Trust Management For A Decentralized Service Exposure Marketplace

A Service Exposure Perspective

Ahmed Aly Ahmed Aly Beder

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo, Finland July 30, 2020

Supervisors

Prof. Antti Yla-Jaaski
Prof. Panagiotis Papadimitratos

Advisors

Mohammad Khodaei
Aleksandra Obeso Duque

Aalto University
School of Science

Copyright © 2020 Ahmed Aly Ahmed Aly Beder

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ahmed Aly Ahmed Aly Beder

Title Trust Management For A Decentralized Service Exposure Marketplace: A Service Exposure Perspective

Degree programme Department of Computer Science

Major Security and Cloud Computing **Code of major** SCI3084

Supervisors Prof. Antti Yla-Jaaski
Prof. Panagiotis Papadimitratos

Advisors Mohammad Khodaei
Aleksandra Obeso Duque

Date July 30, 2020 **Number of pages** 54+11 **Language** English

Abstract

Enabling trust between entities to collaborate, without the necessity of a third-party mediator is a challenging problem. This problem is highlighted when the collaboration involves a complicated process, spans multiple systems, and encompasses a large number of entities. This is the case in a decentralized service exposure marketplace. In this work, we design and implement a Proof-Of-Concept (PoC) suite of services to enable a blockchain to become the anchor of trust for a decentralized service exposure marketplace. We first formalize the necessary requirements to enable trust between a consortium of entities hosting the marketplace. We then follow with a threat model against the identified requirement, highlighting misbehaviour from the different entities. Finally, we propose a model, Trust Engine, which facilitates the trust management process and mitigates the identified threats. We showcase a proof-of-concept of our model, utilizing a combination of smart contracts (hyperledger fabric), blockchain, and service mesh technology (Istio). The Trust Engine successfully identifies the misbehaviour, documents it in the blockchain, and enforces polices to remediate the misbehaviour. Furthermore, we examined each component in our suggested system to identify the performance bottleneck. Lastly, we discuss the limitations of our suggested model with regards to other service mesh deployment models as well as potential future work and improvements.

Keywords Trust Management, Service Exposure, Blockchain, Service Mesh, Decentralized Marketplace

Acknowledgements

I would like to thank Prof. Panos Papadimitratos and Mohammad Khodaei of Networked Systems Security Group at KTH for advising and supervising my work. I truly appreciate their help with the report and dedicating their time to not only polish my thesis writing skills, but also help throughout working with the degree project at KTH. In addition to that, I would like to express my gratitude to Antti Ylä-Jääski from Department of Computer Science at Aalto University for helping me with the Aalto University requirements. I thank Aleksandra Obeso Duque and Remi Robert from Edge Service Exposure R&D at Ericsson for guidance and supervision throughout the project. I thank my family for constantly keeping on track and supporting me to reach my goals.

Ahmed Beder

Contents

Abstract 3

1 Introduction 2
1.1 Background and Motivation . . . 2
1.2 Problem Statement . . . 2
1.3 Purpose . . . 3
1.4 Goal . . . 3
1.5 Research Questions . . . 3
1.6 Benefits, Ethics and Sustainability . . . 3
1.7 Methodology . . . 3
1.8 Stakeholders . . . 3
1.9 Delimitations . . . 4
1.10 Outline . . . 4

2 Background and Literature Review 5
2.1 Digital Trust and Trust Management . . . 5
2.2 The Evolution of a Service Mesh . . . 5
2.2.1 From Monolith Architecture to Microservice . . . 6
2.2.2 Orchestration and Emergence of Service Mesh . . . 7
2.2.3 Features of a Service Mesh . . . 8
2.3 Istio Deployments Models . . . 9
2.3.1 Multi-cluster Deployment of Istio . . . 9
2.3.2 Multi-mesh Deployment and Service Mesh Federation . . . 11
2.4 Blockchain Technology and Smart Contracts . . . 12
2.4.1 Blockchain Technology . . . 13
2.4.2 Smart Contracts . . . 13
2.5 “Nubo" Decentralized Marketplace Overview . . . 14
2.5.1 Decentralized Marketplace Design . . . 14
2.6 Literature Review and Related Work . . . 15
2.6.1 Trust Management for Decentralized Systems . . . 15
2.6.2 Trust Management for Cloud Service Exposure Marketplace . . . 16

3 Model, Requirements and Assumptions 18
3.1 System Model . . . 18
3.1.1 Blockchain and Smart Contracts . . . 18
3.1.2 Orchestration clusters . . . 20
3.1.3 Entities and Roles . . . 20
3.2 Requirements to Establish Trustworthy Relationships . . . 21
3.3 Security Assumptions . . . 22
3.4 Adversary Model . . . 22
3.4.1 Disrupting Service Level Agreement (SLA) . . . 23
3.4.2 Bypassing Authorizations . . . 23
3.4.3 Security Properties Violation . . . 23
3.4.4 Disrupting of Availability and Resilience . . . 25

4 Implementation and Contribution 28
4.1 Description of Requirements to Establish Trust . . . 28
4.1.1 Requirements of Adherence to SLA . . . 29
4.1.2 Requirements of Authorized Observability . . . 31
4.1.3 Security properties requirements . . . 34
4.1.4 Requirements of Availability and Resilience . . . 36
4.2 Mapping Requirements to Establish Trust into Service Mesh Features . . . 38
4.3 Trust Engine Design and Proof of Concept . . . 39
4.3.1 Checker . . . 41
4.3.2 Collector . . . 41
4.3.3 Policy Enforcement . . . 41

5 Security Analysis and Performance Evaluation 42
5.1 Security Analysis . . . 42
5.1.1 Disrupting SLA . . . 42
5.1.2 Bypassing Authorization . . . 42
5.1.3 Security Properties Violation . . . 42
5.1.4 Disrupting of Availability and Resilience . . . 42
5.1.5 Attack Scenario Proof-Of-Concept (PoC) . . . 42
5.2 Performance Evaluation . . . 44
5.2.1 Overall Detection and Remediation . . . 44
5.2.2 Scalability and Bottleneck . . . 44
5.2.3 Delivering Predictable Operational Performance . . . 46

6 Discussion and Future Work 49
6.1 Limitations and Consideration for Proposed Trust Engine . . . 49
6.2 Multi-mesh and Multi-cluster Istio Deployments Architectures . . . 49
6.3 Incentives and Deterrents for Blockchain Peers . . . 49

7 Conclusion 51

A Application Programming Interface (API) documentation 56

B Configuration Files 62
B.1 Collector . . . 62
B.2 Policy Enforcer . . . 62
B.2.1 Policy Template . . . 64

List of Figures

2.1 Trust management as the continuous process . . . 6
2.2 Monolithic Architecture (MA) overview . . . 7
2.3 Microservices Architecture (MSA) overview . . . 8
2.4 Sidecar architecture for service mesh . . . 9
2.5 Service mesh . . . 10
2.6 Directions of communications in Istio. . . . 11
2.7 Multi-cluster Istio deployment using shared control plane. . . . 11
2.8 Replicated control planes . . . 12
2.9 Blockchain architecture overview . . . 13
2.10 A block in a blockchain . . . 14
2.11 Overview of “Nubo" reference architecture . . . 15
2.12 Overview of trust management systems . . . 15
3.1 Marketplace system overview . . . 18
3.2 Communication flow in a blockchain . . . 19
3.3 Entities and roles in a marketplace . . . 21
3.4 Attack tree for disrupting SLA . . . 23
3.5 Attack tree for bypassing authorizations . . . 24
3.6 Attack tree for security properties violation . . . 25
3.7 Attack tree for disrupting of availability and resilience . . . 27
4.1 Legend overview . . . 29
4.2 Requirements of adherence to SLA . . . 29
4.3 Requirements of authorized observability . . . 32
4.4 Security properties overview . . . 34
4.5 Requirements of availability and resilience . . . 36
4.6 Mapping of requirements to features of Istio . . . 39
4.7 Trust Engine system design. . . . 39
4.8 Architecture of deployed Trust Engine . . . 40
5.1 Adversary scenario for this work . . . 43
5.2 Service consumption and metric reporting . . . 43
5.3 Trust engine receiving verdict from the Blockchain and applying policy . . . 44
5.4 Time to detect misbehaviour and apply policy . . . 45
5.5 Collection of metrics time for a single Service Provider (SP) with increasing number of Service (S) . . . 46
5.6 Policy creation and applying time for a single SP with increasing number of S . . . 47
5.7 Comparison between metrics collection time for various number of SP and S . . . 47
5.8 Comparison between applying and creation time for various number of SP and S . . . 48

List of Tables

2.1 Difference between single mesh and multi-mesh deployment architecture in Istio 12
4.1 Requirements per trust set of requirements for each entity pairing 28
4.2 Functional requirement of adherence to SLA 30
4.3 Non-functional requirement of adherence to SLA 31
4.4 Authorized observability requirements 33
4.5 Security properties requirements 35
4.6 Availability and resilience requirements 37
4.7 Testbed specification overview 40
5.1 Namespaces overview 45

Acronyms

AI Artificial Intelligence
API Application Programming Interface
CA Certificate Authority
CIA Confidentiality, Integrity, and Authenticity
CSP Communication Service Provider
DevOps Development and Operation
DNS Domain Name System
DoS Denial of Service
FaaS Function as a Service
IaaS Infrastructure as a Service
IoT Internet of Things
ISO International Organization for Standardization
KPI Key Performance Indicator
MA Monolithic Architecture
MSA Microservices Architecture
NIST National Institute of Standards and Technology
PaaS Platform as a Service
PGP Pretty Good Privacy
PKI Public Key Infrastructure
PoC Proof-Of-Concept
RP Resource Provider
S Service
SC Service Consumer
SLA Service Level Agreement
SOA Service Oriented Architecture
SP Service Provider
SSH Secure Shell
SCP Smart Contract-assisted PKI
SaaS Software as a Service
VPN Virtual Protected Network

1 Introduction

A decentralized service exposure marketplace can be a boon for rapid cloud-based services commercialization. However, any decentralized eco-system suffers from limitations, security and privacy of the members of the eco-system being a focal concern. Trust between these members, therefore, becomes a leading deterrent in establishing a decentralized service exposure marketplace. This work investigates smart contracts and service meshes as potential building blocks for a trust-worthy service exposure marketplace infrastructure. The infrastructure ensures that a consortium of entities uphold a set agreed upon requirements for establishing and maintaining trust among themselves.

The upcoming sections in this chapter give a brief overview of previous work done in the trust management and service exposure area, followed by a definition of the problem, scope, and purpose of this thesis work. Finally, an outline of the thesis is presented for the reader.

1.1 Background and Motivation

A service exposure marketplace is a platform that connects a Service Provider (SP) to a Service Consumer (SC) and facilitates provisioning and life-cycle management of the services based on agreed-upon terms and conditions [46]. A service exposure marketplace has two architecture models. The first model consists of a third-party mediator who offers services from multiple SPs. The second model consists of a single provider with a limited selection of services [33]. Both models depend on a single entity, a third-party mediator or a single provider. This dependency is a single point of failure for the system which can be mitigated by a *decentralized marketplace*.

A decentralized marketplace would allow a consortium of entities to offer or provision services without a third-party mediator, or the limited selection of services. However, without a third-party mediator, enabling and managing trust between entities becomes a concern. Trust management is a process to adopt a behavior strategy for a system based on an assessment of the reliability of each entity to other entities in the system [46]. In the scope of a cloud-based service exposure marketplace, trust management is vaguely defined and still in its infancy [48]. For this work, we take a look at a In 2019, Kempf at al. [33] attempted to achieve a trustworthy virtual services marketplace, “Nubo" which leveraged blockchain technology and smart contract rather than a centralized database. The main advantage of using blockchain technology is transparency and non-repudiation. As the transactions cannot be tampered with, this allows members to establish trust without involving intermediaries. Trustworthiness, therefore, is derived from the “Nubo" system, rather than relying on the trustworthiness of providers or third-party mediators. However, blockchain alone can not guarantee the enforcement of Service Level Agreement (SLA) in the transactions [33]. How can a Service Provider (SP) guarantee the usage of its services per the SLA? Do measures and protocols for swiftly resolving disputes exist? These aspects highlight trust management as a priority in the service exposure system. These requirements are an ideal match for what service mesh technology can offer. Service mesh provides fine-grained observably into running deployments, authentication and access control, and a number of key features which would facilitate enforcement of SLA [45]. In this work, we explore the capabilities of service mesh which would enable the blockchain to become the trust anchor for a decentralized service exposure marketplace.

1.2 Problem Statement

A marketplace mediator is the anchor of trust between buyers and sellers in a centralized marketplace. A misbehaving mediator is then, a single point-point of failure for the system. This problem can be mitigated by employing a decentralized model for the marketplace. However, this poses new challenges for the system, most notably, establishing and maintaining trust between the entities of the system.

The problem at hand is the lack of an infrastructure to provide a trusted operation of a decentralized service exposure marketplace hosted by a consortium of entities. Ideally, in a cloud marketplace, a third-party mediator would guarantee that entities are conforming with an agreed-upon SLA. In a decentralized marketplace, however, there is no guarantee that entities will not misbehave; therefore, entities are not obliged to collaborate or trust each other. The problem is enabling and maintaining trust between entities without the necessity of a third-party mediator. The challenge then becomes: how to enable a decentralized system, e.g., blockchain, to be the trust anchor for a decentralized service exposure marketplace?

1.3 Purpose

The purpose of this thesis is two fold: to formalize the requirements for the trusted operation of a decentralized service exposure marketplace, and to illustrate the capabilities of state of the art technologies (service mesh and smart contracts) in enabling a blockchain to become a trust anchor for a decentralized marketplace.

To the best of current knowledge, there are no clear listing of requirements to enable the trust between entities in the context of decentralized service exposure marketplace. Thus, the first step of this work is to formalize the necessary requirements to establish trust between entities. Secondly, we explore the capabilities of service mesh in enforcing these requirements. These requirements need to be mapped to the features of service mesh technology (Istio) to examine if Istio service mesh technology could benefit the trust management in our marketplace. Lastly, we can integrate our solution to existing marketplace as “Nubo".

1.4 Goal

The goal of this degree project is to design and implement a Proof-Of-Concept (PoC) suite of services which would allow the trusted operation of decentralized service exposure marketplace. The degree project will evaluate the ability of the PoC to detect, document, and remediate any misbehaviour by entities in the system. In addition, this work evaluates the operational performance of each service in the PoC to identify the bottleneck.

1.5 Research Questions

This work aims to give the following questions:

- **Which** entities are required to model a service exposure marketplace?
- **What** are the requirements to establish trust between these entities in the context of decentralized marketplace?
- **Who** are the adversaries and **what** are their capabilities within the system?
- **To what extent**, can service mesh in collaboration with blockchain technology and smart contracts mediate the threats of the adversaries and guarantee the fulfillment of the trust requirements?

1.6 Benefits, Ethics and Sustainability

All the data is taken during the project development are open sourced. No fabrication is done while recording or reporting of results. Also, falsification in any kind such as manipulation of research materials is not practiced. Plagiarism of any process or idea is strictly discouraged in any step of the development of this project. All the materials used have been properly cited. The sustainability of this thesis work is supported by using the most popular and up-to-date software framework (Istio). Providing trust requirements for decentralized marketplace could benefit the usage of federation of cloud providers instead of relying on a third-party cloud mediator. Since the thesis work does not collect any personal data, there is no ethical issues related to improper handling of personal data.

1.7 Methodology

We select a qualitative approach for formally defining trust and trust management in the context of our system. We reference a selection of previous literature, which focus on similar trust paradigms, to establish dimensions for measuring trust. Also, we rely on a mixture of industry-standard compliance lists, such as International Organization for Standardization (ISO) 27001, previous literature, and reasoning to model the trust requirements for our system. Moreover, we employ a quantitative approach to assess the performance of the suggested deployment methodology. We simulate abuse cases against our system in a testbed and evaluate its robustness, ease of deployment, and security.

1.8 Stakeholders

The work is done at Ericsson using the internal tools and facilities of the company. The supervision and theoretical part of the work take place at Network Systems Security (NSS) laboratory at KTH.

1.9 Delimitations

The scope of the project is to assess the trust requirements in decentralized marketplace and explore the benefits of service mesh in such an architecture.

First of all, this thesis will focus on using the *Istio* framework for service mesh. The reliability of Istio tool is one of the main assumption for this work. The advantage of Istio over other frameworks as *Linkerd*, *Consul*, *Airbnb Synapse* and *Amazon App Mesh* are connecting the services, securing Confidentiality, Integrity, and Authenticity (CIA) properties of services, controlling the traffic through enforcing the policies, and monitoring of services [25], having a sophisticated documentation and constant contribution of open source community to this framework.

Secondly, we rely on correctness and validity of the blockchain solution in decentralized virtual services marketplace called “Nubo". Having the reliable blockchain solution and service exposure marketplace is another assumption for this work.

1.10 Outline

Chapter 2 provides the necessary theoretical background for the reader to get familiar the terminology of this work, as well as a literature review of similar work in this field. Chapter 3 gives a broad description for the system model, adversary model and the end goal for all entities in this ecosystem. Chapter 4 takes a closer look at formalizing the requirements of establishing trust for entities of a decentralized marketplace system as well discussing a PoC for an integration module which would help enforce these requirements. Chapter 5 discuses the performance of the PoC both in terms of operation and security. Chapter 6 further explores of the limitations of the proposed model as well as the possible future changes and improvements. Finally, chapter 7 concludes the degree project with a summary of the report and its findings.

2 Background and Literature Review

In this section we introduce the important notions that the reader needs to be familiar with before moving further to chapter 3. First of all, in section 2.1 we describe the notions of digital trust and trust management. In section 1.1, we provided a brief description of service mesh technology and blockchain technology. In section 2.2 and section 2.4, we provide a deeper dive into their background and definition. In section 2.5 we discuss “Nubo", the decentralized virtual services marketplace we expand on for this work to build a trust management system. Finally, in section 2.6 we provide a brief overview of the previous works in the field of trust management.

2.1 Digital Trust and Trust Management

The notion of trust plays a pivotal role in social, financial and cultural aspects of life. Taking a bank loan, for example, or deciding on a contractor to hire for a particular job. Trust in the aforementioned cases is based on past experience and perception, and is established through the use of legal frameworks that ensures the trustworthiness of all parties involved [63]. Boon and Holmes pay great importance to the risk aspect of trust, and define trust as the situation when entities have positive expectations about other entity's motives, while being in a risky situation [56]. Carrying over with the notion of trust, Holtmanns and Yan point out a more domain-specific notion: *digital trust*. *Digital trust* is more comprehensive compared to trust, as the establishment of trust in a digital networking environment depends on digital components of the system, not only human factor. Unlike the conventional notion of trust which relies on faith in the trustee's honesty, benevolence and perception, digital trust, is an assessment of a digital entity with regards to competence, security, and reliability [63]. From this point, the notion *trust* will be used to refer the notion *digital trust* for a convenience.

In a digital environment with a convolution of business relationships and regulations, the necessity of trust becomes clear [63]. A digital marketplace, much like its real world counterpart, requires a constant assessment of trustworthiness among all participating entities. This assessment becomes harder as more networking, computing, and virtual elements are introduced. Grandison and Sloman explain that this difficulty comes from expanding to wider range of domains and organizations with different trust relationships [31]. Therefore, the notion of *trust management* becomes important to navigate these complex trust domains and relationships.

According to Louta [46], there are two ways of defining the concept of trust management. The first concept is that trust management is a process in the system consisting of entities, where the entities can be trustworthy to each other. The second concept is that trust management is a process that assesses the reliability of the entities in the system.

Automation of trust management involves the following continuous steps according to Yan and MacLaverty [64]:

- *Trust establishment* is a process of enabling trustworthy relationship between parties.
- *Trust monitoring* is a process of collecting evidence of trustworthy relationship by one parties in regards to another party for assessing the trust.
- *Trust assessment* involves techniques and metrics to assess the trustworthiness of relationship between parties.
- *Trust control and re-establishment* is a set of practices and counter-measures to control whether the parties are not breaking the trust.

Figure 2.1 helps visualize these steps as a continuous process in a cycle.

2.2 The Evolution of a Service Mesh

A typical application consists of multiple components that operate co-dependently to fulfil a function. Take, for example, a website application. A website might contain a backend server, a database, frontend styling and scripts, third-party integrations, and a multitude of other components. Over the past decade these components shifted from existing in a single monolithic boundary to a more lightweight distributed approach [51]. In this section, we explore the shift from a monolith to Microservices Architecture (MSA).

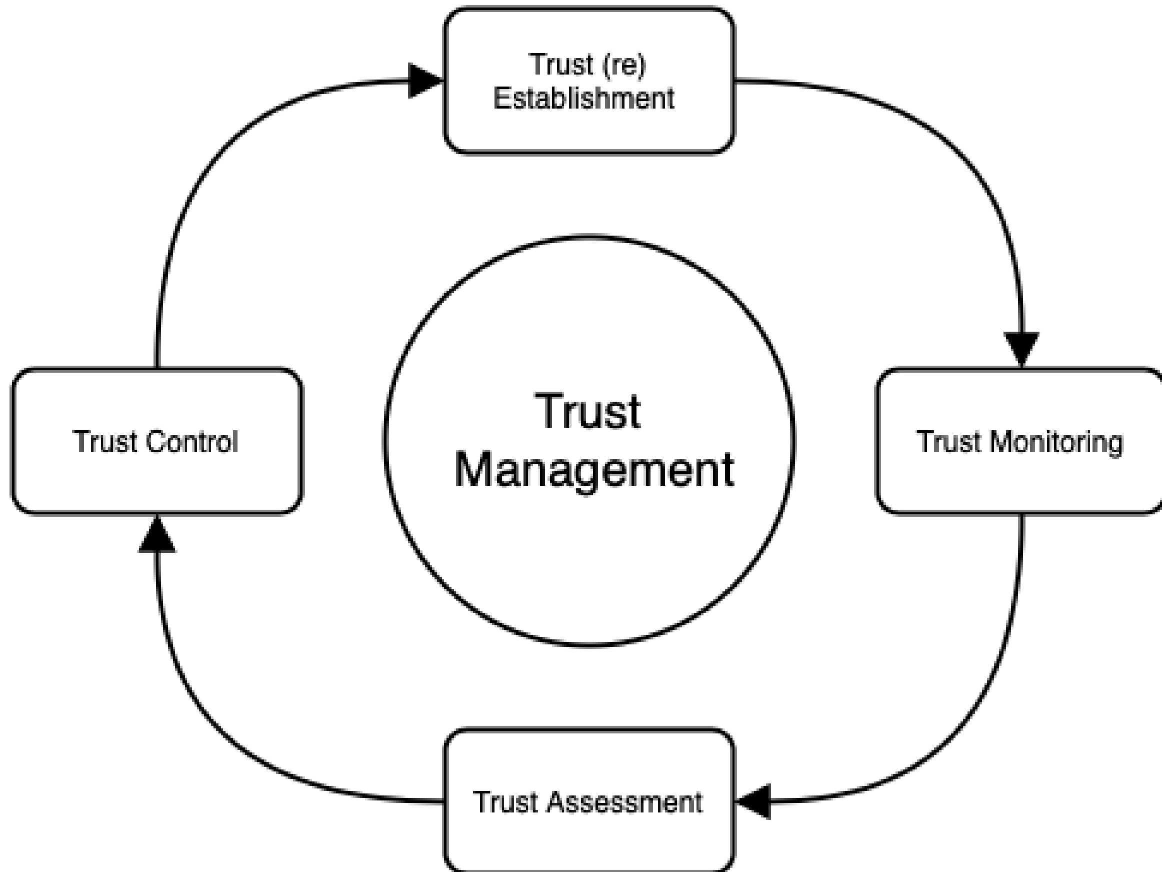

Figure 2.1: Trust management as the continuous process

2.2.1 From Monolith Architecture to Microservice

The literal definition of a *monolith* is a large single block of stone. However, outside the scope of geology, the notion of *monolith* is used for describing other processes and phenomena which are characterized by a common feature, rigidity. Earlier adaptations of cloud development employed a rigid model of service adequately named, Monolithic Architecture (MA). MA is a development technique where all the functionality of a system is placed within the same component [19]. Figure 2.2 depicts an typical MA deployment. An application instance comprises an single component which houses all of the functionality. The application is then replicated and deployed behind a load balancer allowing users to utilize it. According to Chen et al. [19], the non-complex monoliths are easy to develop, test and deploy. However, continuous addition of new functionality and requirements will inevitably lead to having a large complex MA that is overwhelming and difficult to maintain According to Li et al. [45], cloud architecture required a significant change in fundamental abstractions to address the drawbacks of the MA approach, mainly the lack of flexibility and the maintenance complexity. These demanding requirement and apparent limitations gave birth to a new approach for cloud architecture, namely, the micro service architecture.

Di Francisco et al. [22] defines MSA as an architecture model which utilizes multiple small single-function services that communicate using a lightweight communication protocol to conform with the concept of Service Oriented Architecture (SOA). Di Francesco et al. [22] then summarizes the differences between MSA and MA into three main categories: *cloud*, *system quality* and *migration ability*. These categories lead to further research to illustrate the diffrence between the two architecture types. From cloud the research perspective, MSA complies with Development and Operation (DevOps) culture, while MA distinguishes the stages of development and operations. Regarding system quality, there is no final conclusion that MSA outperforms MA in all criteria belonging system quality research perspective as security, scalability and performance. On the migration research perspective, however, MSA can used in a broad range of fields, e.g. Internet of Things (IoT). The shift towards MSA became more prominent after tech giants shifted to using it for their applications [22]. Figure 2.3 depicts a typical MSA deployment. In this model, each service serves a concrete function like logging or web APIs. These services are replicated, and traffic between them is controlled with an ingress controller.

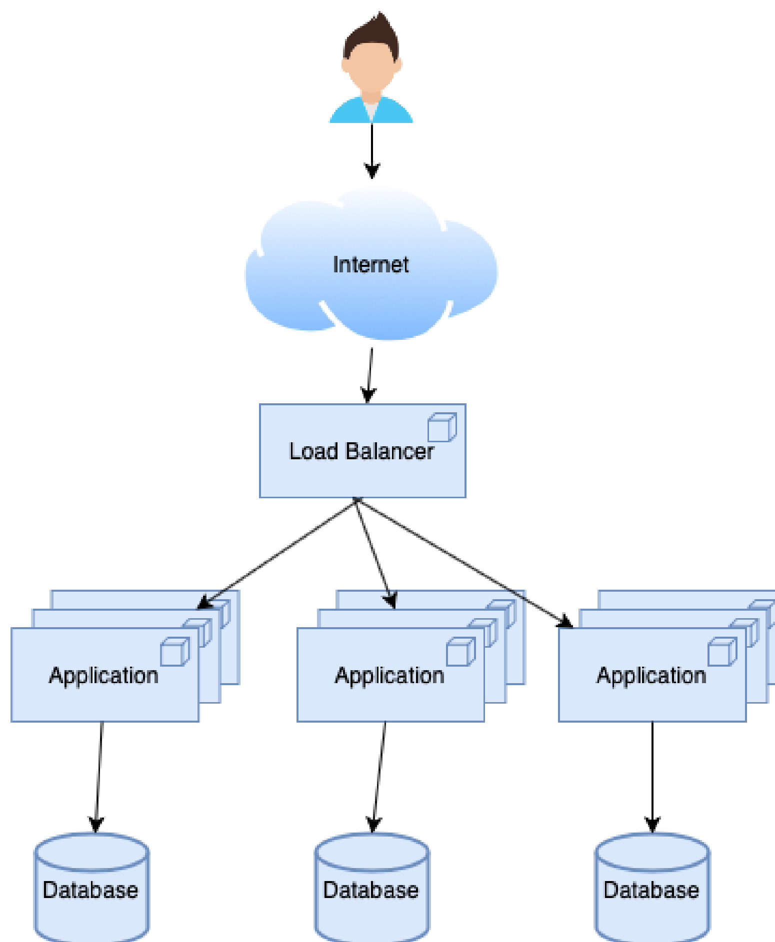

Figure 2.2: MA overview

2.2.2 Orchestration and Emergence of Service Mesh

While the number of microservices increase, the communication between them becomes hectic. Therefore, the choice of MSA over MA led to orchestration of the services. According to Alshuqayran et al. [10], introducing MSA would lead to the need of orchestrating multiple microservices.

The emergence of traffic control [45] led to evolving *service meshes*. **Service mesh** is "an infrastructure layer that enables managed, observable and secure communication between binaries deployed by operators to deliver some function of a service mesh application (referred as "workloads")" [6]. Figure 2.5 represents service mesh architecture. Service mesh architecture uses a dedicated infrastructure layer that logically consists of *data plane* and *control pane*.

Data plane is a composition of proxies that are deployed as sidecars. Figure 2.4 represents the sidecar proxy. The main functions of data plane are service discovery, health checking, routing, load balancing, authentication and access control, and observability [45]. Moreover, enforcing of policies is performed mainly by the data plane.

Control plane serves as the central hub and allows having a unified way of management and configuration of those policies that is coordinated along all components of the data plane. While data plane is visible to every network packet, control plane is not visible [45]. The later improvement involved using the sidecar proxy that interacts with the control plane [54].

The communication between components of the mesh are denoted as *"north-south"* and *"east-west"*. *"East-west"* traffic refers to Application layer service-to-service communication in Istio, while *"north-south"* direction refers to routing the external traffic to internal services of Istio. Services here refer to components of Istio due to terminology standards [6].

Istio uses Ingress and Egress Gateway Controllers as illustrated in figure 2.5 to define entry points for incoming and outcoming traffic. With the help of Ingress and Egress Gateway features of Istio, we can apply features as monitoring and route rules to traffic entering and exiting the service mesh [4].

In figure 2.6, we can view the traffic flow in aforementioned directions [12]. Communication between

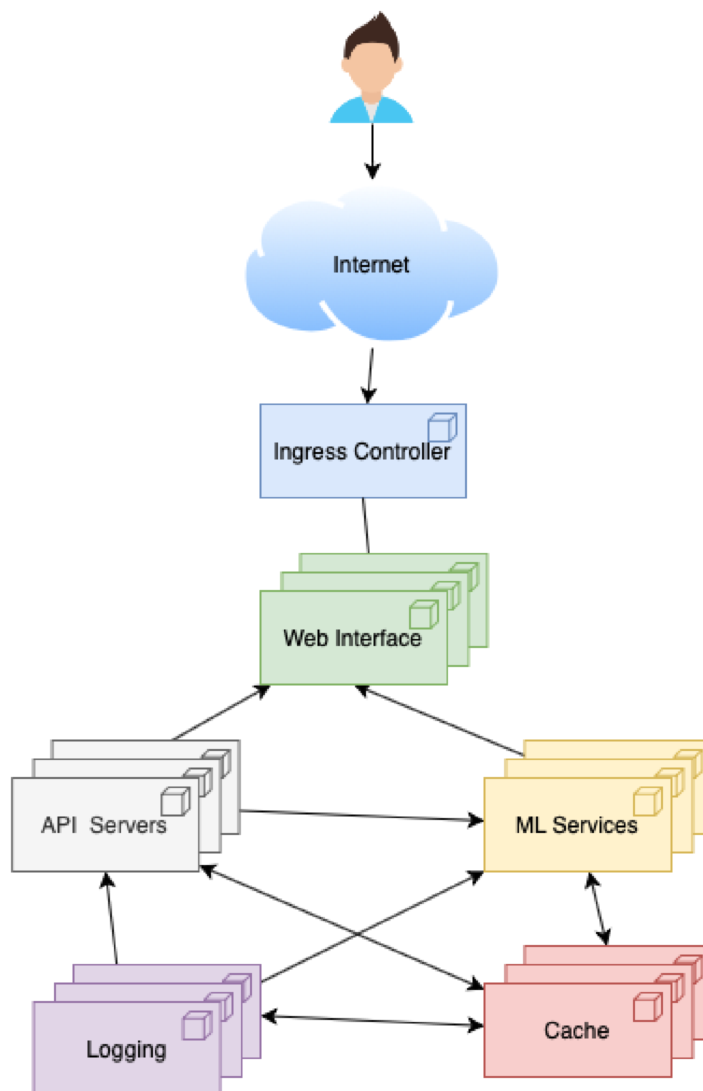

Figure 2.3: MSA overview

two services, as depicted by dotted arrows, represent the *"east-west"* direction. On the other hand, communication coming in or out of the clusters, as depicted by green arrows, represent the *"north-south"* direction.

2.2.3 Features of a Service Mesh

The most common service mesh platforms include *Linkerd*, *Istio*, *Airbnb Synapse* and *Amazon App Mesh*. This document focuses on Istio service mesh platform. Istio is a tool for connecting the services, securing CIA properties of services, controlling the traffic through enforcing the policies, and monitoring of services [25].

According to Li [45], the fundamental features of service mesh are as follows:

- Service discovery
- Load balancing
- Fault tolerance
- Traffic monitoring
- Circuit breaking
- Authentication and access control

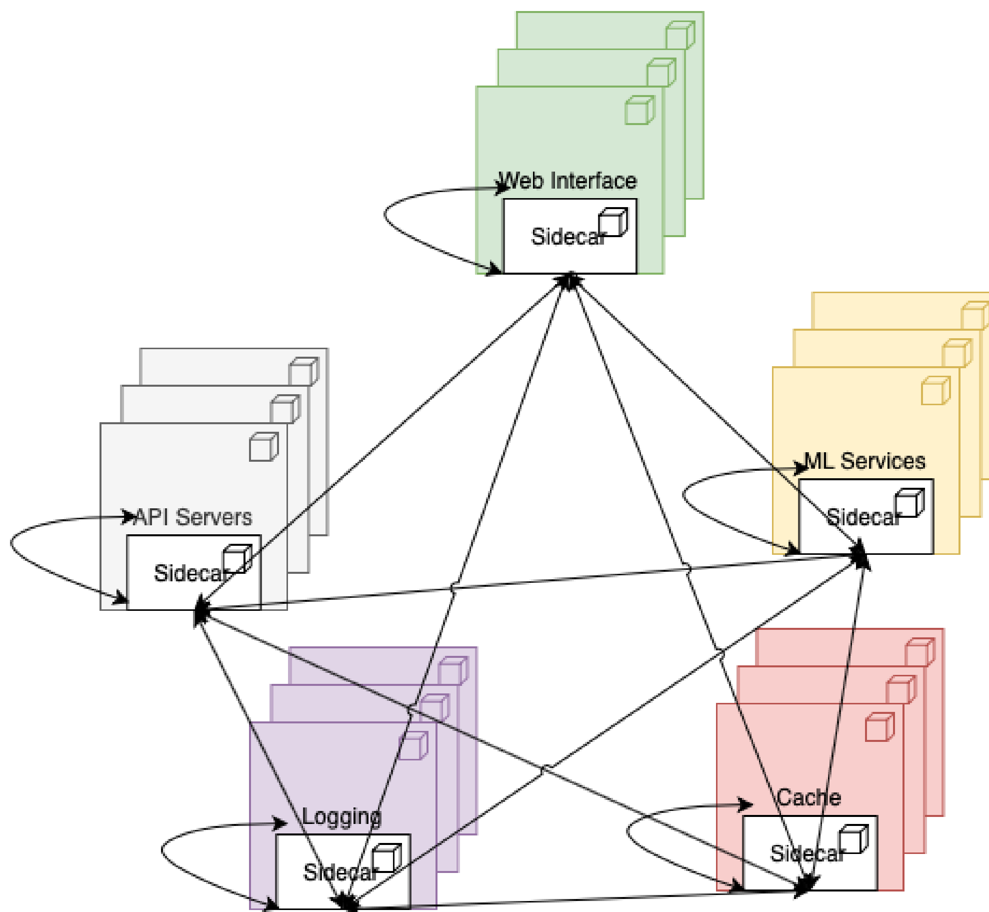

Figure 2.4: Sidecar architecture for service mesh

These features will be discussed in more detail in section 4.2. In the upcoming sections we take a closer look at Istio service mesh technology and its deployment models.

2.3 Istio Deployments Models

When preparing a production deployment of Istio, it is necessary to consider various factors. Therefore, Istio offers several deployment models. Deployments models that Istio offers are as follows [3]:

- Cluster models (single or multiple clusters),
- Network models (single or multiple networks),
- Control plane models (single or multiple control planes),
- Identity and trust models (with Certificate Authority (CA)),
- Tenancy models (namespace tenancy and cluster tenancy),
- Mesh models (single or multiple meshes).

For the scope of this work we focus on multi-cluster and multi-mesh deployment models. In further subsections 2.3.1 and 2.3.2 we discuss multi-cluster and multi-mesh deployment models in more details.

2.3.1 Multi-cluster Deployment of Istio

Overall, there are two ways to deploy multi-cluster "control plane": with shared control plane, or with replicated control planes.

The first scheme in multi-cluster scenario is multi-cluster deployment with shared control plane. Figure 2.7 demonstrates Istio deployment using the shared control plane. In this architecture service

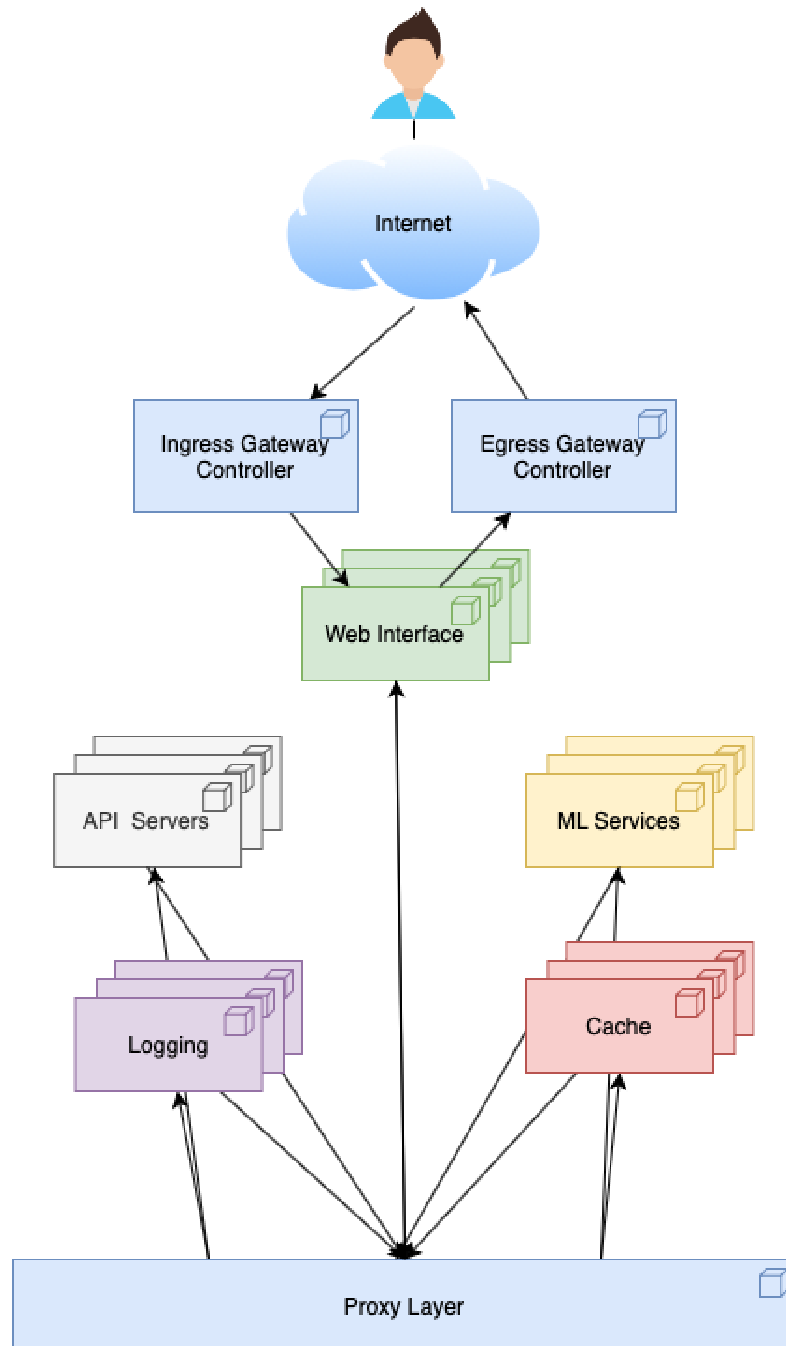

Figure 2.5: Service mesh

mesh is deployed in several clusters running the remote configuration. These clusters connect to the shared control plane, represented in the figure with a blue box, which is located in the main cluster [2]. The shared control plane coordinates all operations across the different clusters, and acts as a central point of control.

An alternative deployment scheme to consider is replicating control planes in each cluster. Figure 2.8 demonstrates the deployment of multi-cluster environment with replicated instances of control plane in each cluster. Compared to the deployment model in figure 2.7, each cluster uses its own endpoints and instance of control plane, and communication between services across clusters is enabled through gateways. All instances of control plane are managed by *shared administrative control plane* that handles the security issues, policy configuration and management [1].

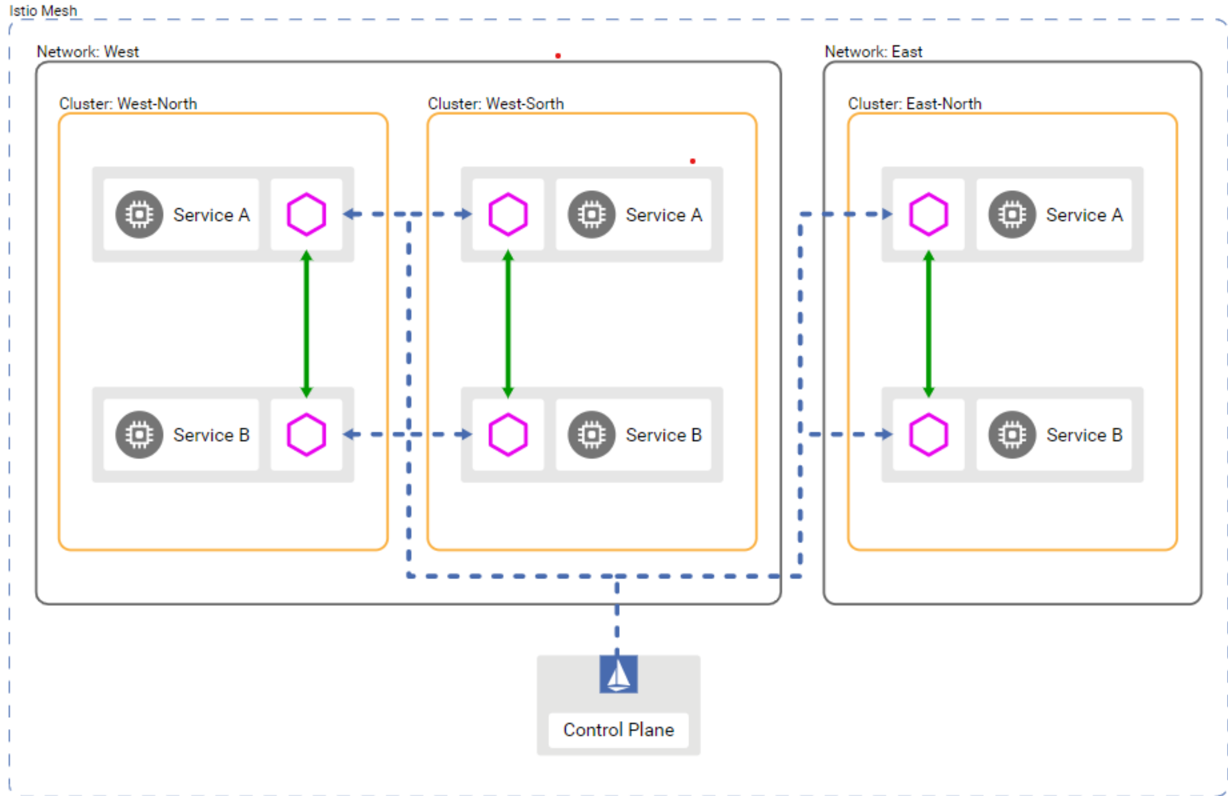

Figure 2.6: Directions of communication in Istio. Figure taken from [5]

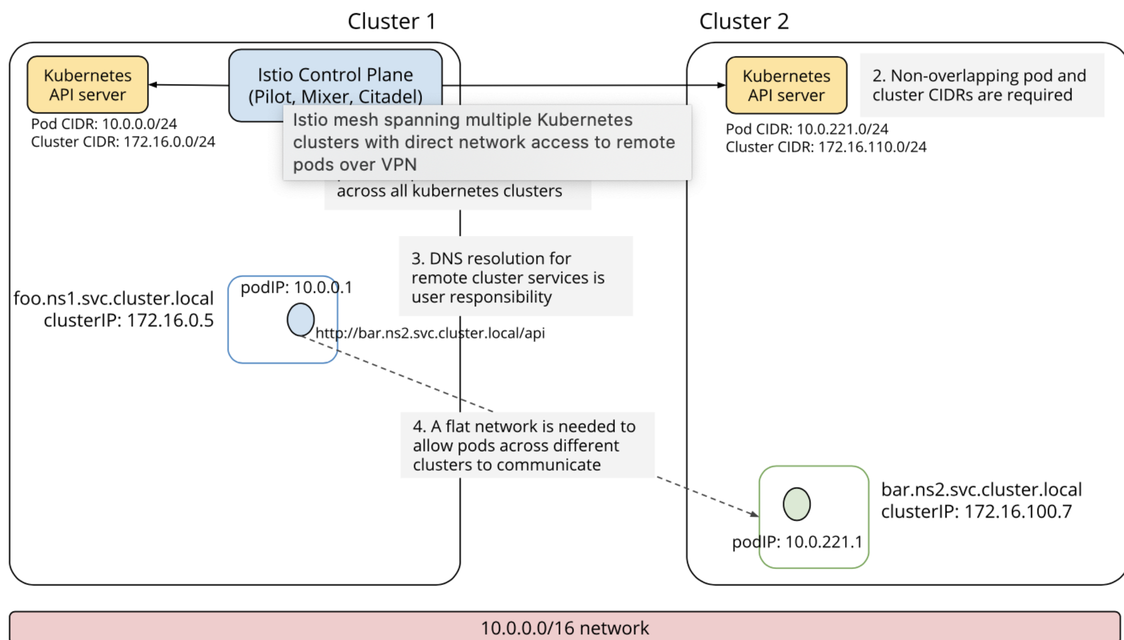

Figure 2.7: Multi-cluster Istio deployment using shared control plane. Figure taken from [2]

2.3.2 Multi-mesh Deployment and Service Mesh Federation

Single mesh deployment is a simple deployment model which provides simplicity and functionality [5]. However, large-scale systems can quickly outgrow this deployment model, for example, by needing

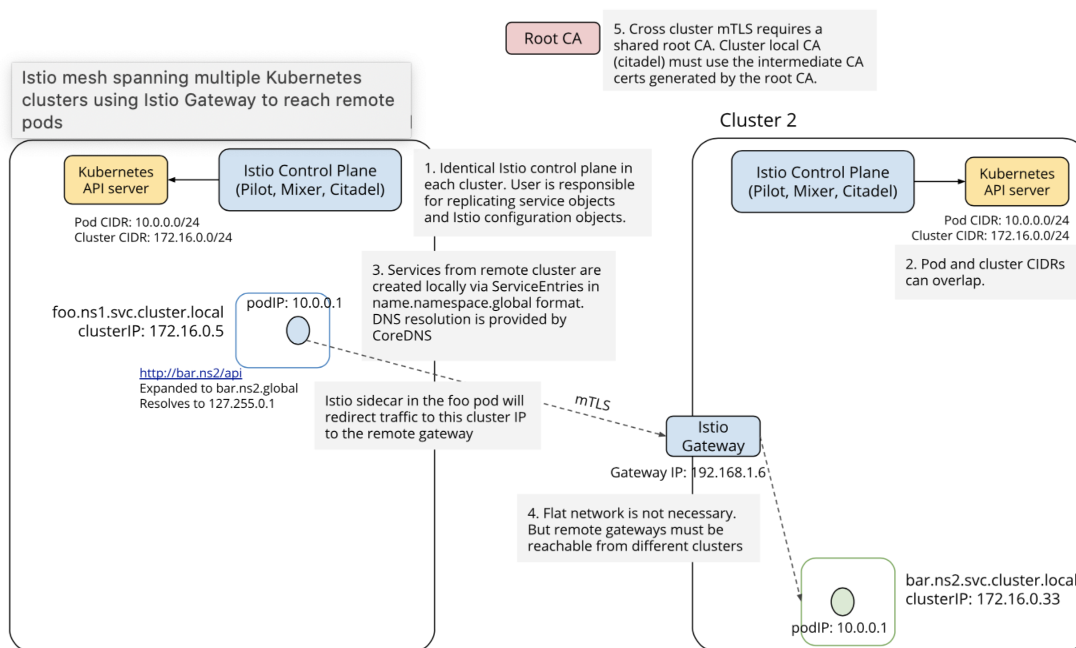

Figure 2.8: Replicated control planes. Figure taken from [1]

geographically distant deployments or multiple vendor technologies. Multi-mesh deployment then becomes a more suitable deployment strategy. Table 2.1 highlights some the key differences between these two deployment models.

Single mesh deployment	**Multi-mesh deployment**
Service names are unique within the mesh	Allows namespace or name reuse
Tenancy is enabled via namespaces (namespace tenancy), or clusters (cluster tenancy)	Tenancy is enabled via namespaces, clusters, or meshes
Does not require mesh federation	Requires mesh federation
Does not offer organizational boundaries[1]	Offers organizational boundaries

Table 2.1: Difference between single mesh and multi-mesh deployment architecture in Istio

Mesh federation is a process to "allow multiple meshes to communicate beyond the boundary of their own mesh" [6]. Mesh federation allows a mesh to expose it's internal services and establish a communication with other meshes, typically in a multi-mesh deployment architecture.

2.4 Blockchain Technology and Smart Contracts

The distributed system does not have a central trusted intermediary that controls the behavior of entities in a system. In the real world scenario, the entities in the distributed system do not often interoperate in a deterministic way, as described in "6.4. Federations and Trust Federations at Scale" section in "Cloud Federation Reference Architecture" by National Institute of Standards and Technology (NIST) [15]. This means that we should consider the cases when entities can perform malicious acts, and the system can fail during the scaling process. The questions that we need to answer is how entities can determine whether the system does not malfunction, which can lead to failure of trust between the components of the system. NIST introduced the concept *"establishing trust in an untrusted world"*, which describes the

[1]Organizational boundaries are the "demarcation between an organizational entity and its external environment" [58].

aforementioned issues. The possible solution for aforementioned case is a "distributed consensus method", one of examples of which is a *blockchain technology* described further in 2.4.1. The concept of blockchain technology is connected with the notion of *smart contracts* described further in 2.4.2.

2.4.1 Blockchain Technology

Blockchain is a "public ledger where all committed transactions are added and stored in a list of blocks" [66]. The members of the distributed system (*"peers"*) agree on the state of the system to preserve the consistency of the blockchain. The key concept in the blockchain technology is mining. *Mining* is a process of creating a new valid block in a blockchain. This is performed by *miners* (system devices or people with their own specialized devices).

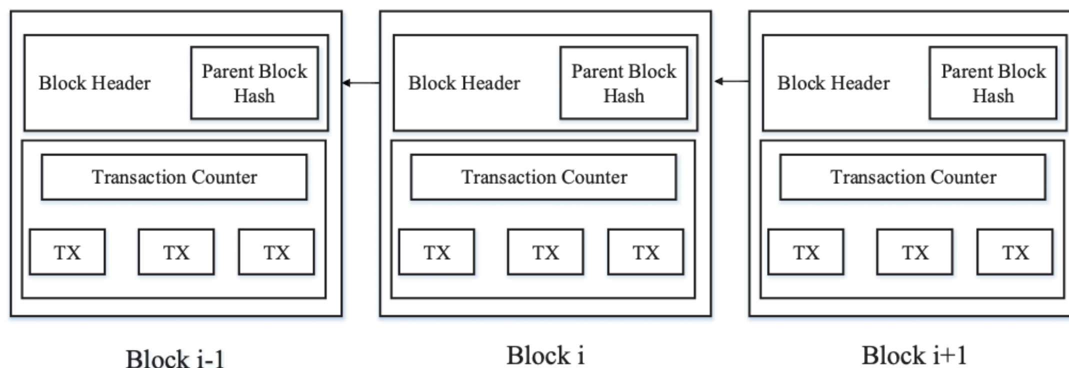

Figure 2.9: Blockchain architecture overview. Figure taken from [66]

Figure 2.9 showcases the general blockchain architecture. Each peer in a blockchain makes a *transaction* (e.g. sending a cryptocurrency to a peer, selecting a service, make a record). Every transaction is signed with the private key of a peer. Several transactions chained together and are stored in a *block*. Transactions are validated by *miners* before adding to a block. Each block is stored in the blockchain with the pointer to a previous block (hash of previous block), nonce, timestamp, indicator of which rule set to follow for validation of a block (version of a block), and hash value of all the transactions in the block (Merkle tree root hash). Figure 2.10 gives an overview of a block. Blockchain validates the new block with respect to previous block, and determines the next state of the system by shifting the pointer to the new block [15]. Because of this architecture, the main characteristics of the system are decentralization (no need for a central trust anchor to control the system), persistency (the information in a blockchain cannot be changed) and auditablity (the information in a blockchain can be easily verified and tracked).

Since blockchain technology does not require a central trusted intermediary as financial or legal institution, the number of business fields favor blockchain technology [66]. The range of application of blockchain technology is wide and include financial and payment services, legal institutes, public services and decentralized systems (e.g., decentralized service exposure marketplace) [52].

2.4.2 Smart Contracts

Nofer et al. [52] mentions that the taxonomy "smart contract" was proposed by Szabo in 1997. According to Szabo, smart contracts are computerized transaction protocols that are used to implement legal contracts as terms of a contract [57]. Smart contacts have a good potential to replace lawyers and banks that are responsible for legal contacts [52]. With the emerge of blockchain technology, the importance of application of smart contracts in a blockchain technology increased [66]. In a blockchain, smart contracts are automatically executed programs. Potential usage of smart contracts in the fields of decentralized service exposure marketplaces is checking whether the terms and conditions (SLA) were fulfilled between entities in this marketplace, e.g. a customer and a provider .

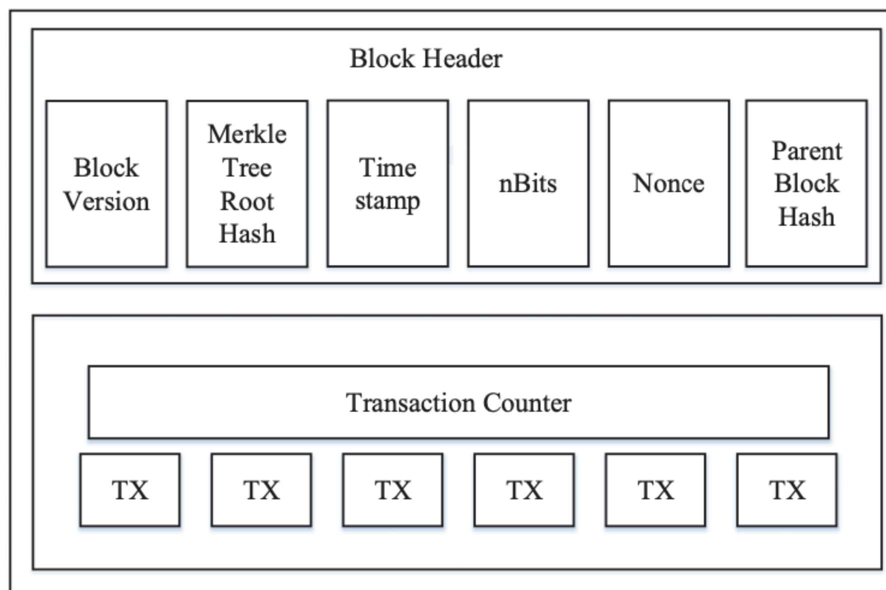

Figure 2.10: Block in a blockchain. Figure taken from [66]

2.5 “Nubo" Decentralized Marketplace Overview

For this work, we focus on a decentralized virtual service marketplace “Nubo". This marketplace uses a blockchain technology explained in 2.4 in more details as a distributed ledger and a trust anchor, which is aligned with our work. But “Nubo" lacks the trust management system, and this makes this marketplace a good candidate to introduce our suggested trust management system. In chapter 4, we discuss how can we build a trust management system in a decentralized service exposure marketplace based on smart contracts and service mesh.

2.5.1 Decentralized Marketplace Design

The reference architecture of this system consist of *Nubo portal*, *Saranyu Blockchain dApp*, *Hyperledger Fabric blockchain microservice*, *distributed NoSQL database* and *Service Manager*. Figure 2.11 demonstrates the “Nubo" reference architecture.

- The web application called “Nubo portal" provides a consumer of virtual services (referred as “tenant" in [33]) an opportunity to browse and request for virtual services (referred further as “services" in this report).
- Saranyu dApp provides REST Application Programming Interface (API) that communicates with the blockchain technology. Saranyu dApp is referred as a *Blockchain API* in this work.
- Hyperledger Fabric blockchain records the information provided by Saranyu dApp about the service offerings, smart contracts between services and consumers of services explained in 2.4.2 in more details, and contracts for subscriptions for consuming services. Quorum is referred as a *Blockchain* in this work.
- Cassandra distributed NoSQL database [42] that stores usage data for each service that is later assessed for business and operation purposes. Cassandra distributed NoSQL database is not considered for the scope of this work.
- Service Manager, a microservice that maintains and orchestrates the services in Docker containers to support isolation and deliver to multiple consumers of virtual services. Originally, Saranyu was not designed to support multitenancy, and, therefore, Service Manger is necessary for “Nubo". Our work focuses on possible improvement of Service Manager in chapter 4.

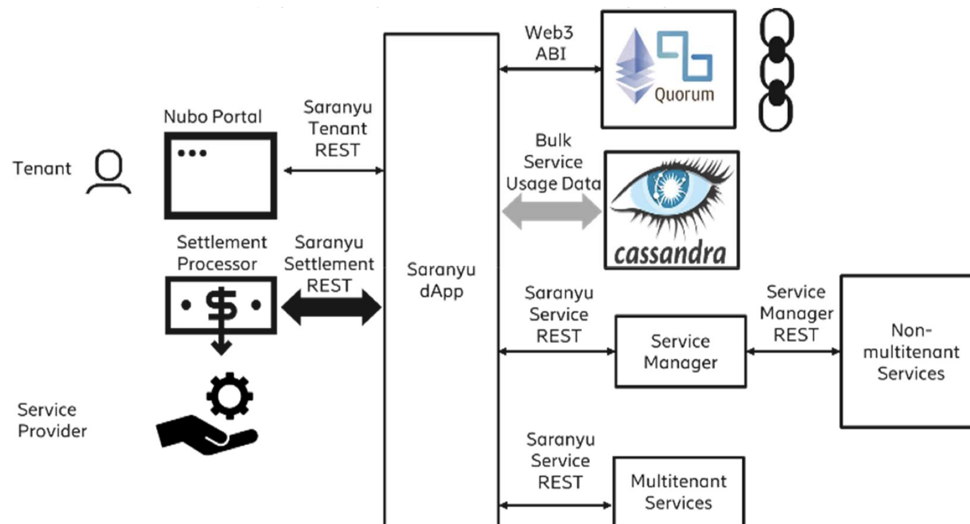

Figure 2.11: Overview of "Nubo" reference architecture. Figure taken from [33]

2.6 Literature Review and Related Work

With the basic concepts defined in the previous subsections, we now move to explore previous work in the area of trust management, service exposure and decentralized marketplaces. Figure 2.12 depicts the trust management systems that we will discuss in upcoming subsections in a form of time line.

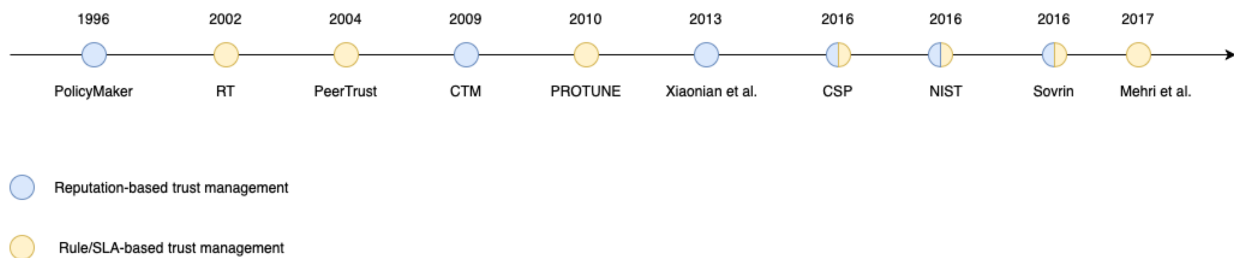

Figure 2.12: Overview of trust management systems

2.6.1 Trust Management for Decentralized Systems

The issue of trust management in a decentralized system is a topic thoroughly explored by multiple works. These works about trust management system or model of a decentralized system, typically, falls under one of two categories: reputation based or rule based. The trust management systems in a decentralized system are generally divided into two categories: *reputation-based* and *role-based* (also referred as *rule-based*). The *reputation-based* relies on the ratings or feedback of peers in a decentralized system to evaluate trustworthiness of members in the system [17]. The *rule-based*, on the other hand, relies on policies and credentials to decide what a member in the system can do [17].

In 1996 Blaze et al. introduced one of the earliest and most influential works on trust management systems *PolicyMaker* [14]. Blaze et al. designed the model to address the limitation of traditional security policy approaches, like, Pretty Good Privacy (PGP) and X.509 [21] which only focused on authentication and identity, but do not focus on the continuous process of trust management. *PolicyMaker* implements a trust management layer that focuses on adjusting the set of allowed actions per a public key holder. Moreover, by leveraging the concept of delegations, one entity can share some of its authority with another entity. Nevertheless, Blaze et al. mentions that the proposed system lacks diversity in application contexts, as the trust management functions can only cover a few aspects of the architecture. In 2009 Lee et al. [43] presented a policy language *"CTM"* language from the family of reputation-based languages. The reputation score in this work is integrated to the policy. Entities request for the reputation score from other entities and perform computation to assess whether to establish the communication with

another entity or not. The main drawback based on remark from Lee et al. [43], however, is that policy evaluations are computationally expensive.

The more modern public key infrastructures (PKIs) designs offers a wide range of support large-scale mobile, multi-domain systems [39], e.g., connecting to web approaches [27]. Modern PKIs offer a number of credential management options and features [34] [35], efficient revocation [37] [38], and cloud based scalable deployment [36]. These capabilities extend to allow inter-domain routing [29], and conditional anonymity [9] [39].

On the other spectrum for trust management for decentralized systems, in 2010, Bonatti et al. [16] proposed a rule-based trust negotiation system *PROTUNE* to protect the privacy of users. *PROTUNE* utilizes policies, which can be encoded in a node's digital credentials, to manage the disclosure of sensitive data, and peers to request or release information. By establishing a peer that conforms to the policies in each node, a *trust negotiations* procedure can automate message exchange between nodes. However, *PROTUNE* lacks the ability for policy negotiations which might hinder the completion of some transactions. Another disadvantage is the lack of usage control constructs. Multiple rule-based systems follow in the suit of *PROTUNE*, like *RT* [44] and *PeerTrust* [26], they all however suffer from the general disadvantage. Rule-based trust management systems are intended for systems with a strong protection requirements.

Finally, more recent approaches combine the aforementioned traditional trust management approaches to achieve a trustworthy decentralized system. One such instance is *Smart Contract-assisted PKI (SCP)* by Ahmed and Aura [8] in 2018. They presented a decentralized trust management system using PKI and smart contracts and combined both rule-based and recommendation-based systems. Their work offered a set of trust policies registered to the blockchain and visible to all members of the decentralized system. However, it was still limited by the "publish/subscribe design pattern" of SCP.

A more relevant resource to our current work is the Sovrin Governance Framework V2. Sovrin proposes a set of protocols and guidelines to enable the creation of domain specific governance frameworks. We can classify this work as the combination between reputation-based and rule-based approaches, since the Sovrin Governance Framework V2 uses the assessment of reputation for nodes by *Trust Institutes*, and combines it with the set of rules to protect the identity of users. Overall, the framework uses the concepts of "Trust Anchors" and "Trust Levels". Trust Anchor is an authoritative entity that all other entities trusts inevitably (e.g. distributed ledger or a blockchain technology). Trust Levels correspond to the reputation scores that each entity posses. Upon increasing the reputation score, the entity can turn into trust anchors, scaling the web of trust of the foundation.

The trust management protocols by Sovrin outperforms similar trust management protocols in functionality, for example, revocation of credentials by the issuer [50]. However, Sovrin protocols lack the technological maturity and stability, as these set of protocols are relatively new [50].

2.6.2 Trust Management for Cloud Service Exposure Marketplace

Much like in a decentralized system, trust management for cloud environment is not a novel topic. Trust management systems in this domain typically fall under three categories: recommendation based, SLA based, and reputation based. In a *recommendation-based trust management system*, trust is established by a third party's recommendation [40], when two entities had no communication before, and therefore, could not establish a trust relationship. This category of trust models does not fit our current need for a decentralized service exposure marketplace. A *SLA-based trust management system* is akin to rule-based systems mentioned in the previous section. A *SLA-based trust management system* relies on policies and rules in the form of service contracts and reports [30]. Finally, a *reputation-based trust management system*, in this context, shares the same definition as the one defined in the previous section.

Xiaonian et al. introduced a *reputation-based trust management system* in 2013 with a focus on identifying malicious entities in cloud environment [59]. The model utilized evidence theory, which leverage direct interactions as first hand evidences and recommendation trust values as second hand evidences, to determine trust in an entity. Xiaonian et al. determine the reputation of an entity based on the cumulative trust values acquired through recommendations by other members of the system [25]. By using this technique, the trust degree of entities increases slowly and decreases quickly which leads to the effective identification of malicious entities. Nevertheless, like other models that follow *reputation-based trust management*, the model by Xiaonian et al. does not take into account collusion between entities [25]

On the other hand, in 2017, Mehri and Tutschku introduced an *SLA-trust management system* for a cloud-based marketplace for Artificial Intelligence (AI) [48]. The approach used by Mehri and Tutschku is based on *"Privacy-by-Design"*, which focuses on controllability and transparency of operations and data

in a cloud marketplace. Mehri and Tutschku utilised “Virtual Premise".“Virtual Premise" is a concept that enforces controlled access and monitoring of data usage in a cloud marketplace. This work, however, does not describe the mechanisms of enforcing policies when the trust is broken, or protection mechanisms for this system.

From 2016-2020, NIST develops the guidelines for trust management for Federated Cloud Reference Architecture model. In chapter 6 “Deployment Governance: Requirements and Options", NIST discusses trust requirements and relations to enable interactions of two or more geographically distant entities. NIST provides protocols, governance strategies, and best practices to establish large-scale federated cloud systems [15].

The works in the field of trust management in decentralized service exposure marketplaces are not common due to the recent emergence of this field and to the best of our knowledge there is no work in this specific field. Thus, this work aims to define a trust management model for the a decentralized service exposure marketplace that would address the limitations discussed for decentralized systems and cloud marketplaces.

3 Model, Requirements and Assumptions

In section 2.1, we defined trust management as a continuous process ensuring that trustworthiness of entities. To utilize this definition, we first define our system model. Section 3.1 describes the components of a decentralized service exposure marketplace and their interactions within the system. We then move to discuss our desired security and operational properties for the introduced system in section 3.2. In section 3.3 we discuss security assumptions for the aforementioned system. In section 3.4, we turn our attention to the security concerns of our system model. We define an adversary model under these assumptions, listing possible threats and their subsequent effects towards our system.

3.1 System Model

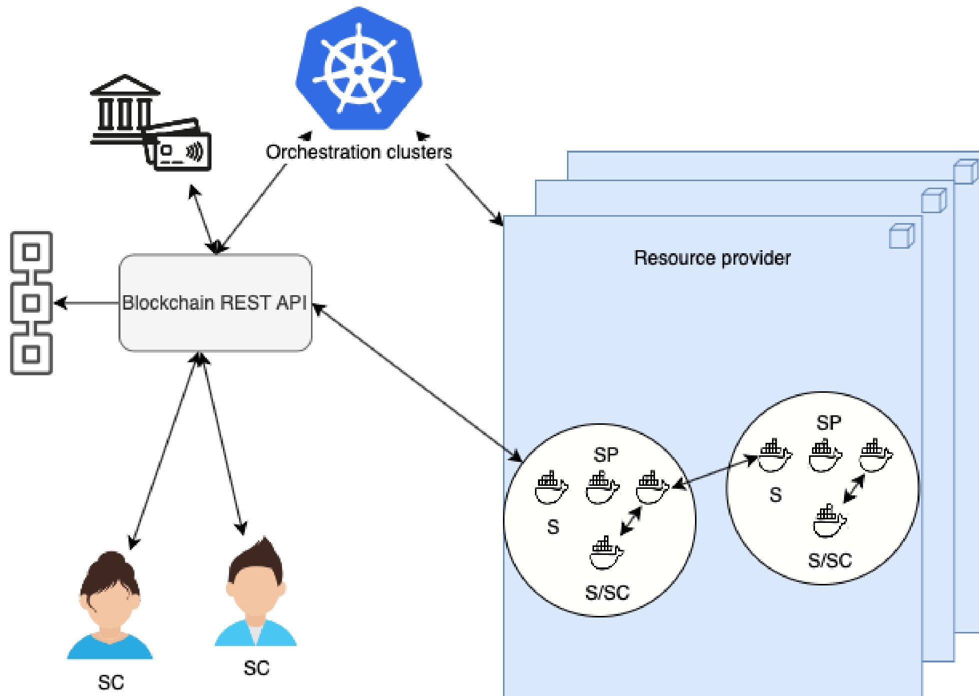

Figure 3.1: Marketplace system overview

A traditional marketplace consists of several components: sellers, products, buyers, terms of the deal, enforcers, and payment systems. With these components in mind, figure 3.1 depicts a model of a decentralized service exposure marketplace system. As illustrated in the figure, components of the traditional marketplace can be mapped to our decentralized marketplace. Products map to Service (S), sellers to SP, and buyers to SC. Moreover, the blockchain ledger and smart contracts records the terms of the deals or SLA, and the orchestration clusters act as enforcers of policies. Finally, a combination of smart contracts and third party payment providers facilitate the payment service for the marketplace. For this work, payment service is outside of scope as it plays no role in a trust management system.

3.1.1 Blockchain and Smart Contracts

As described in section 2.4, a *Blockchain* plays the role of a digital ledger [62] in a distributed system. This entails storing information in a transparent and accessible manner for all entities of the system. For a service exposure marketplace, the information to store is listed as follows:

- **SLAs**: The agreement between a SP and SC to use an S with an agreed specification.
- **Service offerings details**: A description from a SP detailing the specification of S, the usage quota and metric, charging rate, and variation/tiers for the S
- **Service consumption data**: The consumption data of an S by a SC in the agreed upon metric. Note that the blockchain is not responsible for enforcing the quota, but only documenting the consumption in a transparent and accessible manner.

- **The identity of entities**: Each action documented to the blockchain ledger is associated with an identity. These identities are documented to the blockchain ledger which ensures transparency of all actions on the blockchain.

As described in section 2.5, the *Blockchain API* facilitates interaction with the blockchain. This component is in an intermediate software that enables interaction with the blockchain through a simple REST API. The main function exposed by the API allows entities to:

- Create offering detail;
- Provision of a Service (S);
- Query available offerings;
- Register a Service Provider (SP) or a Service Consumer (SC);
- Query a Service (S) consumption.

Finally, an integral part of the operation of the marketplace relies on *Smart contracts*. A Smart contract is a general-purpose computational mechanism, designed as small programs that are stored in a blockchain. Smart contracts are used to safely and securely execute agreements between the parties [11].

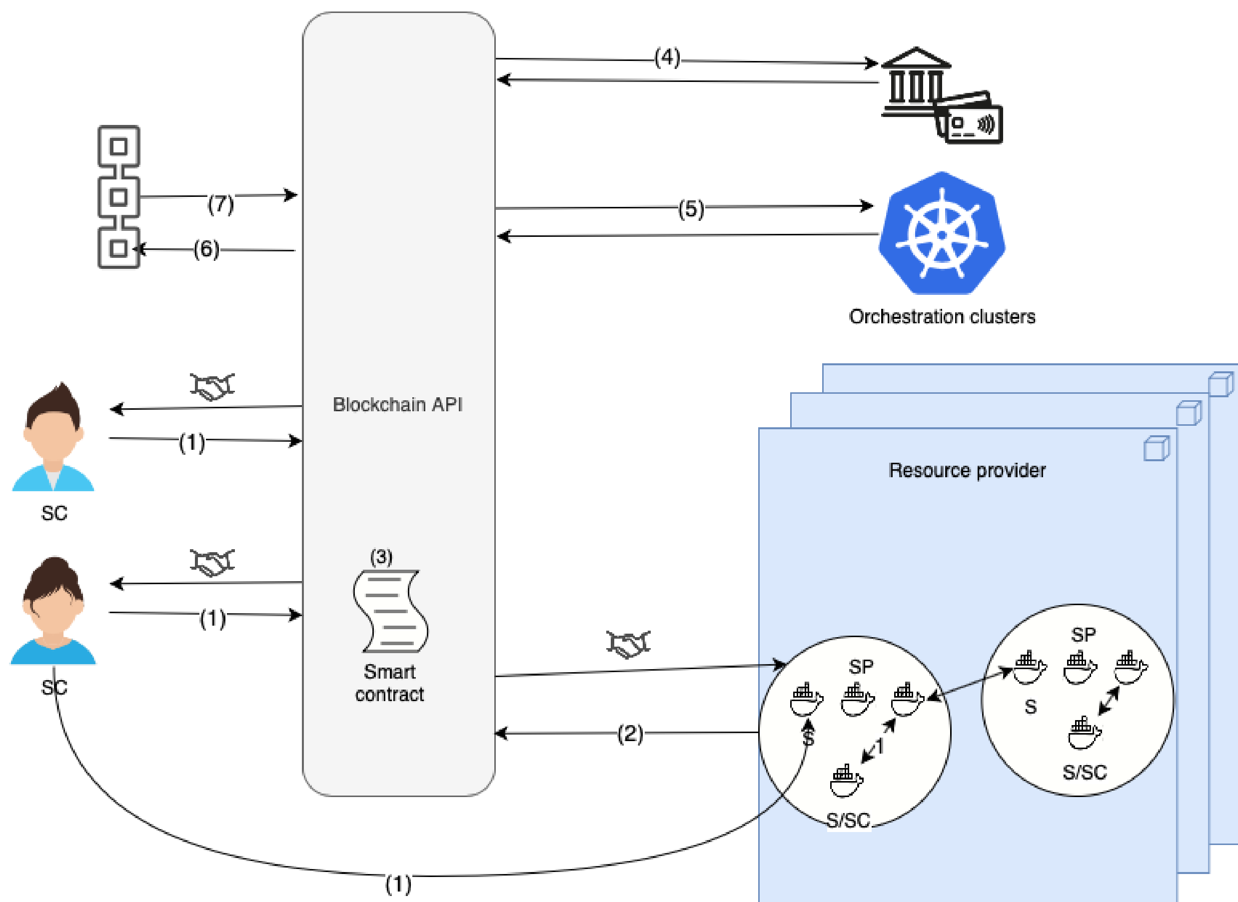

Figure 3.2: Communication flow in a blockchain

Figure 3.2 demonstrates the communication flow in and out blockchain API and blockchain ledger. SC can explore the available Services (Ss) and select one S as it can be seen from (1). SP offers the available S and delivers the requested one (2). The agreement between SC and SP is performed with the Smart contract (3). After obtaining the S the payment mechanism is initiated by smart contract by sending the payment request from Blockchain API (4). There are other tools and components that further support the marketplace functionality and integrity. These tools, as well as others, are generalized into the notion of “Orchestration tools" (5) and discussed in more detail in the upcoming section. Blockchain API records the information in blocks and adds the block to the blockchain (6). All entities in a marketplace can obtain information in the blockchain (7).

3.1.2 Orchestration clusters

Orchestration clusters can provide mechanisms for the following functions in the marketplace [18]:

- Registry, configuring and updating Docker containers,
- Managing resource requirements and quotas,
- Applying configuration files to running jobs in a dynamic manner,
- Service discovery and Domain Name System (DNS),
- Load balancing,
- Metrics, logs and state of the system,
- Scaling of the system,
- Public Key Infrastructure (PKI) and CA,
- Fault tolerance,
- Traffic monitoring,
- Circuit breaking,
- Authentication and access control.

As described in figure 3.2, Blockchain API can request for additional information that are needed for system sustainability. Resource Provider (RP) can update the Orchestration clusters with the metrics of SP and S that can be later used for health check purposes.

3.1.3 Entities and Roles

Entities are referred to as the roles of the system rather than physical components of the system. Figure 3.3 summarizes the system model to have a graphical representation of the system described in 3. Furthermore, these roles are not entitled to a particular actor in the ecosystem. Hence, the same SP can become a SC towards another SP in case a required service is not provided by the same actor, e.g., due to geographic location or availability.

Generally, in a digital marketplace as described in 2.5 for services there are different actors that interact with each other: cloud platform providers, infrastructure providers, application providers, among others. From this point, decentralized service exposure marketplace is also referred as a marketplace. For our purpose, we generalised the most relevant components as follows: **SP**, **RP**, **S** and **SC**. Each entity can hold different meanings, and, thus, we formally define each component as follows:

- Service Consumer (SC) is an entity that is requesting a service from the SP. This entity consumes a Service (S). This entity can take many forms either a virtual service or a user.
- Service Provider (SP) is an entity in the marketplace, which has a number of services and can provide other entities upon request with these services. It can provision a Service (S) instance for a specific SC, deploy a Service (S), monitor S consumption, update, maintain, and tear this Service (S) down.
- Service (S) is a specific set of procedures provisioned and deployed by SP and serve SC. S are categorised into three groups: Function as a Service (FaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). S can inter-operate with each other by either S-to-S or S-to-SC relationship. S-to-S relationship takes place when both Ss are provided by the same SP. S-to-SC relationship takes place, when two services are offered by different SP. In this case one S becomes a SC for another S. Multiple services can operate to become a *service composition*.
- Resource Provider (RP) is an entity that provides the infrastructure usage (Infrastructure as a Service (IaaS)). Resources can be either a compute, a network or a storage resource. A RP can have resources at different points of presence, central site, national sites, edge sites, among others. For our purpose, we generalize RP into two categories: *Edge resource provider* and *Central cloud resource provider*. *Edge resource provider* is an entity that provides infrastructure-level resources to

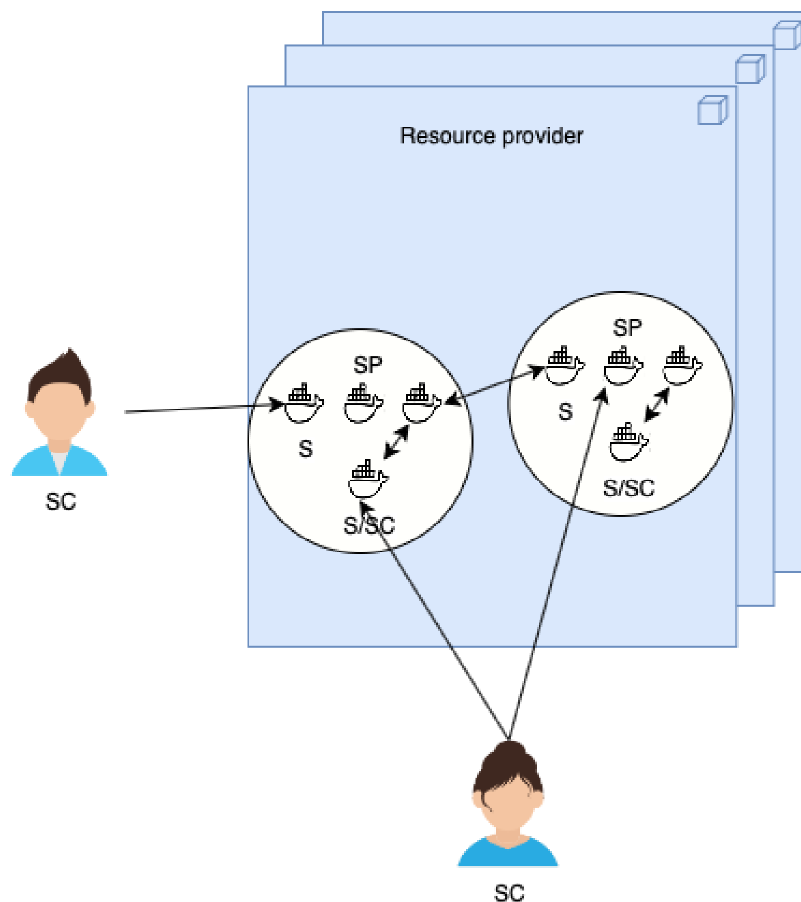

Figure 3.3: Entities and roles in a marketplace

last-mile entities close to the end-user; however, staying at the boundary or within the last-mile Communication Service Provider (CSP) [23]. *Central cloud resource provider* is the other type of distributed RP with more centralized resource location, unlike the aforementioned Edge resource provider. Moreover, a coalition of central cloud RPs which is better known as *"hyper-scale cloud provider"* falls under our definition of our central cloud RP [20].

3.2 Requirements to Establish Trustworthy Relationships

After defining the essential entities to establish a marketplace system in section 3.1, we move to address our desired properties for system operation. We define our requirements under the notion of *trustworthiness*, keeping in mind the complex need for trust management in our decentralized marketplace system. To reiterate the definition of trust management defined in section 2.1, it is the process by which an entity in the system becomes trustworthy to other entities. As such, we define a trustworthy entity as an entity that fulfills a group of trust requirements agreed upon by other entities of the ecosystem. This definition poses the question: what are the requirements needed to enable trustworthy collaboration among the entities of this marketplace ecosystem? These requirements are then further elaborated in chapter 4.

- **Requirements of adherence to SLA**. This dimension regards the capability of entities to provide the agreed upon offering, this can take the form of data, service or otherwise. Trust requirements in this dimension correspond to the behaviour of an offering, its runtime environment limitations and specifications and its response reliability and output validity with regard to an SLA.
- **Requirements of authorized observability**. This dimension refers to the ability of entities to trace, maintain, and perform their respective roles. Trust requirements in this dimension relate to authorized entities that analyze performance analytic, service monitoring, and log data retention policies.

- **Security properties requirements**. This category refers to the *group of security properties* as CIA of the secure communication between the entities. Trust requirements in this dimension correspond to access control, protocol specifications, authentication and identity management.
- **Requirements of availability and resilience**. This category covers the scenarios when the systems expands, and if the trust of processing remains transparent and predictable. Compared to aforementioned operation dimension, requirements in this dimension correspond to availability, fault tolerance, reliability of the runtime system environment and scalability of the system.

3.3 Security Assumptions

In this section we establish security assumptions and considerations that are made for our system.

- The Blockchain technology used as a decentralized ledger is reliable, with a rational underlying ledger scheme.
- Communication between entities is encrypted.
- Entities in the marketplace can be malicious.
- RP entity is assumed to be honest.
- The computation abilities of adversaries are not limited.
- Adversaries are not capable of controlling the blockchain, blockchain API and smart contracts.

As for honesty of RP, in real brokerless marketplace, as decentralized service exposure marketplace, this assumption is feasible. In the case where multiple SPs might be utilizing a single impartial RP. This is often the case in the mobile industry, when multiple mobile service providers share a radio tower or a resource node.

The *target assets* for adversaries include S,SP, SC,RP, cloud resources, privileged accounts, configuration files and availability of the system. The *adversaries* against the system can be either internal or external. An internal adversary is an already existing entity in our system that decided to be malicious. This type of adversary can be a Service (S), a SP, or a SC. An external adversary, on the other hand, is an entity from outside of the system which would try to first get access to the system and perform malicious actions. In this work, we focus mainly on internal adversaries who are actively trying to disrupt the system.

3.4 Adversary Model

In this work, the adversary model refers to attacks against our system, described in 3.1, with the goal of breaking our desired requirements, described in 3.2. The main goal of this adversary model is to break trustworthy relationships between the entities of the system. In 3.3, we define some assumptions, without which we cannot ensure the validity of our adversary model.

The methods that adversaries could use to achieve the target assets will be based on tactics from MITRE[2]. Microsoft mapped these tactics to cloud infrastructure and designed the attack matrix for Kubernetes [49]. We use this matrix to develop an adversary model for this system. The goals of attackers are as follows:

- Disrupting SLA,
- Bypassing authorization,
- Security properties violation,
- Disrupting of availability and resilience.

Subsections 3.4.1, 3.4.2, 3.4.3 and 3.4.4 provide more detailed overview of aforementioned goals of attackers. Each goal is achieved through attack steps. These attack steps are graphically demonstrated in an attack tree[3].

[2]MITRE is the US-based non-profit organization that supports agencies in a security research

[3]Attack tree is a formal description of a security of the system via representing potential attack steps in a tree structure. The main goal is placed in the root node, and different ways of achieving that goal are placed in leaf nodes. When the node has a label "OR", attacker can choose either ways to reach this node. When the node has a label "AND", attacker needs to fulfill all sub-goals to reach the goal [60].

3.4.1 Disrupting SLA

Disrupting SLA involve actions that can allow an attacker to tamper with logs or get more resources than allocated. This can be performed by an entity of a marketplace that either would like to be allocated with more resources than stated in SLA or modify the logs that contains the information about the usage of some resources. In order to either tamper with logs or get more resources than allocated, the malicious entity needs to compromise a privileged container that allows to access the logs or resources. Figure 3.4 represents the attack tree for disrupting SLA.

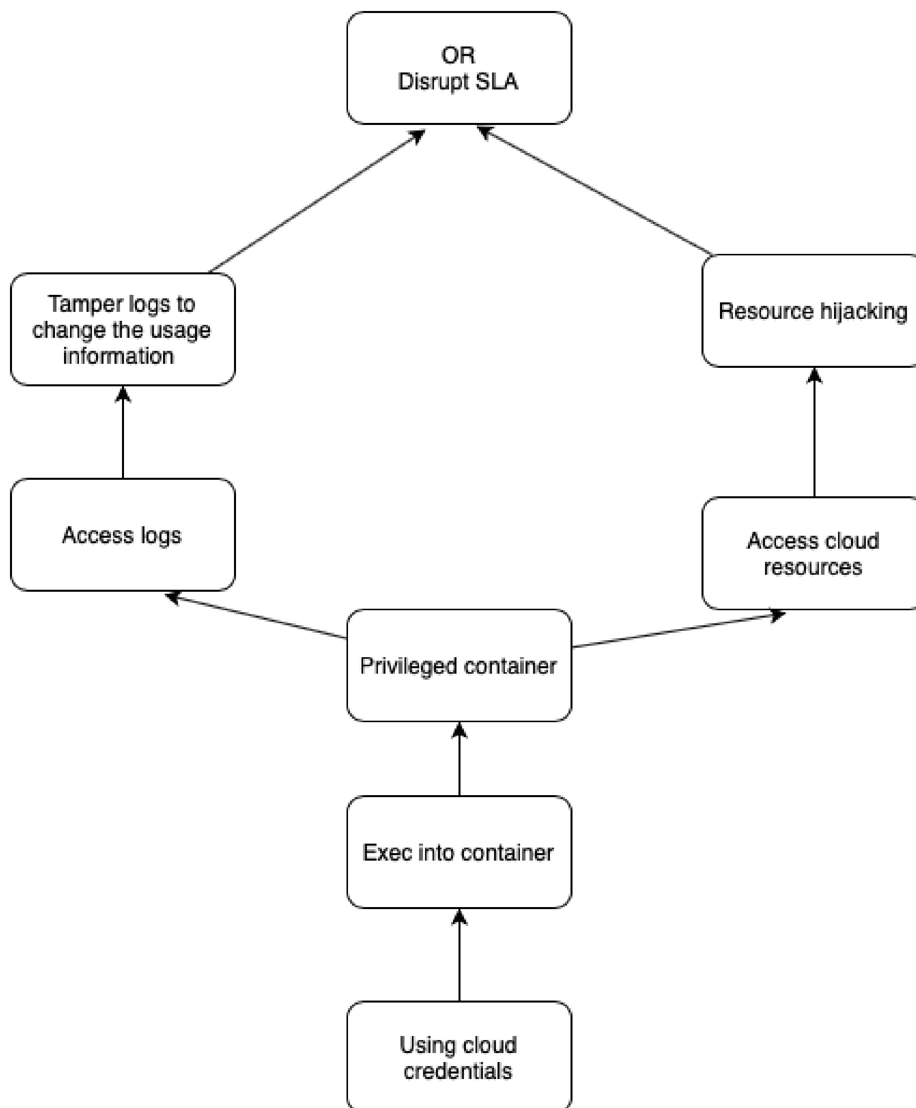

Figure 3.4: Attack tree for disrupting SLA

3.4.2 Bypassing Authorizations

Internal adversaries can focus on escalating the privilege to get administrative access to the resources. An attack strategy would involve obtaining *kubeconfig file* that contains credentials to clusters and their locations or *dashboard* for monitoring and managing a Kubernetes cluster via ClusterIP service. In case the permissions were configured incorrectly, malicious entities can create a new pod or container and execute their malicious code. Figure 3.5 represents the attack tree for bypassing authorizations.

3.4.3 Security Properties Violation

External attackers might focus on initial access tactic [49] to get into the system. If the adversaries manage to get Secure Shell (SSH) credentials via brute-force or phish, they can get an access to SSH

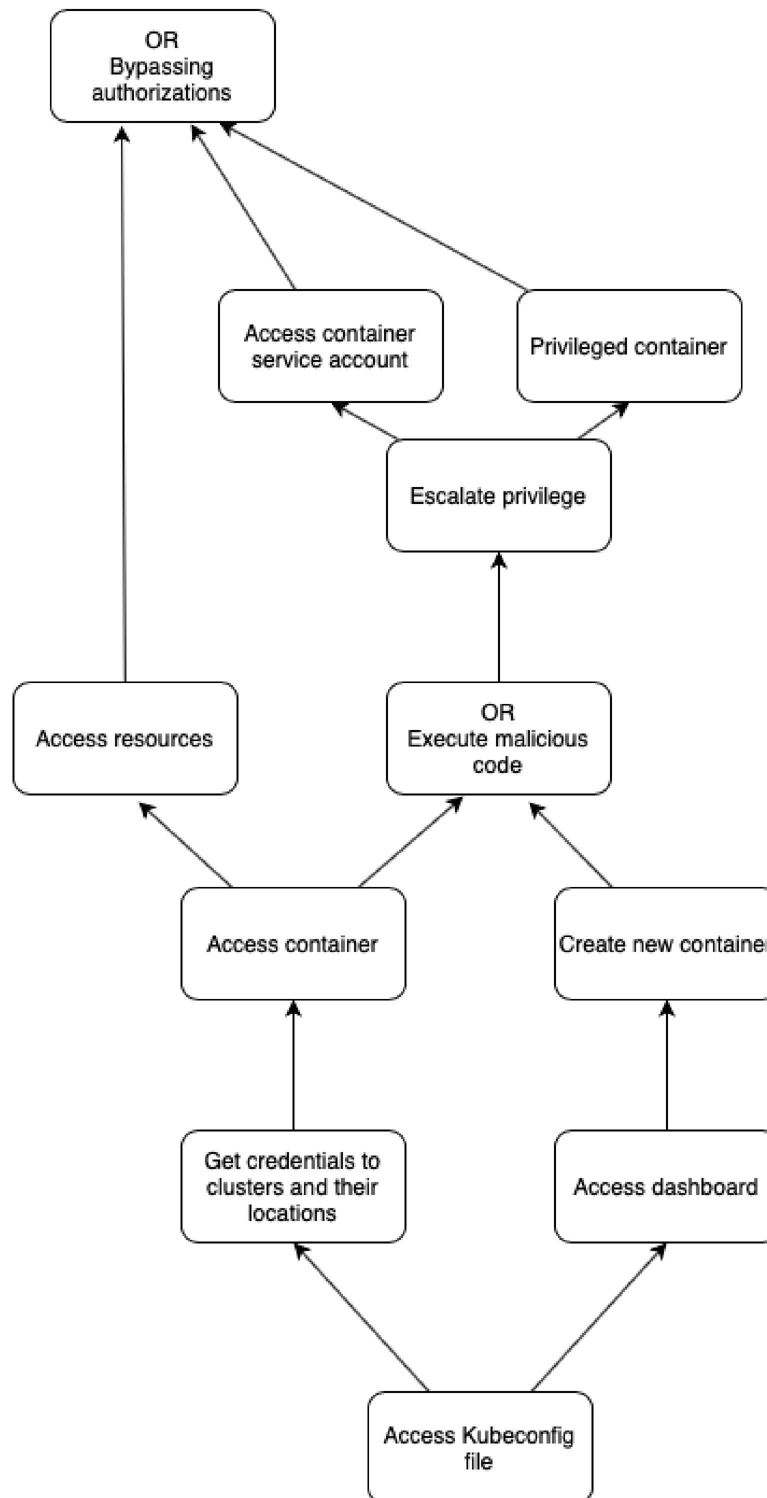

Figure 3.5: Attack tree for bypassing authorizations

server. If cloud providers in decentralized marketplace are public (e.g., AKS[4], EKS[5], GKE[6]), attacker can focus on finding leaked credentials available in open access. Another entry point is placing the compromised Docker image in the private container registry in case the attacker managed to get access to this private registry. If one of the S that faces users is compromised, the attacker can focus on Remote Code Execution vulnerability. Figure 3.6 represents the attack tree for security properties violation.

[4]Azure Kubernetes Service
[5]Amazon Elastic Kubernetes Service
[6]Google Kubernetes Service

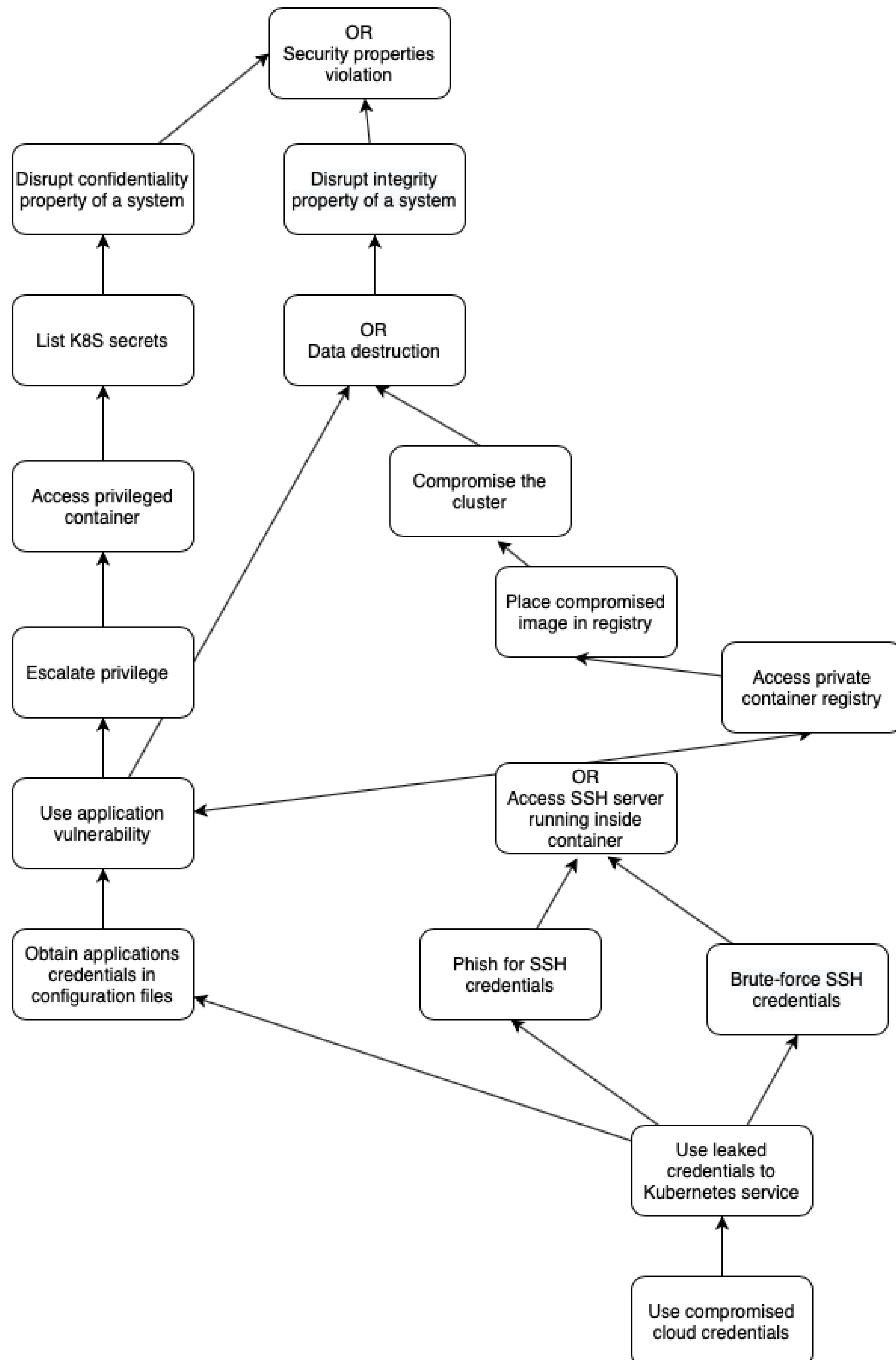

Figure 3.6: Attack tree for security properties violation

3.4.4 Disrupting of Availability and Resilience

This category of attacks includes disrupting, abusing and destroying the normal behavior of the system. The functioning of system may be disrupted, if attackers manage to delete configuration files, deployment

files, or storage. If the resource was compromised, the attacker can perform *resource hijacking* attack to run a computation expensive operation, e.g. mining the blocks in a blockchain. The system may fall into Denial of Service (DoS) attack, when the containers, nodes and API become unavailable. This way the marketplace cannot operate and function accordingly for legitimate users. Figure 3.7 represents the attack tree for disrupting of availability and resilience.

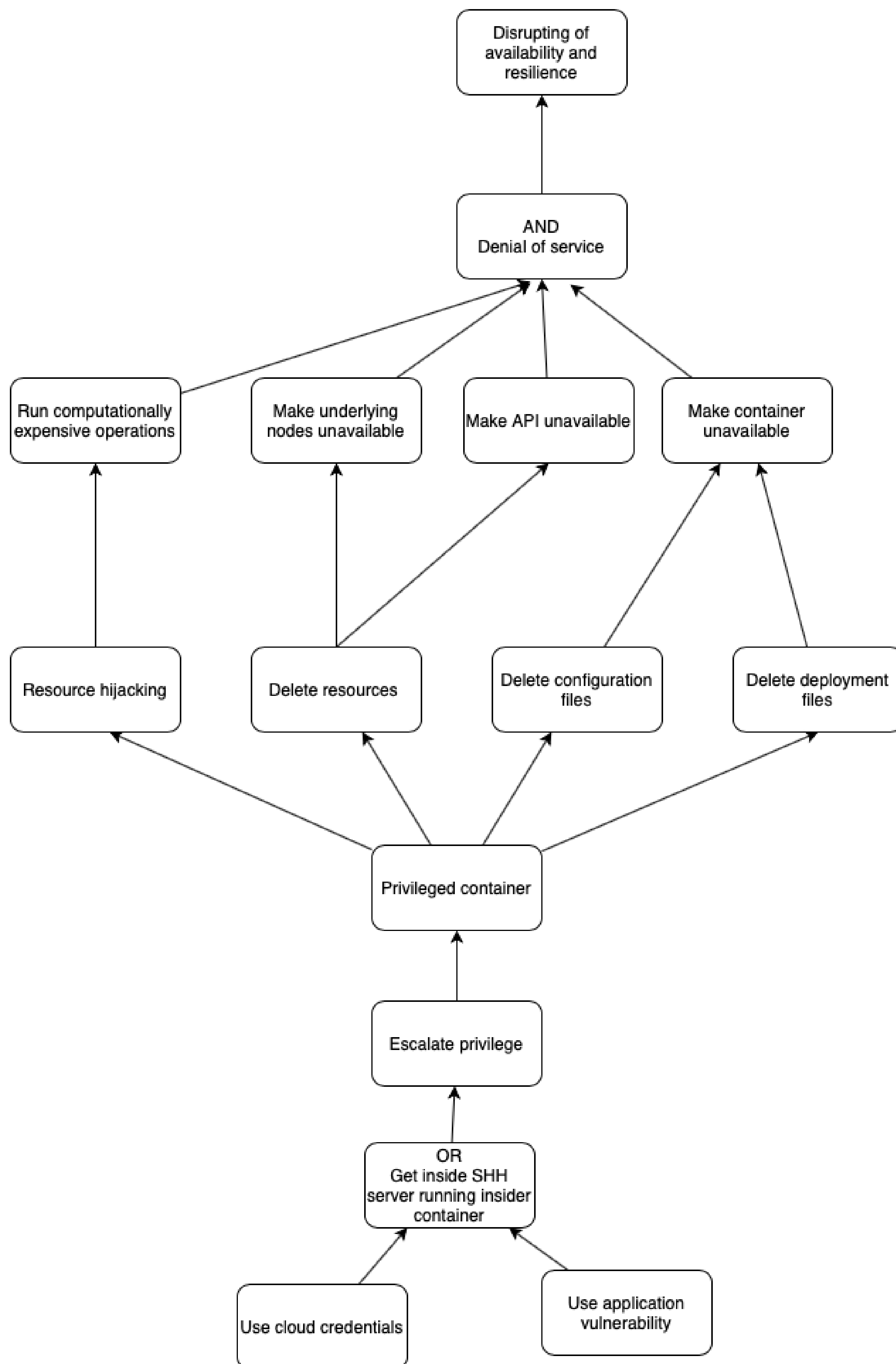

Figure 3.7: Attack tree for disrupting of availability and resilience

4 Implementation and Contribution

This chapter describes how the combination of smart contracts and service mesh technology could serve as trust anchor to establish trust management in decentralized service exposure marketplace. Section 4.1 provides the list of requirements that are divided into four groups described in section 3.2. Section 4.2 provides the mapping between these aforementioned requirements into features of service mesh technology (e.g., Istio). In section 4.3 we describe our solution *"Trust Engine"* that uses the features of Istio and smart contract to build a trust management system in "Nubo" decentralized virtual services marketplace.

4.1 Description of Requirements to Establish Trust

Table 4.1 represents the trust requirements and relations between entities in an adjacency matrix. Each group hosts several requirements, which we explore in detail in sections 4.1.1-4.1.4. In table 4.1, we give each group a color, as depicted at the end of the table, and list the trust requirements (No. 1,2,3, etc) for each entity pairing. Furthermore, by reading the entity paring as (row, column) we can determine the direction of the the relation. For example, the pairing (SP-S) has the following requirements: No. 1,2,3,4,5 from the Security properties requirements and No. 1,2,4 from Requirements of authorized observability. We notice that SP - RP pairing has a high number of requirements, especially in terms of security properties and requirements of availability and resilience. Overall, SP has higher requirements in all requirements regardless of pairing. We can explain this behavior as the SP entity plays a pivotal role between all other entities, i.e., SC, S, and RP. Furthermore, high requirements are most apparent in the SC-SP pairing and S-S pairing, where the requirements are across all groups. We can reasonably relate these findings to the very nature of a service exposure platform, where north-south and east-west communications, as discussed before in 2.2, are integral.

	RP	SP	SC	S
RP		1, 2, 3, 4, 5 4 5		
SP	2, 3 1, 2, 3, 4, 5 1,3,5 1, 2, 3, 4, 5, 7 1	6	1, 2, 3, 4, 5 5	1, 2, 3, 4, 5 1, 2, 4
SC		2 1, 2, 3, 4, 5, 6 2, 3, 4, 5 1,2	1, 2, 3, 4, 5	1, 3, 4 1, 2, 3, 4, 5 4, 5
S		4 1, 2, 3, 4, 5, 6	1, 2, 3, 4, 5	1, 3 1, 2, 3, 4, 5 5 6

Adherence to SLA **Security properties** **Authorization** **Availability**

Table 4.1: Requirements per trust set of requirements for each entity pairing

Subsections 4.1.1, 4.1.2, 4.1.3, 4.1.4 will demonstrate the aforementioned group of requirements and illustrate them schematically. Figure 4.1 represents the notation used for visually illustrating the relations

between entities and group of requirements. Figures 4.2, 4.3, 4.4, and 4.5 uses this as a legend.

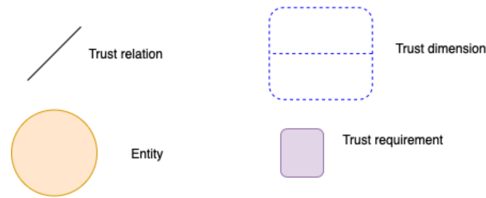

Figure 4.1: Legend overview

4.1.1 Requirements of Adherence to SLA

This group of requirements relates to reliability and validity of the data and provided service between entities in the system. It is based on the need for a marketplace to ensure the quality of its products. This group of requirements can be formally categorized into two groups: *functional* and *non-functional* requirements. The terms of *functional* and *non-functional requirements* are widely utilized in engineering. In this document we use the notion defined by Glinz et al. [28], functional requirements describe how the offering should work, while non-functional requirement relate to environment and conditions of offering.

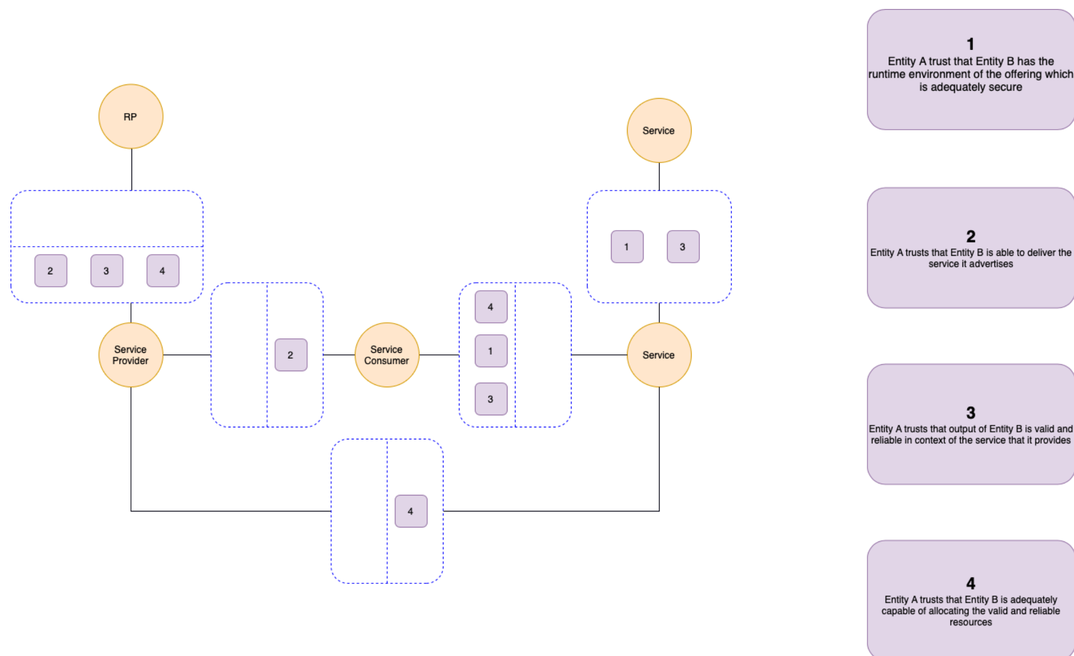

Figure 4.2: Requirements of adherence to SLA

Figure 4.2 depicts the brief overview of requirements of adherence to SLA between entities in the context of our ecosystem. In this figure 4.2 both S entities are provided by SP, representing the relation between Ss offered by the same SP. Meanwhile, S-S relations between different SPs are depicted as SC-S. This relation representation is discussed in 3.1. Table 4.2 and 4.3 demonstrate the list of functional and non-functional requirements respectively. In figure 4.2 each pairing has a set of requirements. Each set is split into two areas, which hold a requirement from the close entity to the further entity from that area. As an example, taking RP - SP pairing, SP has three requirements: **1.2**, **1.4** and **1.7** towards RP. These requirement must hold so that both entities can fulfill adherence to SLA and establish trustworthy relationships.

Explanation per each functional requirements in table 4.2 is as follows:

- 1.1 The environment of the offering here corresponds to technical specifications as operating system, version of compiler, used libraries and virtualization technology. The environment of the offering should not contain any known vulnerabilities (e.g., vulnerable kernel version)

ID	Description	Affected Entities	Reference
1.1	Entity A trusts that Entity B has the runtime environment of the offering which is adequately secure	SC - S S - S	
1.2	Entity A trusts that Entity B is able to deliver the service it advertises	SC - SP SP - RP	
1.3	Entity A trusts that data input of Entity B is valid and reliable in context of the service that it provides	SC-S	
1.4	Entity A trusts that output of Entity B is valid and reliable in context of the service that it provides	SC - S S - S SP - RP	

Table 4.2: Functional requirement of adherence to SLA

- 1.2 This requirement stems from the need for marketplace entities to provide the services they advertise.
- 1.3 According to the report provided by Accenture [7], the ethical concerns about the validity of the input data are connected with the possibility of the malicious actors accessing the entire data set and making harmful modifications. This can lead to making an unintended offering.
- 1.4 This requirement ensures that the output, whether it's data or resource or otherwise, conforms with the agreed upon specifications between the two entities.

ID	Description	Affected Entities	Reference
1.5	Entity A trusts that Entity B is capable of providing the agreed performance during offering the service in terms of specific KPI	SC-S	
1.6	Entity A trusts that Entity B does not perform any unintended or harmful action while using the offered service	S-SC	
1.7	Entity A trusts that Entity B is adequately capable of allocating the valid and reliable resources	S-SP SP-RP	

Table 4.3: Non-functional requirement of adherence to SLA

Explanation per each non-functional requirements in table 4.3 is as follows:

- 1.5 According to Faragardi et al. [24], the performance metrics in terms of services in cloud environment are essential criteria to enable the trust in a cloud environment. KPI includes availability, latency and bandwidth.
- 1.6 Harmful actions may include using extra resources, trying to access other application instances in an obscure way, or modifying the structure and code of the offered service. According to Simplified [55], one of the types of technology ethics is called *Technology transparency*, which states that the principle of work and intentions of the technology should be clearly stated. This can refer to this requirements in terms of declaring that the work and purpose of the offering.
- 1.7 This is a requirement to RP that can ensure its reliability and validity of the data and functionality of RP.

4.1.2 Requirements of Authorized Observability

Requirements of authorized observability relate to roles that handle maintenance of the entities, routing and collecting logs for enabling operation and business processes in the system. This group of requirements is based on the dynamic nature of marketplace to continuously integrate and deliver the new roles that are responsible for vital changes towards entities.

Figure 4.3 depicts the brief overview of operation between entities in marketplace. In this figure, a trust relation between SP - S despite S belonging to SP's needs to establish requirement with S due to cases, when S becomes compromised, leading to invalid log data or otherwise analytic metrics. This would lead for a need to reestablish the trust.

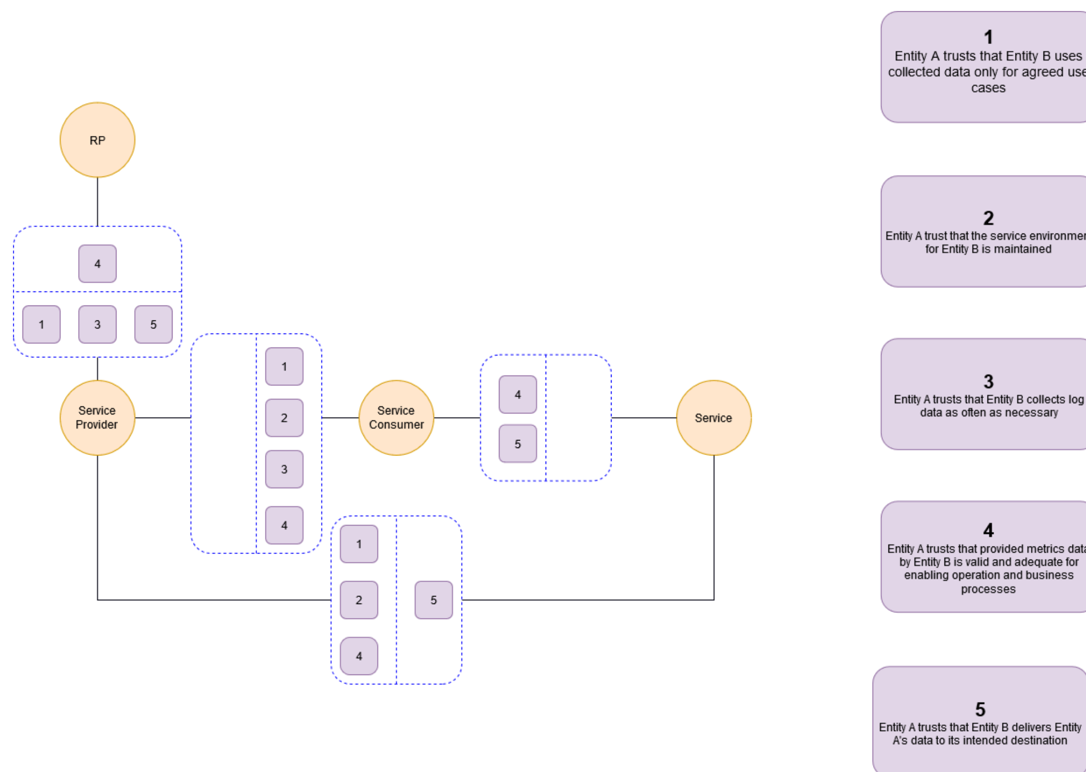

Figure 4.3: Requirements of authorized observability

This group of requirements relates to the A.12.1.2 Change Management, A.12.1.3 Capacity Management, A.14.1.3 Protecting Application Services Transactions and A.12.1.4 Separation of Development, Testing And Operational Environments chapters of ISO 270001 standard [32]. Table 4.4 represent the list of requirements that need to be fulfilled to establish authorized observability.

Explanation per each requirements in table 4.4 is as follows:

- 2.1 Non-agreed use cases might involve using the log data for acquiring privileged information, allowing inference attacks or facilitating competitive advantage.
- 2.2 According to “4.6 Monitoring, Reporting, Accounting, Auditing, and Incident Response" at NIST [15], the system should be monitored for any possible incidents as, e.g. exfiltration of data or changing the access policy for the services and resources. Monitoring should support incident response. Entity B should be able to identify the potential malicious actions. SC believes that it can obtain the up-to-date service (e.g., newest technology specification) from SP. Following in the footsteps of 4.1.1, an SP should continuously ensure that it's offerings are up-to-date and patched against the latest security vulnerabilities that might hinder the operation of its SC.
- 2.3 In case the log data is collected in more than necessary times, then the possibility of inference attacks increases. This requirement ensures that log data is collected with the time interval which is enough to assess the state of entities and guarantee continuous service according to agreements.
- 2.4 According to “3.7.5. Federation Accounting And Billing" at “Cloud Federation Reference Architecture" by NIST [15], collecting the logs for the business processes involve accounting and correct billing and risk assessment. This requirement ensures that entities provide up-to-date and correct data for operation and business processes, such as analytics metrics.
- 2.5 This requirement ensures that entities are responsible for successful communication between entities through enforcing valid routing rules.

ID	Description	Affected Entities	Reference
2.1	Entity A trusts that Entity B uses collected data only for agreed use cases	SP - RP SC - SP	**ISO A.12.1.2:** Change Management Standard [32] **NIST:** Acceptable Use Policies [15]
2.2	Entity A trust that the service environment for Entity B is maintained	SC-S SC - SP	**NIST 4.6:** Monitoring, Reporting, Accounting, Auditing, and Incident Response [15]
2.3	Entity A trusts that Entity B collects log data as often as necessary	SP - RP SC - SP	**ISO A.12.1.2:** Change Management standard [32]
2.4	Entity A trusts that provided metrics data by Entity B is valid and adequate for enabling operation and business processes	RP - SP SC - S SC - SP SP - S	**NIST 3.7.5.:** Federation Accounting And Billing [15]
2.5	Entity A trusts that Entity B delivers Entity A's data to its intended destination	S - SP SP - RP SC - S	

Table 4.4: Authorized observability requirements

4.1.3 Security properties requirements

Security properties relate to confidentiality, authenticity and integrity of communication between entities, controlling of unsolicited action in the network and principles of firewalls work in the system. According to Martucci et al. [47], the confidentiality and integrity of communication between entities are considered as “hard trust mechanisms" that can resolve the data privacy issues, however, need to be accompanied with “soft trust" mechanism that involves the human emotion factor in establishing requirements.

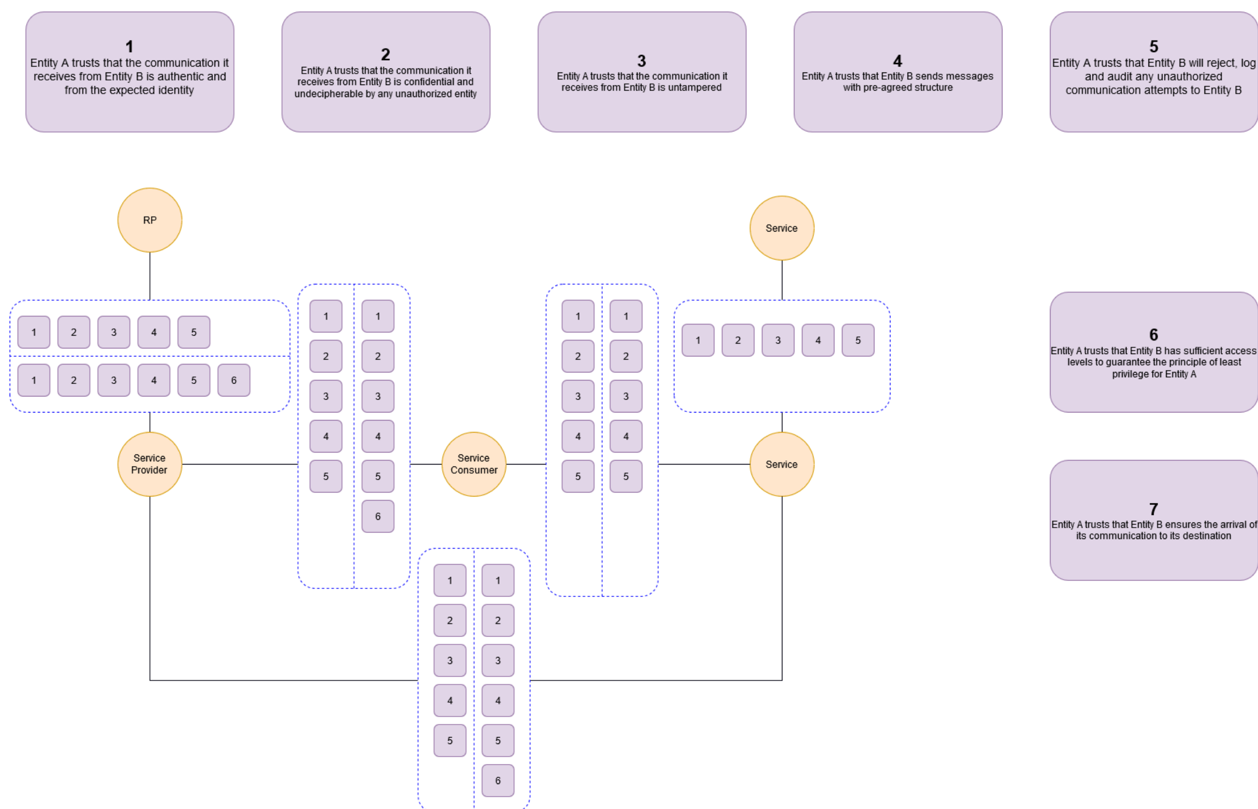

Figure 4.4: Security properties overview

Figure 4.4 depicts the brief overview of security properties of communication between entities in marketplace.

This group of requirements relates to the ISO 27001 - Annex A.9: Access Control, ISO 27001 - Annex A.10: Cryptography and A13 standards of ISO 270001 standards [32] to enhance this group of requirements [32]. Table 4.5 represent the list of requirements that need to be fulfilled to ensure that security properties of the system were not broken.

Explanation per each requirements in table 4.5 is as follows:

- 3.1 According to “4. General Cloud Environments" at NIST [13], external entities should not be able to communicate with entities in the system. The entities in the marketplace are advised to be placed in Virtual Protected Network (VPN), and external access should be controlled by firewalls. This requirement persists between all entities which actively communicate withing the context of the marketplace. In this case, identities of entities in marketplace are known for all entities.

- 3.2 ISO A.10: Cryptography ensures that communication between entities is encrypted. This requirement persists between all entities which actively communicate withing the context of the marketplace.

- 3.3 This requirement persists between all entities which actively communicate within the context of the marketplace. This requirement ensures that the content of messages between entities and communication means between entities are not modified.

- 3.4 According to section “3.7.6. Federation Portability & Interoperability" of “Cloud Federation Reference Architecture" by NIST [15], the ideal deployment of the system should have the entities

ID	Description	Affected Entities	Reference
3.1	Entity A trusts that the communication it receives from Entity B is authentic and from the expected identity	RP - SP SP - SC SC - S S - S	**NIST 4:** General Cloud Environments
3.2	Entity A trusts that the messages it receives from Entity B is confidential and undecipherable by any unauthorized entity	RP - SP SP - SC SC - S S - S	**ISO A.10:** Cryptography [32]
3.3	Entity A trusts that the messages it receives from Entity B is untampered	RP - SP SP - SC SC - S S - S	
3.4	Entity A trusts that Entity B sends messages with pre-agreed structure	RP - SP SP - SC SC - S S - S	**NIST 3.7.6.:** Federation Portability & Interoperability [15]
3.5	Entity A trusts that Entity B will reject, log and audit any unauthorized communication attempts to Entity B	RP - SP SP - SC SC - S S - S	**ISO A.13.2.3:** Electronic Messaging [32] **ISOA.13.2.4:** Confidentiality or Non-Disclosure Agreements [32]
3.6	Entity A trusts that Entity B has sufficient access levels to guarantee the principle of least privilege for Entity A	SP - RP S - SP SC - SP	**ISO A.9.1.1:** Access Control Policy [32].

Table 4.5: Security properties requirements

that follow unified management interface. In real life scenario, this can be achieved through API and unified data format that can boost the *service interoperability*. This requirement persists between all entities which actively communicate within the context of the marketplace. This requirement relates to firewall policies and ensures that the entity will not get any malfunctioning or malicious packets.

- 3.5 This requirement persists between all entities which actively communicate within the context of the marketplace. This requirements showcases the need for access control between entities. According to A.13.2.3 Electronic Messaging and A.13.2.4 Confidentiality or Non-Disclosure Agreements [32], the access policy should be constantly reviewed taking information security risks into consideration.
- 3.6 This requirement ensures that each entity has enough access levels to authorize access to resources and services based on their least need [53].

4.1.4 Requirements of Availability and Resilience

Group of requirements of availability and resilience relate to reliability, fault recovery, separation of the tenancies of entities. These requirements need to be ensured during the deployment of the system.

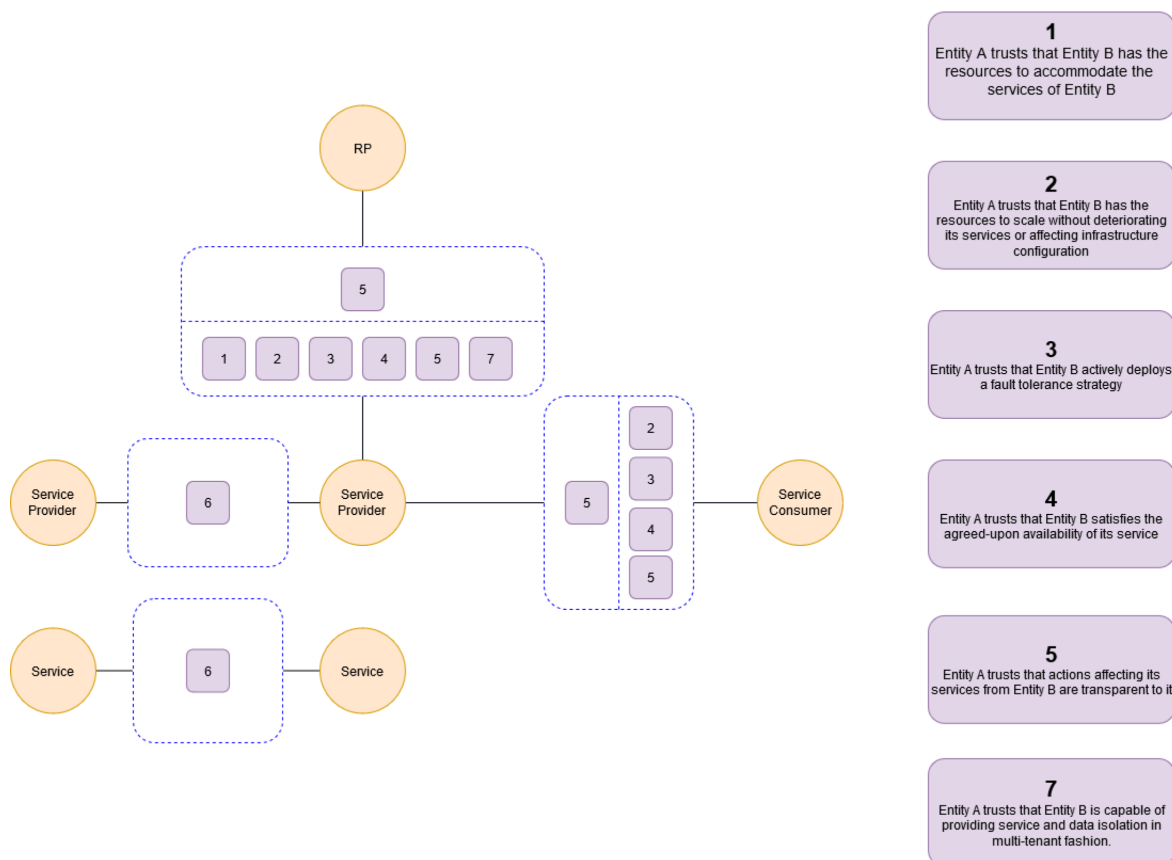

Figure 4.5: Requirements of availability and resilience

Figure 4.5 depicts the brief overview of availability between entities in marketplace. This group of requirements relates to the A.12.3.1 Information Backup and A.12.1.3 Capacity Management A.14.2.3 Technical Review of Applications After Operating Platform Changes of ISO 270001 standards [32]. Table 4.6 represent the list of requirements that need to be fulfilled to achieve the requirements for ensuring availability and resilience of infrastructure. In this figure 4.5, we can examine the requirement between SP and SP, unlike in aforementioned requirements. This subsection provides the rules that need to be established between SP - SP, and, therefore, another SP is depicted in the figure 4.5.

Explanation per each requirements in table 4.5 is as follows:

- 4.1 According to “2.1.4. Federation Governance" at “Cloud Federation Reference Architecture" by NIST [15], the sites need to have the list of available resources that it can use. This sites may

ID	Description	Affected Entities	Reference
4.1	Entity A trusts that Entity B has the resources to accommodate the services of Entity B	SP - RP	**NIST: 2.1.4.**: Federation Governance
4.2	Entity A trusts that Entity B has the resources to scale without deteriorating its services or affecting infrastructure configuration	SP - RP SC - SP	**NIST: 6.4.:** Federations and Trust Federations at Scale
4.3	Entity A trusts that Entity B actively deploys a fault tolerance strategy	SC - SP SP - RP	**ISO: A.12.3.1**: Information Backup [32]
4.4	Entity A trusts that Entity B satisfies the agreed-upon availability of its service	SC - SP SP - RP	
4.5	Entity A trusts that actions affecting its services from Entity B are transparent to it	SP - RP SP - SC	**NIST: 8.5:** Information Security
4.6	Entity A trusts that it operates in a separate tenancy slice than Entity B	SP - SP S - S	**ISO A.12.1.4:** Separation And Development, Testing And Operational Environments [32].
4.7	Entity A trusts that Entity B is capable of providing service and data isolation in multi-tenant fashion	SP - RP SC - SP	**ISO A.14.1.2:** Securing Application Services on Public Networks [32]

Table 4.6: Availability and resilience requirements

selectively allow other entities to access resources. This should be agreed with resource metadata. The resource metadata should be placed in a persistent location. RP needs to provide the technical capabilities to all service providers to operate adequately.

- 4.2 According to “6.4. Federations and Trust Federations at Scale" at “Cloud Federation Reference Architecture" by NIST [15], as the scale of the marketplace increases, the challenges of the general distributed system can occur in the system that includes replication, estimation, management and caching. Scaling of the system should update path to relevant resource information. This requirement ensures that adding new services and service providers or removing them will not affect the fulfillment of the agreed service quality.

- 4.3 This requirement ensures the data recovery and protection against data loss.

- 4.4 According to Martucci et al. [47], cloud service as an infrastructure can be a strategic asset at the national level, and needs be fully functioning and producible withing the national borders. Thus, availability of these services is essential to enable trustworthy relationship between entities, especially in the context of a marketplace. Moreover, system resiliency is based on self-recovery process or backup services for the situations when the main service is not available.

- 4.5 According to “8.5 Information Security" at “Cloud Federation Reference Architecture" by NIST [15], all federations would want to have an opportunity to track the usage of the system and record auditing logs. This ensures that the performed actions are transparent, e.g. all actions are logged in the system, and can be assessed later for auditing purposes. This information could be primary used for cost measurements. In addition to that, this will eliminate the issues during the scaling, when the entities could consume the non-trivial number of resources. Therefore, all operations that take place in the system should be transparent.

- 4.6 A tenancy slice in this context relates to an entity’s ability to perform its function without the interference from another entity. As an example, a tenancy slice can reflect as a separation in the virtual environment such as two Ss operating in different containers or virtual machines. The separation of each entity should comply with given rules, and user tenancies should be defined in the system.

- 4.7 This requirement complies with “Privacy by Design" strategy described by Zeng et al [65]. In this requirement, the entity must have an architecture that allows other entities belonging to it to reside in the separate environment, e.g. virtual box or container.

4.2 Mapping Requirements to Establish Trust into Service Mesh Features

Section 3 proposed requirements to establish trust in a decentralized service exposure marketplace. This section focuses on mapping these requirements into the features of service mesh technology. Figure 4.6 demonstrates how the proposed requirements can be mapped into features of service mesh federation.

Service discovery [45] is a feature in Istio that is relevant to having a globally unique namespaces across the clusters. This helps to find the needed service in the marketplace and can be mapped into *Security properties requirements* and *Requirements of availability and resilience*.

Load balancing is a feature that provides the traffic routing across the network in a service mesh, and can be mapped into *Requirements of availability and resilience* and *Requirements of authorized observability*.

Fault tolerance is a feature that provides the handling of erroneous state of servers and handling the failure the network in a service mesh and can be mapped into *Requirements of availability and resilience*.

Traffic monitoring/ Transparency / Logging is a feature that allows to log data transmission in the system, authorized or unauthorized access to resources. This Istio feature can be mapped into *Requirements of authorized observability*, *Requirements of availability and resilience* and *Security properties requirements*

Circuit breaking [45] is a feature in Istio that help to prevent latency spikes and impact of failures. Circuit breaking can be mapped into *Requirements of availability and resilience*.

Authentication and access control is a feature of service mesh that prevents from unauthorized access and can be mapped into *Security properties requirements*.

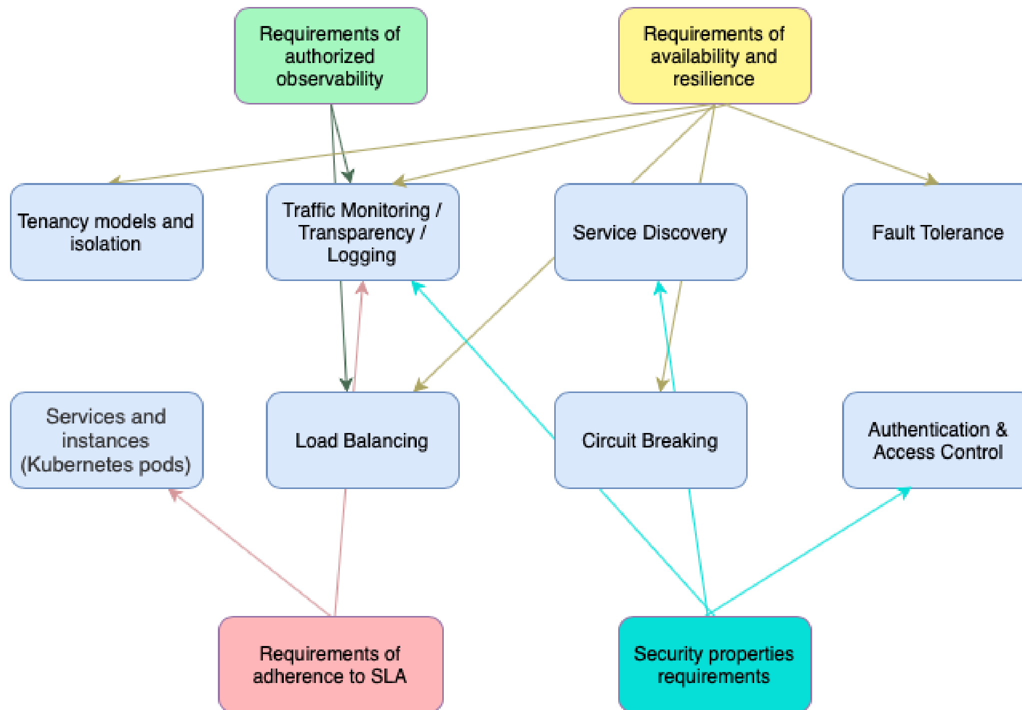

Figure 4.6: Mapping of requirements to features of Istio

4.3 Trust Engine Design and Proof of Concept

This work presents a PoC for *Trust Engine*, a trust management module for a decentralized virtual services marketplace "Nubo" described in section 3.1. The Trust Engine mitigates the limitations of other trust management systems discussed in section 2.6, mainly the lack of policy enforcement and the computational complexity of trust evaluation.

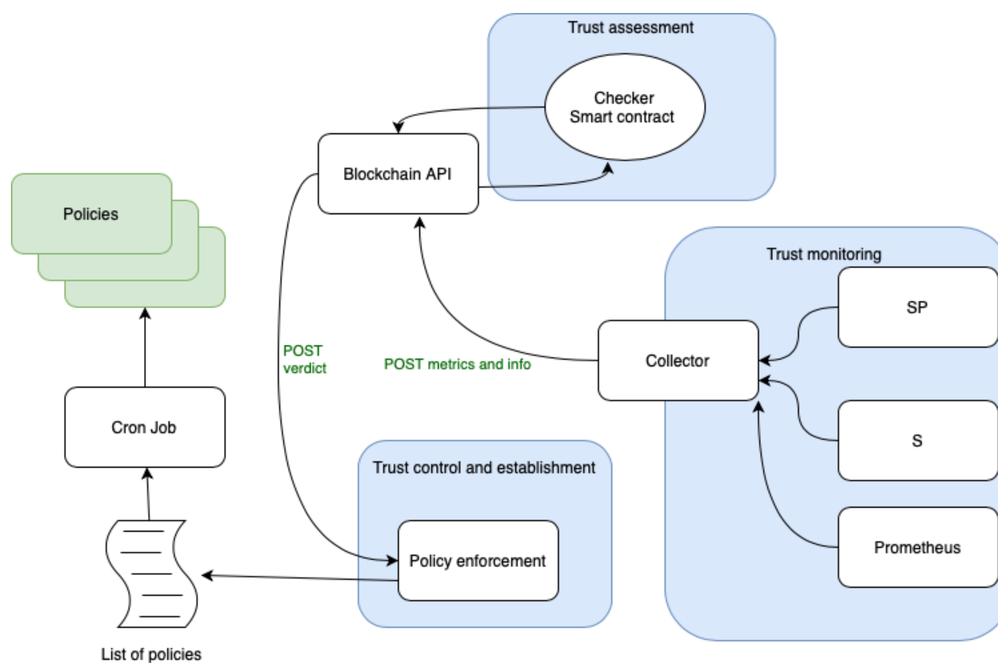

Figure 4.7: Trust Engine system design.

Figure 4.7 depicts interaction between the three main components of the Trust Engine: *Checker*, *Collector*, and *Policy enforcement*. In short, the Collector gathers metrics and other relevant information about entities in the system, and provides them for Checker smart contract. The Checker then assesses

the trustworthiness of system entities based on the requirements to establish trust discussed in section 3.2, and provides a verdict for the Policy enforcement. Then Policy enforcement applies the necessary actions to ensure the fulfilment of the requirement. This process ensures that the Trust Engine follows the guidelines of the trust management process discussed in section 2.1.

Component	Technology	Description
Service Mesh	Istio v1.4.3	Single mesh deployment with a single control plane and namespace tenancy model
Orchestration	Kubernetes on Minikube	Single cluster, with one master node
Virtual Machines	Openstack Nova	Debian based with 16GB of allocated ram and 4 processing cores
Blockchain	Hyperledger fabric	Single peer and single orderer

Table 4.7: Testbed specification overview

Table 4.7 gives an overview of used technologies and specs for developing the practical implementation of the PoC. This work uses Istio service mesh technology deployed along a single Kubernetes cluster, comprised of a single node. We utilize Minikube's Kubernetes implementation to run our single cluster. The cluster is deployed on the Openstack compute virtual machines (Nova). Figure 4.8 depicts the brief architecture of Trust Engine deployed of the PoC.

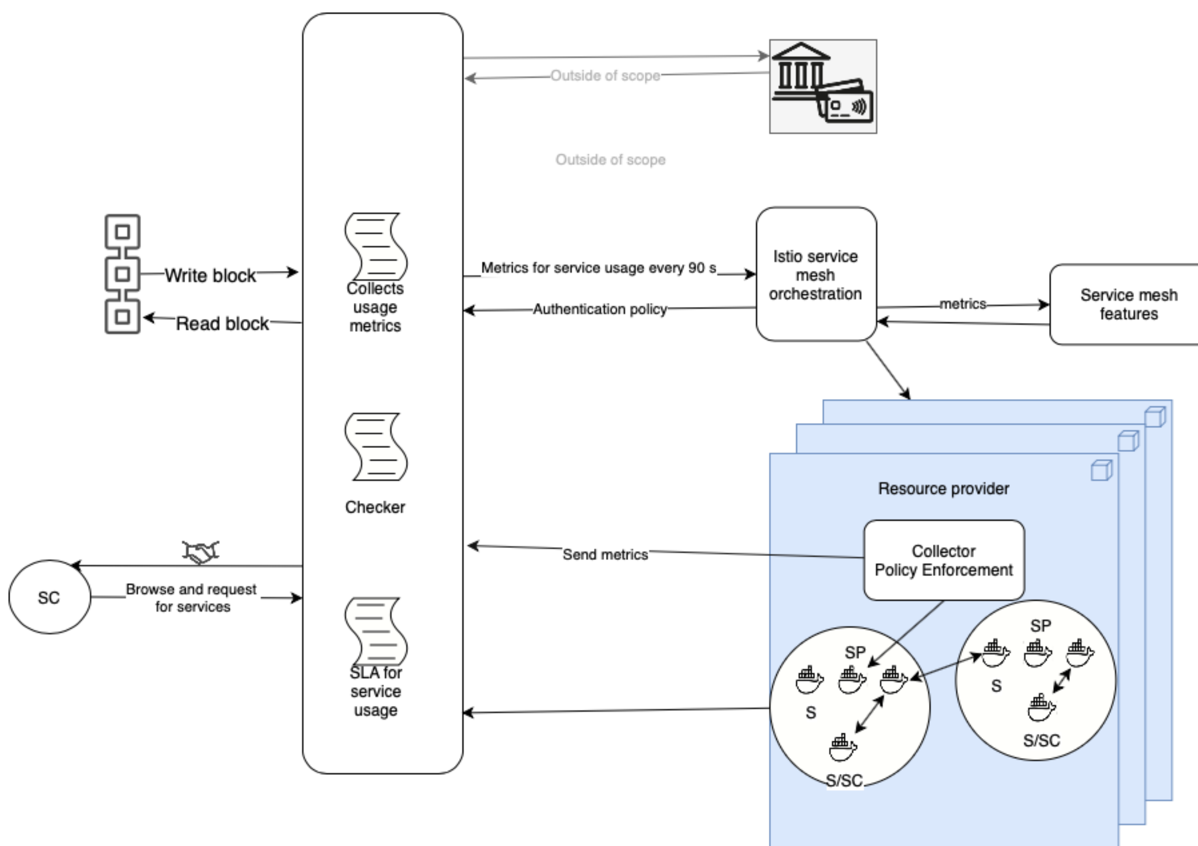

Figure 4.8: Trust management system in a decentralized service exposure marketplace using smart contracts and service mesh technology Istio

In the upcoming subsections 4.3.1, 4.3.2 and 4.3.3, we take a closer look at each of the aforementioned Trust Engine components.

4.3.1 Checker

Checker is a component of a trust management service that is responsible for *trust assessment*. Checker is a smart contract running in the Blockchain API. This program gets the list of the requirements, metrics and logs stored in the Blockchain. These logs and metrics are gathered by Collector described in 4.3.2, and contain information about the entities of the system. The execution of the Checker is triggered by the Collector, which runs every 1 minute. Then Checker applies each of requirements to the collected metrics and logs. As the result, a verdict per each entity is given. The information about the how Checker verdicts and API documentation is given in appendix A.

4.3.2 Collector

Collector is a component of a trust management service that is responsible for *trust monitoring*. This component of Trust Engine collects *Telemetry data* from monitoring tools in the deployment, e.g. *Prometheus*. This information is supplemented with hashes of performed queries and results from the monitoring tools, nuances and timestamps before being sent to the Blockchain API which documents the information to the Blockchain before triggering the Checker component of the trust-engine.

For the PoC implementation, this component is deployed in the trust-engine namespace. The collector is configures as a CronJob that runs every 1 minute[7], a Kubernetes resource which mimics the behaviour of a Crontab in Linux. The configuration file for the resource can be viewed in B.1. In line 12, we can observe that the Istio sidecar injection is not enabled for this component. This stems from the nature of operation of sidecars and CronJob resources. When a CronJob pod terminates after performing the job, the Istio sidecar prevents the deployment from terminating. This required us to either adjust the sidecar deployment with custom health-check metrics or turn it off for the PoC. We chose the latter option for this work, since the sidecar was not necessary for demonstrating the capabilities of the Trust Engine. The metrics collected by the Collector are described in the API documentation, given in appendix A.

4.3.3 Policy Enforcement

Policy enforcement is a component of a trust management service that is responsible for *trust control, establishment and reestablishment*. Policy enforcement is necessary for ensuring the consequences of malicious behavior between entities in the marketplace. Policy enforcement gets a verdict from the Checker per each entity. Thus, the execution of Policy enforcement takes place whenever the Checker gives a new verdict. After getting the verdict, the responsibilities of Policy enforcement mechanism include allowing or denying the traffic between entities, revoking access to some resources and shutting the entities. Policy enforcement gets a new template from the already existing list of templates based on type of policy, and creates a new policy. This new policy is added to a directory of policies. After several policies were created, they are all applied to the system. For this work, we prepared a PoC policy *poc-policy*. For the PoC implementation, this service is deployed in the trust-engine namespace. It requires a service account to control plane to be able to apply policies.

This component utilizes the orchestration feature of a service mesh. Policy enforcement is in a charge of reconfiguration of the orchestrated clusters and making sure that the component-level policies are in agreement with the requirements to establish trust. The information about which policies are getting applied and API documentation is given in appendix A.

[7] https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

5 Security Analysis and Performance Evaluation

This chapter provides the security and performance analysis of the system after integrating the developed PoC that we discussed in chapter 3. Section 5.1 provides a security analysis of the system described in chapter 4, while in section 5.2 we can get a performance evaluation of our PoC system.

5.1 Security Analysis

As described in 3.4, the main goals of an adversary are: *Disrupting SLA, Bypassing authorization, External Security properties violation* and *Disrupting of availability and resilience*. In this section, we provide a security analysis for each goal, stating what was covered in the PoC.

5.1.1 Disrupting SLA

The attacks that can lead to disrupting SLA can be prevented by the Trust Engine by using *Traffic monitoring/Transparency/Logging* feature of service mesh described in section 4.2. Trust Engine collects logs from sidecar containers and Prometheus, not from tampered logs, and then ensures that SLA is satisfied by using smart contact. The results are then transparent for all peers of the system, along with the hash of collected telemetry and verdict. Moreover, attacks towards hijacking of resources can be detected using the same manner which triggers Trust Engine to enforce a policy to mitigate the resource hijacking.

5.1.2 Bypassing Authorization

The attacks that can lead to bypassing authorization can be prevented by Trust Engine by using *Traffic monitoring/Transparency/Logging* feature of service mesh described in 4.2. Trust Engine collects logs from sidecar containers and Prometheus, and then ensures that any unauthorized access was detected and documented in a Blockchain. The results are then transparent for all peers of the system, along with the hash of collected telemetry and verdict. This in turn triggers the Trust Engine to enforce a restricting policy towards a compromised instance.

5.1.3 Security Properties Violation

The attacks that can lead to security properties violation can be prevented by Trust Engine by using *Traffic monitoring/Transparency/Logging*, *Service discovery* and *Authentication and access control* features of service mesh described in 4.2. However, this attack goal is outside of the scope of this current work's implementation.

5.1.4 Disrupting of Availability and Resilience

The attacks that can lead to disrupting of availability and resilience can be prevented by Trust Engine by using *Traffic monitoring/Transparency/Logging*, *Service discovery*, *Circuit breaking*, *Load balancing*, *Tenancy models and isolation* and *Fault tolerance* features of service mesh described in 4.2. However, this attack goal is outside of the scope of this current work's implementation.

5.1.5 Attack Scenario Proof-Of-Concept (PoC)

Figure 5.1 demonstrates an attack scenario towards disrupting SLA. In this scenario malicious entity (SP for this scenario) would report an agreed upon gauge metric (60 seconds). Gauge metrics, unlike counter metric, can go up and down by definition. This scenario represents a consumption of a Service (S) by our SC. For this work we call the gauge metric a PoC metric. In the consumption increases over the allowed rate, the SP is allowed to increase the reported gauge metric. The SC regularly consumes the Service (S) at a constant interval of 1 request/min. The SP reports the gauge metric at a regular interval of every 15 seconds. The SP then attempts to misbehave and report higher gauge metric (90 seconds) despite the constant rate of consumption of the SC. In this aforementioned scenario the requirement *"Entity A trusts that Entity B collects log data as often as necessary"* in 4.1.2 and the requirement *"Entity A trusts that Entity B is able to deliver the service it advertises"* in 4.1.1 were not fulfilled, such that the trustworthy relationship between malicious SP and other entities were not established.

In this scenario, the Trust engine collects the actual usage information through Traffic monitoring feature (Prometheus) of a service mesh and documents that in a Blockchain along with the hash of

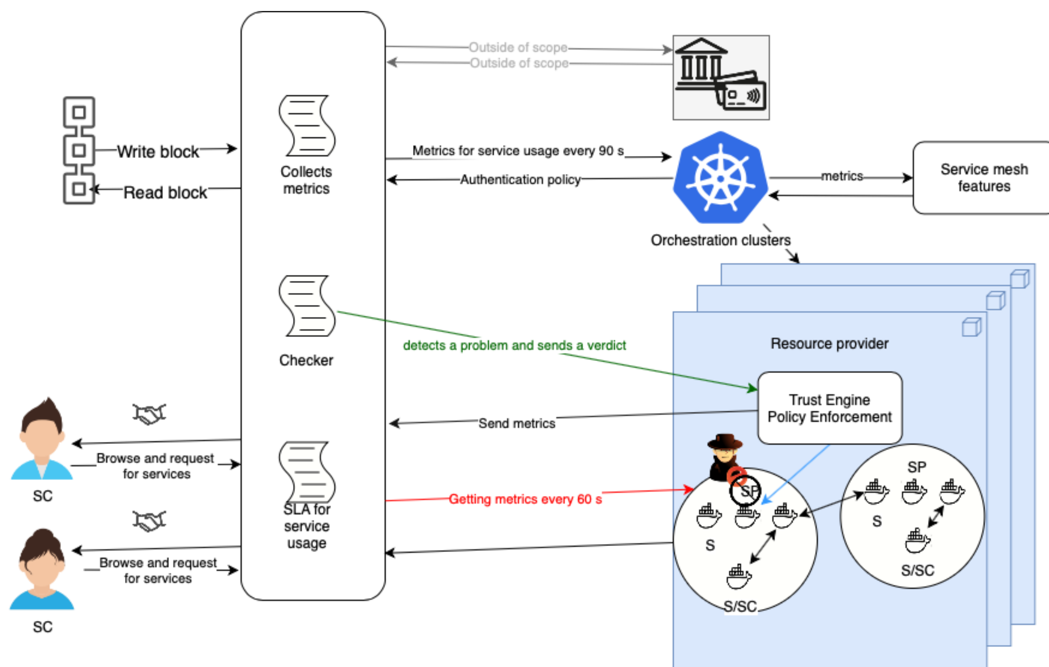

Figure 5.1: Adversary scenario for this work

the log and timestamp to ensure transparency and non-repudiation. This triggers the smart contract running on a Blockchain API which ensures that the collected information conforms with SLA stored on a Blockchain. Then a verdict is sent to a Policy enforcement to penalize the malicious entity and reconfigure metric collection to conform with SLA by applying an updated policy. The success of the Trust engine showcases its ability to enforce policies and verdicts, which is, as mentioned in section 4.3, one of the limitations of current trust management systems for decentralized marketplaces.

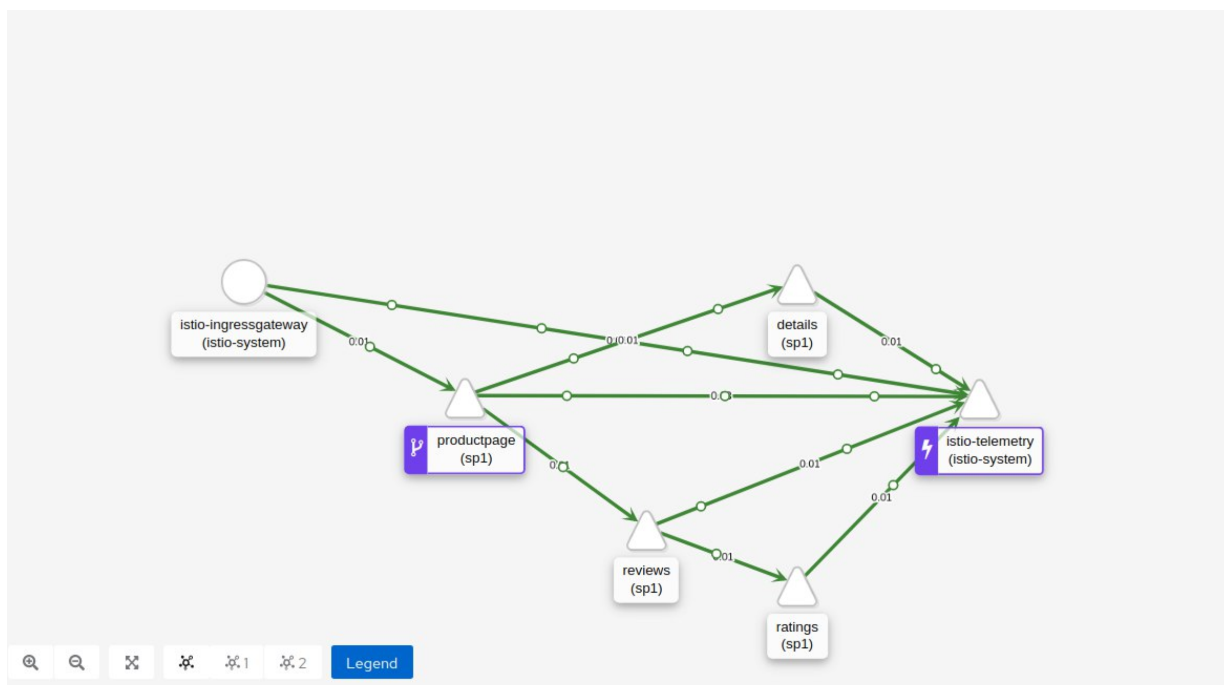

Figure 5.2: SC consuming S and SP reporting metrics

For this scenario we utilize the same setup described in section 4.3. The cluster consists of seven namespaces: *sp1*, *trust-engine*, *istio-system*, *default*, and three Kubernetes system namespaces. Table 5.1 showcases the deployed namespaces and pods in each namespace. For this PoC we use the *bookinfo* Istio

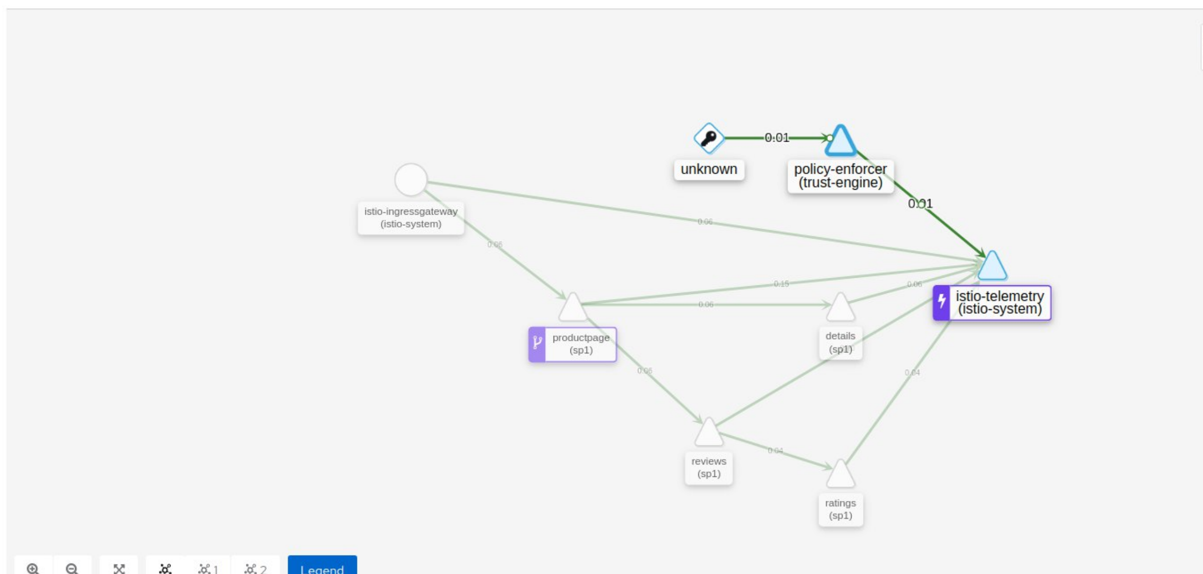

Figure 5.3: Trust engine receiving verdict from the Blockchain and applying policy

example as the services provided by SP. Figure 5.2 highlights the deployed services in the mesh. The SC consumes the Service (S) *productpage* at a regular interval of 1req/min, as illustrated on the edges of the graph. The Service (S) *productpage* then invokes several internal communication. Each service generates metrics which is collected by the Istio-telemetry service, most notably our PoC metric. Figure 5.3 highlights the event of misbehaviour which invokes the *policy-enforcer* module from the trust-engine. The Policy enforcement receives verdicts from the Blockchain, illustrated as the key entity and named unknown to indicate that it's outside the mesh, and applies the verdict to the mesh.

5.2 Performance Evaluation

In this section we take a look at the performance of the Trust Engine, when it comes to applying policies, and collecting telemetry. Further subsections 5.2.1, 5.2.2 and 5.2.3 provides more details on the experiments.

5.2.1 Overall Detection and Remediation

Figure 5.4 showcases the misbehaviour detection in our previously explained scenario. This figure is pulled directly from Prometheus in the running PoC deployment. The metric collected is our PoC metric, and the plot represents the reporting of the metric by the SP. The red spikes represent a metric reporting conforming with the SLA agreement, in this scenario the reported gauge value was set to a value of 120. The green spikes represent the misbehaviour, reporting a value of 180 instead. As this graph is pulled from Prometheus, the x-axis represent the minutes of the hour, the query was run for an interval of 15 previous minutes. The reporting by the SP was performed once every minute. From the figure, we see that it took an interval of 5 minutes to detect, and remediate the misbehaviour. Ideally, a lower number of green spikes is desired as that would indicate faster remediation of misbehaviour. However, the current findings indicate acceptable performance. The multiple green spikes can be explained by the execution time of the blockchain smart contract in addition to the network latency.

5.2.2 Scalability and Bottleneck

To measure the scalability of the Trust Engine, we inspect the execution time of key operations in its components under an increasing load.

Figure 5.5 depicts the performance of the key execution block for the Collection module in the Trust Engine. For this experiment, we provided the collection module with mocked PoC metrics, increasing in a logarithmic step. The mocked PoC metrics simulate metrics provided by a single SP, reporting a single PoC metric for each Service (S) it provides. For each step, we measure the execution time of the code block responsible for parsing the metrics and creating the request for the Checker module. We

Namespace	Description	Pods
default	This namespace is devoted for objects with no other namespace	
istio-system	Namespace that contains services as Prometheus, Grafana, Kiali, and other	
kube-node-lease	Kubernetes system namespace that is responsible for improvement of performance while cluster scales, and help determine the availability of a node [41]	
kube-public	Kubernetes system namespace that stores all resources that needs to be publicly available for all users	
kube-system	This namespace is for objects created by the Kubernetes system, and contains service accounts that are needed for execution of the Kubernetes controllers	
sp1	PoC SP	detals-1, bookinfo-1, productpage-1, ratings-1, reviews-1, reviews-2, reviews-3
trust-engine	This namespace contains Collector, Checker and Policy enforcement services	collector-1, collector-2, collector-3, policy-enforcer-1

Table 5.1: Namespaces overview

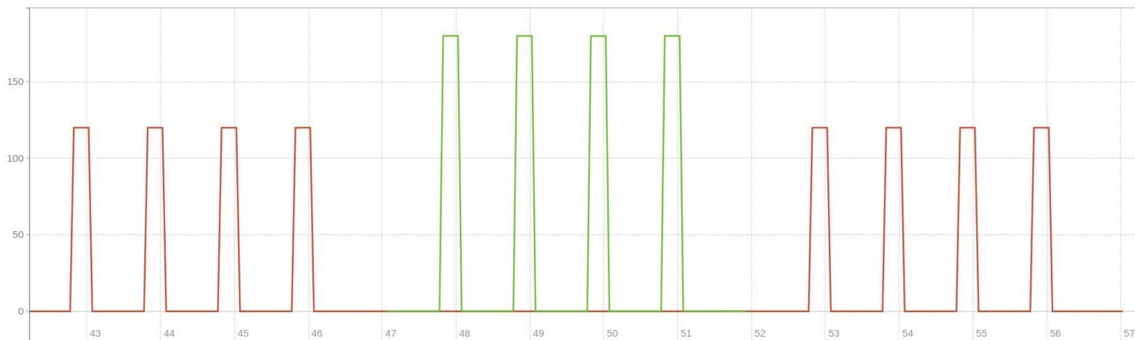

Figure 5.4: Time to detect misbehaviour and apply policy

preform this measurement 30 times for each logarithmic step, collecting the results in a data bucket per logarithmic step. The red marker in figure 5.5 represent the mean value of each data bucket. The green bars in figure 5.5 represents a confidence interval of 95 % for this measurement, based on the standard deviation of the data bucket from the mean value.

The results indicate that the overhead for this operation is almost negligible as the execution time for each data bucket increase with a linear distribution. At very low number of metrics of 1 or 10 we can see almost identical execution time. This can indicate that the operation overhead for the utilized programming language overshadows the execution time for the block. As the number of metrics increase to 100 and 1000, we notice that the increase in execution is proportional to the increase in metrics. This indicates the linearity of the execution time for the block as the load increases. The higher variance of for the confidence interval at largest data bucket can be explained by the lack of threading in the current implementation, which may lead to slight variance (2-4 millisecond) in function calls executed for this code block, most notably the hashing function used to provide a mac for the received metrics.

Figure 5.6 depicts a similar experiment, this time, however, we measure the performance of the two

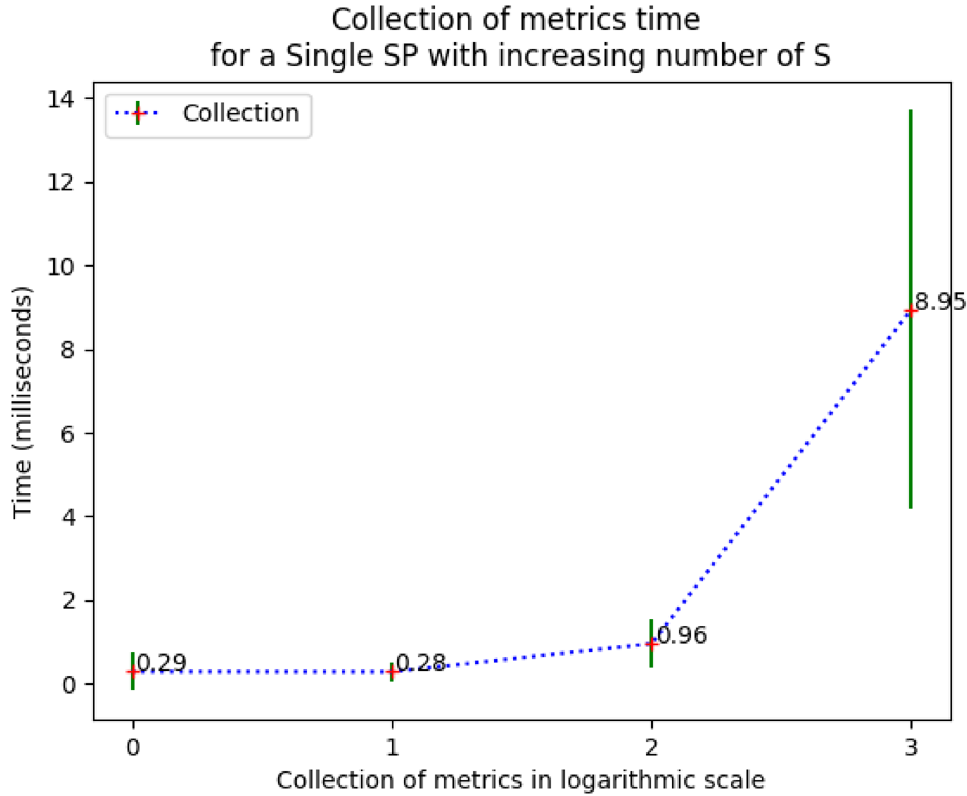

Figure 5.5: Collection of metrics time for a single Service Provider (SP) with increasing number of Service (S)

key execution blocks for the Policy enforcement module in the Trust Engine. The first code block relates to creating a policy from an existing policy template. This block entails multiple I/O operations such as opening, reading and writing to files. The second block relates to applying the created policies. This block entails some network communications with the control plane of Kubernetes and Istio. For the experiment, we provided mocked verdicts to an increasing number of Service (S) of a single SP in a logarithmic step. In each step, we measure the performance of each code block for 30 times and collect the data in a bucket. The red markers over the blue dotted line in figure 5.6 depict the mean execution time for the policy creation for each data bucket. We calculate a confidence interval of 95 % based on the standard deviation from the mean for each measurement. This is depicted in figure 5.6 by a green line, and purple line for policy creation and applying policies respectively.

The results show that the apply policy time is significantly greater than the create policy time across all loads. This can be explained by the network overhead, or the implementation for the Kubenetes/Istio command line tool utilized by the apply policy code block. As the number of files to apply increases, the command line tool requires more requests to the control plane. This network overhead also explains the difference in the confidence interval range between the two block. Finally, despite the large number of I/O operations for the policy creation, the performance of the code block seem to increase linearly and with limited variance. This can be explained by the programming language optimization for utilizing the file descriptors.

5.2.3 Delivering Predictable Operational Performance

In previous section, we observed the execution time of the Trust Engine components with an increasing number of Services (Ss) and a single SP. In this section, we perform experiments to guarantee the same performance would hold under different SP-S permutations. For this work, we decided to pursue the factors of 10 (1, 2, 5, 10) for our collected metrics, which would make the permutations of SP-S pairs: (1,10), (5,2), (2,5) and (10, 1). Choosing permutations of 10 Ss would be resource-effective, since the number of permutations of 100 Ss is 9, while number of permutations of 1000 Ss is 16.

For the experiments setup, we perform 30 trial runs. We repeat these experiments to compute the

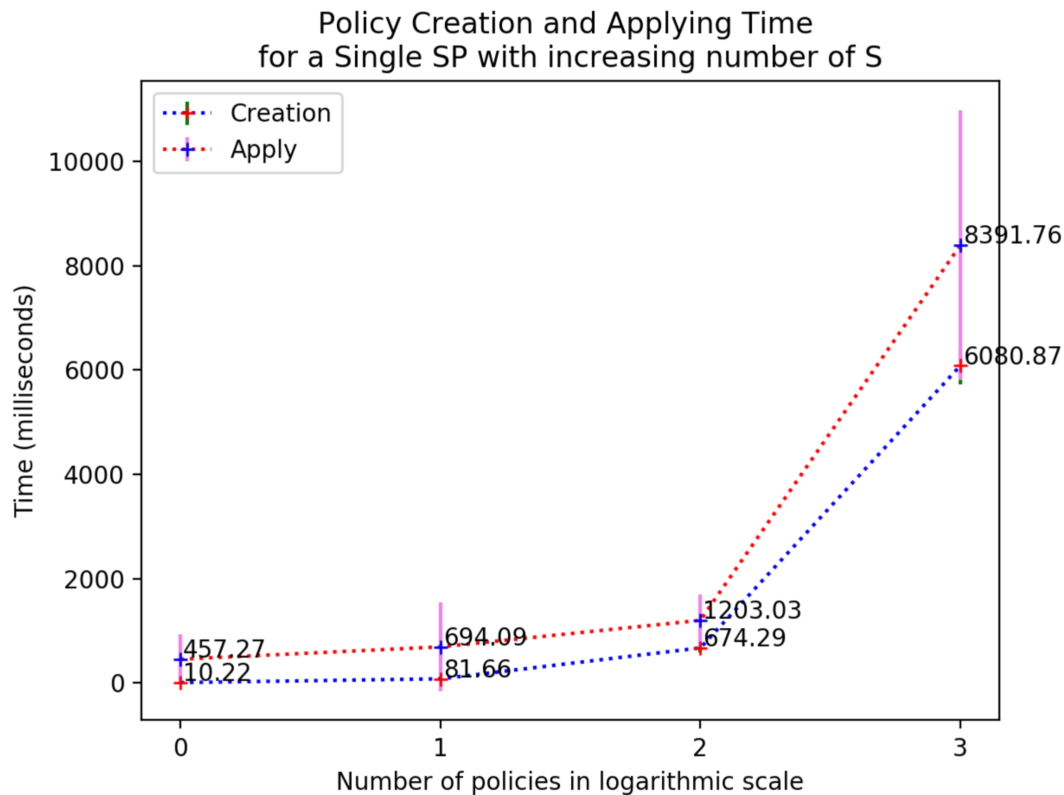

Figure 5.6: Policy creation and applying time for a single SP with increasing number of S

time with a confidence interval of 95 %.

In figure 5.7 show, we compare the time to collect PoC metric, and vary the number of SP, S. For each step, we measure the execution time of the code block responsible for parsing the metrics and creating the request for the Checker module. We expect to parse and create single request with 10 PoC metric in different permutations: 1 SP with 10 Ss, 5 SPs with 2 Ss per each SP, 2 SPs with 5 Ss per each cases. As we can observe, the average time to collect 10 PoC metric between all 4 permutations is 0.3 milliseconds (denoted as dotted green line). The results showcase that the mean of each permutation is relatively close to the mean of all the permutation. This shows that the distrubution of SP and S does not affect the end performance for the Collector module.

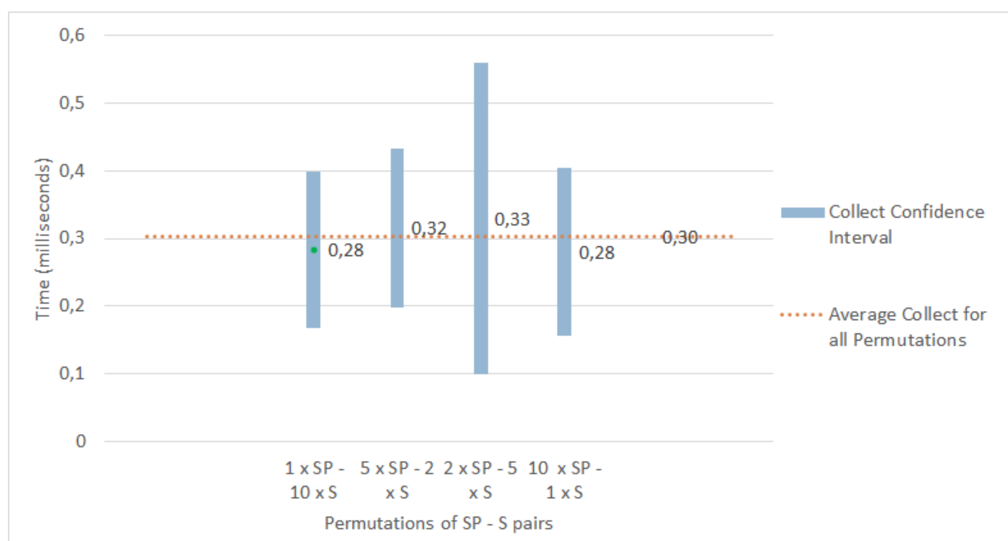

Figure 5.7: Comparison between metrics collection time for various number of SP and S

In figure 5.8, we compare the time to create and apply *poc-policy*, and vary the number of SP, S. As the previous experiment, we expect to create 10 PoC metrics in different permutations: 1 SP with 10 Ss, 5 SPs with 2 Ss per each SP, 2 SPs with 5 Ss per each SP, and 10 SPs with 1 S per each SP. As we can observe, the average time to apply 10 *poc-policies* is 573 milliseconds (noted with dotted green line) between all 4 permutations, and 87 milliseconds for creation. As we can observe in figure 5.8, the is a higher variance from average value for applying policies for all 4 permutations. This variance is connected with the network communication issues that accompany policy applying in Istio.

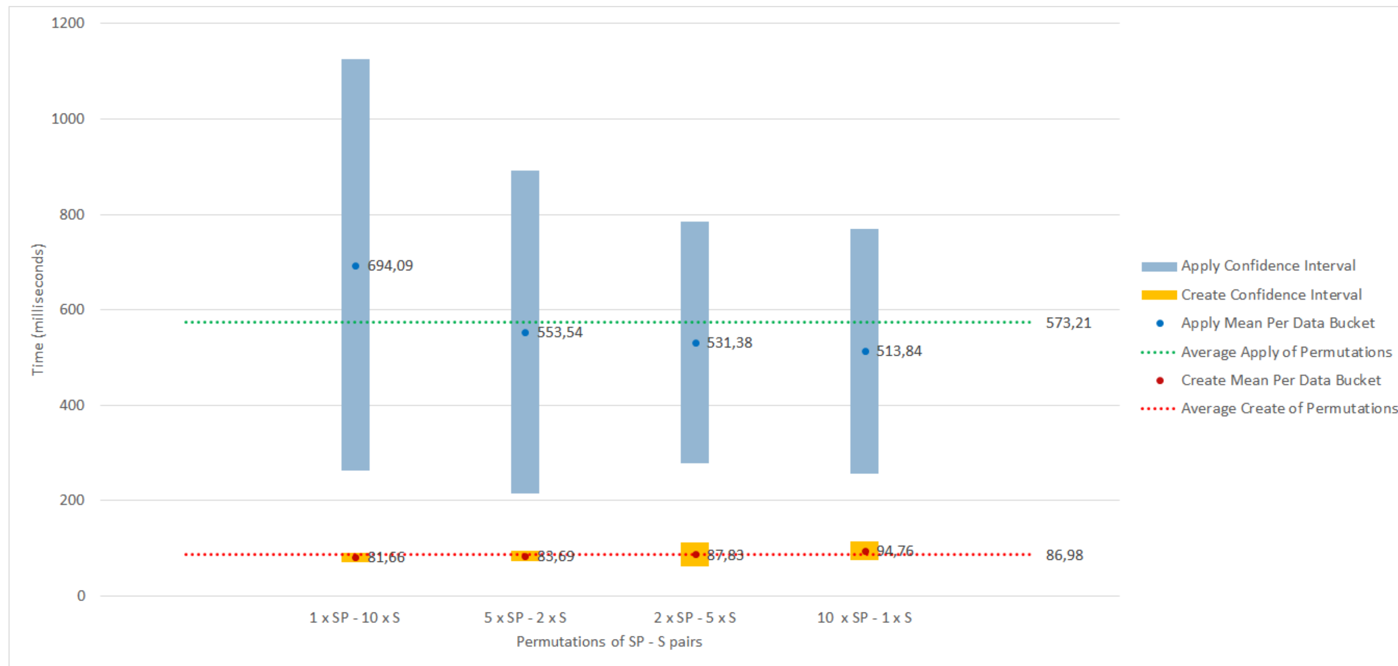

Figure 5.8: Comparison between applying and creation time for various number of SP and S

We ensured that operations by the components of Trust Engine behave as expected, and, thus, predictable operational performance is fulfilled for the them.

6 Discussion and Future Work

As showcased in the previous chapters, trust management as a continuous process in a decentralized marketplace can be built by using the combination service mesh, smart contracts and blockchain technologies. We showed in 4.2, that service mesh technology offers a range of features, like observability and traffic management, which are crucial for all entities partaking in the decentralized marketplace ecosystem. In the upcoming sections, we discuss some of the limitations of the proposed trust engine as well as the necessary changes and consideration when applying the trust engine in a different deployment model. Finally, we discuss potential improvements for the current engine.

6.1 Limitations and Consideration for Proposed Trust Engine

The main point of consideration for the current Trust Engine proposal is the need to trust the entity hosting the service mesh. Parts of the Trust Engine, the Collector and Policy Enforcer, reside in the service mesh. A misbehaviour from the mesh-controlling entity would entail unreliable behaviour of the Trust Engine. A tampered Trust Engine could be configured to always report honest behaviour even when misbehaviour should be detected. This can be achieved by tampering with the Collector module. The challenge is to ensure that the Collector module is not tampered.

Our initial approach was to store a hash of the expected runtime state of the Collector on the blockchain. The Collector module would then generate a hash of its own state on runtime as part of the metric collection. This approach failed to provide the necessary guarantee as the Collector could be tampered to always provide the expected hash. Further investigation for this approach, such as providing a non static secret hash, is a possibility for future work.

6.2 Multi-mesh and Multi-cluster Istio Deployments Architectures

In section 2.3.1, we discussed the possible deployment architectures for service mesh technology. The tenancy models for mutli-mesh, and multi-cluster deployments are typically cluster, or mesh tenancy respectively. This poses new considerations in terms of the proposed Trust Engine model.

The current Trust Engine relies on key components from the service mesh control plane. In a multi-cluster deployment, the control plane is either shared or replicated. In a shared control plane, the current trust engine would require minimal modification to maintain the same level of operation. The Policy Enforcer templates, as well as the necessary queries for metric collection would need to reflect the new tenancy model. A replicated control plane, on the other hand, requires additional modification. A shared root CA in each cluster is needed to enable the Policy Enforcer module to operate on each cluster.

As discussed in the previous section, a limitation of the Trust Engine is the need to trust the entity hosting the service mesh. In a multi-mesh deployment, multiple entities host their respective service mesh. The problem crystallizes when each SP hosts its own service mesh. Moreover it is necessary to have some form of mesh federation in multi-mesh deployments. This entails multiple CAs which further complicates the operation of the Trust Engine, in particular the Policy Enforcer module. Reshaping the current Trust Engine to facilitates multi-mesh deployment is a future work possibility.

6.3 Incentives and Deterrents for Blockchain Peers

Incentives are the foundation of blockchain as software connectors [61]. In order for a group of peers to willingly collaborate, a blockchain typically offers incentives for participants to remain honest. The incentives can be, as discussed in 2.6, in the form of reputation or ratings. This can be incorporated in the current Trust Engine model, to increase the viability of a decentralized service exposure marketplace. In this section we discuss possible incentives for SPs, and SCs which can be integrated in the proposed Trust engine.

Possible incentives for an SP to remain honest are benefits for the operation of their services. These benefits could be in the form of priority for resources, increased quota, or subsidized resource costs. A similar set of incentives could be offered for the honest operation of SC: priority for services, or subsidized service costs.

One way to incorporate these incentives would be to utilize a reputation system for each entity. The reputation can slowly build up for each entity based on their continuous honest operation. This would require only changes only in the Checker module of the Trust Engine. Upon arriving to a verdict, the Checker module can in turn increment, or decrements the reputation of a certain entity. Furthermore, a

deterrent strategy could be to apply weighted changes for the reputation in a way that gaining reputation would be slow, while losing it would be fast. This suggested model, or other models for incentives and deterrents are future work possibilities.

7 Conclusion

In recent years, researchers have proposed various trust management schemes for security in different networks. In this work, we tackled trust management in the context of a decentralized service exposure marketplace.

In order to model this system, we identified the key actors to be: Service Providers (SPs), Service Consumers (SCs), Services (Ss), and Resource Providers (RPs). These actors are enough to model a marketplace system regardless of deployment model. In addition, a marketplace system requires multiple orchestration components which are necessary for the operation of the system. Each pair of actors participating in this system require a set of conditions to be continuously fulfilled in order to continue participating in the service exposure marketplace. In this work we split these conditions into four categories of requirements: Adherence to Service Level Agreement (SLA), Authorized Observability, Communication Security, and Availability and Resilience. We noted that the largest trust burden across all categories lies on the Resource Providers (RPs) toward the SPs. Moreover, we noted that the requirements of Communication Security are the most common between each pair of actors.

When modeling misconduct, this work heavily focused on entities and actors already operating in the system. We defined possible attacks from SPs, SCs and Ss against the desired requirements to enable trust.

Further, this work showed the potential benefits of utilizing smart contract, blockchain and service mesh technology in establishing and maintaining a trusted relationship between all participating entities of the system. We designed and implemented a Proof-Of-Concept (PoC), Trust Engine, to integrate with an existing decentralized virtual services marketplace, Nubo. The Trust Engine leverages the transparency of blockchain, observability and managing capabilities of service mesh and automated guarantee of smart contracts to allow for continuous automated, non-repudiable detection and correction of breaches to the previously identified requirements of trust. The Trust Engine is comprised of three main components, Collector, Checker, and Policy Enforcer modules. The PoC was deployed in a single cluster - single mesh deployment and integrated with a single peer - single orderer blockchain marketplace.

Moreover, the security and operational performance of the suggested model was tested via an abuse scenario towards one of the trust requirement: Tampering of logs and metrics reporting by SPs. The Trust Engine successfully identified, and corrected the misbehaviour in a single check run. Currently, the performance bottleneck for the suggested Trust Engine is the smart contract chaincode. This conforms with our initial assessment as the burden of identifying misconduct lies entirely on the smart contract chaincode. The bottleneck crystallizes with the increase of reported metrics, or trust requirements to check.

Finally, we explored possible improvements for the current trust engine model in terms of providing incentives and deterrents for the different actors in the system. We suggested a reputation based approach which incentives SPs to stay honest by providing them with operation benefits from RPs such as priority resources, increased quota, and subsidized prices for hosting their Services (Ss). Further, we discussed potential alteration to the Trust Engine model in order to operate in other deployment models, notably multi-cluster or multi-mesh. The current Trust Engine, can be easily migrated to a multi cluster with minor alteration in the Collector and Policy Enforcer modules. However, the current Trust Engine can not be easily deployed in a multi mesh deployment due to the need for identity federation which would lead to complication with Policy Enforcers modules. Thus, an alternate approach for trust management in multi mesh can be an opportunity for future research.

References

[1] 1.5, ISTIO. "Replicated control planes". In: (Accessed 03.05.2020). URL: https://istio.io/docs/setup/install/multicluster/gateways/.

[2] 1.5, ISTIO. "Shared control plane (single and multiple networks)". In: (Accessed 03.05.2020). URL: https://istio.io/docs/setup/install/multicluster/shared/.

[3] 1.5, Istio. "Deployment models". In: (2020). Accessed on 27.06. URL: https://istio.io/docs/ops/deployment/deployment-models.

[4] 1.5, Istio. "Egress gateways". In: (2020). Accessed on 27.06. URL: https://istio.io/latest/docs/reference/config/networking/gateway/.

[5] 1.6, ISTIO Prelim. "Deployment models". In: (Accessed 03.05.2020). URL: https://preliminary.istio.io/docs/ops/deployment/deployment-models/.

[6] 1.6, ISTIO Prelim. "Glossary". In: (Accessed 03.05.2020). URL: https://preliminary.istio.io/docs/reference/glossary/.

[7] Accenture. "Building digital trust: The role of data ethics in the digital age". In: (Retrieved 21.03.2020). https://www.accenture.com/_acnmedia/PDF-22/Accenture-Data-Ethics-POV-WEB.pdf.

[8] Ahmed, Abu Shohel and Aura, Tuomas. "Turning trust around: Smart contract-assisted public key infrastructure". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 104–111.

[9] Alexiou, Nikolaos, Laganà, Marcello, Gisdakis, Stylianos, Khodaei, Mohammad, and Papadimitratos, Panagiotis. "Vespa: Vehicular security and privacy-preserving architecture". In: *Proceedings of the 2nd ACM workshop on Hot topics on wireless network security and privacy*. 2013, pp. 19–24.

[10] Alshuqayran, Nuha, Ali, Nour, and Evans, Roger. "A systematic mapping study in microservice architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE. 2016, pp. 44–51.

[11] Amani, Sidney, Bégel, Myriam, Bortin, Maksym, and Staples, Mark. "Towards verifying ethereum smart contract bytecode in Isabelle/HOL". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2018, pp. 66–77.

[12] Ambassador. "Istio integration". In: (Accessed 04.05.2020). URL: https://www.getambassador.io/docs/latest/howtos/istio/.

[13] Badger, Mark Lee, Grance, Timothy, Patt-Corner, Robert, and Voas, Jeffery M. *Cloud computing synopsis and recommendations*. National Institute of Standards & Technology, 2012.

[14] Blaze, Matt, Feigenbaum, Joan, and Lacy, Jack. "Decentralized trust management". In: *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE. 1996, pp. 164–173.

[15] Bohn, Robert B, Lee, Craig A, and Michel, Martial. "The NIST Cloud Federation Reference Architecture". In: (2020).

[16] Bonatti, Piero, De Coi, Juri Luca, Olmedilla, Daniel, and Sauro, Luigi. "A rule-based trust negotiation system". In: *IEEE Transactions on Knowledge and Data Engineering* 22.11 (2010), pp. 1507–1520.

[17] Bonatti, Piero, Duma, Claudiu, Olmedilla, Daniel, and Shahmehri, Nahid. "An integration of reputation-based and policy-based trust management". In: *networks* 2.14 (2007), p. 10.

[18] Burns, Brendan, Grant, Brian, Oppenheimer, David, Brewer, Eric, and Wilkes, John. "Borg, omega, and kubernetes". In: *Queue* 14.1 (2016), pp. 70–93.

[19] Chen, Rui, Li, Shanshan, and Li, Zheng. "From monolith to microservices: A dataflow-driven approach". In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2017, pp. 466–475.

[20] "Cloud infrastructure ready for 5G". In: (2020). URL: https://www.ericsson.com/en/digital-services/offerings/nfvi-cloud-infrastructure.

[21] Cooper, David, Santesson, Stefan, Farrell, Stephen, Boeyen, Sharon, Housley, Russell, Polk, W Timothy, et al. "Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." In: *RFC* 5280 (2008), pp. 1–151.

[22] Di Francesco, Paolo, Malavolta, Ivano, and Lago, Patricia. "Research on architecting microservices: Trends, focus, and potential for industrial adoption". In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, pp. 21–30.

[23] "Edge computing is key for enabling 5G & IoT enterprise services". In: (2020). URL: https://www.ericsson.com/en/digital-services/trending/edge-computing.

[24] Faragardi, Hamid Reza. "Ethical considerations in cloud computing systems". In: *Multidisciplinary Digital Publishing Institute Proceedings*. Vol. 1. 3. 2017, p. 166.

[25] Gannon, Dennis, Barga, Roger, and Sundaresan, Neel. "Cloud-native applications". In: *IEEE Cloud Computing* 4.5 (2017), pp. 16–21.

[26] Gavriloaie, Rita, Nejdl, Wolfgang, Olmedilla, Daniel, Seamons, Kent E, and Winslett, Marianne. "No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web". In: *European Semantic Web Symposium*. Springer. 2004, pp. 342–356.

[27] Gisdakis, Stylianos, Laganà, Marcello, Giannetsos, Thanassis, and Papadimitratos, Panos. "SEROSA: SERvice oriented security architecture for Vehicular Communications". In: *2013 IEEE Vehicular Networking Conference*. IEEE. 2013, pp. 111–118.

[28] Glinz, Martin. "On non-functional requirements". In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE. 2007, pp. 21–26.

[29] Gómez-Arevalillo, Alfonso de La Rocha and Papadimitratos, Panos. "Blockchain-based public key infrastructure for inter-domain secure routing". In: 2017.

[30] Govindaraj, Priya and Jaisankar, N. "A review on various trust models in cloud environment." In: *Journal of Engineering Science & Technology Review* 10.2 (2017).

[31] Grandison, Tyrone and Sloman, Morris. "A survey of trust in internet applications". In: *IEEE Communications Surveys & Tutorials* 3.4 (2000), pp. 2–16.

[32] "ISO/IEC 27001:2013(en) Information technology — Security techniques — Information security management systems — Requirements". In: (2020). URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en.

[33] Kempf, James, Nayak, Sambit, Robert, Remi, Feng, Jim, Deshmukh, Kunal Rajan, Shukla, Anshu, Duque, Aleksandra Obeso, Narendra, Nanjangud, and Sjöberg, Johan. "The Nubo virtual services marketplace". In: *arXiv preprint arXiv:1909.04934* (2019).

[34] Khodaei, Mohammad, Jin, Hongyu, and Papadimitratos, Panagiotis. "SECMACE: Scalable and robust identity and credential management infrastructure in vehicular communication systems". In: *IEEE Transactions on Intelligent Transportation Systems* 19.5 (2018), pp. 1430–1444.

[35] Khodaei, Mohammad, Jin, Hongyu, and Papadimitratos, Panos. "Towards deploying a scalable & robust vehicular identity and credential management infrastructure". In: *2014 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2014, pp. 33–40.

[36] Khodaei, Mohammad, Noroozi, Hamid, and Papadimitratos, Panos. "Scaling pseudonymous authentication for large mobile systems". In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. 2019, pp. 174–184.

[37] Khodaei, Mohammad and Papadimitratos, Panagiotis. "Scalable & Resilient Vehicle-Centric Certificate Revocation List Distribution in Vehicular Communication Systems". In: *IEEE Transactions on Mobile Computing* (2020).

[38] Khodaei, Mohammad and Papadimitratos, Panos. "Efficient, scalable, and resilient vehicle-centric certificate revocation list distribution in VANETs". In: *Proceedings of the 11th ACM conference on security & privacy in wireless and mobile networks*. 2018, pp. 172–183.

[39] Khodaei, Mohammad and Papadimitratos, Panos. "The key to intelligent transportation: Identity and credential management in vehicular communication systems". In: *IEEE Vehicular Technology Magazine* 10.4 (2015), pp. 63–69.

[40] Kong, Dehua and Zhai, Yuqing. "Trust based recommendation system in service-oriented cloud computing". In: *2012 International Conference on Cloud and Service Computing*. IEEE. 2012, pp. 176–179.

[41] Kubernetes. "Namespaces". In: (2020). Accessed on 27.06. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

[42] Lakshman, Avinash and Malik, Prashant. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[43] Lee, Adam J and Yu, Ting. “Towards a dynamic and composable model of trust”. In: *Proceedings of the 14th ACM symposium on Access control models and technologies*. 2009, pp. 217–226.

[44] Li, Ninghui and Mitchell, John C. “RT: A role-based trust-management framework”. In: *Proceedings DARPA Information Survivability Conference and Exposition*. Vol. 1. IEEE. 2003, pp. 201–212.

[45] Li, Wubin, Lemieux, Yves, Gao, Jing, Zhao, Zhuofeng, and Han, Yanbo. “Service Mesh: Challenges, state of the art, and future research opportunities”. In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2019, pp. 122–1225.

[46] Louta, Malamati and Michalas, Angelos. “Towards efficient trust aware e-marketplace frameworks”. In: *Encyclopedia of E-Business Development and Management in the Global Economy*. IGI Global, 2010, pp. 273–283.

[47] Martucci, Leonardo A, Zuccato, Albin, Smeets, Ben, Habib, Sheikh M, Johansson, Thomas, and Shahmehri, Nahid. “Privacy, security and trust in cloud computing: The perspective of the telecommunication industry”. In: *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*. IEEE. 2012, pp. 627–632.

[48] Mehri, Vida, Tutschku, Kurt, et al. “Flexible privacy and high trust in the next generation internet: The use case of a cloud-based marketplace for AI”. In: *SNCNW-Swedish National Computer Networking Workshop, Halmstad*. Halmstad university. 2017.

[49] Microsoft. “Attack matrix for Kubernetes”. In: (Accessed 12.04.2020). URL: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/.

[50] Nauta, JC and Joosten, R. “Self-Sovereign identity: A Ccmparison of IRMA and Sovrin”. In: (2019).

[51] Newman, Sam. *Building microservices: designing fine-grained systems*. " O’Reilly Media, Inc.", 2015.

[52] Nofer, Michael, Gomber, Peter, Hinz, Oliver, and Schiereck, Dirk. “Blockchain”. In: *Business & Information Systems Engineering* 59.3 (2017), pp. 183–187.

[53] Schneider, Fred B. “Least privilege and more [computer security]”. In: *IEEE Security & Privacy* 1.5 (2003), pp. 55–59.

[54] Sheikh, Ozair, Dikaleh, Serjik, Mistry, Dharmesh, Pape, Darren, and Felix, Chris. “Modernize digital applications with microservices management using the istio service mesh”. In: *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. IBM Corp. 2018, pp. 359–360.

[55] Simplified. “20 Types of technology ethics”. In: (Accessed on 21.03.2020). https://simplicable.com/new/technology-ethics.

[56] Susan, D and Holmes, John G. “The dynamics of interpersonal trust: Resolving uncertainty in the face of risk”. In: *Cooperation and prosocial behaviour* 190 (1991).

[57] Szabo, N. *The idea of smart contracts. Nick Szabo’s papers and concise tutorials*. 2018.

[58] Walter W. Powell, Birthe Soppe. “Organizational boundaries”. In: (2020). Accessed on 25.05. URL: https://www.sciencedirect.com/topics/computer-science/organizational-boundary.

[59] Wu, Xiaonian, Zhang, Runlian, Zeng, Bing, and Zhou, Shengyuan. “A trust evaluation model for cloud computing”. In: *Procedia Computer Science* 17 (2013), pp. 1170–1177.

[60] Xie, Feng, Lu, Tianbo, Guo, Xiaobo, Liu, Jingli, Peng, Yong, and Gao, Yang. “Security analysis on cyber-physical system using attack tree”. In: *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE. 2013, pp. 429–432.

[61] Xu, Xiwei, Pautasso, Cesare, Zhu, Liming, Gramoli, Vincent, Ponomarev, Alexander, Tran, An Binh, and Chen, Shiping. “The blockchain as a software connector”. In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE. 2016, pp. 182–191.

[62] Yaga, Dylan, Mell, Peter, Roby, Nik, and Scarfone, Karen. “Blockchain technology overview”. In: *arXiv preprint arXiv:1906.11078* (2019).

[63] Yan, Zheng and Holtmanns, Silke. "Trust modeling and management: from social trust to digital trust". In: *Computer security, privacy and politics: current issues, challenges and solutions.* IGI Global, 2008, pp. 290–323.

[64] Yan, Zheng and MacLaverty, Ronan. "Autonomic trust management in a component based software system". In: *International Conference on Autonomic and Trusted Computing.* Springer. 2006, pp. 279–292.

[65] Zeng, Ke and Cavoukian, Ann. *Modelling cloud computing architecture without compromising privacy: A privacy by design approach.* Information and Privacy Commissioner of Ontario, 2010.

[66] Zheng, Zibin, Xie, Shaoan, Dai, Hongning, Chen, Xiangping, and Wang, Huaimin. "An overview of blockchain technology: Architecture, consensus, and future trends". In: *2017 IEEE international congress on big data (BigData congress).* IEEE. 2017, pp. 557–564.

A API documentation

API endpoints

Blockchain-communicate (http://blockchain:8080)

Communicates with Saranyu, and gets recent verdict that it passes to Policy Enforcer and provides the information from Checker.

<table>
<tr>
<td>POST /verdict</td>
<td>Request Structure:

Body Parameters:
[{
  “service_provider_did”: <string>,
  “services”: [{
    “Service_did”: <string>,
    “service_name": <string>,
    "service_verdict": <string>,
    "timestamp": <int>,
    "nonce": <int>
    }]
}]</td>
</tr>
<tr>
<td colspan="2">Sends verdicts to Policy Enforcer
Service_provider_DID is a unique String that is assigned to entities in marketplace;
Service_did is a unique String;
Service_name is stored in Saranyu;
Service_verdict is one of two types: String of Policy Name// [Denied, Approved] that is assigned to each service in SP. Based on this value, Policy-Enforcer applies policies to entities.
Timestamp is a unique UNIX time that is used to guarantee freshness and non-tampering;
Nonce is a random integer that is used to guarantee freshness and non-tampering.</td>
</tr>
<tr>
<td>POST /metric</td>
<td>Request Structure:

Body Parameters:

{
  "prometheus_query_hash": <string>,
  "prometheus_response_hash": <string>
  "prometheus_metrics": [{
      "service": {
          "name": <string>
          "did": <string>
          "requests_sent": [{
            "destination_name": <string></td>
</tr>
</table>

```
            "destination_id": <string>
            "number_of_requests": int
          }],
          "requests_received":[{
            "sender_name":  <string>
            "sender_id": <string>
            "number_of_requests": int
          }],
        },
      }],
    "timestamp": <long>
    "nonce": <long>
  ...(other_metrics?)
}
```

<u>Sends the metrics from the communication between entities in the marketplace to Saranyu.</u>
Prometheus_query_hash is a hash of Prometheus query is an expression to be evaluated;
Prometheus_response_hash is sent to ensure that any component of decentralized marketplace can check the validity of Checker result with the original Prometheus query;
Prometheus metrics is array of metrics per each service;
Service name is is stored in Saranyu;
Service_did is a unique String;
Requests_sent is an array of services (name, did, number of requests) that obtained packets from this service
Requests_received is an array of services (name, did, number of requests) that sent packets to this service;
Timestamp is a unique UNIX time that is used to guarantee freshness and non-tampering;
Nonce is a random integer that is used to guarantee freshness and non-tampering.

Other metrics will follow in suit, providing a hash for query and response

Policy-enforcer (http://policy-enforcer:9090)

Gets verdict from Blockchain Communication and applies policies on entities.

<table>
<tr><td>POST /verdict</td><td>Output from a Blockchain Communication:

Request Structure:

Body Parameters:
[{
  “service_provider”: <string>,
  “services”: [{
    “Service_did”: <string>,
    “service_name": <string>,
    "service_verdict": <string>,
    "timestamp": <int>,
    "nonce": <int>,
    }]
}]</td></tr>
<tr><td colspan="2"><u>Gets the verdicts from Blockchain Communication.</u>
Service_provider_DID is a unique String that is assigned to entities in marketplace;
Service_did is a unique String;
Service_name is stored in Saranyu;
Service_verdict is one of two types: [Denied, Approved] that is assigned to each service in SP. Based on this value, Policy-Enforcer applies policies to entities.
Timestamp is a unique UNIX time that is used to guarantee freshness and non-tampering;
Nonce is a random integer that is used to guarantee freshness and non-tampering.</td></tr>
</table>

Other endpoints can include:
- Endpoint to update policy templates

Checker-deployment

As for the current implementation, Checker posts the result of requirements check as “failed” or “passed”. Later the Checker will post the data about the situation between entities and ISTIO components, which will be used by Smart contract to run checkers for requirements.

Generic Responses

200	{“status”: “ok”}
400	{“message”: “invalid parameters”}
401	{“message”: “ unauthorized”}
403	{“message”: “not allowed”}

B Configuration Files

B.1 Collector

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: collector
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
          - name: collector
            image: ccbeder/test-private:collector-latest
            imagePullPolicy: Always
            args:
            - python
            - collector.py
          restartPolicy: OnFailure
          imagePullSecrets:
            - name: regcred
```

B.2 Policy Enforcer

```
apiVersion: v1
kind: Service
metadata:
  name: policy-enforcer
  labels:
    app: policy-enforcer
    service: policy-enforcer
spec:
  ports:
  - port: 8080
    name: http
  selector:
    app: policy-enforcer
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: internal-kubectl
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: modify-policy
rules:
  - apiGroups:
    [ "authentication.istio.io", "security.istio.io", "config.istio.io"]
    resources:
```

```
28        - policies
29        - authorizationpolicies
30        - instances
31      verbs:
32        - get
33        - list
34        - delete
35        - update
36        - create
37        - patch
38        - watch
39        - deletecollection
40  ---
41  apiVersion: rbac.authorization.k8s.io/v1
42  kind: ClusterRoleBinding
43  metadata:
44    name: modify-policy-to-sa
45  subjects:
46    - kind: ServiceAccount
47      name: internal-kubectl
48      namespace: "trust-engine"
49  roleRef:
50    kind: ClusterRole
51    name: modify-policy
52    apiGroup: rbac.authorization.k8s.io
53  ---
54  apiVersion: apps/v1
55  kind: Deployment
56  metadata:
57    name: policy-enforcer-v1
58    labels:
59      app: policy-enforcer
60      version: v1
61  spec:
62    replicas: 1
63    selector:
64      matchLabels:
65        app: policy-enforcer
66        version: v1
67    template:
68      metadata:
69        labels:
70          app: policy-enforcer
71          version: v1
72      spec:
73        serviceAccountName: internal-kubectl
74        containers:
75        - name: blockchain
76          image: ccbeder/test-private:policy-latest
77          imagePullPolicy: Always
78          ports:
79          - containerPort: 8080
80        imagePullSecrets:
81          - name: regcred
```

B.2.1 Policy Template

```
apiVersion: config.istio.io/v1alpha2
kind: instance
metadata:
  name: pocmetric
  namespace: '{{SP_NAMESPACE}}'
spec:
  compiledTemplate: metric
  params:
    value: "90" # poc value
    dimensions:
      reporter:>-
        conditional(
        (context.reporter.kind | "inbound")
        ==
        "outbound", "client", "server"
        )
      source: source.workload.name | "unknown"
      destination: destination.workload.name | "unknown"
      message: "agreed in sla"
      namespace: "{{SP_NAMESPACE}}"
    monitored_resource_type: '"UNSPECIFIED"'
```