An Architecture Pattern for Trusted Orchestration in IoT Edge Clouds

Claus Pahl, Nabil El Ioini, Sven Helmer
Faculty of Computer Science
Free University of Bozen-Bolzano
39100 Bolzano, Italy
{claus.pahl, nabil.elioini, sven.helmer}@unibz.it

Brian Lee
Software Research Institue
Athlone Institute of Technology
Athlone, Ireland
blee@ait.ie

***Abstract*—Fog and edge architectures provide a link between centralised clouds and the world of IoT and sensors. The architectures consist of devices of different sizes that coordinate the communication with sensors and cloud services, and that process data from or for the sensors and the cloud locally. In dynamic settings that allow resources to be created, placed, used and removed at any time, advanced orchestration techniques are needed for the activities. Often, these architectures cross organisational boundaries, which causes trust concerns. The origin of data and the identity of sensors or actuators needs to be identified. Additionally, data needs to be stored securely. Orchestration activities across boundaries are subject to a contractual perspective. We present here an architecture pattern that supports trusted orchestration for edge clouds. The W3C Provenance standard is the conceptual foundation of the pattern. Permissioned blockchain technology serves as the platform to implement the provenance framework.**

Keywords: Architecture Pattern, Blockchain, IoT, Edge Cloud, Fog Computing, Orchestration, Trust.

I. Introduction

An IoT edge cloud architecture is a distributed system, typically consisting of an outer rim of IoT, sensor devices and networks, an intermediate layer of local processing capabilities and more centralised cloud systems for data processing and storage [17]. We specifically consider containerised edge orchestration, since this introduces mobile containers [28] as an additional challenge into this architectural setting. Key benefits of such a platform are (i) a lightweight virtualisation platform for flexible configuration and (ii) a dynamic cluster architecture where devices might join or leave, which are suitable for varying data processing needs for sensors and actuators [19]. The orchestration of all activities, i.e., the management of joining and leaving IoT devices, the dynamic placement and change of containers as software devices [3], the creation and processing of data, all need to be managed in a secure and trustworthy way in an inherently insecure, untrusted fog or edge computing environment. An important concern for edge architectures is the secure management of orchestration tasks. IoT networks are distributed environments, in which trust between sensor owners, network and device providers does not necessarily exist. Equally, infrastructure devices might be owned and managed by parties that do not trust each other.

In order to support key orchestration activities from a trust perspective, we need to consider different security concerns:

- record the identity of all entities in the system
- record the provenance of sensor data and other entities entering the processing system
- record the fact that certain processing steps (in the form of smart contracts) have actually been carried out.

We employ blockchain technology to manage the security concerns by recording the above information in a tamper-proof way to create trust. Our aim is to propose a conceptual reference architecture that enables secure, trustworthy processing of IoT edge architectures. Furthermore, provenance is the central concept to organise the storage of orchestration information.

We present an architectural pattern, with its underlying principles, that combines IoT edge orchestration with a blockchain-based provenance mechanism. We formalise the architecture pattern in the format of a state machine that describes the basic processing of the orchestration activities, supported by a blockchain to enable trusted orchestration management (TOM). We discuss several use cases to motivate the needs, picking one of them later in order to validate the state machine definition.

Our paper is organised as follows. We start with basic architecture, trust and blockchain concepts in Section II, before identifying requirements and architectural principles in Section III. Section IV then defines the architecture pattern conceptually, before we formalise it as a state machine in Section V. We validate the design through a use case in Section VI and discuss implementation options in Section VII, before ending with related work and conclusions.

II. Architecture, Trust and Blockchain Basics

An architectural pattern, sometimes called an architectural style, is a set of principles and coarse-grained structures that provides an abstract framework for a family of software systems [8]. Architecture styles are sets of principles and patterns that shape an application. Garlan and Shaw define an architectural style as 'a family of systems in terms of a pattern of structural organisation. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined' [6]. To

978-1-5386-5896-3/18/$31.00 ©2018 IEEE

make principles and patterns more explicit, we follow [7]: 'an architectural style consists of a set of architectural principles and patterns that are aligned with each other to make designs recognisable and design activities repeatable: The principles express architectural design intent; the patterns adhere to the principles and are commonly occurring in practice'.

A key element of our architecture pattern is the incorporation of assured, maybe certifiable security to enable trust [5]. Edge/fog and cloud computing infrastructures allow users to cut costs by distributing and outsourcing computations on-demand. However, users of these services have often no means of verifying the confidentiality and integrity of data and computation as well as the identity of the service providers [1], [2]. Thus, the trustworthiness of the environment is undermined.

Blockchain technology is a solution for security needs in an untrusted environment. Many security problems can be addressed using the decentralised, autonomous, and trusted capabilities of blockchains, which provide inherent security mechanisms capable of operating in an unreliable network, without relying on a central authority [13]. Furthermore, blockchains are tamper-proof, distributed and shared databases in which all participants can append and read transactions but no one has full control over it. Every new transaction is digitally signed and timestamped, which means that all operations can be traced back, and their provenance can be determined [24].

The security model implemented by blockchains insures data integrity using consensus-driven mechanisms to enable the verification of all the transactions in the network [15], which makes all records easily auditable. This is particularly important since it allows tracking all sources of insecure transactions in the network (e.g., vulnerable IoT devices) [25]. Additionally, blockchains can strengthen the security of edge components in terms of the identity management and access control and prevent data manipulation. Key blockchain principles are:

- Blocks (transactions) are appended to the chain.
- Each transaction is hashed/digitally signed to maintain integrity and enforce non-repudiation. Then this is sent to all nodes to obtain consensus using a specific consensus method and respective coordination protocol. Consensus is essentially driven by collective self-interest.
- Finally, the result is passed on to the recipient. Public keys act as addresses in this context.

A key concept here is a smart contract: if there is agreement (condition met) then contracts (operations) are executed. These contracts are attached to the blockchain. The problem is that due to massive data replication, performance and scalability need to be evaluated. This is of course a challenge in the constrained environment in question here.

III. Trusted Orchestration Management (TOM) Requirements and Principles

The orchestration of data, software and hardware components in edge clouds is the key architectural concern in behavioural terms that will be addressed through the pattern [18]. This will be combined with a security layer to enable trust. We start gathering functional and platform requirements in order identify the core principles that guide the pattern.

A. Core Requirements and Platform Assumptions

We now look at security requirements for IoT edge architectures in more detail. There is a lack of trust in these architectures that requires security features to be applied.

- Things (be that sensors, devices or software) might dynamically join and need to be identified.
- Data is generated and communicated, but needs to be traceable to its origin (provenance).
- Decisions in the architecture management and orchestration need to be taken dynamically and locally, and need to be agreed by all participants involved.

What is needed is a trustworthy mechanism for the secure management of edge architectures that focuses on the transactional aspects, but including here identity, provenance and orchestration as aspects of a facetted smart contracts approach.

To explore this setting deeper, we make some assumptions regarding the specific devices and software platform, although these do not impact on the generality of the approach. We assume a cluster-based architecture for the outer and middle layers, allowing for some hierarchical organisation of the edge architecture to take place. We also introduce lightweight smaller devices such as single-board computers, while we still assume that more capable gateway servers and cloud infrastructures are also integrated into the scenario. Basic cluster orchestration for containers could be provided by platforms like Kubernetes or Docker Swarm [9], [10], [12].

At the core are classical orchestration activities such as deploying activities on nodes of the network. Since all entities (devices, containers, data producers/consumers) can join and leave the system at any time, two additional activities need to be supported: identification and recording of activities, resulting in the following activities to be supported: identify, orchestrate and record.

B. Architectural Principles

In this IoT edge cloud context, a number of key principles arise from the requirements that are security-related:

- Identification: Identify all things as authentication and data origin are issues.
- Data Provenance: important if, for instance, incidents happen and records need to be examined.
- Non-Repudiation of architecture management operations in a contract situation, e.g., change of software in maintenance or emergency situations.

The key aim is to manage trust without central authorities in lightweight edge clusters and possibly limited connectivity. We need technologies that can be applied to identity management, data provenance and transaction processing. What is required is advanced distributed trust management including concepts such as smart contracts of identity and orchestration management in an environment that is constrained in terms of computational power and connectivity.

C. Application Areas and Secure Orchestration Needs

To demonstrate the need for a trusted orchestration management (TOM), we look at different application scenarios. We summarise in a schematic way the key entities and orchestration needs.

Lightweight sensor architectures are about data collection and processing with containers and data input/output. First, we briefly look at an agricultural domain:

- Devices: RPi clusters in remote, exposed areas (assumed to be in fixed locations)
- Sensors: rain, temperature, sun, humidity
- Actuators: irrigation system
- Containers: regular sensor data collection, data filtering and analysis, local storage, maintenance and testing
- Data: sensor data (raw, filtered), analysis results, actuator instructions
- Orchestration: data provenance, container identification, container orchestration on devices (contract execution).

Logistics is another use case with vehicle-based mobile sensors that could detect loading/unloading in different locations:

- Devices: mobile (moving vehicles that might join or leave locations), single-board computers
- Sensors: e.g., RFID to detect item movement
- Actuators: none
- Containers: item movement (origin and target), system testing and maintenance
- Data: item movement, from and to locations
- Orchestration: data provenance, device identification, container identification, orchestration of containers on devices (smart contract execution).

Our proposed pattern addresses security needs as described in the above use cases.

D. Suitability of Blockchains to Support Security Requirements and Principles

We now discuss the suitability of blockchain technologies for providing a secure platform that addresses the above requirements and principles for IoT edge architecture orchestration. Blockchain management forms a distributed software architecture, where agreement on shared state for decentralised and transactional data can be established across a network of untrusted participants – as it is the case in edge clouds. This approach avoids to rely on central trusted integration points that control the system in question – which is a single point of failure. Edge platforms that build trust on blockchains can take advantage of common blockchain properties such as data immutability, integrity, fair access, transparency, and non-repudiation of transactions.

By building on container-based orchestration, software becomes another artefact that is subject to identity and authorisation concerns, since edge computing is essentially based on the idea to bring software to the edge (to process data locally) rather than to bring data to the cloud centre. Often, large amounts of data are collected at the edge, transferred to and stored in the cloud, causing high resource consumption there without a clear need for all the data. Edge cloud computing involves resource-constrained devices that link sensors with central cloud processing. These devices can carry out critical filtering and processing activities, relieving the central components of the architecture from overload.

IV. A Blockchain-based Architectural Pattern for Trusted Orchestration Management (TOM)

The two extended trusted orchestration management (TOM) activities in addition to deployment and execution of the components that we need to take care of are identity management and logging (recording) of all orchestration-related actions. Identity management is supported for instance in cluster management tools like Docker Swarm. Here authentication is used in the cluster setup to define the capabilities of processing nodes. Logging orchestration actions and generated data supports auditability, which is of importance in untrusted environments. As explained, we propose a blockchain to record all orchestration actions.

A. Blockchains for Edge Architecture Orchestration

Blockchain technology has been applied for transactional processing, but the novelty here is the application to lightweight dynamic edge architectures, i.e., blockchain methods and protocols in localised clusters of edge devices. Smart contracts define orchestration decisions in the architecture (as transactions). We use blockchains to manage security (identity, origin, non-repudiation) in distributed autonomous clusters.

Specifically, the following blockchain architectural configurations [14] are proposed:

- permissioned blockchains with brokers to enable a degree of local control by the orchestrator.
- partially centralised/decentralised settings with permissioned blockchains with permissions for fine-grained operations on the transaction level (e.g., permission to create assets such as data).
- consider both permissioned blockchains with permissioned miners (write) and also permission-less normal nodes (read).

With respect to concerns such as cost efficiency, performance, and flexibility, a key problem for an orchestrator is choosing what data and computation should be placed on-chain and what should be kept off-chain. This might, however, be decided depending on the concrete implementation of the pattern.

Smart contracts are another aspect of blockchain configuration. Smart contracts are not processed until their invoking transactions are included in a new block. Blocks impose an order on transactions, thus resolving non-determinism which might otherwise affect their execution results. This can be implemented for device and container orchestration [16], [22] within the contract transaction of the blockchain.

B. Provenance to Enable Trust

We singled out data origin and provenance, identity management and orchestration non-repudiation as important security concerns in our framework. We describe now how

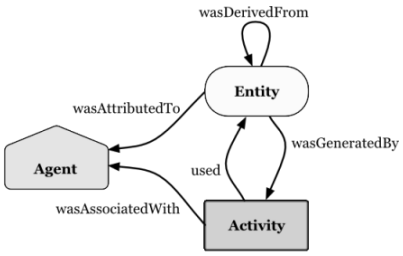


Fig. 1. W3C PROV Provenance Model [based on https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/].

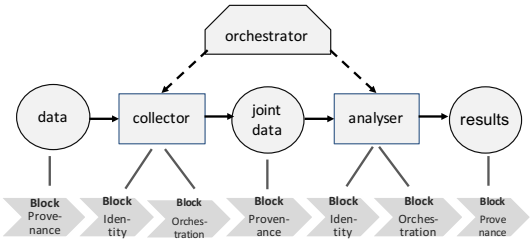


Fig. 2. Structural Architecture: Sample Configuration

a blockchain can be utilised to support the requirements. Provenance actually plays the key role here, as it is the mechanism to capture all identity, orchestration and data origin actions in a systematic uniform way. Our starting point is the W3C PROV standard (https://www.w3.org/TR/prov-overview/). According to PROV, provenance is information about entities, activities, and people involved in producing in our case data. This provenance data in general aids the assessment of quality, reliability or trustworthiness in the data production and processing. The goal of PROV is to enable the representation and interchange of provenance information.

Fig. V-B shows the PROV conceptual model with entities, activities and agents. Here, entities are data objects as well as software containers and hardware devices. Provenance records describe the provenance of entities. An entity's provenance can refer to other entities, e.g., compiled sensor data to the original records. Activities, which are the dynamic parts (i.e. processing components), create and change entities, often making use of previously existing entities to achieve this. The two fundamental activities are generation and usage of entities, which are represented by relationships in the model. There are also two types of activities – usage and generation. Those carried out by devices (sensors): sensors can generate data, actuators can consume/use them. Those carried out by containers (deployed on devices), which can also generate and use data. Activities are carried out on behalf of agents that also act as owners of entities, i.e. are responsible for the processing. An agent takes a role in an activity such that the agent can be assigned some degree of responsibility for the activity taking place. Actors are for instance orchestrators in charge of deploying software and managing the infrastructure.

All entities (device, container, data) are dynamic, i.e., can join (be created or deployed) or leave (be destroyed or terminated) the system. Three operations arise: (i) on joining, or before accepting data as a processing component, they need to be identified; (ii) data origins need to be remembered (stored); (iii) placement of entities, which is an orchestration decision.

C. An Architecture Pattern for TOM

We propose to implement TOM through provenance-based orchestration logging.[1] Following Section II, we define the architecture pattern for trusted orchestration management in terms of: (i) a structural architecture consisting of entities: orchestrator (agent), containers (running on devices, implementing services on data), data (being processed by container-based services), (ii) a set of core activities, (iii) a behavioural architecture (interaction of key entities). Furthermore, we provide a mapping of PROV to the architecture pattern in this section, which we formalise as a state machine in Section V.

We first outline some core properties of the architectural pattern. The orchestrator decides where the data and activities are located. Data processing means to identify a source of data, create a PROV record and orchestrate (i.e., assign a target service for the data processing by orchestrator), which is based on the execution of a smart contract. The smart contract is based on the provenance record (which is in our case more declarative in nature, but the orchestrator can interpret this). Table 1 (Section V) will formalise this. We note that in case of orchestrator failure, other nodes can be promoted to a manager role, or a if an orchestration mechanism such as docker swarm is used, multiple nodes can be defined with a manager role, then a consensus mechanism (i.e., raft) can be used to make sure that all the managing nodes have an exact copy of the orchestration configuration. This removes the problem of single point of failure.

In the sample architecture scenario in Fig. 2, the orchestrator is the agent that orchestrates, i.e., deploys in this case the collector and analyser containers on some available computing devices. This effectively forms a contract between orchestrator and nodes, whereby the nodes are contracted to carry out the activities, i.e., in this case the collection of data into a joint representation and the consecutive analysis, providing some results. We can define a set of principle activities for the collection and analyser functions from Fig. 2 based on the PROV relationships USES and GENERATES: the collector USES sensor data and GENERATES the joint data; the analyser USES the joint data and GENERATES the results. This sequence of activities forms an orchestration plan. This plan is enacted based on the blockchain smart contract concept, requiring the contracted activity to (i) obtain permissions (credentials) to retrieve the data (USES) and (ii) create output entities (GENERATE) as an obligation defined in the contract. A smart contract is defined through a program that defines the implementation of the work to be done. It includes the obligations to be carried out, the benefits (in terms of SLAs) and the penalties for not achieving the obligations. Generally, fees

[1]We can expand this idea by considering the provenance of an agent, and not only activities and entities. In our case, the orchestrator is also a container, though one with a management rather than an application role. In order to make provenance assertions about an agent in PROV, the agent must then be declared explicitly both as an agent and as an entity.

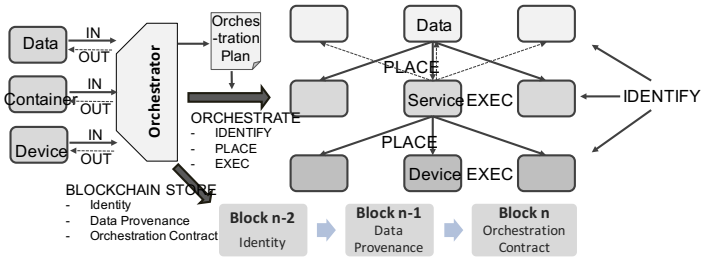


Fig. 3. Structural Architecture: Blockchain-based Orchestration – with an Orchestration Layer (based on PROV) and a Trust Layer (based on blockchains, reflecting the PROV activities)

paid to the contractor and possible penalties to compensate the contract issuer shall be neglected here. Each processing step based on the contract is recorded in the blockchain: (i) the generation of data through a provenance entry: what, by whom, when; (ii) the creation of a credentials object defining, based on the identity of the processing component, the authorised activities; (iii)the formalised contract between the orchestrator and the activity node. The obligations formalised include in the IoT edge context data-oriented activities such as storage, filtering and analysis and container-oriented activities such as deploying or redeploying (updating) a container.

The structural architecture is presented in Fig. 3. Some key assumptions are as follows: sensors are also considered as devices, as are computational devices such as gateway servers or smaller computers. Sensors in this way are also considered as being service providers, e.g. providing monitored data as a stream or in discrete packages. Actuators are in a similar way services that process incoming instructions and configuration data. We assume the computational devices to host containers that are dynamically updateable. Sensors and actuators might not have that dynamic capability.

Next, we provide a PROV-to-Pattern Mapping, focusing on the Orchestrator (PROV-AGENT) and the Orchestration Plan (formalised by a BCOR state machine in the next section). The orchestration plan is based on the following ORCHESTRATION operations (PROV-ACTIVITY) on the entities (PROV-ENTITY) in Fig. 3:

- IDENTIFY (Device/Host, Container/Service, Data)
- PLACE (Container on Host; Data on Service), which is a send/deployment action
- EXEC (Host.Container $\rightarrow$ Service; Service.Data $\rightarrow$ Results), which is one of two processing actions, CONSUME and PROVIDE, that are applicable to hosts (consumes container or provides service) or services (consumes (in)data and provides (out)data/results). Sample operations are Collect and Analyse. These correspond to PROV.USES and PROV.GENERATES.

Additionally, the following BLOCKCHAIN STORAGE operations from Fig. 3, generally for accountability, are needed for the following entities to be stored:

- Identity: for authentication
- Data Provenance: for data origin and integrity
- Orchestration: for contract non-repudiation

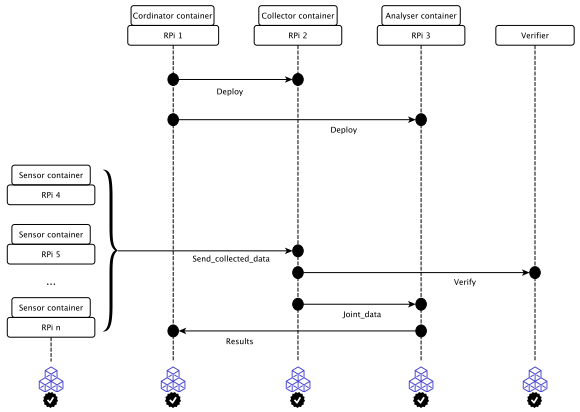


Fig. 4. Behavioural Architecture of the Blockchain Orchestration Integration.

Table I summarises the dependencies between the PROV elements ENTITY, ACTIVITY and AGENT.

TABLE I
ENTITY TO ACTIVITY, MANAGED BY ORCHESTRATOR (AGENT).

ENTITY ACTIVITY	IDENTIFY (who)	PLACE [REMOVE] (where)	EXECUTE (what)
Data	x	x	
		Consumed by $\downarrow$	Produces (service) $\uparrow$
Container	x	x	x
		Consumed by $\downarrow$	Produces (data) $\uparrow$
Device	x		x

Fig. 4 shows the behavioural interaction architecture of the solution focusing on the interactions between the components. All actions are recorded in the blockchain to guarantee data provenance. Additionally, the identity of all entities is stored. The actions are executed by invoking the appropriate smart contract. For instance, when a sensor container collects data, it invokes the *send_collected_data* smart contract defined by the collector container by passing a signed hash of the collected data. At this point the collected container checks the identity of the sensor container (e.g., signature) and the integrity of the data (e.g., the hash of the data), then downloads the data in order to process it.

V. Formalisation – A State Machine for Blockchain-based Orchestration

In the previous sections we introduced the principles of our architecture and the basic characteristics of the architectural pattern. Now, we formalise the concept using a state machine definition: we start with the data structures underlying the operation and the introduction of key features of the operations. We introduce the syntax of the state machine specification and specify the state transitions in a table format. We finally define the full state machine based on the transition table.

A. Data Structure and Operations

The blockchain data structure is a timestamped sequence of blocks, which records and aggregates data about transactions that have occurred within the blockchain network. A block consists of: (i) a hash for maintaining integrity, (ii) a pointer

to the previous block and (iii) the stored content. The data structure for the stored content on the blockchain contains one of following three structures:

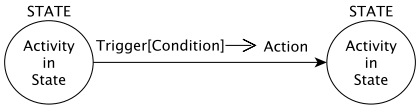
- PROV(data:entity, origin:activitity-generates, receiver: activity-uses)
- IDENTITY(object:identity, id-proof:proof)
- ORCHESTRATE(deployed:entity,location:entity)

Operations on blockchains include in principle Read (extract content or execute smart contract), Validate and Insert. The second is concerned with the chain management and validation process and shall be ignored here. Our focus is reading/executing and creating/inserting blocks. We can detail them according to their task in the IoT edge orchestration context. The full list of actions includes (in three categories):

- Orchestration Actions: Device-identify, Device-terminate, Container-identify, Container-terminate, Activity-invoke SmartContract-execute
- Blockchain Actions: store prov(id-create), store prov(id-delete), store prov(data), store prov(orchestrate)
- Auxiliary: Ack - Acknowledgements of received input

B. State Chart Definition

We assume a basic state machine here that formalises the sequence of Fig. 4. States reflect the results of IoT processing and are stored on the blockchain. Outcomes of the transitions below that result in blocks added to the blockchain, i.e. the state is reflected in the blockchain through a new block. All transitions are orchestrator-managed: triggered through previous states, constrained by conditions and having an orchestration action executed. The state machine syntax is based on transitions in the form



The BC-based ORchestration (BCOR) State Machine is defined as a 5-tuple: BCOR-SM = [S, T, C, A, δ] where:

- S = (BCS1, BCS2, . . . BCSn) is the set of (blockchain-oriented) states that indicate the record created in the blockchain, i.e., (PROV-ID-create, PROV-ID-delete, PROV-DATA, PROV-ORCH);
- δ : S×T×C×A $\rightarrow$ S is the high-level state transition function. It specifies the next state depending on the present state and the conditions; see Table 1;
- T = (T1, T2, . . . Tk) is the set of triggers, i.e., events that initiate the state transition for entities joining or leaving (container-join/leave, device-join/leave);
- C = (C1, C2, . . . Cl) are control conditions that constrain a transition depending on a previous state, e.g., devices or containers need to be joined before they can leave or data needs to have left its generating entity, i.e., the conditions are (joined(device), joined(container), left(data));
- A = (A1, A2, . . . Am) is the set of actions carried out by the orchestrator, which here are (identify-device, identify-container, terminate-device, terminate-container, determine-activity, orchestrate(smart contract)).

VI. Validation Example

In order to validate the architectural pattern, we discuss a use case – smart cities – to illustrate the need for identity management (for devices, software, data), contracts, and data provenance. As an example, we take a closer look at cars. These are mobile devices, which can determine their position (e.g., consume information from beacons). They can also provide location and direction of travel to traffic management, which results in some requirements. For identification of the device, they need to register to a local network (similar to mobile phone hand-over). Traffic management is a component that might use an analyser function (which might be a third-party service). This needs the identification of the software service. The platform could use a flexible software infrastructure with frequent updates of the containers that run the analysis and other features. Another separate component is the road infrastructure management, which requires data identification.

Data structures – as defined above – provide structure for the parameters of the orchestration and blockchain actions.

We can make good use of contracts in this example. For instance, the traffic management could invoke a contract on the traffic infrastructure, i.e., could instruct the traffic lights to be changed (actuator setting). A dynamic contract management scenario is, e.g. a car connecting to traffic management: in this case a passive contract is created:

```
IF USES(GENERATES [car, location, direction])
    % Requires a certain state sequence to have
    % been passed through
THEN car.ACK; bc.PROV(ID-CREATE(car)),
     bc.PROV(DATA(car:origin, (location, direction):data,
                  trafficmgmt: receiver))
```

In a second scenario, the traffic management connects to the traffic infrastructure: in this case an executable contract is created:

```
IF USES(GENERATES [SET(light,colour)])
   % Requires a certain state sequence to have
   % been passed through
THEN TRFMgmt.ACK, bc.PROV(ORCH(Set(light,colour)))
```

The benefits include a tamper-proof record that can be examined in the case of accidents.

VII. Implementation - Orchestration and Blockchain Engines

In this section, we discuss possible implementation platforms for the architecture pattern. In Fig. 3, we identified two layers: orchestration and blockchain. We illustrate them in the next two subsections.

A. Orchestration Engines

Docker Swarm and Kubernetes are possible container cluster engines. Container technologies are lightweight virtualisation mechanisms. Docker and Kubernetes as the most prominent examples have been successfully placed on edge devices such as Raspberry Pis [20], [26] to form local computation clusters. We note that our architecture is orchestration engine agnostic, meaning that any engine that implements an orchestration mechanism can be used. As our focus is here on

TABLE II
STATE MACHINE DEFINITION.

Transition ON Trigger	Transition IF .. Condition	Transition DO Orchestrator Action	State IN STATE DO .. Blockchain Action
Join device	-	identify device	store PROV-ID-create record
Leave device	IF joined(device)	terminate device	store PROV-ID-delete record
Join container	-	identify container	store PROV-ID-create record
Leave container	IF joined(container)	terminate container	store PROV-ID-delete record
Leave data (from sensor/activity)	-	determine processing activity	store PROV-DATA record
Join data (at activity)	IF left(data)	instruct activity (EXEC CONTRACT)	store PROV-ORCH record

edge cloud architectures, we investigate the core components for a container middleware platform for edge clouds. Different containers for a specific processing problem are composed, which forms an orchestration plan. This plan defines the edge cloud topology on which the orchestration is enacted.

1) Swarm Cluster Architecture and Security: To illustrate the orchestration need better, we select Docker Swarm that we also used to carry out experiments [12], [26]. The cluster nodes can have different roles. A selected node becomes the user gateway into the cluster. Docker Machines can manage remote hosts by sending the commands from the Docker client to the Docker daemon on the remote machine over a secured connection. When the first Docker Machine is created, in order to create a trusted network, new TLS protocol certificates are created on the local machine and then copied to the remote machines. For the management of the cluster we use Docker Swarm. Normal nodes run one container that identifies them as a Swarm node. The Swarm managers deploy an additional dedicated container that provides the management interface. There are mechanisms to avoid inconsistencies in the Swarm that could lead to potential misbehaviour. If several Swarm managers exist, they can also share their knowledge about the swarm by communicating information from a non-leading manager to the one in charge. While some identity and reliability management is in place, provenance of these activities is important to enable trust. These activities can be captured in the form of smart contracts, e.g., ORCH(create(DockerMachine)), ORCH(deploy(Container)) or ORCH(retrieve(StatusInformation)) in our solution.

2) Orchestration Information: Information about a Swarm needs to be shared. In a multi-host network, we can use a key-value store that keeps information about network state (e.g., discovery, networks, endpoints and IP addresses). Information about an image and how it can be reached needs to be provided. The orchestrator uses this key-value store for selecting a lead node in a cluster and managing information distribution across the nodes. This is similar to data generated by sensors subject to provenance activities. The data can be used in the smart contracts that implement the orchestration: NetworkState(Endpoints,IPAddresses) or ManagerInfo(Status).

B. Blockchain Engines

While a container engine can form the core of the orchestrator, we need a blockchain engine to run the provenance layer. A permissioned blockchain platform supporting smart contracts is Hyperledger Fabric. Hyperledger fabric is an open-source, distributed ledger computing platform featuring smart contract (Chaincode) functionality. It provides a decentralised environment replicated across different network participants. Hyperledger fabric employs smart contracts to manage controlled access to the ledger as well as a modular and extendable architecture that allows plug-in/out of different components to perform specific functions. As a permissioned blockchain, it supports different types of transactions and variable block sizes, while maintaining a high level of control over transaction execution and validation time.

Within fabric, nodes assume different roles (i.e., clients, peers and orderers), where clients act on behalf of the client applications, the peers maintain copies of the ledger and orderers provide the communication channels and generate the new blocks (mining). This approach reduces latency and increases throughput. Since in a permissioned blockchain all the parties are known (identity vs anonymity), to increase privacy fabric uses channels, which are private subnets of communication between two or more participants in the network.

VIII. RELATED WORK

Our pattern resembles a reference architecture for an edge cloud. For instance, Kratzke has investigated lightweight architectures for cloud environments, focusing on the clustering mechanism [11], without considering security and trust in-depth. In [27], Zhu et al. have used the Hyperledger Fabric blockchain as a platform to implement a distributed control system based on the IEC 61499 standard. The standard is designed to support distributed automation control systems through the implementation of function blocks containing the logic of each participating component of the system. The function blocks are implemented as Docker containers and Kubernetes is used for orchestrating the execution of the containers. The blockchain layer is used to secure and validate all transactions as a result of executing the different containers.

If we want to consider a management platform for an IoT edge architecture, be it in constrained or mobile environments, the functional scope of a middleware layer needs to be suitably adapted [4], [21]. Key requirements are robustness and security – both significant contributors to trust. In [29], Stanciu presents a blockchain-based architecture named EdgeChain, with the corresponding algorithm to manage a decentralised placement service for mobile edge applications (MEC). While in traditional cloud computing environments the resource management is the responsibility of the cloud provider, in MEC, a degree of collaboration between service providers is needed for an effective use of resources. The main aspects addressed by EdgeChain are *i)* providing a transparent selection process for hosts to process the incoming requests, *ii)* a trustworthy and democratic environment to determine the best place to deploy

applications, and *iii)* high service availability independent from vendors or service providers.

Robustness is a requirement that needs to be facilitated through fault tolerance mechanisms that deal with failure of connections and nodes. Flexible orchestration and load balancing are such functions. Security is another requirement, here relevant in the form of identity management in unsecured environments. Other security concerns such as data provenance or smart contracts accompanying orchestration instructions are also relevant. De Coninck et al. [23] also approach this problem from a middleware perspective. Dupont et al. [21] look at container migration to enhance the flexibility, which is an important concern in IoT settings. In [30], Hardjono and Smith present a privacy preserving technique that relies on blockchain to anonymously record device commissioning. The technique assures the different parties about the identity of the devices and their manufactures while incentivising device owners to share sensor data in a privacy-preserving fashion.

IX. Conclusion

Trust is an essential ingredient to make the Internet of Things work. Edge and fog computing can provide an operational platform to manage the behavioural aspects. However, in an open environment that is subject to security problems and that crosses organisational boundaries, trust needs to be addressed. The problems are the identity of hardware devices and software applications, the origin and integrity of data and the contractual nature of orchestration in this context.

We have presented an architecture pattern for trusted orchestration management (TOM) in an edge cloud, guided by the security principles identity, data provenance and non-repudiation. An architecture pattern was presented in terms of its structural and behavioural properties. A state machine definition formalises the conceptual architecture presentation.

We have demonstrated that a blockchain-based solution can ideally map trust concerns to an architecture level, which are conceptually captured by the W3C Provenance framework.

We have presented an architecture pattern, i.e., a solution that is abstracted from more specific performance and other implementation concerns. For instance, the complete storage of extensive sensor data in a blockchain is not feasible due to the performance problems of most blockchain solutions. Here, off-chain storage and other specific blockchain configurations, which we only briefly touched, need to be further investigated.

Acknowledgment

The authors would like to thank the Free University of Bozen-Bolzano for supporting the ECO and ECORE projects.

References

[1] Cloud Security Alliance. 2018. https://cloudsecurityalliance.org/group/cloudtrust.

[2] N. Santos, K.P. Gummadi, R. Rodrigues, “Towards trusted cloud computing”, HotCloud Conference on Hot topics in cloud computing, 2009.

[3] C. Pahl, P. Jamshidi, D. Weyns, “Cloud architecture continuity: Change models and change rules for sustainable cloud software architectures”, Journal of Software: Evolution and Process 29 (2), 2017.

[4] A. Chandra, J. Weissman, and B. Heintz, “Decentralized Edge Clouds”, IEEE Internet Computing, 2013.

[5] C.A. Ardagna, R. Asal, E. Damiani, T. Dimitrakos, N. El Ioini, C. Pahl, “Certification-Based Cloud Adaptation”, IEEE Transactions on Services Computing, 2018.

[6] D. Garlan and M. Shaw, “An introduction to software architecture”. Vol. 1. World Scientific, 1994.

[7] C. Pahl, P. Jamshidi, O. Zimmermann, “Architectural principles for cloud software”, ACM Transactions on Internet Technology (TOIT), 2018.

[8] Microsoft, “Software Architecture and Design”, 2009.

[9] C. Pahl, “Containerisation and the PaaS Cloud”. IEEE Cloud Comp. 2015.

[10] C. Pahl and B. Lee, “Containers and clusters for edge cloud architectures - a technology review”. Int Conf on Future Internet of Things and Cloud (FiCloud). 2015.

[11] N. Kratzke, “A Lightweight Virtualization Cluster Reference Architecture Derived from Open Source PaaS Platforms”, Open Journal of Mobile Computing and Cloud Computing vol. 1, no. 2, 2014.

[12] D. von Leon, L. Miori, J. Sanin, N. El Ioini, S. Helmer, C. Pahl, ”A Performance Exploration of Architectural Options for a Middleware for Decentralised Lightweight Edge Cloud Architectures”, International Conference on Internet of Things, Big Data and Security, 2018.

[13] C. Pahl, N. El Ioini, S. Helmer, ”A Decision Framework for Blockchain Platforms for IoT and Edge Computing”, International Conference on Internet of Things, Big Data and Security, 2018.

[14] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, S. Chen, “The blockchain as a software connector”. In Working IEEE/IFIP Conference on Software Architecture (WICSA), 2016

[15] J.A. Garay et al. “The Bitcoin Backbone Protocol: Analysis and Applications”. https://eprint.iacr.org/2014/765.pdf 2015.

[16] V. Andrikopoulos, S. Gomez Saez, F. Leymann, and J. Wettinger, “Optimal distribution of applications in the cloud”. In Adv Inf Syst Eng, pp. 75-90. 2014.

[17] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things”. Workshop Mobile Cloud Comp, 2012.

[18] A. Gember, A Krishnamurthy, S. St John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, “Stratos: A network-aware orchestration layer for middleboxes in the cloud”. Duke University, 2013.

[19] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, “Clouds of virtual machines in edge networks”. IEEE Comm. 2013.

[20] P. Tso, D. White, S. Jouet, J. Singer, and D. Pezaros, “The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures”. Int. Works Resource Management of Cloud Comp. 2013.

[21] C. Dupont, R. Giaffreda, and L. Capra, “Edge computing in IoT context: Horizontal and vertical Linux container migration”. In Global Internet of Things Summit (GIoTS), 2017.

[22] P. Bellavista and A. Zanni. “Feasibility of fog computing deployment based on docker containerization over raspberrypi”. International Conference on Distributed Computing and Networking. ACM, 2017.

[23] E. De Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, B. Dhoedt and P. Simoens, “Middleware Platform for Distributed Applications Incorporating Robots, Sensors and the Cloud”. IEEE Intl Conference on Cloud Networking, 2016.

[24] A. Dorri, S. S. Kanhere, and R. Jurdak “Towards an Optimized BlockChain for IoT”. Intl Conf on IoT Design and Implementation. 2017.

[25] N. Kshetri, “Can Blockchain Strengthen the Internet of Things?” IT Professional 19.4, 68-72, 2017.

[26] C. Pahl, S. Helmer, L. Miori, J. Sanin and B. Lee, “A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters”. IEEE Intl Conference on Future Internet of Things and Cloud Workshops (FiCloudW), pp. 117-124, 2016.

[27] H. Zhu, C. Huang, J. Zhou, “EdgeChain: Blockchain-based Multi-vendor Mobile Edge Application Placement”. arXiv preprint 1801.04035, 2018.

[28] C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, “Cloud container technologies: a state-of-the-art review”. IEEE Transactions on Cloud Computing, 2017.

[29] A. Stanciu, “Blockchain Based Distributed Control System for Edge Computing”. IEEE International Conference on Control Systems and Computer Science (CSCS), pp. 667-671, 2017.

[30] T. Hardjono and N. Smith, “Cloud-Based Commissioning of Constrained Devices using Permissioned Blockchains”. ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS ’16). ACM, pp. 29-36, 2016.