**Westfälische Hochschule - Gelsenkirchen Bocholt Recklinghausen - 45877 Gelsenkirchen**

Studiengang: M. Sc. Internet-Sicherheit


Sommersemester 2019

3. Fachsemester

# Automatisiertes Erzeugen eines Sovrin Validator-Nodes und des zugehörigen Clients als Container

Abgabedatum:

7. August 2020

Aktualisierungsdatum:

7. August 2020

Autor:

Tobias Spielmannn (Matr. 201222808) – tobias.spielmann@studmail.w-hs.de

Betreuer:

Prof. N. Pohlmann

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

Inhalt dieses Projektes ist es, Sovrin Stewards die Erstellung und Einrichtung eines Validator-Nodes zu erleichtern. Hierbei ist es unerheblich, ob der Node zum ersten oder zum wiederholten Male erstellt wird. Wichtig ist nur, dass bei erstmaliger Einrichtung alle öffentlichen und privaten Schlüssel, sowie sonstige Identifier an einem sicheren Ort aufbewahrt werden.
Der Validator sowie der dazugehörige Client werden in Form von zwei unabhängigen Docker Containern erstellt und zur Ausführung gebracht. Dies bewirkt eine maximale Entkopplung der beiden Komponenten.

## 1.1 Was ist Sovrin?

Sovrin ist ein dezentrales Netzwerk basierend auf der Blockchain-Technologie Hyperledger Indy zur Bereitstellung von Self-Sovereign Identity (SSI) Dienstleistungen. Es ist Teil des Web of Trust und trägt dazu bei, Prozesse und vor Allem Kommunikation im Internet vertrauenswürdiger zu gestalten. Dabei nehmen verschiedene Komponenten unterschiedliche Aufgaben innerhalb des Netzwerkes wahr. Dieses Modell macht eine zentrale Komponente unnötig. Somit sind die Daten des einzelnen Teilnehmers auf ihren eigenen privaten Endgeräten gespeichert. Lediglich die Informationen zur Überprüfung der Daten des Nutzers liegen öffentlich in diesem dezentralen Netzwerk. Jedoch ist es mit diesen öffentlichen Informationen nicht möglich, Rückschlüsse auf die privaten Informationen zu ziehen.

## 1.2 Einordnung des Vorhabens

Die in diesem Projekt erstellten Container sind Teil des Sovrin-Netzwerkes und somit Teil eines SSI-Netzwerkes. Der Validator-Node des if(is) ist im innersten Ring angeordnet. Die Aufgabe dieser Komponenten besteht darin, schreibende Zugriffe auf die Blockchain entgegenzunehmen, zu prüfen und ggf. auszuführen. Das bedeutet sie sind wesentlicher Bestandteil der Integrität des Sovrin-Netzwerks.

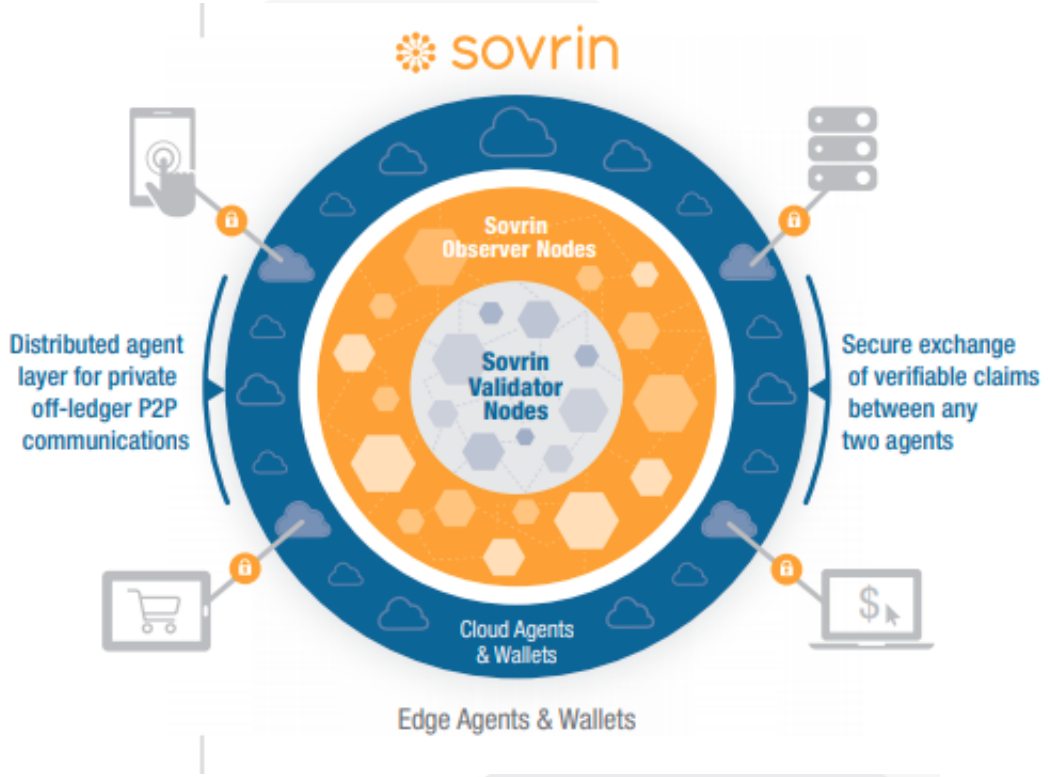


Abbildung 1.1: Sovrin Ring Model [1]

## 1.3 Relevanz des Projektes

Dezentrale Netzwerk, oder in diesem speziellen Fall SSI-Netzwerke, haben die Hauptaufgaben die Integrität von Daten, Kommunikationen und derer Beteiligten sicherzustellen. Hierfür sind die Validator-Nodes maßgeblich verantwortlich. Deren eigene Integrität und fehlerfreie Arbeit ist unerlässlich. Aus diesem Grund sind nur ausge-

wählte Institutionen dazu berechtigt einen solchen Validator-Node im Sovrin-Netzwerk betreiben zu dürfen. Es ist eine Anerkennung auf Gegenseitigkeit. Sovrin gewinnt an Anerkennung durch die Teilhabe namenhafter Partner. Die Partner wiederum können ihr Mitwirken am Sovrin-Netzwerk ihrem Portfolio imagefördernd hinzufügen. Daher ist die Teilnahme des if(is) am Sovrin-Netzwerk durch das Betreiben eine Validator-Nodes ein Gewinn für beide Parteien.

# Kapitel 2

# Projektumgebung und -infrastruktur

Das Projekt und die erstellten Container müssen auf verschiedenen Ebenen in verschieden Umgebungen und Infrastrukturen eingefügt werden. Sehr offensichtlich ist die Eingliederung in das Sovrin-Netzwerk als Validator-Node, da dies der Kerninhalt des Projektes ist. Doch ein dezentrales Netzwerk wie Sovrin bringt auch die Eigenschaft der Heterogenität mit sich. Das bedeutet, dass die Betreiber der einzelnen voneinander unabhängig laufenden Komponenten ebenso unabhängig voneinander ihre internen IT-Infrastrukturen betreiben. Daher müssen die Container auch in die interne IT-Infrastruktur des if(is) eingegliedert werden. Diese beiden Aspekte gilt es in diesem Kapitel zusammenzubringen.

## 2.1 Einbindung in das Sovrin-Netzwerk

Das Einbinden eines Validator-Nodes in das Sovrin-Netzwerk ist gerade für neue Stewards nicht ganz einfach. Angegfangen mit dem Punkt, dass grundsätzlich zwei unabhängige Systeme eingerichtet werden müssen – der Validator-Node selbst, sowie ein Client zur Kommunikation mit dem Node. Des Weiteren kommen eine Menge neue Begriffe und Werte in der Anleitung vor. Mehr dazu in den entsprechenden Kapiteln.
Da Sovrin auf Hyperledger Indy basiert, ist dies auch die Technologie, mit der man sich am meisten auseinandersetzen muss. Die Einbindung in das Sovrin-Netzwerk findet nahezu vollständig über die Indy-Cli statt. Die eigentliche Funktionalität für Sovrin wird durch das Installieren eines entsprechenden Paketes hinzugefügt.

## 2.2 Einbindung in die interne Infrastruktur

Das Einbinden des Validator-Nodes und des Clients in die interne Infrastruktur wird unabhängig von deren konkreten Aufgaben umgesetzt. Einzige Voraussetzung in diesem Fall ist das Betriebssystem Linux Ubuntu Version 16.03. Diese eine Voraussetzung stellt eine Herausforderung dar, da im if(is) keine Server mit einem Ubuntu-Betriebssystem betrieben werden. Hieraus ergibt sich eine weitere Notwendigkeit für eine Abstraktion von der ursprünglichen Hardware. Hier steht eine virtuelle Maschine und eine Containerumgebung zur Auswahl. Eine virtuelle Maschine stellt einen zwar virtueller aber in der Funktionalität und Ressourcenverbrauch einen vollwertigen Rechner dar. Eine Containerumgebung hingegen ist etwas schlanker gestaltet. Diese wird in der Regel für die Ausführung einer einzigen Aufgabe oder auch für einen einzigen Prozess angewendet. Beide Varianten sind für dieses Projekt theoretisch geeignet. Für die Praxis ist die Containerumgebung jedoch passender. Die beiden Komponenten haben jeweils eine Aufgabe wahrzunehmen, was für die Verwendung der Containerumgebung spricht.

Diese größere Flexibilität einer Containerumgebung bietet mehrere infrastrukturelle Möglichkeiten für den Betrieb der beiden Knoten bzw Container. Um Ressourcen zu sparen, empfiehlt es sich beide Container auf einem Host zur Ausführung zu bringen. Diesem Host sollten zwei verschiedene IP-Adressen zugewiesen werden, wovon jeweils eine auf den Client und eine auf den Validator gemappt wird. Anschließend können beide Container betrieben werden. Während der Validator-Node durchgehend aktiv sind wird, ist das Starten des Client lediglich bei Bedarf notwendig.

# Kapitel 3

# Projektumsetzung

Die Projektumsetzung lässt sich grob in zwei Phasen einteilen. Zunächst wurden beide Komponenten – Validator-Node und Client – direkt und ohne Kapselung durch eine Containerumgebung erstellt. Anschließend wurde auf Basis der so gewonnen Erfahrung je ein Container-Image erstellt, und der Container probehalber eingerichtet.

## 3.1 Erstellung der Nodes ohne Docker

Die direkte Erstellung ohne Containerumgebung stellt die erste Phase der Projektumsetzung dar. Auf diese Weise ist es möglich Erfahrungen in der Einrichtung und dem Betrieb der Nodes zu sammeln, da hier ein direkter Zugriff zu jeder Zeit möglich ist. Auf zwei virtuellen Maschinen wurde das Betriebssystem Linux Ubuntu in der benötigten Version 16.03. installiert. Für die Erstellung des Validator-Nodes und des zugehörigen Clients stellt die Sovrin-Foundation eine Guideline für neue Stewards zur Verfügung.[2]

**Allgemeine Vorbereitungen** Grundlegende Voraussetzungen sind zwei Linux Maschinen mit Ubuntu Version 16.03 und einer Internetverbindung. Als nächstes muss die Konfiguration der Firewall des Validators so angepasst werden, dass die Ports 9701 und 9702 geöffnet sind. Hierbei nutzt der Validator den Port 9701 für die Kommunikation mit dem Sovrin-Netzwerk und 9702 für die Kommunikation mit seinem Client. Auch am Client sollte ein Port für die Kommunikation mit dem Node freigegeben werden, dieser kann hier jedoch frei gewählt werden. Abschließend sind die zugewiesenen IP-

Adressen der Komponenten wichtig, da diese im Laufe der Einrichtung jeweils in beiden Komponenten eingetragen werden müssen.

**Vorbereitung des Client** Die Vorbereitungen werden in der Guideline in den Punkten 3.1.1 bis 3.1.3 beschreiben. Angefangen mit dem Client werden nach einem Systemupdate die benötigten Pakete installiert. Hierunter befindet sich sowohl Standardsoftware, als auch ein spezielles Paket. Hierbei handelt es sich um das Paket für die indy-cli. Laut Guideline sollen die Standardpackete "software-properties-common", "pwgen" und "python-software-properties" installiert werden. Zusätzlich ist es notwendig die Pakete "apt-transports-https" zur Übertragung von externen Paketen per HTTPS und "ca-certificates" mit einer Liste von häufig genutzten Zertifizierungstellen zu installieren. Nach dem Hinzufügen zwei weiterer spezieller Sovrin-Repositories kann auch die Cli von Hyperledger Indy installiert werden.
Bevor die Cli gestartet werden kann, muss der Acceptance Mechanism festgelegt werden. Hierzu wird in einer separaten JSON-Datei festgelegt, auf welche Art und Weise das Transaction Author Agreement (TAA) akzeptiert wird.

```
1  {
2   "taaAcceptanceMechanism": "for_session"
3  }
```

In diesem Fall wird der Acceptance Mechanism auf "for_session" festgesetzt. Somit muss das TAA bei jedem Verbindungsaufbau in den Pool erneut akzeptiert werden.
Zum Schluss der Vorbereitungen müssen noch drei Konfigurationsdateien für die Indy-Cli heruntergeladen werden. Diese können unmittelbar ohne Anpassungen genutzt werden.

**Erster Start des Client** Nach dem alle Pakete installiert und benötigten Dateien heruntergeladen sind, ist der Client nahezu bereit für den ersten Start.

*pwgen -s 32 -1 » pwgen.txt*

Mit diesem Befehl kann ein neuer Key-Seed erzeugt und in einer .txt-Datei zwischengespeichert werden. Sollte dies schon existieren, kann dieser Schritt übersprungen werden. Dieser Seed bildet später die Berechnungsgrundlage für Generierung des privaten und öffentlichen Schlüsselmaterials des Clients.

*indy-cli –config <path/to/config>/cliconfig.json*

Mit diesem Befehl wird die Indy-Cli unter Anwendung der zuvor erstellen JSON-Datei gestartet. Anschließend wird innerhalb dieser Cli die weitere Einrichtung vorgenommen.

*indy> pool create buildernet gen_txn_file=pool_transactions_builder_genesis*

Zu Beginn muss der Pool erstellt werden. Die referenzierte Datei enthält hierzu die benötigten Konfigurationsanweisungen.

*indy> wallet create buildernet_wallet key=<key_for_wallet>*

Anschließend muss die Wallet erstellt werden, in welcher das private Schlüsselmaterial für diesen Pool aufbewahrt werden soll. Hierbei kann sowohl die Bezeichnung "buildernet_wallet" als auch der key frei gewählt werden. Beide Werte werden für die weitere Verwendung benötigt.

*indy> pool connect buildernet*

Der Client ist jetzt bereit für den ersten Verbindungsaufbau zum Pool. Während der Ausführung dieses Befehls muss das TAA akzeptiert werden. Gemäß dem gesetzten Wert in der JSON-Datei muss dies bei jedem Ausführen des Befehls wiederholt werden.

*pool(buildernet):indy> wallet open buildernet_wallet key=<key_for_wallet>*

Um die Wallet innerhalb des Pools verwenden zu können, muss diese nach dem erfolgreichen Verbindungsaufbau geöffnet werden. Hierzu ist das bei der Erstellung der Wallet gesetzte Passwort notwendig.

*pool(buildernet):buildernet_wallet:indy> did new seed=<key-seed>*

Hier wird der Anfangs erwähnte und ggf. neu generierte Key-Seed benötigt. Dieser wird als Parameterwert in den Befehl eingefügt. Die Ausgabe zeigt die generierte DID und den VERKEY. Diese Werte müssen nicht explizit zwischengespeichert werden, sie können in der Wallet eingesehen werden.

*pool(buildernet):buildernet_wallet:indy> did list*

Mit Hilfe dieses Befehls können alle in der Wallet gespeicherten DIDs mit den dazugehörigen VERKEYs angezeigt werden. Daher ist ein Zwischenspeichern in einer separeaten Datei nicht notwendig.

**Vorbereitung des Nodes** Die Vorbereitung des Nodes wird in der Guideline in Abschnitten 3.2.2 und 3.2.3 beschrieben. Diese Schritte sind sehr vergleichbar mit denen zur Vorbereitung des Clientes. Zu Beginn wird vom Ubuntu Keyserver Schlüsselmaterial

heruntergeladen. Anschließend werden Standard-Pakete installiert, inkl der zusätzlich zur Anleitung benötigten Pakete "ca-certificates" und "apt-transport-https". Nun kann ein sorvin-spezifisches Repositorie referenziert und das darin enthalte Sovrin-Pakete installiert werden. Nach der Änderung einer Netzwerkkonfiguration können analog zum Client Konfigurationsdateien heruntergeladen werden.

**Erster Start des Nodes** Nach den Vorbereitungen kann der Node zum ersten Mal gestartet werden.

*sudo -i -u indy init_indy_node <ALIAS> <node_ip> <node_port> <client_ip> <client_port> » init-indy.txt*

Mit diesem Befehl wird Node als Indy-Netzwerkknoten initialisiert. Ihm wird neben seinem Alias und der eigenen Netzwerkkonfiguration auch die Verbindung zum Client bekannt gegeben. Diese Daten müssen im Vorfeld ermittelt werden. Genaueres dazu befindet sich im Absatz Allgemeine Vorbereitungen. Die während der Ausführung generierten Schlüssel und deren Proofs werden in der Ausgabe angezeigt. Da diese Daten von großer Wichtigkeit sind, sollten diese Ausgabe unbedingt bspw. in eine .txt-Datei umgeleitet werden. Bei erster Einrichtung des Validator-Nodes und damit der Neugenerierung von neuem Schlüsselmaterial muss dieses an die Sovrin-Foundation weitergegeben werden. Wenn bei der Regenerierung der selbe Seed genutzt wird, ist dies nicht notwendig, da hierbei auch das selbe Schlüsselmaterial wieder erzeugt wird. Ab diesem Punkt müssen die benötigten Schlüssel zwingend im Sovrin-Netzwerk bekannt sein, da ein Verbindungsaufbau mit dem Pool anderenfalls nicht erfolgreich sein würde.

Als vorerst letzte Aktion auf dem Validator-Node gilt es zu überprüfen, ob die aktuellste Version des Sovrin-Paketes installiert ist und diese ggf. manuell nachträglich zu installieren.

**Abschließende Verbindung des Nodes mit dem Netzwerk und Start des Nodes** Im letzten Teil muss der Node mit dem Pool verbunden werden. Dies geschieht über den Client.

*indy-cli –config <path/to/config>/cliconfig.json*
*indy> pool connect buildernet*
*pool(buildernet):indy> wallet open buildernet_wallet key=<key_for_wallet>*

Zu Beginn muss der Client in Form der Indy-Cli gestartet, mit dem Pool verbunden, und die Wallet geöffnet werden.

`pool(buildernet):buildernet_wallet:indy> ledger node target=<node_identifier> node_ip=<validator_`

Mit diesem Befehl wird der Client mit dem Node zusammengeführt. Die IP-Adressen und Ports sind identisch mit dem Befehl zur Einrichtung des Nodes. Der Identifier, sowie der BLS-Key und der BLS-POP können aus der Ausgabe des init_indy_node Befehls entnommen werden.

*sudo systemctl start indy-node*

Mit diesem Befehl kann von dem Validator-Node selbst aus der Indy-Knoten nun gestartet werden.

## 3.2 Erstellung der Nodes als Container

Nach dem der Validator-Node und der zugehörige Client unmittelbar als einzelne Linuxmaschinen aufgesetzt und lauffähig gemacht wurden, konnten diese in eine Containumgebung überführt werden. Die Reihenfolge der auszuführenden Schritt ist sehr vergleichbar zum vorherigen Abschnitt, müssen jedoch mit einer anderen Tatik umgesetzt werden. Eine Containerumgebung bietet die Möglichkeit die grundlegende Initialiseriung automatisiert vorzunehmen.

**Client Dockerfile** Das Client-Dockerfile[3] zeigt alle Vorbereitungsmaßnahmen der Kapitel 3.1.1 bis 3.1.3 einschließlich des Sovrin-Guideline für Stewards. [2] Die hier zu sehenden Befehle erfüllen den selben Zweck wie die händische Eingabe im vorherigen Kapitel.

*docker build -t <tag> <path/to>/Dockerfile*

Mit diesem Befehl wird ein Docker-Container auf Basis dieses Dockerfiles gebaut. Hierbei werden die folgenden Aktionen ausgeführt:

- Schlüsselmaterial vom Ubuntu-Keyserver herunterladen
- Standardpaket nach Anleitung ergänzt durch die zusätzlich benötigten Pakete
- Referenzieren eine sovrin-spezifischen Repositories
- Installation der Indy-Cli aus dem soeben hinzugefügten Repositorie

- Erstellung eines Arbeitsbereiches in Form eines neuen Verzeichnisses und Navigation hinein
- Kopieren der JSON-Datei mit dem Acceptance-Mechanism in den Arbeitsbereich im Container
- Herunterladen der Konfigurationsdateien in den Arbeitsbereich

Bis auf die Erstellung eines neuen Verzeichnisses sind alle Schritte identisch zur händischen Eingabe der Befehel aus dem vorherigen Kapitel.

```
1  # Dockerfile to build a Container for a sovrin client node
2
3  # container will base on ubuntu 16.04
4  FROM ubuntu:16.04
5
6  # install node packages
7  RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-
       keys CE7709D068DB5E88
8  RUN apt-get update \
9    && apt-get install -y software-properties-common python-software-
         properties apt-transport-https ca-certificates pwgen \
10   && rm -rf /var/lib/apt/lists/*
11
12 RUN add-apt-repository "deb https://repo.sovrin.org/sdk/deb xenial
       stable"
13 RUN apt-get update \
14   && apt-get install -y indy-cli \
15   && rm -rf /var/lib/apt/lists/*
16
17 # add accpetance_mechanism
18 RUN mkdir /workspace
19 WORKDIR /workspace
20 COPY acceptance_mechanism.json ./cliconfig.json
21
```

```
22 # get genesis files
23 ADD https://raw.githubusercontent.com/sovrin-foundation/sovrin/
       master/sovrin/pool_transactions_builder_genesis
       pool_transactions_builder_genesis
24 ADD https://raw.githubusercontent.com/sovrin-foundation/sovrin/
       stable/sovrin/pool_transactions_sandbox_genesis
       pool_transactions_sandbox_genesis
25 ADD https://raw.githubusercontent.com/sovrin-foundation/sovrin/
       stable/sovrin/pool_transactions_live_genesis
       pool_transactions_live_genesis
```

Listing 3.1: Client-Dockerfile

**Validator Dockerfile** Das Validator-Dockerfile beinhaltet die Befehle die im vorherigen Kapitel manuell eingegeben und ausgeführt wurden. Somit werden hier die Schritte aus dem Kapiteln 3.2.1 bis Mitte 3.2.2 angewendet. Auch dieses Dockerfile wird über den build-Befehl (siehe oben) erstellt. Die hier ausgeführten Befehle sind wie beim Client, aus der Anleitung übernommen und in Dockerfile-Syntax übertragen.

```
# Dockerfile to build a Container for a Sovrin Validator Node

# based on ubuntu 16.04
FROM ubuntu:16.04

# Exposing only needed Ports
EXPOSE 9701/tcp
EXPOSE 9702/tcp

# install sovrin packages and update
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
    CE7709D068DB5E88 \
  && apt-get update \
  && apt-get install -y software-properties-common apt-transport-
     https ca-certificates curl pwgen \
  && add-apt-repository "deb [arch=amd64] http://repo.sovrin.org/
     deb xenial stable" \
  && apt-get update \
  && apt-get install -y sovrin
ADD indy_config.py /etc/indy/indy_config.py

USER indy
RUN mkdir /var/lib/indy/net3
WORKDIR /var/lib/indy/net3
ADD https://raw.githubusercontent.com/sovrin-foundation/sovrin/
    master/sovrin/domain_transactions_builder_genesis
    domain_transactions_genesis
ADD https://raw.githubusercontent.com/sovrin-foundation/sovrin/
    master/sovrin/pool_transactions_builder_genesis
    pool_transactions_genesis

USER root
```

Listing 3.2: Validator-Dockerfile

# Kapitel 4

# Probleme/Herausforderungen im Projektablauf

Auch wenn die grundlegende Aufgabenstellung realtiv einfach formuliert werden kann: Automatisiertes Erstellen eines Sovrin Validator-Nodes mit dem zugehörigen Client. Jedoch haben sich im Laufe der Umsetzung einige Herausforderungen ergeben, sowohl bei der Erstellung der Nodes als Linux-Maschinen, als auch in der Überführung in die Containerumgebung.

## 4.1 Probleme bei Erstellung der Nodes

Auch wenn die Sovrin-Guideline für Stewards [2] bereits realtiv ausführlich und informativ formuliert ist, reicht diese nicht vollständig aus. Um die Rolle und topologische Position des zu erstellenden Validator-Nodes besser einordnen zu können, hilft das Sovrin-Whitepaper [1]. Das größte Potential für Verwirrung bieten die verschiedenen Arten von Schlüsselmaterial, welche im Laufe des Projektes zum Einsatz kommen.

## 4.2 Probleme im Zusammenhang mit Containerumgebung

Docker ist eine Abstraktionsebene zwischen einer physischen und virtuellen Maschine. In dieser Eigenschaft liegt die Herausforderung. Ein Dockerfile übernimmt die auto-

matisierte und damit leicht reproduzierbare Initialisierung des Containers. Dies erreicht jedoch schnell ihre Grenzen. So kann die Installation von Paketen oder das Kopieren von Dateien in den Container automatisiert erfolgen.

**Die Änderung einer Datei** innerhalb des Containers erfordert einen kleinen Workaroud, welcher sind sofort offensichtlich ist. Diese Änderung per Skript zu automatisieren ist relativ umständlich, da dieses erst bei der ersten Ausführung des Containers möglich ist. Einfacher ist es hier den Container mit Originaldatei einmal auszuführen und diese Datei auf dem Host zu speichern. Anschließend wird diese Datei entsprechend angepasst und per Dockerfile wieder in den Container hinein kopiert.

**Automatisiertes Einrichten nach dem ersten Start** ist ebenfalls nicht so einfach umzusetzen. Es ist möglich als Entrypoint ein Skript aufzurufen. Dieses wird jedoch bei jedem Start aufgerufen. Somit muss hier unterschieden werden können, ob es sich um den ersten oder einen wiederholten Start des Containers handelt. Aus diesem Grund erfolgt in der ersten Version die Einrichtung noch manuell durch den Anwender.

# Kapitel 5

# Zusammenfassung und Ausblick

Dieses abschließende Kapitel gibt eine Zusammenfassung und ein Fazit über die ausgeführten Tätigkeiten. Des Weiteren soll ein Ausblick gegeben werden, welche Funktionen und Erweiterungen noch möglich sind.

## 5.1 Zusammenfassung/Fazit

Die Eingangsproblematik einer einfachen und automatisierten Einrichtung eins Validator-Nodes mit Client für das Sovrin-Netzwerk wurde grundlegend umgesetzt. Die Initialisierung der Komponenten kann automatisiert als eigenständige Container mit Hilfe eines Dockerfiles durchgeführt werden. Dies bildet die Basis für die Weiteren Maßnahmen. Auf diese Art und Weise kann bei einem Fehler im weiteren Verlauf der Einrichtung schnell die Basis wieder hergestellt werden. Die anschließende weitergehende Einrichtung des Komponenten erfolgt händisch.

Um jedoch zunächst Erfahrungem im Umgang mit den Komponenten sammeln zu können wurden diese auf vollwertigen Maschinen lauffähig gemacht. Hierzu bietet die Guideline der Sovrin-Foundation eine gute Arbeitsgrundlage. Es waren nur leichte Anpassungen, sowie die Installation von zwei zusätzlichen Paketen notwendig. Die Komponenten konnten so schnell eingerichtet und genutzt werden.

Die anschließende Übertragung der gewonnen Erfahrungen auf die Einbettung der Komponenten in jeweils eine Containerumgebung stellte die eigentliche Herausforderung dar. Da ein Container zwar eine Abstraktion von der Hardware des Hosts aber keine vollwertige Maschine darstellt, musste für ein paar Schritte ein Workaround geschaffen werden. So musste zum Beispiel das Hinzufügung von Dateien in den Container entweder im Dockerfile geschehen, oder bei laufendem Container händisch erfolgen. Auch notwendige Anpassungen von Konfigurationsdateien waren nicht ohne Weiteres möglich. Des Weiteren stellte die automatisierte Einrichtung der Komponente nach dem ersten Start eine Herausforderung dar. Hier muss der aktuelle Status der Einrichtung festgestellt, interpretiert und ggf. fortgesetzt werden.

Alles in Allem ist es gelungen die Probleme zu lösen. So ist eine zumindest teilweise automatisierte aber in jedem Falle einfachere Lösung zur Einrichtung der Komponenten entstanden.

## 5.2 Ausblick

Wie jedes Projekt hat auch dieses einen abgesteckten inhaltlichen Rahmen. Dieser beinhaltet in diesem Fall die grundlegende Vereinfachung der Einrichtung eins Sovrin-Validator Nodes und des zugehörigen Clients. Diese Anforderungen wurden grundlegend umgesetzt. Jedoch besteht in der Regel immer Bedarf und/oder Potential für Erweiterungen und Verbesserungen.
Der größte Punkt für eine Erweiterung ist die automatisierte Einrichtung der Komponenten nach dem Start des Containers. Dazu muss grundlegend festgestellt werden können, ob und welcher Teil der Einrichtung noch durchgeführt werden muss. Das bedeutet es muss eine Möglichkeit geschaffen werden, den aktuellen Status des Knotens zu erfassen und zu interpretieren. Zusätzlich müssen teilweise die Ausgaben einzelner Befehle interpretiert und partiell weiterverarbeitet werden. Diese Maßnahmen sind automatisiert nur sehr aufwendig umzusetzen. Die Ausgaben folgen keinem vordefinierten Format. Daher ist die Anwendung von bereits vorhandenen Parsern nicht möglich.
Neben diesem größten Punkt gibt es noch zahlreiche kleinere Punkte die mit verhältnismäßig viel Aufwand anwenderfreundlicher gestaltet werden können.

# Literaturverzeichnis


[1] Sovrin Foundation. Sovrin™: A protocol and token for selfsovereign identity and decentralized trust; a white paper from the sovrin foundation. https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf, Jan. 2018. zugetriffen am 21.07.2020.

[2] Sovrin Foundation. Sovrin steward validator preparation guide. https://docs.google.com/document/d/18MNB7nEKerlcyZKof5AvGMy0GP9T82c4SWaxZkPzya4/edith Aug. 2019. zugegriffen am 20.07.2020.

[3] Docker Inc. Dockerfile reference. https://docs.docker.com/engine/reference/builder/, 2013. zugegriffen am 07.08.2020.