Zayed University
ZU Scholars

All Works

1-1-2022

Multi-party Contract Management for Microservices

Zakaria Maamar
Zayed University

Noura Faci
Laboratoire d'Informatique en Images et Systèmes d'Information

Joyce El Haddad
Paris Dauphine University

Fadwa Yahya
Prince Sattam Bin Abdulaziz University

Mohammad Askar
Paris Dauphine University; Centre for Ethnology and Comparative Sociology

Follow this and additional works at: https://zuscholars.zu.ac.ae/works

Part of the Computer Sciences Commons

Recommended Citation

Maamar, Zakaria; Faci, Noura; El Haddad, Joyce; Yahya, Fadwa; and Askar, Mohammad, "Multi-party Contract Management for Microservices" (2022). *All Works*. 5283.
https://zuscholars.zu.ac.ae/works/5283

This Conference Proceeding is brought to you for free and open access by ZU Scholars. It has been accepted for inclusion in All Works by an authorized administrator of ZU Scholars. For more information, please contact scholars@zu.ac.ae.

Multi-party Contract Management for Microservices

Zakaria Maamar[1][a], Noura Faci[2][b], Joyce El Haddad[3][c], Fadwa Yahya[4][d] and Mohammad Askar[3,5]
[1]*Zayed University, Dubai, U.A.E.*
[2]*Université Calude Bernard, CNRS, LIRIS, 69622 Villeurbanne Cedex, France*
[3]*Université Paris Dauphine-PSL, Paris, France*
[4]*Prince Sattam Bin Abdulaziz University, Al kharj, Saudi Arabia*
[5]*Université Paris Nanterre, Nanterre, France*

Keywords: Cloud, Contract, Edge, Internet of Things, Microservices, Quality-of-Service.

Abstract: This paper discusses the necessary steps and means for ensuring the successful deployment and execution of software components referred to as microservices on top of platforms referred to as Internet of Things (IoT) devices, clouds, and edges. These steps and means are packaged into formal documents known in the literature as contracts. Because of the multi-dimensional nature of deploying and executing microservices, contracts are specialized into discovery, deployment, and collaboration types, capturing each specific aspect of the completion of these contracts. This completion is associated with a set of Quality-of-Service (QoS) parameters that are monitored allowing to identify potential deviations between what has been agreed upon and what has really happened. To demonstrate the technical doability of contracts, a system is implemented using different datasets that support experiments related to assessing the impact of the number of microservices and platforms on the performance of the system.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

A cloud/edge-based Internet-of-Things (IoT) environment comprises a number of interconnected devices (*aka* things) working together to provision services that end-user applications would compose together. To sustain this provisioning and avoid the pitfalls of monolithic applications, the best architectural styles should be adopted with focus lately on microservices (Brito et al., 2021; Butzin et al., 2016). In a recent post by NGINX[1], Netflix shared its experience of transitioning "*from a traditional development model with 100 engineers producing a monolithic DVD-rental application to a microservices architecture with many small teams responsible for the end-to-end development of hundreds of microservices that work together to stream digital entertainment to millions of Netflix customers every day*".

Tapping into microservices' core characteristics as per (Lewis and Fowler, 2014), high-cohesion and loosely-coupled, the trend nowadays is to deploy microservice-based applications on a mix of cloud and edge platforms despite their differences. According to Khebbeb et al., cloud means more resources, more reliability, and more latency, and edge means less resources, less reliability, and less latency (Khebbeb et al., 2020). In fact, they complement each other (De Donno et al., 2019; Singh, 2017).

In a previous work, we designed and implemented the deployment and execution of microservices in the presence of many stakeholders exemplified with things, edge platforms, and cloud platforms (Maamar and Faci, 2021). The design and implementation took into account constraints related to things' limited technical capabilities, edges' closeness to things, and clouds' inappropriateness for real-time applications[2] along with additional characteristics like types

[a] https://orcid.org/0000-0003-4462-8337
[b] https://orcid.org/0000-0001-7428-6302
[c] https://orcid.org/0000-0002-2709-2430
[d] https://orcid.org/0000-0003-4661-1344
[1] https://tinyurl.com/ojm9zgp.

[2] Puliafito et al. report that "*the average round trip time between an Amazon Cloud server in Virginia (U.S.A.) and a device in the U.S. Pacific Coast is 66ms; it is equal to 125ms if the end device is in Italy; and reaches 302ms when the device is in Beijing*" (Puliafito et al., 2019).

Maamar, Z., Faci, N., El Haddad, J., Yahya, F. and Askar, M.
Multi-party Contract Management for Microservices.
DOI: 10.5220/0011266200003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 276-287
ISBN: 978-989-758-588-3; ISSN: 2184-2833
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

of things (static *versus* mobile), forms of interactions (vertical *versus* horizontal), and properties of resources that are consumed (limited *versus* limited-but-renewable *versus* non-shareable). In this paper, we identify and afterwards formalize the necessary steps and mechanisms that would first, confirm the binding of things/edges/clouds as stakeholders to microservices and second, allow these stakeholders to collaborate together, should they run into any obstacles that these constraints and characteristics could cause. We package the steps and mechanisms into contracts and track their satisfaction through a set of non-functional properties forming what the ICT community refers to as Quality-of-Service (QoS) model (Menascé, 2002).

The adoption of contracts is commonly reported in the ICT literature as per the survey paper (Marino et al., 2019). However, to the best of our knowledge, little exists when it comes to first, identifying contracts in the context of microservices, things, and edge/cloud platforms and second, defining contracts' types, lifecycles, and adjustments, should the QoS non-functional properties become unsatisfied. To address this gap, we proceed with (*i*) defining contracts to regulate microservices' deployment and execution, (*ii*) identifying types of contracts to ensure the success of this deployment and execution, (*iii*) specifying lifecycles of and dependencies between contracts, and, finally, (*iv*) demonstrating contract management through a system. The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 presents our microservices' deployment and execution approach that is referred to as chore*O*rchest mixing choreography and orchestration to achieve this deployment and execution. Section 4 examines contracts in terms of types, clauses, and lifecycles. Section 5 discusses the system that was implemented and the results obtained out of the experiments. Finally, Section 6 concludes and points out some future work.

2 RELATED WORK

To the best of our knowledge, there are not dedicated works that examine contracts and their complete lifecycles in an ecosystem of microservices, IoT, cloud, and edge. To address this gap, we discuss some works that adopt contracts for multiple purposes like monitoring, regulation, and security.

Balint and Truong propose a contract-aware IoT framework to manage and monitor IoT data marketplaces in (Balint and Truong, 2017). Contracts refer to data rights (e.g., derivation and reproduction), quality of data (e.g., completeness and conformity), pricing model (e.g., charges and subscription period), purchasing policy (e.g., contract termination and refund) and control (e.g., warranty and indemnity). Contracts are established between customers (either persons or software) and things' providers. Both providers and customers engage in contract negotiation to specify contractual terms when purchasing or selling data. To address the scalability of data marketplaces, the framework is designed as a microservices architecture allowing each service to scale without disrupting other services in the framework.

Longo et al. discuss the importance of public contracts to regulate the management of public services such as data services in (Longo et al., 2019). The authors note that the rapid and continued change of these services' requirements and expectations, is making contracts "obsolete" calling for their regular adjustment. To keep the contracts up-to-date, Longo et al. propose a cloud-based approach for assessing the QoS of local Transportation Services (TS) in Apulia Region (Southern Italy). SLA between TS providers and the Regional Authority, as well as the minimal guaranteed QoS levels between TS providers and passengers, are modeled as contracts enacted via a cloud-based system, which gathers data from sensors embedded into passengers' smartphones. As a result, changes in contracts' conditions to improve the perceived and delivered QoS have been quick and facilitated based on collected data.

Pan et al. report about first, the security and scalability challenges that IoT is facing because of the limited capabilities of IoT devices and second, the role that edge could have in helping IoT tackle these challenges in (Pan et al., 2019). The authors designed and prototyped an edge-based IoT framework, EdgeChain, that capitalizes on blockchain and smart contract technologies. EdgeChain uses a credit-based resource management system to control how much edge resources are made available for IoT devices with respect to predefined policies that consider priority, application types, and past behaviors. To enforce these policies, smart contracts regulate IoT devices' behaviors in a non-deniable and automated manner. In addition, all IoT devices' activities and transactions are recorded using blockchain for secure data logging and auditing. As a result, contracts permit carrying out trusted transactions.

Singh et al. propose a Service Level Agreement (SLA)-aware autonomic Technique for Allocation of Resources (STAR) given that current resource management solutions may not provision efficient services for cloud nodes and may end-up vio-

lating the SLA's clauses in (Singh et al., 2020). STAR aims at mitigating such violations and improving user satisfaction. Furthermore, STAR considers different QoS parameters such as execution time, cost, latency, reliability and availability to analyze the impact of QoS parameters on SLA violation and to dynamically manage resources based on QoS requirements.

Sun et al. present a contract-based resource sharing approach to schedule tasks in a fog-cloud environment in (Sun et al., 2020). Due to clouds' limitations, the authors address how to take advantage of clusters of fog resources so, that, more tasks can run on these resources. Using a sealed-bid bilateral auction mechanism (buyers, sellers, and auctioneers) and constructing functional domains, Sun et al. identify the best fog nodes (where some are mobiles) in each cluster with respect to the betweeness centrality, computing performance, and communication delay to the IoT nodes. During contract formation, a fog cluster can be either a buyer in the sense of having the right to use some fog nodes in other fog clusters or a seller in the sense of giving the right to other fog clusters to use the fog nodes that fall into its cluster, the cloud is seen as a trusted third party, and the commodity in the auction is the specific type of fog nodes in a particular fog cluster under corresponding time slot.

Truong and Klein adopt DevOps contracts to ensure the proper execution of IoT microservices over edge resources in (Truong and Klein, 2020). The authors note that many stakeholders participate in preparing the contracts, for instance IoT service users, IoT service providers, IoT unit providers, IoT developers, and IoT gateway/platform and edge platform providers. All these stakeholders share many concerns about IoT units, services, and, infrastructure resources. Terms such as access to data, quality of data, QoS, and price are included in the contracts.

It is clear that the works above (although a sample) expose the gap in examining contracts in an ecosystem of microservices, IoT, edge, and cloud. Our approach to manage contracts for microservices sheds light on types of contracts, lifecycles of contracts, interactions between contracts, and properties of contracts. This management also considers the platforms upon which microservices will be deployed and executed. Some platforms like things have limited processing capabilities while others like clouds are not fit for satisfying real-time applications' requirements.

3 MICROSERVICES DEPLOYMENT AND EXECUTION

This section summarizes our previous work on deploying and executing microservices over things, edges, and/or clouds.

Three elements namely, types of things, forms of interactions, and availabilities of resources, were taken into account:

Types of things. We identify 2 types of things labelled as either static or mobile. By considering these 2 types, we identify the strengths and limitations of things when it comes to interacting with the environment and consuming resources. On the one hand, a static thing is assigned to a physical location and cannot be moved because of its size, security concerns, and safety regulations for example. On the other hand, a mobile thing is fitted with wireless communication means so, that, it roams the environment. It happens that a mobile thing becomes temporarily static for reasons like running out of resources before resuming its planned/unplanned roaming and suspending roaming until some conditions are met.

Forms of interactions. We identify 2 forms of interactions labelled as either horizontal, occurring between homogeneous peers, or vertical, occurring between heterogeneous peers. A peer is either a thing, an edge, or a cloud. Interactions are a mix of bottom-up from things to edges then clouds conveying data, and top-down from clouds to edges then things conveying commands. These interactions permit to form coalitions of peers to handle complex users' demands and offload demands from one peer to another and even from one coalition to another.

Availabilities of resources. We consider availabilities of resources since they impact the deployment and execution of microservices on things, edges, and/or clouds. Some resources are limited like storage while others are (temporarily) non-shareable like data. Building upon our previous work on resource management (Baker et al., 2018), we associate resource availabilities with 5 consumption properties referred to as unlimited, shareable, limited (the consumption of a resource is restricted to a particular capacity and/or period of time), limited-but-renewable (the consumption of a resource continues to happen since the (initial) agreed-upon capacity has been increased and/or the (initial) agreed-upon period of time has been extended), and non-shareable (the concurrent consumption of a resource must be coordinated - e.g., one at a time -). Unless stated, a resource is by default unlimited and/or shareable.

Deploying microservices would be either orches-

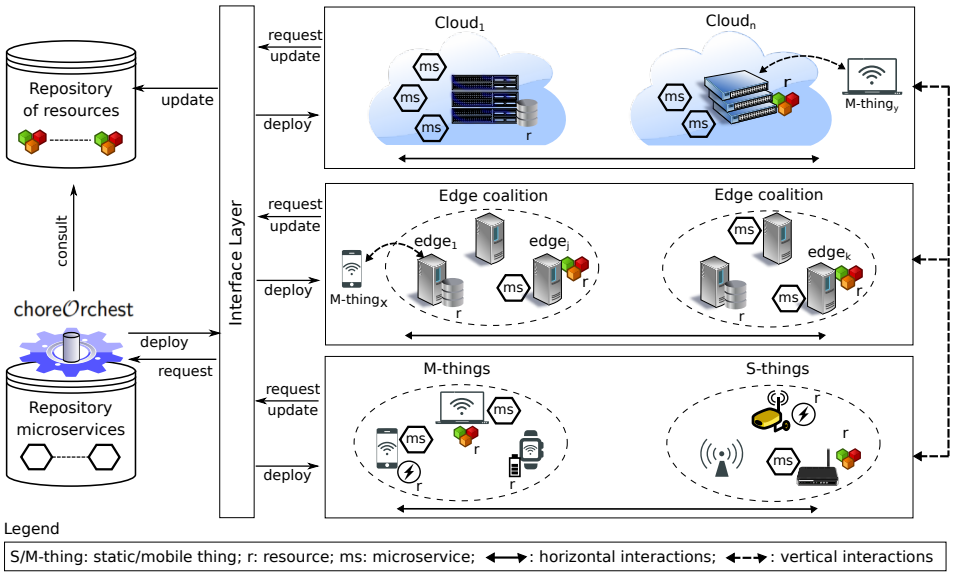

Figure 1: choreOrchest in-action.

trated or choreographed. Orchestration would rely on a centralized component that would decide on where microservices would be deployed. Contrarily, choreography could rely on peer-to-peer interactions to let communicating platforms decide on where microservices would be deployed. To cater for the needs of our cloud, edge, and IoT ecosystem, we mix choreography and orchestration into a dedicated component that we refer to as choreOrchest. Fig. 1 illustrates how choreOrchest, running on top of the pool of microservices, is responsible for selecting those that will be deployed on the available platforms (clouds, edges, static things, and/or mobile things). choreOrchest is aware of the platforms' resource availabilities along with the microservices' needs of resources. These availabilities are reported in the repository of resources that all platforms regularly update. Once choreOrchest consults the repository of resources and pool of microservices, it discovers relevant platforms that match with microservices. This matching aims at maximizing the income of each platform (i.e., total sum of incomes for resources that a platform secures when hosting a microservices for deployment and execution) under the constraint that deployment and execution of microservices must not exceed a certain budget. Upon matching completion, choreOrchest selects on which platforms the microservices will be deployed and then requests the interface layer to deploy and track them. In conjunction with the orchestration-based deployment, it happens during execution that the platforms would offload some of the hosted microservices to other peers. Offloading supposes that each platform maintains a vicinity list containing those collaborative peers that would be willing to support this platform in hosting some microservices in return of a fee. These offloading opportunities that choreOrchest may not be aware of could permit to ensure that next microservices or a group of microservices are executed in the same platforms to avoid for instance, unnecessary data transfer between platforms. It happens that a microservice would finish the execution earlier than expected and that a platform would secure additional resources through virtualization. As a result the platforms request from the interface layer to interact with the choreOrchest that would pull relevant microservices from the pool for deployment through the interface layer again. For more details, readers are invited to consult (Maamar and Faci, 2021).

4 CONTRACT MANAGEMENT

This section introduces the concepts underpinning contract management for microservices deployment and execution. First, types of contracts with identified QoS non-functional properties are discussed, and then interactions between contracts as well as contracts' lifecycles are presented.

4.1 Types of Contracts

Fig. 2 illustrates the 3 types of contracts that we deem necessary for managing microservices deployment and execution over thing, edge, and cloud platforms. These contracts are discovery, deployment, and collaboration. The figure also illustrates how the

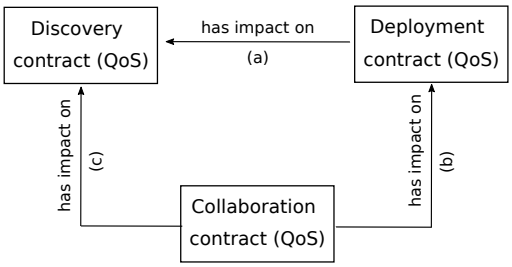

Figure 2: Types of contracts.

contracts impact each other. Indeed, the value of a QoS non-functional property in a contract is changed because of a counterpart property in another contract. In the following, each contract is defined along with its QoS non-functional properties.

Discovery contract is established between microservices, a third party (e.g., broker), and potential hosting platforms.The following QoS non-functional properties could populate a discovery contract (Fig. 3 as example):

- Discovery time that the third party would need to connect microservices to platforms.
- Discovery quality that microservices and platforms would each expect from the third party. Because of microservices' and platforms' separate expectations, we specialize discovery quality into dQuality_{ms} for the microservices that target a certain hosting level like reliability by the platforms and dQuality_{pl} for the platforms that target a certain execution time of the hosted microservices. Since discovery quality is assessed after the hosting of microservices over platforms occurs effectively, the deployment contract's QoS non-functional properties (discussed next) will impact the discovery contract as per Fig. 2 (a). Any deviation from a discovery contract's agreed-upon clause like drop in execution time due to a poor hosting level should be communicated the third party that could consider this deviation when recommending potential platforms to microservices in the future.

Deployment contract is established between microservices and confirmed platforms upon the third party's recommendations as stated in the discovery contract description. The following QoS non-functional properties could populate a deployment contract (Fig. 4as example):

- Deployment time that a platform would need to have a microservice ready for execution, should the microservice require a particular set-up.
- Execution time that a platform would need to have a microservice executed.

```
"name" : "DiscoveryContract",
"id": 1,
"partners" : [
    {"FirstParty" : "Broker"}, {"SecondParty" : "Microservice ms"},
    {"ThirdParty": "Platform p"}
],
"terms" : [
    {
        "qosMetric": "DiscoveryTime",
        "constraint" : "less than",
        "value" : 20,
        "unit " : "minute"
    },
    {
        "qosMetric": "dQuality_ms",
        "constraint" : "more than",
        "value" : 20,
        "unit " : null
    },
    {
        "qosMetric": "dQuality_pl",
        "constraint" : "less than",
        "value" : 180,
        "unit " : "second"
    }
]
```

Figure 3: Example of JSON instantiated discovery-contract.

- Hosting level that a microservice would require without degrading its performance nor the performance of the platform. A platform's hosting level could be related to its capacity of performing without failures nor interruptions over a period of time, and would depend on its technical capabilities.
- Delegation quality that a platform would use to make a microservice aware of the (positive/negative) offloading impact on this microservice's deployment time and execution time. Like with the the discovery contract's discovery-quality property, we specialize delegation quality into eQuality_{ms} for the microservices that end-up executed on different platforms and eQuality_{pl} for the platforms that receive microservices for hosting upon the requests of other platforms. Since delegation quality is assessed after the offloading of microservices occurs effectively, the collaboration contract's QoS non-functional properties will impact the deployment contract as per Fig. 2 (b). Any deviation from a deployment contract's agreed-upon clause like increase/decrease in an offloaded microservice's hosting level due to

```
"name" : "DeploymentContract",
"id": 2,
"partners" : [
    {"FirstParty" : "Platform p"}, {"SecondParty" : "Microservice ms"}],
"terms" : [
    { "qosMetric": "DeploymentTime",
      "constraint" : "less than", "value" : 3, "unit " : "minute"
    },
    {  "qosMetric": "ExecutionTime",
       "constraint" : "less than", "value" : 24, "unit " : "minute"
    },
    {  "qosMetric": "HostingLevel",
       "constraint" : "more than", "value" : 90, "unit " : null
    },
    {  "qosMetric": "DelegationQuality",
       "terms" : [
           { "ThirdParty" : "Platform p2",
              "depTimeImpact" : "positif", "depTimeImpactValue" : 1, "depTimeImpactUnit " : "minute",
              "execTimeImpact" : "negatif", "execTimeImpactValue" : 2, "execTimeImpactUnit " : "minute"
           },
           { "ThirdParty" : "Platform p3",
              "depTimeImpact" : "negatif", "depTimeImpactValue" : 5, "depTimeImpactUnit " : "minute",
              "execTimeImpact" : "positif", "execTimeImpactValue" : 3, "execTimeImpactUnit " : "minute"
           }
       ]
    },
    {  "qosMetric": "eQualityms",
       "value" : null
    },
    { "qosMetric": "eQualitypl",
      "value" : null
    }
]
```


Figure 4: Example of JSON instantiated deployment-contract.

a better/worse platform should be communicated to this microservice's owner so, that, he decides in the future on accepting/rejecting offloading demands.

Collaboration contract is established between either homogeneous peers like things-things, edges-edges, and clouds-clouds or heterogeneous peers like things-edges, things-clouds, and edges-clouds. The following QoS non-functional properties could populate a collaboration contract (Fig. 5 as example):

- Offloading time that a platform would need to transfer a microservice to another platform for deployment and execution.
- Offloading quality that a microservice would use to report its experience of being deployed and executed on a different platform from the one that is reported in the discovery contract. This experience refers to deployment-time and execution-time properties that should be benchmarked to the same properties in the deployment contract. Along with this experience, offloading quality is shared with the third party involved in the discovery so, that, future recommendations of platforms to host microservices could be adjusted, whether the quality turns out positive or negative. This means that a collaboration contract's QoS non-functional properties will impact the discovery contract as per Fig. 2 (c).
- Collaboration quality that a platform would use to decide in the future on offloading microservices to other platforms. Collaboration quality should be benchmarked to the deployment contract's delegation-quality property as per Fig. 2 (b). Like with both the discovery contract's discovery-quality property and the deployment contract's delegation-quality property, we specialize collaboration quality into $\text{cRecommendingQuality}_{pl}$ for the platform recommending a peer and $\text{cRecommendedQuality}_{pl}$ for the platform that is recommended by a peer and is dependent on the deployment contract's eQuality_{ms}.


```
"name" : "CollaborationContract",
"id": 3,
"partners" : [
    {"FirstParty" : "Platform p1"}, {"SecondParty" : "Platform p2"},
    {"ThirdParty" : "Microservice ms"}
],
"terms" : [
    {
        "qosMetric": "offloadingTime",
        "constraint" : "less than",
        "value" : 5,
        "unit " : "minute"
    },
    {
        "qosMetric": "offloadingQuality",
        "value" : null
    },
    {
        "qosMetric": "cRecommendingQualitypl",
        "value" : "high"
    },
    {
        "qosMetric": "cRecommendedQualitypl",
        "value" : "high"
    }
]
```


Figure 5: Example of JSON instantiated collaboration-contract.

4.2 Interactions between Contracts

In Fig. 2, interactions (a), (b), and (c) capture the impacts that some contracts could have on each other. Indeed, the deployment contract impacts the discovery contract and the collaboration contract impacts both the discovery contract and the deployment contract. By impact, we mean the completion of a contract at run-time leads into results that would be integrated into the preparation of another contract or updating an existing one.

1. **from:** Deployment Contract **to:** Discovery Contract. On the one hand, the satisfaction level (abstracting deploying-time and execution-time properties) of a microservice towards a platform upon which it has been deployed (i.e., dQuality_{ms}) needs to be reported to the third party so, that, future discovery cases that could involve this platform could be handled differently. On the other hand, the satisfaction level (abstracting hosting-level property) of a platform towards the microservices it has received for hosting (i.e., dQuality_{pl}) needs to be reported to the third party so, that, future discovery cases that could involve these microservices would be handled differently. Thanks to details reported to the third party, this one does not rely on what microservices and platforms announce in terms of technical requirements and capabilities, respectively. But, the third party also relies on what happens at run-time when completing deployment contracts.
2. **from:** Collaboration Contract **to:** Deployment Contract.] On the one hand, the satisfaction level (abstracting eQuality_{ms} property) of a microservice towards a new platform, that is different from the initial platform reported in the discovery contract, should be reported to this initial platform so, that, future delegation cases that could involve this new platform would be handled differently. On the other hand, the satisfaction level (abstracting eQuality_{pl} property) of a platform towards the microservices it has received for hosting upon the request of the initial platform reported in the discovery contract should be reported to this initial platform so, that, future delegation cases that could involve these microservices could be handled differently. Thanks to details reported to the initial platform, this one does not rely on what microservices and other platforms announce in terms of technical requirements and capabilities, respectively. But, the initial platform also relies on what happens at run-time when implementing collaboration contracts.
3. **from:** Collaboration Contract **to:** Discovery Contract. The satisfaction level (abstracting offloading-quality property) of a microservice towards a new platform, that is different from the one identified during the discovery, needs to be reported to the third party so, that, future discovery cases that could involve the first platform that recommends this new platform would be handled differently.

4.3 Lifecycles of Contracts

To track and manage different aspects of contracts like performance and compliance, we define their lifecycles represented as a state diagram (Fig. 6). States include *initiated* (initial state), *revised* (entry-point state), *performed* (initial state), *completed* (final state), *canceled* (final state), and *suspended* (final state) and are connected together forming Sequences (Seq_i). Prior to listing these sequences, we recall that a contract could be subject to changes(e.g., a new platform is assigned) and/or could run into obstacles (e.g., late confirmation of a delegation request) that both could result into either canceling or suspending the contract. For the sake of simplicity, sequences having *performed* as an initial state are not listed below since they are already parts of the sequences having *initiated* as an initial state.

1. $Seq_1 = initiated \xrightarrow{start} performed \xrightarrow{success} completed$. A contract is completed successfully without being subject to any changes nor running into any obstacles.
2. $Seq_2 = initiated \xrightarrow{change} revised \xrightarrow{approval} performed \xrightarrow{success} completed$. A contract is completed successfully after being subject to some changes and not running into any obstacles.
3. $Seq_3 = initiated \xrightarrow{start} performed \xrightarrow{failure} canceled$. A contract is canceled although it was not subject to any changes but has run into some obstacles.
4. $Seq_4 = initiated \xrightarrow{change} revised \xrightarrow{approval} performed \xrightarrow{failure} canceled$. A contract is canceled because it has been subject to some changes and has run into some obstacles.
5. $Seq_5 = initiated \xrightarrow{start} performed \xrightarrow{violation} suspended$. A contract is suspended without being subject to any changes but has run into obstacles that resulted into its violation and hence, suspension.
6. $Seq_6 = initiated \xrightarrow{change} revised \xrightarrow{approval} performed \xrightarrow{violation} suspended$. A contract is suspended after being subject to some changes and running into

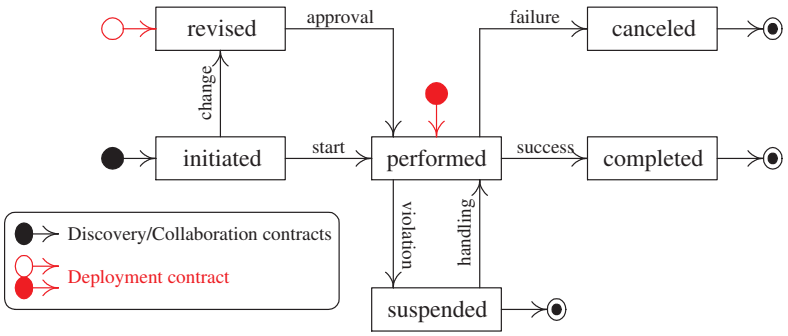

Figure 6: Contract's lifecycle as a state diagram.

some obstacles. Both have resulted into its violation and hence, suspension.

7. $Seq_7 = initiated \xrightarrow{start} performed \xrightarrow{violation} suspended \xrightarrow{handling} performed \xrightarrow{success} completed$. A contract is completed successfully without being subject to any changes but has run into some obstacles that have been handled, which has allowed its successful completion.

8. $Seq_8 = initiated \xrightarrow{change} revised \xrightarrow{approval} performed \xrightarrow{violation} suspended \xrightarrow{handling} performed \xrightarrow{success} completed$. A contract is completed successfully after being subject to some changes and running into some obstacles that have been handled, which has allowed its successful completion.

9. $Seq_9 = initiated \xrightarrow{start} performed \xrightarrow{violation} suspended \xrightarrow{handling} performed \xrightarrow{failure} canceled$. A contract is canceled without being subject to any changes but has run into some obstacles that although these obstacles have been handled the contract has been canceled.

10. $Seq_{10} = initiated \xrightarrow{change} revised \xrightarrow{approval} performed \xrightarrow{violation} suspended \xrightarrow{handling} performed \xrightarrow{failure} canceled$. A contract is canceled after being subject to some changes and running into some obstacles that although these obstacles have been handled the contract has been canceled.

After listing a contract's different sequences of states, we proceed, hereafter, with identifying the relevant lifecycle per type of contract. Each lifecycle draws the necessary states from these sequences along with the option of dropping some states because of the nature of each contract.

4.3.1 Discovery Contract's Lifecycle

A discovery contract formalizes the first steps that lead to identifying the platforms upon which the microservices will be deployed and then, executed. This contract's lifecycle takes on the following states:

- In the *initiated* state, the discovery contract's necessary attributes are instantiated in terms of which microservice needs hosting, which potential platforms are contacted to provide this hosting, and which third party drives the interactions between the microservice and platforms. Additional attributes to instantiate could include deadlines to complete the interactions and particular criteria to shortlist the platforms.
- In the *revised* state, the discovery contract's instantiated attributes could be adjusted, should some changes blacklike criteria for shortlisting platforms and/or technical details for hosting microservices arise.
- In the *performed* state, the discovery contract is implemented by making the third party match the microservice to the adequate platform.
- In the *suspended* state, the under-performing discovery contract could run into obstacles such as violating the deadline to identify a platform for a microservice. Should these obstacles end-up being handled properly, then the completion of the discovery contract would resume. Otherwise, it would be stopped.
- In either *completed* or *canceled* state, the under-performing discovery contract ends with either success or failure, respectively, depending on the outcome of assigning a microservice to a platform. A successful completion of the discovery contract triggers the instantiation of the deployment contract's necessary attributes.

4.3.2 Deployment Contract's Lifecycle

Once a discovery contract completes with success, a deployment contract is drawn between the microservice and platform. This contract's lifecycle takes on the following states:

- In the *performed* state, the deployment contract is implemented by making the microservice run over the platform.
- In the *revised* state, the instantiated attributes of the already-prepared deployment contract (outcome of the discovery contract) could be adjusted, should some changes arise impacting the deployment of the microservice on the platform. An example of change could be deploying the microservice

  on a different platform from the one that is mentioned in the discovery contract. Should this change happen, a collaboration contract would need to be prepared.
- In the *suspended* state, the under-performing deployment contract is running into obstacles such as violating the agreed-upon deployment time and execution time. Should these obstacles end-up being handled, then the completion of the deployment contract would resume. Otherwise, it would be stopped.
- In either *completed* or *canceled* state, the under-performing deployment contract ends with either success or failure depending on the outcome of having the microservice run over the platform.

In the afore-mentioned states, it is worth noting the absence of *initiated* state making *performed* the initial state and *revised* an entry-point state for the deployment contract.

4.3.3 Collaboration Contract's Lifecycle

A collaboration contract formalizes the process of transferring a microservice from a platform to another prior to its execution. This contract's lifecycle takes on the following states:

- In the *initiated* state, the collaboration contract's necessary attributes are instantiated in terms of which microservice will be transferred to which platform and which platform is recommending the transfer.
- In the *revised* state, the collaboration contract's instantiated attributes could be adjusted, should some changes arise impacting for instance, the recommended platform for hosting the microservice.
- In the *performed* state, the collaboration contract is implemented through the effective transfer of the microservice to the recommended platform.
- In the *suspended* state, the under-performing collaboration contract is running into obstacles such as violating the agreed-upon offloading time and offloading quality time. Should these obstacles be handled, then the completion of the collaboration contract would resume. Otherwise, it would be stopped.
- In either *completed* or *canceled* state, the under-performing collaboration ends with either success or failure depending on the outcome of having the microservice transferred to the recommended platform.

5 IMPLEMENTATION AND EVALUATION

This section first describes the architecture of the system for managing contracts and then, the experiments that were carried out.

5.1 Implementation

Fig. 7 is the architecture of the system we developed in Java 1.8 on a Windows 10, Intel Core i5-8300H processor, 16 GB RAM, GPU Nvidia GTX 1050 4GB desktop. The system consists of 3 repositories (platforms, microservices, and contracts), 3 managers (contract, monitoring, and execution), and one *log* file. In this figure, pr, ex, and po stand for pre-execution, execution and post-execution stages, respectively, and arrowed lines correspond to interactions between all the repositories, managers, and *log*.

During the pre-execution stage, the contract manager prepares all types of contracts (discovery, deployment, and collaboration) based on first, microservices' technical requirements and platforms' technical capabilities and second, these platforms' ongoing/changing loads. During this stage, different values are assigned to the QoS non-functional properties according to their roles in finalizing the contracts. For instance, Discovery time, dQuality_{ms}, dQuality_{pl}, and Deployment time are assigned random values according to a specific range, e.g., $[5, 10]$, while eQuality_{ms}, eQuality_{pl}, and Offloading quality are assigned *null*, and properties $\text{cRecommendingQuality}_{pl}$ and $\text{cRecommendedQuality}_{pl}$ are assigned *high*. To address the cold-start concern, we assumed that, at initialization time, all platforms trust each other confirming the high level of collaboration between them.

During the execution stage, the execution manager consults the repository of contracts to deploy the microservices and then, proceeds with tracking their execution and potentially offloading some to other platforms along with measuring the effective values

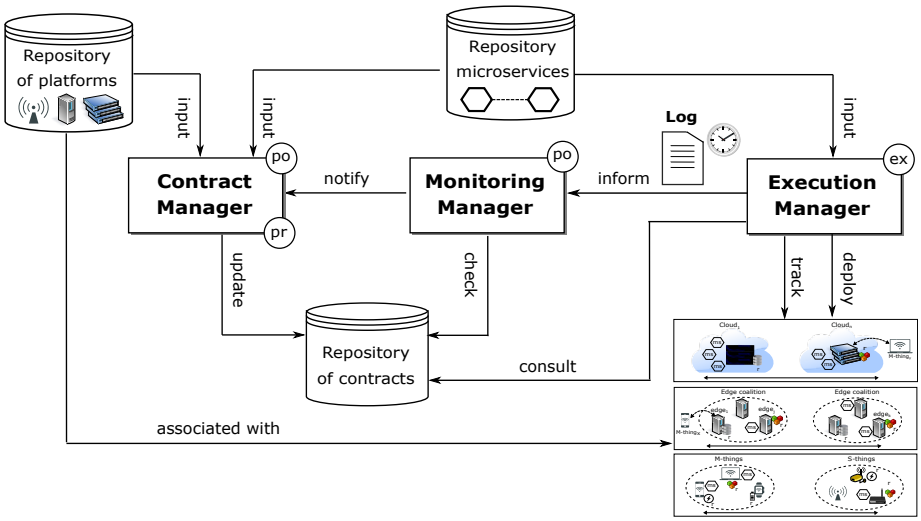

Figure 7: Architecture of the contract-management system.

of relevant QoS non-functional properties like Discovery time, Deployment time, and Execution time. These values are stored in the *log* that the monitoring manager uses during the post-execution stage for benchmarking against the corresponding values in the deployment contracts.

Should there be any discrepancy according to some thresholds, the monitoring manager would notify the contract manager that would flag a contract as either suspended or canceled in compliance with this contract's lifecycle (Fig. 6). Otherwise, the contract manager flags the contract as completed in compliance again with this contract's lifecycle.

5.2 Evaluation

Experiment Setup. Due to the limited availability of real datasets that could satisfy our technical needs and requirements, we resorted to creating 2 datasets (*d1* and *d2*) during the pre-execution stage and using an existing dataset (*d3*) that was obtained in a previous work (Maamar and Faci, 2021). Table 1 reports details about each dataset in terms of number of microservices, number of platforms, and QoS values whether real or generated.

Results and Discussions. We conducted a series of experiments to compute the average execution time of the monitoring and contract managers. Each experiment was executed 10 times. The first series of experiments were applied to *d1* evaluating the impact of incrementing by 100 the number of microservices from 50 to 1050 on the average execution time of both managers. Table 2 shows that the average execution time increases exponentially with the number of microservices. However, even with a large number of microservices, the achieved performance remains acceptable.

In the second series of experiments that were applied to *d2*, we evaluated the impact of incrementing by 50 the number of platforms from 50 to 500 on the average execution time of both managers. As expected, the average execution time increases exponentially with the number of platforms as per Table 3. However, even with a large number of platforms, we still achieve an acceptable performance.

To conclude the series of experiments, we conducted two more. The first one was applied to *d2* with exactly 30 platforms resulting into an average execution time of 187,4 ms for both managers. Finally, the second one was applied to *d3* checking the validity of the previous experiment's results. The average execution time of both managers is 184,4 ms which is in line with these results.

6 CONCLUSION

This paper presented an approach for regulating microservices' deployment and execution over platforms using contracts. On the one hand, the platforms are specialized into IoT devices, edges, and clouds. On the other hand, contracts are specialized into discovery, deployment, and collaboration defining who has done what, when, where, and for what purpose. For instance, discovery contracts regulated the "deals" between microservices' owners and platforms' providers while collaboration contracts regulated the "deals" between platforms offloading their micorservices to other platforms. Contracts were also

Table 1: Details about the datasets of the experiments.

dataset	# of microservices	# of platforms	real QoS values	generated QoS values
d1	1050	30	–	discovery time, deployment time, execution time, hosting level, offloading time
d2	20	510	–	discovery time, deployment time, execution time, hosting level, offloading time
d3	15	30	deployment time, offloading time	discovery time, execution time, hosting level

Table 2: Impact of number of microservices on the monitoring and contract managers.

# of microservices	average execution time in ms
50	709,1
150	2587,3
250	5567,0
350	7837,3
450	11344,7
550	16044,2
650	20823,4
750	27248,6
850	34761,7
950	42227,3
1050	49351,0

Table 3: Impact of number of platforms on the monitoring and contract managers.

# of platforms	average execution time in ms
50	479,7
100	2087
150	4998
200	10116,6
250	15316,4
300	22947,9
350	31323,8
400	39518,9
450	55676,8
500	69392,2

associated with lifecycles allowing to monitor their progress towards either successful completion or failure. To demonstrate the technical doability of the contract management approach, a system's architecture was first, designed identifying the necessary repositories and managers and then, implemented in Java. The system supported different experiments examining for instance, the impact of increasing the number of microservices on the performance of the system's managers.

In term of future work, we would like to examine contract monitoring and enforcement. On the one hand, the former would track contract completion from creation until expiry going through review and execution. How to assess the performance of contracts and how to proceed with their renewals without impacting ongoing contracts are some monitoring-related questions that need to be addressed. To this end, we plan to develop techniques that could help for instance, identify reasons of renewals, recommend amendments to expedite renewals, and predict these amendments' (financial) impacts on stakeholders. On the other hand, the latter, contract enforcement, would ensure the full compliance of all stakeholders with contracts' clauses. How to build trust among the stakeholders for long-term collaboration is a question that needs to be addressed. To this end, we plan to develop techniques that could foster trust and incentive/penalize good/bad behaviors of stakeholders linked to contracts.

REFERENCES

Baker, T., Ugljanin, E., Faci, N., Sellami, M., Maamar, Z., and Kajan, E. (2018). Everything as a Resource: Foundations and Illustration through Internet-of-Things. *Computers in Industry*, 94.

Balint, F. and Truong, H. (2017). On Supporting Contract-aware IoT Dataspace Services. In *IEEE Int. Conf. on Mobile Cloud Computing, Services, and Engineering*, San Francisco, USA.

Brito, M., Cunha, J., and de Sousa Saraiva, J. (2021). Identification of Microservices from Monolithic Applications through Topic Modelling. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, Republic of Korea (virtual event).

Butzin, B., Golatowski, F., and Timmermann, D. (2016). Microservices Approach for the Internet of Things. In *Proceedings of the 21st IEEE Int. Conf. on Emerging Technologies and Factory Automation*, Berlin, Germany.

De Donno, M., Tange, K., and Dragoni, N. (2019). Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. *IEEE Access*, 7.

Khebbeb, K., Hameurlain, N., and Belalab, F. (November 2020). A Maude-based Rewriting Approach to

Model and Verify Cloud/Fog Self-Adaptation and Orchestration. *Journal of Systems Architecture*, 110.

Lewis, J. and Fowler, M. (2014). Microservices.

Longo, A., Zappatore, M., and Bochicchio, M. (2019). A Cloud-based Approach to Dynamically Manage Service Contracts for Local Public Transportation. *International Journal of Grid and Utility Computing*, 10(6).

Maamar, Z. and Faci, N. (2021). Microservices Deployment and Execution in a Cloud, Edge, and IoT Configuration. Technical report, Zayed University.

Marino, F., Moiso, C., and Petracca, M. (2019). Automatic Contract Negotiation, Service Discovery and Mutual Authentication Solutions: A Survey on the Enabling Technologies of the Forthcoming IoT Ecosystems. *Comput. Networks*, 148.

Menascé, D. A. (2002). QoS Issues in Web Services. *IEEE Internet Computing*, 6(6).

Pan, J., Wang, J., Hester, A., AlQerm, I., Liu, Y., and Zhao, Y. (2019). EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts. *IEEE Internet Things J.*, 6(3).

Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., and Rana, O. (2019). Fog Computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology*, 19(2).

Singh, S. (2017). Optimize cloud computations using edge computing. In *IEEE International Conference on Big Data, IoT and Data Science (BID)*.

Singh, S., Chana, I., and Buyya, R. (2020). STAR: SLA-aware Autonomic Management of Cloud Resources. *IEEE Transactions on Cloud Computing*, 8(4).

Sun, H., Yu, H., and Fan, G. (2020). Contract-Based Resource Sharing for Time Effective Task Scheduling in Fog-Cloud Environment. *IEEE Transactions on Network and Service Management*, 17(2).

Truong, H. and Klein, P. (2020). DevOps Contract for Assuring Execution of IoT Microservices in the Edge. *Internet Things*, 9.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS