen in Verteilten Systemen
Zustand	nur Log mit Transaktionen	Log und aktueller Zustand jeder Tabelle
Kern des Protokolls	Log	Log und der Code, der den Log modifiziert
Master	Leader	Primary
Anhänger	Follower	Backup
Amtsperiode des Leaders	Term	View
Lese-Operationen	alle Knoten	alle Knoten
Schreib-Operationen	nur Leader	nur Primary (aka Leader)
Verwaltung der Zugangsrechte	-	+
Verschlüsselung	-	+
Integrität des Logs	-	Merkle Tree statt einer Blockkette, Checkpoints werden auf dem Ledger gesichert
Technologien auf Basis dieses Algorithmus	• HashiCorp Consul • RethinkDB • Hazelcast • CockroachDB • etcd	• CCF Open Source • Azure Confidential Ledger

Tabelle I
VERGLEICH DER RAFT UND CCF ALGORITHMEN

Während der Recherche wurde der Fokus auf Projekte, Blockchain Clients und Netze gelegt, die auf Basis von Kubernetes in der Cloud deployed werden können [6]. Dabei sind vor allem diese zwei Blockchains ins Visier geraten, und zwar BESU und Goquorum. Was diese Blockchains vereint ist die Tatsache, dass sie entweder als private, oder als permissioned Blockchains eingesetzt werden können.

A. BESU

Hyperledger BESU ist ein in Java geschriebener Opensource Ethereum Client [7]. Veröffentlicht wurde dieser Client im Jahr 2018 von der Firma ConsenSys [7]. Er kann auf dem Ethereum Mainnet und auch auf den Testnetzwerken laufen (Rinkeby, Ropsten, Goerli), gedacht wurde dieser Client allerdings vor allem für Enterprise Anwendungen mit der Option Zugriffsrechte für bestimmte Nutzergruppen einzuschränken [7].

Da BESU hauptsächlich für private / permissioned Anwendungsfälle vorgesehen wurde, bietet er zusätzliche zu den gewöhnlichen Funktionen eines Ethereum Clients eine API, um Metriken zu sammeln [7]. Über die Kommandozeile kann dies mit zwei Flags konfiguriert werden, nämlich –metrics-enabled und –metrics-port. Über diese API können knapp 100 Metriken gesammelt werden, unter anderem Gas Limit, Transaction Count, Difficulty, Anzahl der Knoten und viele weitere Metriken. Die Metriken können in Elastic, Grafana oder in Splunk visualisiert werden. Von Vorteil ist auch, dass BESU das OpenTelemetry Format unterstützt.

Besu unterstützt folgende Konsensprotokolle:

- **Proof of Work** (Ethash): PoW kommt auf dem Mainnet und kleinen (Pre-Production) Testnetzen zum Einsatz.
- **Proof of Authority** (QBFT, IBFT 2.0, Clique): Für neue private Netze wird QBFT empfohlen, zumal bei IBFT Abzweigungen und Reorganisierungen der Blockkette passieren können. Clique wird bei produktiven Netzen nicht empfohlen.
- **Proof of Stake**: Dieser Algorithmus kann auf dem Mainnet verwendet werden.

Weitere Besonderheiten von BESU:

- Eine dritte Person (external parties) hat keinen Zugriff auf die Liste von involvierten Parteien in einer Transaktion, ebenso wie den Inhalt dieser Transaktion.
- Hyperledger Besu unterstützt Schlüsselmanagement innerhalb des Clients aus Sicherheitsgründen nicht. Stattdessen muss ein Ethereum compatibles Wallet oder ein Werkzeug wie EthSigner verwendet werden. EthSigner ist ein weiteres Projekt der Firma ConsenSys für die Unterzeichnung der Transaktionen. Kryptographische Schlüssel können auf der Festplatte, in der Cloud (e.g. Azure) oder in einem Secrets Manager wie Hashicorp Vault gesichert werden.

Einer der bekanntesten Anwendungsfälle von BESU ist die pharmazeutische Industrie. Das Problem der Arzneimittelverschwendung in den USA ist eines der wichtigsten Probleme mit einer jährlichen Verschwendung von verschreibungspflichtigen Medikamenten im Wert von fast 2 Milliarden US-Dollar.

Riesige Mengen verschreibungspflichtiger Medikamente bleiben in Kliniken, Heimen und bei einzelnen Patienten ungenutzt. Die einfachste Lösung in dieser Situation könnte darin bestehen, ungenutzte Medikamente zu entsorgen. Die Kosten dafür sind jedoch ein weiterer kritischer Punkt mit weitreichenden finanziellen Folgen. Darüber hinaus könnten die Präzedenzfälle für die Entsorgung nicht verwendeter Medikamente durch Verbrennung zu erheblichen Umweltproblemen führen. Die beste Lösung wäre in diesem Fall ein Service, über den ungenutzte Arzneimittel zurückgeben und den bedürftigen Patienten zur Verfügung gestellt werden könnten. Bei der Umsetzung dieser Lösung müssen folgende praktische Aspekte berücksichtigt werden:

- Anreiz für Apotheken Medikamente anzunehmen
- Anreiz für Patienten Medikamente zurückzugeben

- Datenschutz
- Eine umfassende Kontrolle über die Qualität gespendeter Medikamente

Ein Projekt in dem diese Idee umgesetzt wurde ist “Save Pharmaceutical”. Dieser Service nutzt BESU, um die oben genannte Parteien miteinander zu verbinden und einen Anreiz für die Wiederverwendeung der Medikamente zu schaffen.

B. Go Quorum

Go Quorum ist ein weiteres Open Source Projekt der Firma ConsenSys [8]. Go Quorum ist ein Fork von Go Ethereum (der offizielle Ethereum Go Client). Laut der Beschreibung hat Go Quorum genau dieselbe Funktionalität wie BESU [8]. Außerdem existiert eine Bibliothek “web3js-quorum”, die mit diesen beiden Clients kompatibel ist [8]. Genauso wie bei BESU kann ein Go Quorum Testnetz auf einem Kubernetes Cluster aufgestellt werden.

LITERATUR

[1] Github.com. The Confidential Consortium Framework. https://learning.oreilly.com/library/view/modelling-and-simulation/9780857090782/ (abgerufen am 03.06.2022).

[2] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cedric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olga Ohrimenko, Felix Schuster, Roy Schuster, Alex Shamis, Olga Vrousgou, Christoph M. Wintersteiger. CCF: A Framework for Building Confidential Verifiable Replicated Services. Microsoft Research Microsoft Azure (April 2019). https://github.com/microsoft/CCF/blob/main/CCF-TECHNICAL-REPORT.pdf (abgerufen am 03.06.2022).

[3] Wikipedia. The Free Encyclopedia. K-onfidential Consortium Framework“. https://en.wikipedia.org/wiki/Confidential_Consortium_Framework (abgerufen am 03.06.2022).

[4] Wikipedia. The Free Encyclopedia. Trusted computing base“. https://en.wikipedia.org/wiki/Trusted_computing_base (abgerufen am 03.06.2022).

[5] Microsoft Azure. Azure Confidential Ledger. Architecture. https://docs.microsoft.com/en-us/azure/confidential-ledger/architecture (abgerufen am 03.06.2022).

[6] Wikipedia. The Free Encyclopedia. “Kubernetes“. https://de.wikipedia.org/wiki/Kubernetes (abgerufen am 03.06.2022).

[7] Hyperledger Besu. Besu Ethereum client https://besu.hyperledger.org/en/stable/ (abgerufen am 03.06.2022).

[8] ConSensys. GoQuorum https://consensys.net/docs/goquorum/en/latest/ (abgerufen am 03.06.2022).