A Survey on Microservice Security–Trends in Architecture, Privacy and Standardization on Cloud Computing Environments

Luciano de Aguiar Monteiro[1], Washington Henrique Carvalho Almeida[1], Raphael Rodrigues Hazin[1], Anderson Cavalcanti de Lima[1], Sahra Karolina Gomes e Silva[2] and Felipe Silva Ferraz[1]
[1]Center of Advanced Studies and Systems of Recife
Recife, Brazil
[2]Nassau Mauritius College
Teresina, Brazil
E-mail: {lucianoaguiarthe, washington.hc.almeida, raphaelhazin, andclima, sahrask}@gmail.com
E-mail: fsf@cesar.org.br

Abstract — **Microservices have been adopted as a natural solution for the replacement of monolithic systems. Some technologies and standards have been adopted for the development of microservices in the cloud environment. Application Programming Interface and Representational State Transfer were used on a large scale for the implementation. The purpose of the present work is to carry out a bibliographic survey on the microservice security trends focusing mainly on architecture, privacy and standardization aspects in Cloud Computing environments. This paper presents a bundle of elements that must be considered for the construction of solutions based on microservices.**

Keywords- Microservice; Security; Cloud; Architecture; API; Monolithic

I. INTRODUCTION

Migration of the monolithic architecture to the cloud has been a major problem. In this paper a research was carried out on the topic of microservices that have been adopted as a natural solution in the replacement of monolithic systems. The main question lies in how its architecture has been used and issues of security and privacy keys on a Cloud Computing environment. Cloud Computing provides a centralized pool of configurable computing resources and computing outsourcing mechanisms that enable different computing services to different people in a way similar to utility-based systems such as electricity, water, and sewage.

The motivation for this collection was the fact that more and more microservices have been found as a solution for cloud applications. This paper analysis in further details aspects related to Survey on Microservice Architecture [1]. Due to its architecture, a concern about security issues is fundamental, unlike a monolithic architecture where security is implemented in physical barriers and limiting access to resources, the microservice architecture has its main characteristic in interoperability, reuse and scalability. The purpose of this paper is to compile security issues in microservices, as shown in the following sections.

For the recent advances of Cloud Computing technologies, the use of microservices on applications has been more widely addressed due to the rich set of features in such architecture. These are cloud-based applications that make users use it at low cost, threshold, and risk. Therefore, their practical use in business can be expected as a trend for the next generation of business applications [2].

Scaling monolithic applications is a challenge because they commonly offer a lot of services. Some of them are more popular than others. If popular services need to be scaled because they are highly demanded, the whole set of services will also be scaled at the same time, which implies that unpopular services will consume a large amount of server resources even when they are not going to be used [3].

The microservice-based architecture has emerged to simplify this reality and is a natural evolution to application models.

Microservices are a software oriented entity, which have the following features [4]:

Isolation from other microservices, as well as from the execution environment based on a virtualized container;

Autonomy – microservices can be deployed, destroyed, moved or duplicated independently. Thus, microservices cannot be bound to any local resource because microservice environment can create more than one instance of the same microservice;

Open and standardized interface that describes all specific goals with effectiveness, efficiency and available communication methods (either Application Programming Interface (API) or Graphical User Interface (GUI));

Microservice is fine-grained – each microservice should handle its own task.

Microservice architecture does not make an application any simpler, it only distributes the application logic into multiple smaller components, resulting in a much more complex network interaction model between components. When a real-world application is decomposed, it can easily create hundreds of microservices [5]. For this reason, this paper presents basic principles for the implementation of microservices aimed at classic security aspects for commercial applications. Organization of applications based on these standards mitigates common security issues.

The microservice architecture is a cloud application design pattern that implies that the application be divided into a number of small independent services, each of which is responsible for implementing a certain feature, as noted in Figure 1.

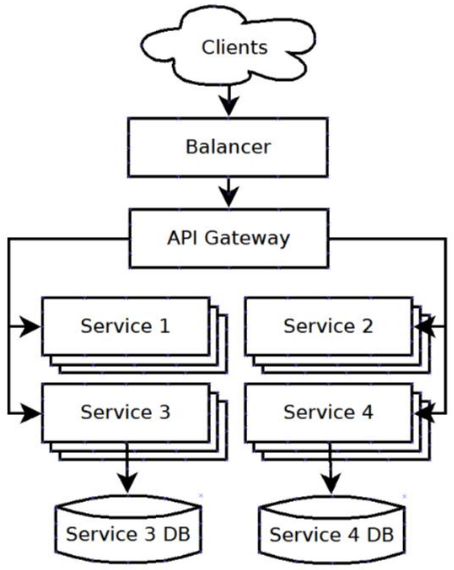

Figure 1. Microservice system architecture [4].

Microservices can be considered meta-processes in a Meta Operating System; they are independent, they can communicate with each other using messages and they can be duplicated, suspended or moved to any computational resource and so on [4]. Meta-modeling process is a type of modeling for analysis and modeling applicable to some known problems. Meta process modeling supports the effort of creating flexible process models.

The adopted methodology for this paper included a research in IEEE Xplore Digital Library, ACM Digital Library and Web of Science sources to provide all necessary information through published works. The strings Microservice AND security; Microservice AND Privacy; Microservice AND Cloud Computing was used to identify these works.

The remainder of this article is structured as follows: Section II is an overview of microservice in research topics; Section III presents security in Cloud Computing environment and Section IV shows the privacy model adopted in cloud applications for microservices and we present the standards of cloud environment and then conclude and summarize all results of that exercise in Section V.

II. MICROSERVICE

A. *State of the Art*

The microservice architecture was first approached in May 2011 at the workshop of software architecture [6], and since then it has been evolving and being adopted and implemented in Cloud Computing servers like Amazon AWS, Google Cloud and Azure.

From the technological perspective, early microservice applications were strongly influenced by a new generation of software development, deployment, and management tools [6]. Figure 2 show timeline with the technologies that drove the microservice architecture. The use of Linux Containers (LXC) was the first widely used container technique. It uses kernel namespaces to provide resource isolation [7], until the service mesh which build on sidecar technologies to provide a fully integrated service-to-service communication monitoring [6].

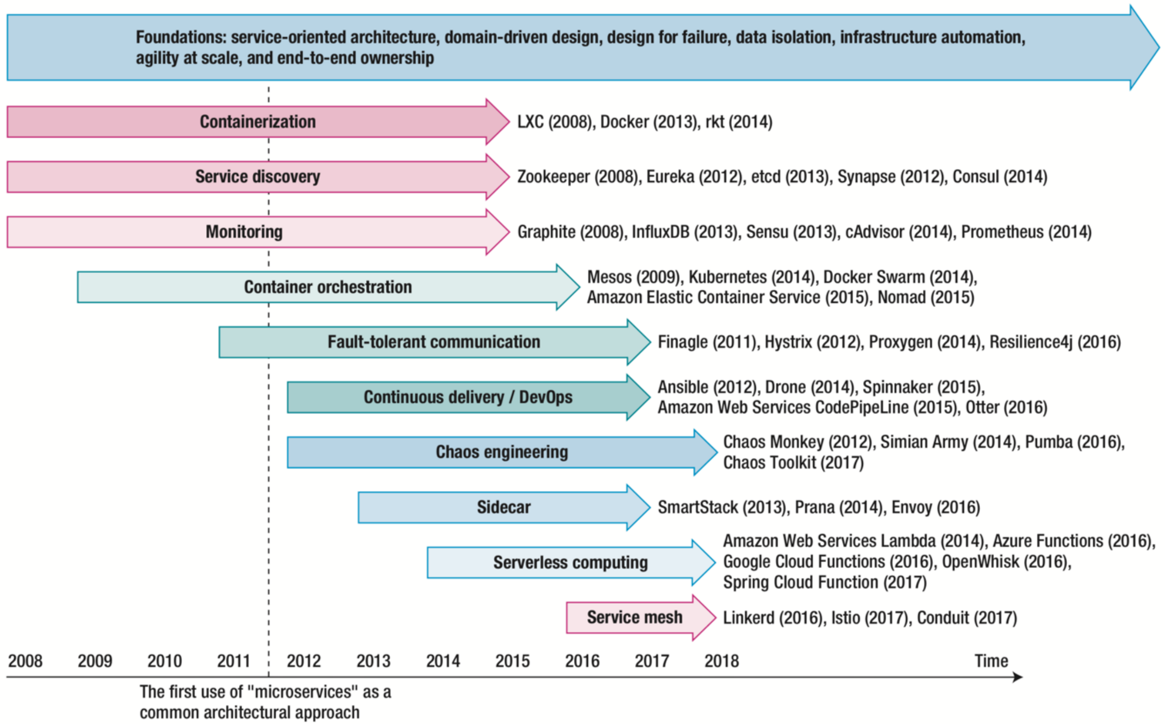

Figure 2. A microservice technologies timeline [6].

State of the art in microservices focuses its searches on the question of architecture analysis, performance, maintenance and security [5]. With the migration to continuous delivery culture, the organization of companies has changed from long processes which sometimes went through different areas to smaller teams, where each is responsible for developing their own microservice and for providing an API that will be used by other teams. On this point, the DevOps culture ends up being the most commonly used.

Information security according to ISO/IEC 27001 standards is based on three principles of confidentiality, integrity and availability. The microservices implementation is heavily bogged down in the irrevocability guarantee of these principles, for that reason some measures must be taken due to its complexity, the architecture proposed in this work treats numerous benefits for the guarantee of information security.

The development of solutions based on microservices has naturally used Cloud Computing environments so as to make the most of the best characteristics and functionalities provided by various solutions in the market. In this study we identified 3 relevant topics: the question of granularity, the deployment process and the resulting patterns.

A Microservice Architecture is a way of architecting software applications as independently deployable services. Based on Fowler, microservices can be characterized by a number of principles [8]:

- organization around business capability
- evolutionary design
- deployment / infrastructure automation
- intelligence in the endpoints
- heterogeneity and decentralized control
- decentralized control of data
- design for failure

The aforementioned principles are fundamental in the architecture that will be better described in the next section. The implementation of microservices is based on trade-offs between security and performance. This research found that the implementation of microservices uses the most advanced resources from Cloud Computing. The main characteristics of Cloud Computing can be summarized in the following points [9]:

- *Multi-Tenancy*

Refers to having more than one occupant of the cloud living and sharing with other occupants of the provider's infrastructures, including computational resources, storage, services, and applications. Through multi-tenancy, clouds provide simultaneous, secure hosting of services for various customers using the same cloud infrastructure resources. It is an exclusive characteristic to resource sharing on clouds.

- *Elasticity*

Another important aspect of Cloud Computing implies that the user is able to scale up or down resources assigned to services or resources based on the current demand. For providers, scaling up and down of a tenant's resources gives a prospect to other tenants to use the tenant's previously assigned resources.

- *Availability of Information*

Service Level Agreement (SLA) is a trust bond between cloud provider and customer. It defines a maximum time for which the network resources or applications may not be available for use by the customer. Due to the complex nature of customer demands, a simple measure and trigger process may not work for SLA enforcement.

- *Multiple Stakeholders*

In a Cloud Computing model, there are different stakeholders involved: cloud provider (an entity that delivers infrastructures to the cloud's customers), service provider (an entity that uses the cloud infrastructure to deliver applications/services to end users), and customer (an entity that uses services hosted on the cloud infrastructure).

Another important characteristic is the deployment of microservices. A cloud deployment model signifies a specific type of Cloud Computing environment, renowned by ownership, size, and access. There are three common cloud deployment models, namely private cloud, public cloud, and hybrid cloud [9].

Figure 3 shows differences between two architectures and demonstrates the microservices implantation. Their independence and granularity can be provided in several infrastructures.

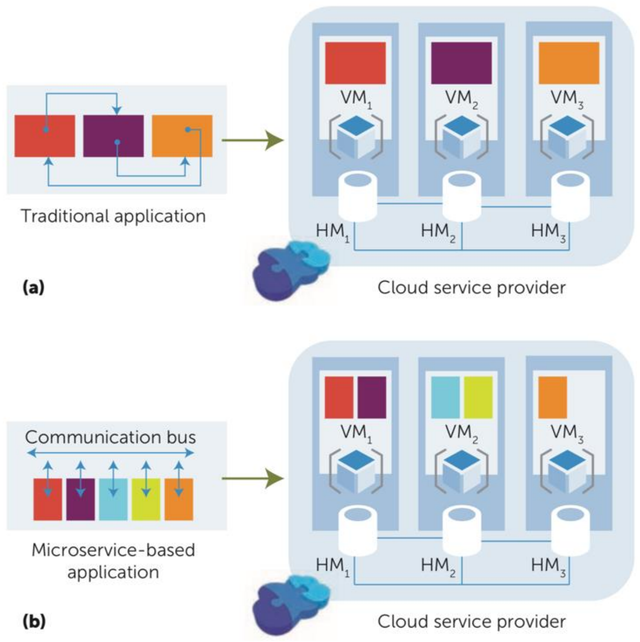

Figure 3. Software deployment in a cloud platform using (a) conventional and (b) microservice-based software [10].

Microservices are relatively small and autonomous services deployed independently, with a single and clearly-defined purpose. Because of their independent deployment, they have a lot of advantages. They can be developed in different programming languages, they can scale independently from other services and they can be deployed in the hardware that best suits their needs [10].

Moreover, because of their size, they are easier to maintain and more fault tolerant since the failure of one service will not break the whole system the way it could happen in a monolithic system [10].

Another characteristic of microservices is cloud native applications, the support of the IDEAL properties: Isolation of state, Distribution, Elasticity, Automated management and Loose Coupling. Microservices propose to vertically decompose the applications into a subset of business-driven services. Every service can be developed, deployed and tested independently by different development teams, and by means of different technology stacks. The responsibility of the development of a microservice belongs only to one team, who is in charge of the whole development process, including deploying, operating and upgrading the service when needed [10]. Figure 4 shows the complexity related.

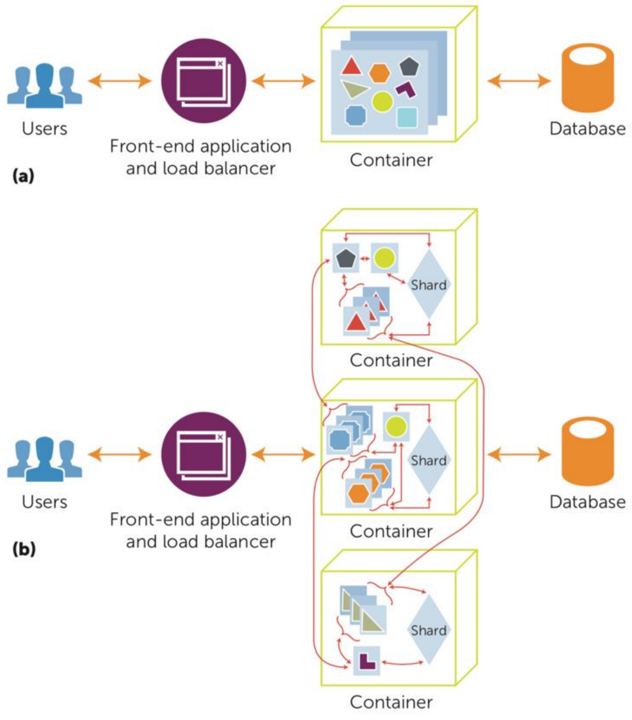

Figure 4. Architectural complexity of (a) monolithic and (b) microservice-based software [10].

Decoupling applications in this manner yields several benefits: it simplifies scaling (each service can be scaled independently), provides greater flexibility in resource allocation and scheduling, allows greater code reuse, enables new fault tolerant mechanisms, provides better modularity, and allows application developers to take advantage of services from other provides e.g., Amazon S3. As a result, this architecture has been widely adopted by both startups and large established companies (e.g., Uber and Netflix), and is being deployed at significant scale (e.g., Uber's application is composed of over 1000 microservices) [12].

The use of microservices can reduce the operational costs, as shown in the study [10]. The comparison was made in a cloud and monolithic solutions environment.

1) Cost comparison

In the study carried out in paper [13], it is shown a cost comparison in the various commercially used architectures of software development. In summary, use of microservices brings lower infrastructure spending by allowing scalability as well as scalability since the measurement of operating cost is done by use. In the old monolithic architecture many resources end up being loaded to memory even without being used, this is one of the great differences for the strong diffusion of this new architecture in software industry.

Given that each architecture was deployed in different infrastructures, we defined and calculated the metric Cost per Million of Requests (CMR) for each architecture in the three scenarios, in order to easily compare their execution costs. For each scenario and architecture, this metric was calculated by dividing the monthly infrastructure costs by the number of requests supported per month, which is calculated by multiplying the number of requests supported per minute by 43,200 —the number of minutes per month (60*24*30)—. We assumed a constant throughput per minute during a month [13]. The CMR metric for each architecture is shown in Figure 5.

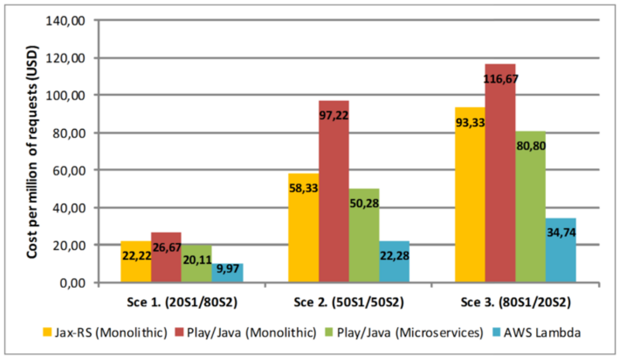

Figure 5. Cost Comparison of The Three Architectures per Million of Requests [13].

2) Granularity

Microservices can be declared with varying levels of capability, and the size of this functionality is typically referred to as its granularity, that is, the functional complexity coded in a service or number of use cases implemented by a microservice. Since microservices are discrete and must be composed into greater functional entities to support business workflows, it follows that message passing between microservices (as a result of method invocation) increases as the microservices become finer-grained.

The 'building-block' approach to service composition is attractive from an architectural perspective; arguments for service reuse can be made, and the gap between application design and the user requirements documentation can be reduced. However, the increase in communication between services (manifesting as out-of-process calls and the number of service calls made) also increases the response time of an application, particularly when many small increases in latency are compounded together [7].

Container-based technologies, and in particular its best known implementation Docker, made deployment of our new application possible through several characteristics [12]:

- A Docker image contains all of its dependencies, which means a given service can be treated as a black box, only exposing its API in exchange for resources.
- The containers are by default sealed from one to another, which results in guaranteed low coupling, without the high cost associated with virtual machines.
- Docker Compose made it possible to easily deploy any number of services, by composing in a text file an application made of several services.
- Docker Swarm mode allows for complete decoupling of the containers and the machines supporting them. In its recent version 3, Docker Compose allows for Distributed Application Bundles, which define applications made of several services without any dependence other than the presence of a Docker host IP address and access credentials.

Recommended patterns on how to compose microservices together [8]:

1. *Aggregator Microservice Design Pattern* – e.g., a service invoking others to retrieve / process data.
2. *Proxy Microservice Design Pattern* – a variation of the Aggregator with no aggregation.
3. *Chained Microservice Design Pattern* – produces a single consolidated response to a request.
4. *Branch Microservice Design Pattern* – extends the Aggregator and allows simultaneous response processing from possibly mutually exclusive chains of microservices.
5. *Shared Data Microservice Design Pattern* – towards autonomy through full-stack services with control of all components.
6. *Asynchronous Messaging Microservice Design Pattern* – use message queues instead of Representational State Transfer (REST) request/response pattern.

Integration is another important feature. The architecture of microservices allows for better integration of corporations where there are areas that handle a number of business activities. As this pattern is based on the independence of technologies, the services made available can be developed without a change in technology, which is usually expensive. Throughout this study we present data that show how this pattern brought about significant improvements in the development of solutions for the software industry.

B. Architecture

Microservice architecture has become a dominant architectural style choice in the service-oriented software industry. Microservice is a style of architecture that puts the emphasis on dividing the system into small and lightweight services that are purposely built to perform a very cohesive business function, and is an evolution of the traditional service oriented architecture style [14], in which what is presents a scenario of microservice architecture (Figure 6) in which five services working independently provide requests of a mobile app through an API [15].

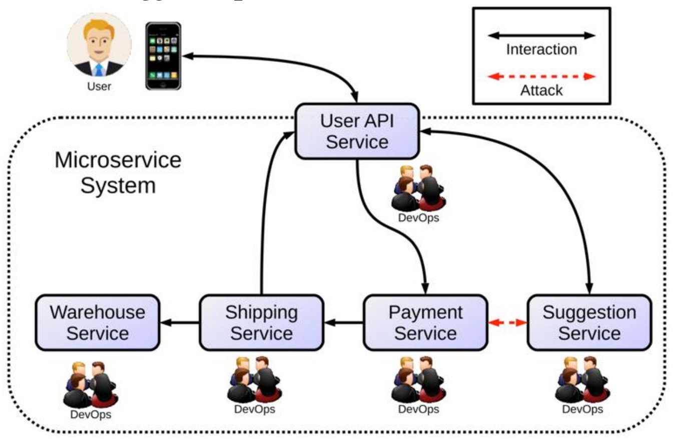

Figure 6. Example scenario of a microservice system[15].

The idea of splitting an application into a set of smaller and interconnected services (microservice) is currently attracting the interest of many application developers and service providers (e.g., Amazon [16][17], Netflix [12][18][19] and eBay [20][5]).

A Microservice based architecture has a pattern for development of distributed applications, where the application is composed of a number of smaller "independent" components; these components are small applications in themselves [21].

A microservice normally comprises three layers as a typical 3-tiered application[22], consisting of an interface layer [23], a business logic layer [20] and a data persistence layer, but within a much smaller bounded context. This sets a broad scope of the technical capabilities that a microservice could possess. However, not every microservice provides all capabilities. This would vary depending on how the function provided is meant to be consumed. For example, a microservice used primarily by providers of APIs would have a communications interface layer, business logic and data persistence layers but not necessarily have user interfaces [21].

We are considering a reference architecture model of microservices, demonstrating the main components and elements of this standard [21]. Table I presents a comparison between monolithic architecture and microservice architecture.

Table I. Comparing monolithic and microservice architecture [21].

Category	Monolithic Architecture	Microservice Architecture
Code	A single code base for the entire application.	Multiple code bases. Each microservice has its own code base.
Understandability	Often confusing and hard to maintain.	Much better readability and much easier to maintain.
Deployment	Complex deployments with maintenance windows and schedule downtimes.	Simple deployment as each microservice can be deployed individually, with minimal or zero downtime.

Category	Monolithic Architecture	Microservice Architecture
Language	Typically, entirely developed in one programming language.	Each microservice can be developed in a different programming language.
Scaling	Requires you to scale the entire application even though bottlenecks are localized.	Enables you to scale bottle-necked services without scaling the entire application.

In this paper, we will cover the following main elements:

1) API Proxy

To "de-couple" the microservice from its consumers, this proxy pattern is applied at the microservice interface level, regardless of the "API proxy" component. Organizations will provide APIs to different consumers, some of whom are within and others outside of the enterprise. These microservices would differ in SLA, security requirements, access levels, etc. [21].

2) Enterprise API Registry

The "discovery" requirements of the microservices are met through the use of the API registry service. Its purpose is to make the interfaces exposed by the microservice visible to consumers of the services both within and outside of the enterprise. An "Enterprise API registry" is a shared component across the enterprise, whose location must be well known and accessible. Its information content is published in a standard format, information should be in consistent and human readable format, and must have controlled access. It must have search and retrieval capabilities to allow users to look up details on available API specifications at design time [21].

3) Enterprise Microservice Repository

The "enterprise microservice repository" would be a shared repository for storing information about microservices. It provides information such as microservice lifecycle status, versions, business and development ownership, detailed information like its purpose, how it achieves the purpose, tools, technologies, architecture, the service it provides, any APIs it consumes, data persisted and queried and any specific non-functional requirements. In the absence of well-defined repository standards, the enterprise must define its own standard specification artefacts for microservices [5].

These elements are fundamental to the organized implementation of microservices and have been considered in this survey.

C. Microservice Standards and Solutions

In the centralized structure, the standardization becomes almost a natural way, but in microservices implementation this philosophy changes.

Traditional enterprise applications are divided into the front-end User Interface (UI), service-side logic components, and database. Front-end UI components run on user devices, such as web pages or mobile-side interfaces. Server-side logic components run on a server or in the cloud. The back-end database hosts application data. Server-side components work in conjunction with the database to handle requests issued by users [24].

Teams building microservices prefer a different approach to standards too. Rather than using a set of defined standards, written down somewhere on paper, they prefer the idea of producing useful tools that other developers can use to solve problems similar to the ones they are facing. These tools are usually harvested from implementations and shared with a wider group, sometimes, but not exclusively. Using a git and github has become the *de facto* version control system of choice. Open source practices are becoming more and more common in-house [25].

A microservice is an application on its own to perform the functions required. It evolves independently and can choose its own architecture, technology, platform, and can be managed, deployed and scaled independently with its own release lifecycle and development methodology. This approach takes away the construct of the Service-Oriented Architecture (SOA) and Enterprise Service Bus (ESB)and the accompanying challenges by making "smart endpoints" and treating the intermediate layers as network resources whose function is that of data transfer [21].

Unlike SOA, microservices do not have integration components responsible for service orchestration and prefer choreography. Business processes are embedded in services and there is no logic in the integration. Thus, Microservices themselves are responsible for the interaction with others. This gives limited flexibility to design or adjust business processes over the company’s IT. It is a payoff for microservice independent service management. However, Netflix considers even the option to orchestrate microservices, which is not a mainstream path [26].

The applications that expose interfaces that can be used by other applications to interact with are defined as API [5]. Microservice APIs which are built using internet communication protocols like HTTP adhere to open standards like REST [27][28] and SOAP [3] and use data exchange technologies like XML [29] and JavaScript Object Notation (JSON) [5].

Applications developed in a monolithic architecture perform multiple functions such as providing address validation, product catalogue, customer credit check, etc. When using the microservice based architecture pattern, applications are created for specific functions, such as address validation, customer credit check and online ordering; these applications are cobbled together to provide the entire capability for the proposed service. The approach to application development based on microservice architecture addresses the challenges of "monolithic" application and services [21].

In the research undertaken in this paper, the microservices are implemented and documented as follows [14]:

1) Architectural views/diagrams

- UML
- Standard modeling languages, e.g., RAML and YAML.
- Specifically designed modeling languages, e.g., CAMLE.

- Standard specification languages, e.g., Javascript (Node.js), JSON and Ruby.
- Specifically designed specification languages, e.g., Jolie.
- Pseudocode for algorithms.

2) REST

Representational State Transfer (REST) consisting of a set of architectural principles that, when followed, allows a well-defined interface design to be created. Applications that use REST principles are called RESTFul. REST [5][29][28][30] is often applied to provide services to other services (web services) and to the same full use of messages. To better understand the architectural style, it is important to highlight three important concepts: (i) feature; (ii) operations and (iii) representations. Resource is any information that is made available to customers through a Unique Identifier (URI). We can also define resource as being the source of representations. The representations are a set of data that explains the state of the requested resource. URIs must have a notation pattern, be descriptive, and have a previously defined hierarchy. The same resource can be identified by one or more URIs, but a URI [31], [32] identifies only one resource.

3) API

API is a basic authentication, including API user registration with strong password protection, (b) modern security mechanisms such as message level security, web signature and web encryption, and (c) security mechanisms within API and its backend services as a third security factor such as token based API for backend authentication, public key infrastructure and transport layer handshake protocol [23].

REST APIs [18] are developed in many technologies and microservices developed using different types of programming languages (Java, .NET, PHP, Ruby, Phyton, Scala, NodeJs etc.) and persistent technologies (SQL, No-SQL, etc.) [3][8][33]. They can be managed and exposed to web clients, who can then access the microservices and receive their responses through a "*livequery*" mechanism whereby updates to database data are instantly communicated to subscribing clients [29]. Figure 7 best presents categories of practices for designing REST-based web services.

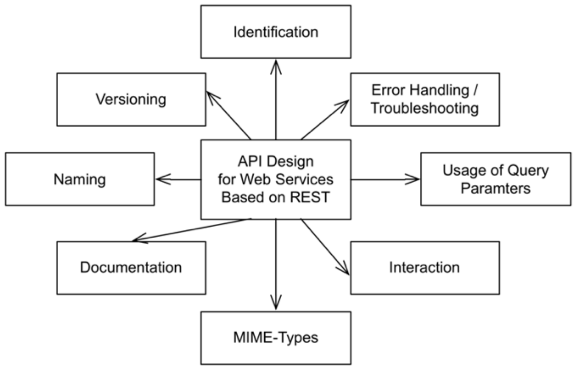

Figure 7. Categories of best practices for designing REST [34].

NoSQL databases are used in these implementations [29][35][36][37]. The NoSQL nature of the database is essential for providing the scaling, sharing and replication functionality expected from modern architectures, as well as to better support hierarchical data required for collaborative document editing [29].

The popularity of microservice-based architecture is evident from the report by the popular jobs portal *indeed.com*, in which the number of job openings on microservices-related technologies such as JSON [5][38][32] and REST [3][29][28] has grown more than 100 times in the last six years, whereas jobs in similar technology areas like SOAP and XML have remained nearly identical [5].

Solutions for microservices seek to implement simple algorithms that meet specific needs with the elements presented in this section. security on Cloud Computing

III. SECURITY ON CLOUD COMPUTING MICROSERVICES

Switching from a monolithic or centralized architecture to a decentralized architecture requires some care. In the past, security was focused on a single point [15], responsible for receiving all service requests. In the microservice-based architecture, the resources are offered through several points of access that interconnect each other, forming a unique solution.

Microservices combined with secure containers can facilitate new ways of building critical applications. These applications will benefit from tools and services built for less critical software. Secure containers and compiler extensions can help address more stringent requirements of critical applications. Although this approach is sufficient for implementing fail-stop applications, there are still several open research questions regarding whether and how it might support fail-operational applications [40].

Monolithic security services are relatively easier to implement than microservices. Monolithic services have a clear boundary and encapsulate their intercommunications. This will obscure security vulnerabilities [41][42] within the inner layers of the system. A microservice also encapsulates its communications. Both microservices and services are based upon clear requirements.

In a microservice-based system a simple routine completion requires the microservices to communicate with each other over network, for example. This will expose more data and information (endpoints) about the system and thus it expands the attack surface [19]. Some care must be taken in the communication between other services in the same network, and this is one of the major challenges [23][43][29] in this approach.

Monolithic applications, as previously explained, have a single and shared code base where all the developers work together. This development methodology has a few downsides, as it needs to struggle with handling cases in which the number of users exceeds the capacity of the server and it is hard to manage and maintain due to the lack of mechanisms aimed at modularization [26]. The evolution of the development of an application in monolithic architecture becomes quite complex, considering that in order to add new functionalities, one must change the source code, and still considering the same reasons, making the software hard to maintain. Monolithic architectures are typically difficult to deploy, difficult to upgrade and maintain and difficult to understand [27].

The deployment of monolithic architecture applications in Cloud Computing environments causes a very negative impact: services need to be scaled because they are highly demanding. The whole set of services will also be scale at the same time, which generates unpopular services that consume a large amount of server resources even when they are not going to be used [3].

The organization of teams for the development of a system based on microservices is generally subdivided into teams and services, and these teams are generally responsible for the implementation and delivery of services. For this type of implementation, the teams have to be aligned in the purposes of the microservices and the interconnection between them, thus also synchronizing the protocol [44] used to carry out the communication, thus respecting a standard for access protection or improper interception. Defining the way services are interconnected and interacting is the key point of security [38].

The security challenge brought by such network complexity is the ever-increasing difficulty in debugging, monitoring, auditing and forensic analysis of the entire application [45]. Since microservices are often deployed in a cloud that the application owners do not control, it is difficult for them to construct a global view of the entire application [5].

In microservice architecture, an application is essentially a collection of workflows. These workflows can compose many levels of services, each processing and modifying the data before its final destination. What we need is a way to certify the metadata related to a data stream and manage its validity during time and re-elaboration [46].

Security is a major challenge that must be carefully thought of in microservices architecture. Services communicate with each other in various ways creating a trust relationship. For some systems, it is vital that a user is identified in all the chains of a service communication happening between microservices.

Microservices predominant execution environments are containers, that remove dependencies on the underlying infrastructure services, which reduces the complexity of dealing with those platforms [47], microservices need high availability and scalability characteristics provided by providers of Cloud Computing, environment preferably used by the developers. In this architecture the four security aspects that should be considered are: containers, data, permission, and network [48], as noted in Figure 8.

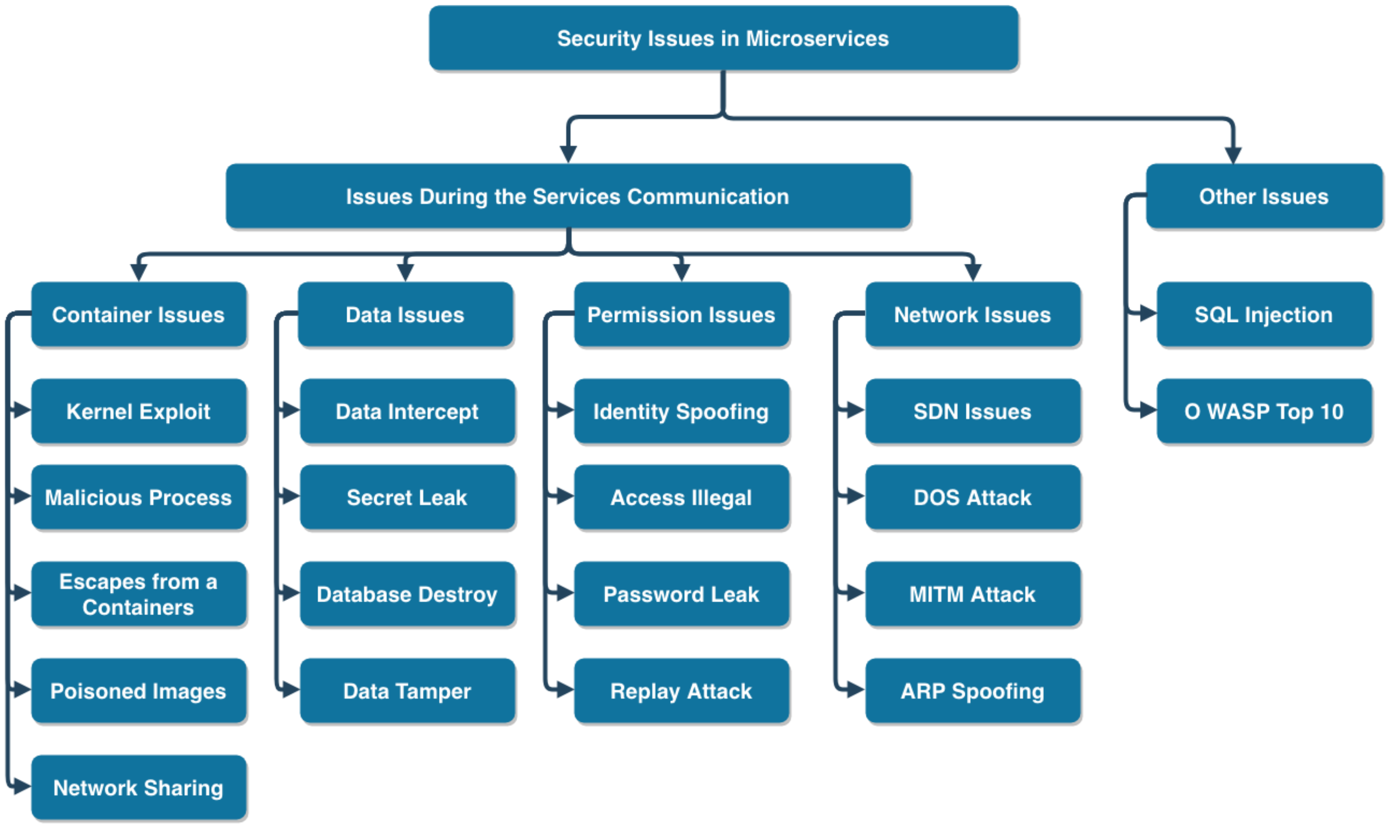

Figure 8. A taxonomy of security issues of Microservices [48]

Based on main security aspects of microservices discussed in Figure 6, the following main safety mechanisms will be presented to prevent safety deficiencies in microservices:

- mutual authentication of Services Using Mutual Transport Layer Security - with a self-hosted Public Key Infrastructure as a method to protect all internal service-to-service communication [49];
- host-authenticated TLS with in-band authentication are well-known solutions that are employed by designers to handle security challenges [14][15];
- principal propagation via Security Tokens: after a user has been authenticated by the gateway, the microservices behind it will be processing user's requests, a security token is created on the server side upon the successful validation of the clients credentials and given to the client for subsequent use [49].

Although the microservices are independent and do not cause dependencies among the modules, the biggest challenge nowadays is to guarantee availability [50]. The DevOps movement (set of practices to integrate the software development to IT operations) is currently collaborating with cloud environments and microservice architecture, providing continuous integration from the code compilation to the availability of the test and production environment, making it a facilitator for systems implementation utilizing microservices.

Ensuring the availability of services is presented as a security requirement facilitated by the use of the microservice architecture. This approach usually works by fragmenting the entire solution in smaller pieces [51]. Considering that these fragments are parts of the code with specific functions (microservices), in the event of a fragment failure, it would not result in the unavailability of all system resources. Availability has some critical points as they are bound to be observed, such as: implementing software versions, software crash recovery, invasions, unavailability of infra features beyond points.

In a microservice architecture, it is typical for many instances of a particular service to be running at any one time and for these instances to stop and start over time [52]. The problem of service discovery is to enable service consumers to locate service providers in real time to facilitate communication [53]. Docker Containers have been gaining a lot of hard work because of their agility and ease of making new services available [50]. The containers allow the microservices to be packaged [54] and available next to their dependencies in a single image, thus facilitating the availability of the service in a timely manner, minimizing downtime. This mode is called code portability [33]. In the context of microservices, the use of Docker containers for service delivery has resulted in benefits under various aspects such as automation, independence, portability and security, especially when considering ease of management, creation and continuous integration of environments systems offered by the Docker platform. In Docker, each container consists of only the application and the dependencies that the application needs to run, ideally no more and no less [33].

Another security concern involves the trust among the distributed microservices. An individual microservice may be compromised and controlled by an adversary. For example, the adversary may exploit vulnerability in a public facing microservice and escalate privilege on the virtual machine that the microservice runs in. As another example, insiders may abuse their privileges to control some microservices. As a result, individual microservices may not be trustworthy [5].

IV. PRIVACY ISSUES

Privacy has been a barrier for the adoption of Cloud Computing [51][55]. The migration to microservices has helped overcome this obstacle due to the scale gains proposed in this architecture.

In general, privacy refers to the condition or state of hiding the presence or view [56]. There is a need to attain this state in the places where confidential things are used such as data and files. In cloud data storage privacy is needed to attain the data, user identity and controls [57].

Trust is a crucial factor in Cloud Computing environments in current practice. It depends mostly on observation of characteristics, and self-evaluation of cloud service vendors. Existing trust mechanisms in the cloud are characteristics-based trust, SLA confirmation-based trust, Cloud transparency techniques, Trust as a service, Formal endorsement, audit and standards. In order to attain the service, it requires to be used in blend with social and technological mechanisms for providing persistent trust [58].

The exchange of sensitive data is intense in large-scale scenarios of Cloud Computing, with several federations, where multiple Identity Providers and Service Providers work together to provide services. Therefore, identity management should provide models and privacy mechanisms in order to manage the sensitive data of its users [43].

For service provider's standards in Cloud Computing environment, the contract is usually based on good deeds of that company, and hence the users need to pay attention to the security requirements, contract terms and other credentials. The users must have clear understanding in detailed terms and conditions of service providers and also the risk involved in signing the service provider's contract before moving to cloud [58].

Cloud service provides various options to the business customers to choose the level of protection needed for their data. The most common of these approaches is encryption. The customer chooses the type of encryption that they prefer and store the encryption key in a safe place under their control [44].

To ensure privacy, a well referenced model is used. This model is presented in Figure 9.

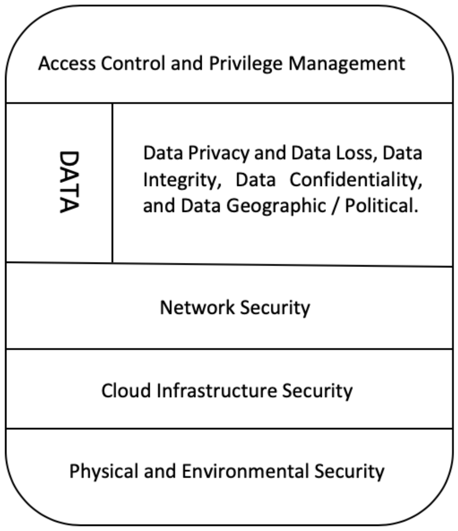
Access Control and Privilege Management
DATA
Data Privacy and Data Loss, Data Integrity, Data Confidentiality, and Data Geographic / Political.
Network Security
Cloud Infrastructure Security
Physical and Environmental Security

Figure 9. Cloud security and privacy model [55].

According to the proposed model in [55], a secure and private cloud model is divided into five layers: Physical and Environmental Security, Cloud Infrastructure Security, Network Security, Data and Access Control and Privilege Management:

A. Physical and Environmental Security

Layer of policies adopted with the objective of protecting physical access to the cloud provider [16]. Another benefit of cloud service is the ability to meet the elasticity of demand. Business processes should consider the availability of open-ended resources at an affordable cost. Use of services such as Software as Service (SaaS) enables the business to focus more on their core strengths. Since availability of computing resource is no longer a constraint, the business should take advantage of computing power to experiment with new ideas to serve the customers better. Since the cost is usage based, changing business processes to take advantage of newer technologies is advantageous to a business. Cloud service addresses an important business process for every business, namely backup and recovery. Many businesses do not pay enough attention to data backup and recovery because it is time consuming and does not provide immediate benefit until some disaster strikes, which is rare. With the cloud taking care of all the management aspects of data backup and recovery, businesses tend to focus on their strengths and a cloud provides the essential service of backup and recovery when needed. A common perception is that in order to provide security the user must have control over the devices. This usually applies to physical security. Given the elastic nature of demand for service and the centralization of service, the cloud environment is in a better position to provide greater physical security to the hardware [44].

B. Cloud Infrastructure Security

Addresses issues with cloud infrastructure security, but specifically with the virtualization environment [59]. Above this, the combination of software layers, the virtualization layer and the management layer allow for the effective management of servers. Virtualization is a critical element of cloud implementations and is used to provide the essential cloud characteristics of location independence, resource pooling and rapid elasticity. Differing from traditional network topologies such as client–server, Cloud Computing is able to offer robustness and alleviate traffic congestion issues. The management layer is able to monitor traffic and respond to peaks or drops with the creation of new servers or the destruction of unnecessary ones. The management layer has the additional ability of being able to implement security monitoring and rules throughout the cloud [60].

C. Network Security

Specifies the medium to which the end user connects to the cloud, comprising browsers and their connection [20]. The client-server in the cloud is accomplished by a client sending a request to the server and waiting for the result, where the server performs the computational process. A connection medium between a client and cloud service provider is the Web browser which relates to the cloud system. As discussed before, a client sends a request and needs to validate it on its own to check the authority of the user on the cloud system. Client credentials are signed by using Extensible Markup Language signature to authenticate and Extensible Markup Language (XML) encryption to encrypt the Simple Object Access Protocol (SOAP) messages [58].

D. Data

Layers cover data privacy, integrity, confidentiality, and geographic location [46]. To prevent data loss in cloud different security measures can be adopted. One of the most important measures is to maintain backup of all data in cloud which can be accessed in case of data loss. However, data backup must also be protected to maintain the security properties of data such as integrity and confidentiality. Different data loss prevention mechanisms have been proposed in research and academics for the prevention of data loss in network, processing, and storage. Many companies, including Symantec, McAfee, and Cisco, have also developed solutions to implement data loss prevention across storage systems, networks and end points [61].

E. Access Control and Privilege Management

Policies and processes used by cloud services provider to ensure that only the users granted appropriate privileges can use or modify data. It includes identification, authentication [62] and authorization issues [55]. The access control and privilege management are policies and processes used by cloud providers to ensure that only the consumers granted appropriate privileges can accede, use or modify data. Lately, researchers have proposed many models (such as Attribute Based Encryption (ABE), Key Policy Attribute Based Encryption, Cipher Text Policy Attribute Based Encryption, etc.) that are useful to provider security and access control. The majority of these proposed models are the modified form of the classical ABE model [63].

The implementation of the architecture proposed in this paper and the use of API brings some issues that must be identified to avoid problems.

API is used by the developers which act as an interface between the cloud service providers and the client. It allows users to manage and get the information from service providers. API and the related software need to be highly secured as it is used by the cloud users to access their data. API is the public front door entry to the data and accessible externally, thus it incorporates many threats in it [61].

V. CONCLUSIONS AND FUTURE WORK

Microservice-based architectures have become increasingly popular as an architectural style for software development. In this architectural style, the services provided by software solutions are divided into smaller parts and focused on the specific service of some functionalities. The approach of developing microservices with the construction of smaller software components has a number of advantages over the traditional monolithic architecture such as increasing the resilience of the software implemented as a microservice and the ease of scaling the solution implemented through the microservices.

Security aspects are critical in this architecture because the widespread use of Cloud Computing services, as demonstrated by the complexity of implementation, requires care with the privacy and security of information that is handled by those services.

The development of software using the microservice-based architecture comprises important aspects that must be observed in order to obtain good results. The objective of this article was to present the elements that should be considered for the development of solutions based on microservices and describing how the microservice-based architecture is defined. In addition to identifying the elements related to their implementation in Cloud Computing environment and explaining the privacy model applicable and relating the elements that intergrade the standards and solutions linked to the microservice-based architecture.

Future work will be developed to present case studies demonstrating the implementation of the microservice architecture in a Cloud Computing environment with the use of Docker containers for its construction and summarization of security troubles.

REFERENCES

[1] W. H. C. Almeida, L. D. A. Monteiro, R. R. Hazin, C. De Lima, and F. S. Ferraz, "Survey on Microservice Architecture - Security , Privacy and Standardization on Cloud Computing Environment," *ICSEA 2017*, no. c, pp. 199–205, 2017.

[2] S. H. Jyhjong Lin, Lendy Chaoyu, "Migrating Web Applications to Clouds with Microservices Arcitectures," *Int. Conf. Appl. Syst. Innov.*, pp. 1–4, 2016.

[3] M. Villamizar and et al, "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud Evaluando el Patrón de Arquitectura Monolítica y de Micro Servicios Para Desplegar Aplicaciones en la Nube," *10th Comput. Colomb. Conf.*, pp. 583–590, 2015.

[4] D. I. Savchenko, G. I. Radchenko, and O. Taipale, "Microservices validation: Mjolnirr platform case study," *2015 38th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2015 - Proc.*, no. May, pp. 235–240, 2015.

[5] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservices-based cloud applications," *Proc. - IEEE 7th Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2015*, pp. 50–57, 2016.

[6] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, 2018.

[7] Á. Kovács, "Comparison of different linux containers," *2017 40th Int. Conf. Telecommun. Signal Process. TSP 2017*, vol. 2017–Janua, pp. 47–51, 2017.

[8] C. Pahl and P. Jamshidi, "Microservices: A Systematic Mapping Study," *Proc. 6th Int. Conf. Cloud Comput. Serv. Sci.*, vol. 1, no. Closer, pp. 137–146, 2016.

[9] H. Bennasar, M. Essaaidi, A. Bendahmane, and J. Ben-Othman, "State-of-The-Art of cloud computing cyber-security," *Proc. 2015 IEEE World Conf. Complex Syst. WCCS 2015*, 2016.

[10] C. Esposito, "Challenges in Delivering Software in the Cloud as Microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 10–14, 2016.

[11] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, "Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages," *Proc. XP2017 Sci. Work.*, p. 23, 2017.

[12] A. Panda, M. Sagiv, and S. Shenker, "Verification in the Age of Microservices," *Proc. 16th Work. Hot Top. Oper. Syst. - HotOS '17*, pp. 30–36, 2017.

[13] M. Villamizar *et al.*, "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures," *Proc. - 2016 16th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2016*, pp. 179–182, 2016.

[14] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," *Proc. - 2016 IEEE 9th Int. Conf. Serv. Comput. Appl. SOCA 2016*, pp. 44–51, 2016.

[15] K. Jander, L. Braubach, and A. Pokahr, "Defense-in-depth and Role Authentication for Microservice Systems," *Procedia Comput. Sci.*, vol. 130, pp. 456–463, 2018.

[16] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," *Proc. - 2015 Int. Conf. Futur. Internet Things Cloud, FiCloud 2015 2015 Int. Conf. Open Big Data, OBD 2015*, pp. 25–30, 2015.

[17] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, "Efficiency analysis of provisioning microservices," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, pp. 261–268, 2017.

[18] R. Heinrich *et al.*, "Performance Engineering for Microservices: Research Challenges and Directions," *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. Companion*, pp. 223–226, 2017.

[19] M. Ahmadvand and A. Ibrahim, "Requirements reconciliation for scalable and secure microservice (de)composition," *Proc. - 2016 IEEE 24th Int. Requir. Eng. Conf. Work. REW 2016*, pp. 68–73, 2017.

[20] T. Q. Thanh, S. Covaci, T. Magedanz, P. Gouvas, and A. Zafeiropoulos, "Embedding security and privacy into the development and operation of cloud applications and services," *2016 17th Int. Telecommun. Netw. Strateg. Plan. Symp.*, pp. 31–36, 2016.

[21] Yale Yu, H. Silveira, and M. Sundaram, "A microservice based reference architecture model in the context of enterprise architecture," *2016 IEEE Adv. Inf. Manag. Commun. Electron. Autom. Control Conf.*, pp. 1856–1860, 2016.

[22] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, "Orchestration of Containerized Microservices for IIoT using Docker," pp. 1532–1536, 2017.

[23] M. B. Mollah, M. A. K. Azad, and A. Vasilakos, "Security and

privacy challenges in mobile cloud computing: Survey and way ahead," *J. Netw. Comput. Appl.*, vol. 84, pp. 38–54, 2017.

[24] C. Y. Fan and S. P. Ma, "Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report," *Proc. - 2017 IEEE 6th Int. Conf. AI Mob. Serv. AIMS 2017*, pp. 109–112, 2017.

[25] J. Fowler, Marthin; Lewis, "Microservices: a definition of this new architectural term," *Microservices:a definition of this new architectural term*, 2014. .

[26] S. Systems, T. Cerny, and M. J. Donahoo, "Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems," pp. 228–235.

[27] S. Yamamoto, S. Matsumoto, and M. Nakamura, "Using cloud technologies for large-scale house data in smart city," *CloudCom 2012 - Proc. 2012 4th IEEE Int. Conf. Cloud Comput. Technol. Sci.*, pp. 141–148, 2012.

[28] J. Bogner and A. Zimmermann, "Towards Integrating Microservices with Adaptable Enterprise Architecture," *Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOCW*, vol. 2016–Septe, pp. 158–163, 2016.

[29] C. Gadea, M. Trifan, D. Ionescu, and B. Ionescu, "A reference architecture for real-time microservice API consumption," *Proc. 3rd Work. CrossCloud Infrastructures Platforms - CrossCloud '16*, pp. 1–6, 2016.

[30] D. Guo, W. Wang, G. Zeng, and Z. Wei, "Microservices architecture based cloudware deployment platform for service computing," *Proc. - 2016 IEEE Symp. Serv. Syst. Eng. SOSE 2016*, pp. 358–364, 2016.

[31] P. Marchetta, E. Natale, A. Pescape, A. Salvi, and S. Santini, "A map-based platform for smart mobility services," *Proc. - IEEE Symp. Comput. Commun.*, vol. 2016–Febru, pp. 19–24, 2016.

[32] A. de Camargo, I. Salvadori, R. dos S. Mello, and F. Siqueira, "An architecture to automate performance tests on microservices," *Proc. 18th Int. Conf. Inf. Integr. Web-based Appl. Serv. - iiWAS '16*, pp. 422–429, 2016.

[33] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," *Conf. Proc. - IEEE SOUTHEASTCON*, vol. 2016–July, pp. 0–4, 2016.

[34] P. Giessler, R. Steinegger, S. Abeck, and M. Gebhart, "Checklist for the API Design of Web Services based on REST," vol. 9, no. 3, pp. 41–51, 2016.

[35] A. Gueidi, H. Gharsellaoui, and S. Ben Ahmed, "A NoSQL-based Approach for Real-Time Managing of Embedded Data Bases," *Proc. - 2016 World Symp. Comput. Appl. Res. WSCAR 2016*, pp. 110–115, 2016.

[36] T. I. Damaiyanti, A. Imawan, and J. Kwon, "Extracting trends of traffic congestion using a NoSQL database," *Proc. - 4th IEEE Int. Conf. Big Data Cloud Comput. BDCloud 2014 with 7th IEEE Int. Conf. Soc. Comput. Networking, Soc. 2014 4th Int. Conf. Sustain. Comput. C*, pp. 209–213, 2015.

[37] R. Simmonds, P. Watson, and J. Halliday, "Antares: A Scalable, Real-Time, Fault Tolerant Data Store for Spatial Analysis," *Proc. - 2015 IEEE World Congr. Serv. Serv. 2015*, pp. 105–112, 2015.

[38] A. Ciuffoletti, "Automated Deployment of a Microservice-based Monitoring Infrastructure," *Procedia Comput. Sci.*, vol. 68, pp. 163–172, 2015.

[39] T. Combe, T. Paris-tech, A. Martin, R. Di Pietro, and N. B. Labs, "To Docker or Not to Docker : A Security Perspective," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, 2016.

[40] C. Fetzer, "Building critical applications using microservices," *IEEE Secur. Priv.*, vol. 14, no. 6, pp. 86–89, 2016.

[41] I. Khalil, A. Khreishah, and M. Azeem, "Cloud Computing Security: A Survey," *Computers*, vol. 3, no. 1, pp. 1–35, 2014.

[42] C. Saravanakumar and C. Arun, "Survey on interoperability, security, trust, privacy standardization of cloud computing," *Proc. 2014 Int. Conf. Contemp. Comput. Informatics, IC3I 2014*, pp. 977–982, 2014.

[43] J. Werner, C. M. Westphall, and C. B. Westphall, "Cloud identity management: A survey on privacy strategies," *Comput. Networks*, vol. 122, pp. 29–42, 2017.

[44] S. Srinivasan, "Data privacy concerns involving cloud," *2016 11th Int. Conf. Internet Technol. Secur. Trans. ICITST 2016*, pp. 53–56, 2017.

[45] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open Issues in Scheduling Microservices in the Cloud," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, 2016.

[46] F. Callegati, S. Giallorenzo, A. Melis, and M. Prandini, "Data security issues in MaaS-enabling platforms," *2016 IEEE 2nd Int. Forum Res. Technol. Soc. Ind. Leveraging a Better Tomorrow, RTSI 2016*, pp. 0–4, 2016.

[47] D. S. Linthicum, "Practical Use of Microservices in Moving Workloads to the Cloud," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 6–9, 2016.

[48] D. Yu, Y. Jin, Y. Zhang, and X. Zheng, "A survey on security issues in services communication of Microservices-enabled fog applications," *Concurr. Comput.*, no. September 2017, pp. 1–19, 2017.

[49] T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," *Proc. - 12th IEEE Int. Symp. Serv. Syst. Eng. SOSE 2018 9th Int. Work. Jt. Cloud Comput. JCC 2018*, pp. 11–20, 2018.

[50] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," *Proc. - 2016 IEEE Int. Conf. Cloud Eng. IC2E 2016 Co-located with 1st IEEE Int. Conf. Internet-of-Things Des. Implementation, IoTDI 2016*, pp. 202–211, 2016.

[51] K. Bao, I. Mauser, S. Kochanneck, H. Xu, and H. Schmeck, "A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings," *Proc. 1st Int. Work. Mashups Things APIs - MOTA '16*, pp. 1–6, 2016.

[52] D. Escobar *et al.*, "Towards the understanding and evolution of monolithic applications as microservices," *Proc. 2016 42nd Lat. Am. Comput. Conf. CLEI 2016*, 2017.

[53] J. Stubbs, W. Moreira, and R. Dooley, "Distributed Systems of Microservices Using Docker and Serfnode," *Proc. - 7th Int. Work. Sci. Gateways, IWSG 2015*, pp. 34–39, 2015.

[54] R. Roostaei and Z. Movahedi, "Mobility and Context-Aware Offloading in Mobile Cloud Computing," *Proc. - 13th IEEE Int. Conf. Ubiquitous Intell. Comput. 13th IEEE Int. Conf. Adv. Trust. Comput. 16th IEEE Int. Conf. Scalable Comput. Commun. IEEE Int.*, pp. 1144–1148, 2017.

[55] K. El Makkaoui, A. Ezzati, A. Beni-Hssane, and C. Motamed, "Data confidentiality in the world of cloud," *J. Theor. Appl. Inf. Technol.*, vol. 84, no. 3, pp. 305–314, 2016.

[56] C. Perra and S. Member, "A Framework for the Development of Sustainable Urban Mobility Applications," 2016.

[57] M. Thangavel, P. Varalakshmi, and S. Sridhar, "An analysis of privacy preservation schemes in cloud computing," *Proc. 2nd IEEE Int. Conf. Eng. Technol. ICETECH 2016*, no. March, pp. 146–151, 2016.

[58] G. Shanmugasundaram and A. P. Cloud, "A COMPREHENSIVE REVIEW ON CLOUD COMPUTING SECURITY," 2017.

[59] H. Gebre-amlak, S. Lee, A. M. A. Jabbari, Y. Chen, and B. Choi, "MIST : Mobility-Inspired SofTware-Defined Fog System," 2017.

[60] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Futur. Gener. Comput. Syst.*, vol. 28, no. 3, pp. 583–592, 2012.

[61] G. N. Dev, "A Survey on Security Threats in Cloud Computing Technology," *Int. J. Res.*, vol. 1, no. 8, pp. 1071–1081, 2014.

[62] R. H. Steinegger, D. Deckers, P. Giessler, and S. Abeck, "Risk-based authenticator for web applications," *Proc. 21st Eur. Conf. Pattern Lang. Programs - Eur. '16*, no. February 2017, pp. 1–11,

2016.

[63] K. E. Makkaoui*, A. Ezzati, A. Beni-Hssane, and C. Motamed, “Cloud Security and Privacy Model for Providing Secure Cloud Services,” 2016.