Journal of Network and Systems Management (2022) 30:34
https://doi.org/10.1007/s10922-022-09641-z

CNS-AOM: Design, Implementation and Integration of an Architecture for Orchestration and Management of Cloud-Network Slices

André Luiz Beltrami Rocha[1] · Celso Henrique Cesila[2] · Paulo Ditarso Maciel Jr.[3] · Sand Luz Correa[4] · Javier Rubio-Loyola[5] · Christian Esteve Rothenberg[2] · Fábio Luciano Verdi[1]

Received: 29 January 2021 / Revised: 14 September 2021 / Accepted: 2 January 2022

Abstract

Cloud-Network Slice (CNS) is defined as an end-to-end infrastructure composed by computing, networking, and storage resources and it is expected to be a key enabler for novel verticals such as Industry 4.0, IoT and Vehicular Networks. This paper presents the design, implementation and integration of the Architecture for Orchestration and Management of Cloud-Network Slices (CNS-AOM), a modular architecture to orchestrate and manage slice resources and services in CNSs. The CNS-AOM is designed and implemented considering three important characteristics: (i) the business model called Slice-as-a-Service (SlaaS); (ii) the multiple administrative and technological domains; and (iii) the slice elasticity, which means the capacity of dynamically growing and shrinking the slice resources to improve service performance. To prove the feasibility of our proposal, two Proofs of Concept (PoC) are implemented in real environments to validate the CNS-AOM. First, an end-to-end content distribution service (CDN) is deployed across three different cities in Brazil to emphasize the multiple domains. Second, we present an IoT service using a fully-featured commercial service platform called dojot, which is instantiated and orchestrated by the proposed architecture. The dojot slice is instantiated overseas in four cities across two countries. The evaluation for the CDN slice considers the appropriate metrics that should be monitored and the actual services that should be instantiated to meet the end-user's requirements depending on its location. Moreover, in the dojot slice, elasticity operations (vertical and horizontal) are tested and evaluated along with the time taken to deploy the slice infrastructure and the service. The main contributions of this paper are: (i) the design, implementation and integration of the CNS-AOM; (ii) the orchestration control-loop of the slice resources; and (iii) the execution of real proof-of-concept scenarios that demonstrate the feasibility of

Extended author information available on the last page of the article

Published online: 30 January 2022

the CNS-AOM to instantiate and orchestrate services across geographically-distanced cities.

Keywords Cloud-network slicing · Cloud-network slice monitoring and management · Cloud-network slice orchestration · NECOS project · Slice-as-a-service · Orchestration closed-loop · Cloud-network slice elasticity

1 Introduction

The *cloud-network slicing* concept [1, 2] emerges from the need to dynamically provide end-to-end infrastructures supporting different services and verticals composed of heterogeneous resources such as computing, networking, and storage. The main driver behind this concept is to support those verticals across different slice parts representing multiple administrative and technological domains. In the Slice-as-a-Service (SlaaS) business model, a tenant makes a request to the slice provider so that a slice is created. Once the slice is instantiated, the tenant must be able to access, configure and manage the slice resources (physical or virtual), and also must be able to perform the service deployment in this Cloud-Network Slice (CNS). These operations are isolated for each CNS, even when subletting those resources and services to other tenants.

With the recent technological expansion of mobile networks (e.g., 5G), Internet of Things (IoT), and smart city deployments, several entities and projects have been considering the slicing concept as a potential enabler of these new technologies. Commonly in the literature [3–8], the concept of network slicing refers to a shared infrastructure composed by only network elements, disregarding other types of resources such as computing.

In the scope of this work, a cloud-network slicing is being formed by computing, networking and storage resources [2]. Considerable efforts have been made to define and standardize architectures that provide *slices or network slices* [9]. However, important operations such as slice resource monitoring, management, and orchestration still have several important challenges that need to be addressed. According to recent works [9–12], some of the most relevant open issues in the context of CNSs are the Slice-as-a-Service (SlaaS) business model, services lifecycle management, CNS orchestration, slice elasticity, and monitoring.

Despite all the efforts to create standards by entities such as 3rd Generation Partnership Project (3GPP) [13], European Telecommunications Standards Institute (ETSI) [14], and Next Generation Mobile Networks (NGMN) [15], there is still no specific proposal to deal with CNS orchestration control-loop. Under current approaches, the orchestration of slice resources cannot be efficiently performed, as they are not capable of managing and monitoring different resources that are allocated across multiple administrative and technological domains. The Architecture for Orchestration and Management of CNSs (CNS-AOM) was designed with this

purpose as part of an international project, called NECOS (Novel Enablers for Cloud Slicing)[1], a collaboration between Brazil and the European Union (EU). The objective of the project was to study and present a slice provider platform responsible for the life-cycle of the CNS; namely, from the CNS provisioning to the slice resource and service management and orchestration, up to the slice decommission. In this paper, we focus on the post-CNS provision. More details about the provisioning phase and other NECOS components can be found in [2].

To the best of our knowledge, the design and implementation of the CNS-AOM is the first proposal intended to address the aforementioned challenges holistically. In this paper, we present the design, implementation, integration and evaluation of the CNS-AOM. The architecture was designed considering state-of-the art challenges, and focusing on slice elasticity, a key feature in the context of CNSs that has not been sufficiently explored in the literature.

Elasticity in CNSs refers to the ability to increase or decrease resources allocated in order to improve service performance or optimize the use of resources. For CNSs, two types of elasticity have been studied, vertical and horizontal elasticity, as presented in [16]. In brief, vertical elasticity considers the addition/removal of resources (hosts, network elements, etc.) in a slice part that is already allocated to the slice. By contrast, horizontal elasticity represents the addition/removal of an entire administrative domain (slice part). Elasticity operations are detailed by a tenant defining policies that will be interpreted by an orchestrator capable of periodically checking the metrics being monitored. When a policy is violated, the orchestrator initiates an elasticity operation. An elasticity operation changes the slice infrastructure leading to readjustments in services as well as the monitoring and management of subsystems. Therefore, the main responsibilities of CNS-AOM are as follows: (i) deployment of the management and monitoring components after slice provisioning; (ii) instantiating services; (iii) reflecting the slice elasticity operations in the services, as well as in the management and monitoring subsystems; (iv) monitoring the infrastructure and services; and (v) managing the services and virtual resources of a CNS.

The PoCs presented in this paper are a key aspect since they show the complexity and challenges introduced by providing and orchestrating services across multiple administrative and technological domains. The PoCs are designed considering the geographical and technological heterogeneity and demonstrate the challenges explored by the CNS-AOM, such as the CNS orchestration control-loop in real scenarios. We evaluate the proposal by presenting two PoCs. The first PoC is a content distribution (CDN) service running on a CNS located in three different cities in São Paulo state, Brazil. The purpose of this service is to deploy edges dynamically according to the end-user's geographic location based on the subnet IP address. The second PoC shows the orchestration and management of a complex IoT service platform called dojot. The results obtained for the dojot slice are from a detailed analysis of the elasticity operations being carried out in a real environment implemented in different countries. One of the most important aspects to be highlighted is the CNS-AOM capacity to instantiate and manage (including monitoring) services, even

[1] http://www.h2020-necos.eu/.

across different cities or countries, always considering technological heterogeneity and slice elasticity.

The remainder of this paper is organized as follows. Section 2 presents a literature review. Section 3 describes the CNS-AOM architecture, specifically discussing the management, monitoring, and orchestration functionalities. Section 4 details the components implementation and shows two workflows representing the main methods implemented. Section 5 presents the CDN and *dojot* PoCs used for validating the architecture. Finally, Sect. 6 concludes our paper, presenting the final remarks and possible future work.

2 Background and Related Work

CNS management involves managing the lifecycle of slice resources and services. To this purpose, slice monitoring is of paramount importance for the efficient orchestration of the requested resources and services by providers and tenants. Several works address such aspects in a variety of contexts. However, to the best of our knowledge, no previous work has tackled these areas in the context of CNSs. We divide this section into three subsections. First, we present the efforts of EU projects and the initiatives from different standardization bodies. Then, we show the main characteristics of the NECOS project. Finally, the last subsection comprises a discussion of papers related to CNS-AOM.

2.1 Projects and Initiatives

This subsection summarizes a set of relevant network slicing standardization efforts and research projects. We present a qualitative comparison of selected projects considering the following (non-exhaustive) list of seven key characteristics:

1. E2E - the end-to-end connectivity of the slice resources[2] to provide services;
2. Multi-domain - the support of multiple administrative and/or technological domains;
3. Slice lifecycle management - the slice realizes lifecycle management of resources;
4. Tenant slice management - the tenant manages the slice resources;
5. SlaaS - the slice is provided as a service, similar to the business models for Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), etc;
6. Virtual Infrastructure Manager (VIM)/WAN Infrastructure Manager (WIM) on demand - the tenant is able to request specific VIMs or WIMs on demand for each slice part;

[2] Slice resources mean the slice parts, i.e, the components that make part of the infrastructure of the slice. Features such traffic steering, load balancing, caching and others may be deployed by the tenant and are beyond the scope of the slice resources.

Table 1 Summary of key characteristics for slicing initiatives, European projects and the NECOS project

Initiatives/ EU Projects	E2E	Multi-domain	Slice lifecycle management	Tenant slice management	Slice as a Service (SlaaS)	VIM/WIM on demand	Marketplace for slices
ITU-T [21, 22]	×	×	×	×	×	×	×
NGMN [23, 24]	×	×	×	×	×	×	×
IETF [3, 4, 18, 25–31]	√	√	√	√	×	×	×
3GPP [5, 32–37]	×	×	√	√	×	×	×
ETSI [6, 7, 38]	√	×	×	√	×	×	×
ONF [39]	×	×	×	×	×	×	×
5G-EX	×	√	√	√	√	×	×
5G-SONATA	×	×	√	√	×	×	×
5G-NORMA	×	×	√	×	×	×	×
5G-Transformer	×	×	√	√	×	×	×
5G-PAGODA	×	×	√	√	×	×	×
5G-SliceNet	×	×	√	×	√	√	×
NECOS	√	√	√	√	√	√	√

Source: Adapted from NECOS deliverable 3.1.[40]

7. Marketplace for slices - the slice resources are negotiated in a marketplace composed of different infrastructure providers that offer their resources as part of a requested slice.

Table 1 relates the aforementioned seven characteristics to six slicing standardization initiatives (lines 2 to 7), seven European 5G Infrastructure Public Private Partnership (5GPPP) projects (lines 8 to 13), and the NECOS project. The positive ticks indicate that the corresponding work considers the applicable slice characteristic in its scope.

The initiatives considered in this analysis were as follows: (i) ITU Telecommunication Standardization Sector (ITU-T), (ii) NGMN, (iii) Internet Engineering Task Force (IETF), (iv) 3GPP, (v) ETSI, and (vi) Open Networking Foundation (ONF). The EU 5GPPP projects studied were as follows: (i) 5GEX[3], (ii) 5G-SONATA[4], (iii) 5G-NORMA[5], (iv) 5G-Transformer[6], (v) 5G-PAGODA[7], (vi) 5G-Slicenet[8],

[3] http://www.5gex.eu/.
[4] http://sonata-nfv.eu/.
[5] http://www.it.uc3m.es/wnl/5gnorma/.
[6] http://5g-transformer.eu/.
[7] https://5g-pagoda.aalto.fi/.
[8] https://slicenet.eu/.

and (vii) the NECOS project. Our main goal was to compare the NECOS project that originated this work with relevant slicing initiatives and similar projects. More details of the architectural survey on network slicing is available in [17].

Most projects and initiatives use the *network slicing* concept, however, in the literature we have different definitions for this term. Therefore, in the context of the NECOS project and in this article, we adopt the term cloud-network slicing as a set of networking, computing and storage resources. Currently and still in definition, IETF is the only standardization body that defines the term network slicing comprising networking, computing and storage resources [18].

The closest related project is the 5G-EX, which considered four of the key characteristics. An important aspect, the marketplace for CNSs is only considered in the NECOS project. The authors in [19, 20] present use cases in this regard. The analysis presented in Table 1 conveys a more flexible conceptualization of slicing, which goes beyond the peer-to-peer orchestration and management interactions of most of the presented initiatives. The proposed architecture, CNS-AOM, is based on this conceptualization, introducing and designing components that allow for a slicing approach that covers the highlighted characteristics, such as management, monitoring, and orchestration.

2.2 Cloud-Network Slicing and the NECOS Project

Figure 1 based on [41], shows six slices of three different verticals (V2X in red, Massive IoT in green and Critical Communications in blue). The slices on the left side are network slices and the infrastructure is composed only by network elements (switches and routers) and network functions (e.g. firewall, load balancing) [8]. On the right side, there are CNSs divided into two administrative domains (slice parts) with cloud computing, storage resources, and network elements [2, 40, 41].

The NECOS project [1, 2] was envisioned during the Fourth EU-BR Collaborative Call by the European Commission and the Brazilian National Network for Education and Research (RNP)[9]. The consortium was devoted to study, propose, and implement a platform to enable the concept of cloud-network slicing. Therefore, the objective of NECOS is to provide CNSs for tenants with features such as the automatic configuration of the infrastructure across multiple federated domains, service-independent operation, and slice adaptation to service needs. Additionally, it has to provide an autonomous platform for managing, monitoring, and orchestrating the slice resources and the internal components without depending on any action from the tenant.

In NECOS, the lightweight slice defined cloud (LSDC) architecture is proposed to showcase the concept of cloud-network slicing, under a SlaaS business model that uses readily available cloud platform features and functions. LSDC introduces a new way to automate the cloud configuration process by being able to deploy CNSs split

[9] http://www.h2020-necos.eu/.

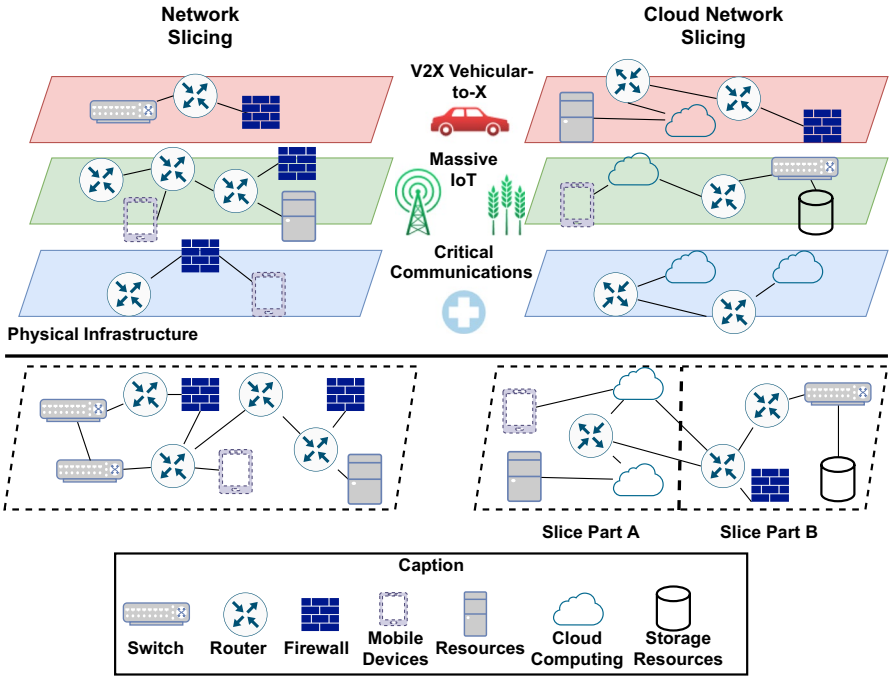


Fig. 1 Network slicing and cloud-network slicing concepts

into slice parts across all the providers as a set of federated data centers, providing uniform management of computing, network, and storage resources.

During the NECOS project, we designed and implemented a subsystem called Infrastructure and Monitoring Abstraction (IMA) responsible for the monitoring and management of services and infrastructure (physical and virtual). We also designed and implemented a minimalist version of a slicing resource orchestrator (SRO) that was responsible for orchestration operations, such as verifying the monitored metrics and testing them against tenant policies in order to trigger elasticity operations, if necessary. In this paper, we present an entirely coupled solution between IMA and SRO called CNS-AOM. The design of our architecture is divided into three subsystems: resource & service (R&S) management, R&S monitoring, and the SRO. The rest of this paper presents an in-depth discussion of the CNS-AOM and implementation, highlighting the challenges in creating a monitoring and management abstraction layer as well as the integration of these layers with the SRO.

2.3 Related Work

In the 5G-SONATA project [42], the authors describe a prototype version of the service programming and orchestration for virtualized software networks, an orchestrator that allows programmability for 5G networks. This proposed orchestrator uses an open-source monitoring model that presents several desirable features, such as

flexibility to operate on different microservice platforms, scalability, and customization. However, it does not consider a multi-domain environment and cannot label metrics or group them in ways required by business models.

Lattice[10] is an intelligent monitoring system presented by Tusa et al. [43]. This tool proposes a monitoring framework for virtualized resources, services, and network elements, geared toward cloud computing systems. The framework monitors a cloud computing environment by collecting metrics and detecting increases or decreases in traffic, with configurable setup parameters such as the interval at which metrics are collected and sent to an orchestrator. However, this framework does not support all the roles of a monitoring solution in the context of CNSs, such as the multiple technological domains and the ability to communicate with several monitoring tools.

The authors in [44] unify components developed within the 5GEX and 5G-SONATA projects to orchestrate the deployment and management of Virtual Network Functions (VNF) services using the Very Lightweight Network & Service Platform [45] (VLSP) as a VIM, allowing multi-domain orchestration of slices. Despite this being a proposal in the context of slices, we can identify the following differences between our work and the one described. (i) The architecture presented by the authors encompasses only the VLSP as a VIM; it does not deal with heterogeneity in terms of different VIMs and/or WIMs in the same end-to-end infrastructure. (ii) Our architecture considers not only the management of the infrastructure and services but also presents the monitoring of both physical and virtual resources per slice.

This paper is a natural evolution of the work presented in [16]. The previous work only includes the monitoring subsystem, presenting the design and evaluation of each sub-component. The main differences between the previous and this work are as follows: (i) the addition of the management subsystem and orchestration to the CNS-AOM, (ii) the inclusion of the service monitoring, and (iii) the presentation of two actual services instantiated across different cities and countries using the proposed architecture.

Cloud computing management has been widely explored in the literature. However, it is necessary to map these studies to the concept of CNSs, which demands the analysis of several topics, such as cloud computing, network, and storage. Two surveys found in the literature helped in the development of the proposed architecture. The first one presented in [46] focuses on the broad study area of cloud management as well as its main challenges. The authors classify eight functional areas in the field: global scheduling of virtualized resources, resource demand profiling, resource utilization estimation, resource pricing and profit maximization, local scheduling of cloud resources, application scaling and provisioning, workload management, and cloud management systems. In the second survey [47], the authors also classify different functional areas in the field, identifying several works in the literature for each of them, and raising open research questions. The theoretical basis

[10] http://clayfour.ee.ucl.ac.uk/lattice/.

of these surveys was used by adapting some of the characteristics presented specifically for cloud computing to the context of the cloud-network slicing concept.

We would like to highlight three papers published in previous editions of the Journal of Network and Systems Management (JONS). The first work [48] presents how to provisioning and maintenance 5G services over network slices with a focus on network elements and Network Function Virtualization (NFV). The second paper [49] focuses on a techno-economic analysis to provide a cost allocation model to network slices. Finally, the authors in [50] propose an architecture responsible for creating network slices to use in the context of wireless networks. The work presented in this paper goes deeply into the cloud-network slicing topic and presents the CNS-AOM design with contributions such as the slice elasticity, the multiple administrative and technological domains and the PoCs on real environments.

3 Architecture

In this section, we present the CNS-AOM architecture describing the subsystems, interfaces, and components. The proposed archicteture covers the CNS post-provision phase, being responsible for the slice resource management, monitoring and orchestration. The architecture was designed in a way to support the technological diversity between slice resources, dealing with multiple administrative domains, performing the orchestration and adapting the slice resources when an elasticity operation is performed.

The CNS-AOM is divided into three subsystems: *Resource & Service (R&S) Management*, *Resource & Service (R&S) Monitoring*, and *Slicing Resource Orchestrator (SRO)*. A brief description of their main features is presented below.

- The SRO is responsible for starting the CNS post-provision phase and create the R&S management and monitoring components according to a given CNS. It is the main entity for performing CNS elasticity operations, dynamically growing or shrinking the slice resources, in case the monitored metrics break any policy. It also supports the policy creation, deletion, and modification. Finally, SRO must adapt the R&S management and monitoring components according to the new CNS configuration after an elasticity operation occurs.
- The R&S management subsystem is responsible for providing the CNS resources management, which means the capacity to manage the slice resources and services for each slice part. It also performs the lifecycle management of instantiated services at run-time, including operations such as the service deployment, reconfiguration and stop.
- The R&S monitoring subsystem collects and stores the infrastructure and service metrics for each slice part. Those metrics will be checked for the SRO to perform the slice elasticity operations.

The CNS-AOM is presented in Fig. 2, showing the R&S management subsystem in yellow, the R&S monitoring subsystem in green, and the SRO subsystem in purple.

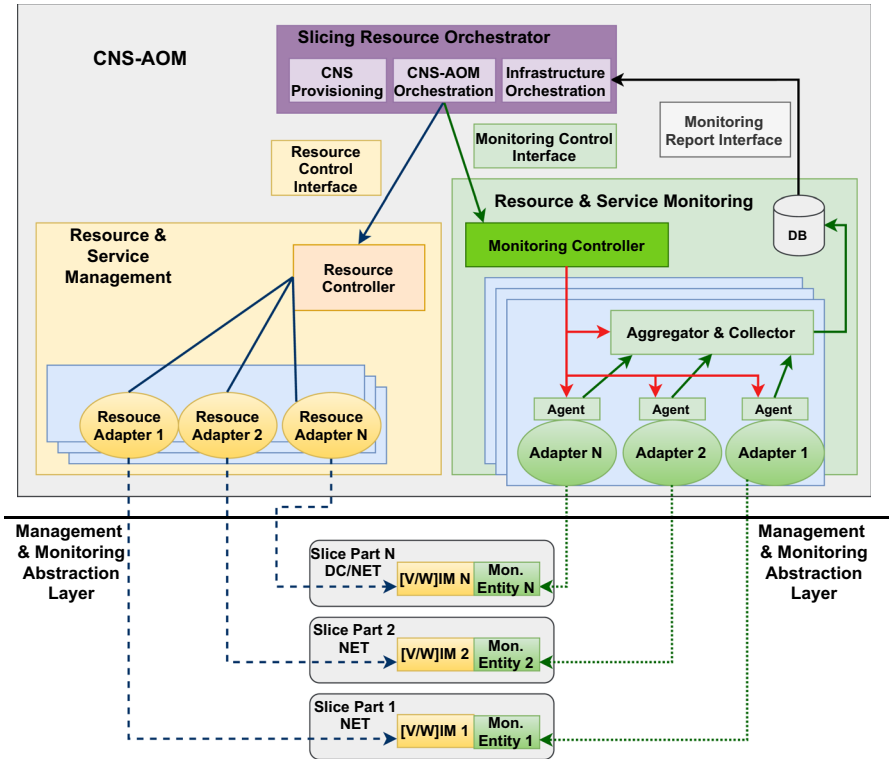


Fig. 2 CNS-AOM architecture

3.1 Slicing Resource Orchestrator

The SRO subsystem is divided into three components, as shown in Fig. 2. The focus of this paper is on post-slice provisioning and the interaction of the SRO with the management and monitoring subsystems. Generally, the CNS provisioning component is responsible for instantiating the CNS infrastructure interacting with each resource provider. This component also binds the slice parts in order to provide an end-to-end slice. After CNS infrastructure instantiation, the *CNS provisioning* component communicates with the *CNS-AOM orchestration* component to start the subsystems' instantiation on-demand.

The *CNS-AOM orchestration* component is the core component of the architecture and is responsible for interacting with the monitoring and management subsystems. These interfaces are omitted from Fig. 2 for sake of simplification. The main responsibilities of each interface are presented bellow.

– **CNS provisioning interface:** after CNS infrastructure provisioning, the *CNS provisioning* component calls the *CNS-AOM orchestration* component to start the deployment of the R&S monitoring and management subsystems;

- **Tenant interface:** once the slice and the CNS-AOM subsystems are running, the tenant is able to request certain operations such as to obtain the infrastructure topology, change the configuration of the monitoring subsystem, service deployment and service deletion. It is the responsibility of the CNS-AOM to interpret these operations and trigger the monitoring and management subsystems;
- **Infrastructure orchestration interface:** after the *infrastructure orchestration* component triggers an elasticity operation (the capacity of growing or shrinking the slice resources), it is necessary to reflect the infrastructure changes in the monitoring and management subsystems. For example, if a new slice part is added to the slice (horizontal elasticity), the monitoring and management subsystems need to deploy the components necessary to interact with this new slice part. In addition, the service could be (re)deployed to consider this new infrastructure. Therefore, to perform these adaptations, the *infrastructure orchestration* component uses this interface to communicate with the *CNS-AOM orchestration* component.

Finally, the *infrastructure orchestration* component is the "brain" of the SRO, performing the elasticity operations based on previously established policies, and considering the infrastructure or services metrics being monitored by the R&S Monitoring. As already described, the SRO through the monitoring report interface is capable of obtaining the monitored metrics and checking periodically if the policies defined by the tenant are being adhered to.

The component could use different algorithms to trigger elasticity operations, such as simple threshold-based algorithms or (reinforcement) learning-based algorithms, which are described in Sect. 4.3. When an elasticity operation is performed, the *infrastructure orchestration* calls the *infrastructure orchestration interface* to adapt the management and monitoring subsystems and, if necessary, to (re)deploy the service.

3.2 Resource & Service Management

The R&S Management subsystem aims to abstract the communication between a slice provider (or a tenant) and the CNS infrastructure. It is also responsible for managing the lifecycle of the virtual resources and services that are instantiated in a slice over multiple administrative and technological domains. This component uses resource adapters to communicate with different VIMs and WIMs. The resource adapters are implemented specifically to request actions for a specific VIM or WIM and include actions to deploy a new virtual machine or service. Examples of well-known VIMs/WIMs are OpenStack, Kubernetes, Docker Swarm, and OpenFlow controllers. This allows infrastructure providers and tenants to use the technology that best suits them.

The *resource controller* component is responsible for managing the adapters and virtual elements of the slice at run-time. Additionally, this component performs the following functionalities:

- **Adapter lifecycle management:** start, stop, update, and configure the resource adapters;
- **Virtual elements (virtual machine or container) management:** start, stop, update, configure, and retrieve information;
- **Service management:** start, stop, update, configure, and obtain information about the services.

In Fig. 2, we show the R&S Management components in yellow and also highlight their two interfaces: the *resource control interface* and the *management & monitoring abstraction layer*. The former is responsible for sending requests from the *SRO* to the *resource controller*. The latter *is* the interface responsible for abstracting the communication between the adapters and the VIMs/WIMs. The abstraction layer is only used by the *resource control interface* in response to a request.

The resource adapters are specific wrappers for different VIM or WIM implementations, whose aim is to provide basic platform-agnostic wrapping services for common slice provider platform functions. In other words, the adapters can be implemented and used according to the VIM/WIM chosen by the tenant for that slice part. To accomplish end-to-end management for each CNS, it is necessary to instantiate one adapter per slice part. Therefore, the right adapter has to be instantiated by the resource controller based on the VIM/WIM for each slice part to perform the aforementioned operations. The design of this component is very important as it supports the CNS characteristics in the CNS-AOM, such as the heterogeneity of VIMs/WIMs, the elasticity operations, and the concept of SlaaS. For example, in elasticity operations, when a new slice part is added to the slice infrastructure, a new adapter is instantiated by our mechanism.

3.3 Resource & Service Monitoring

The R&S Monitoring subsystem is responsible for monitoring the infrastructure and services for a given slice. It needs an abstraction layer that supports distinct monitoring entities and the ability to aggregate metrics from different resource types (computing, networking, storage, and services). The details of this subsystem and each component are presented in green on the right side of Fig. 2.

The communication in this subsystem is divided into two planes. The control plane, represented by red arrows, supports the lifecycle management of monitoring components (instantiation, configuration, and deletion). The data plane, represented by green arrows, represents the path of collected metrics through the components until they are stored in the database. This division in control and data planes brings agility as well as isolation to the monitoring solution.

The *monitoring controller* is the component responsible for instantiating the elements required to monitor a slice and its resources or services. It manages the

aggregator & collector, *agents*, and *adapters*. In addition, the *aggregator & collector* component multiplexes the monitoring information from different parts of an end-to-end slice. It receives the measurements collected by adapters and aggregates them to generate the metrics for an end-to-end CNS. As shown on the right side of Fig. 2, the aggregated metrics are stored in a database, from where they could be queried at any time by the *infrastructure orchestration* component.

To provide an abstraction layer, the *agent* and *adapter* components jointly collect measurements across the different monitoring entities instantiated in each slice part. The *agent* manages the *adapters'* requests and controls their activation. In addition, each *agent* is responsible for aggregating the measurements collected by the *adapters* and sending them to the *collector* component.

The *adapter* is the main component responsible for providing the monitoring abstraction, similar to the *resource adapter* described in the previous subsection. The role of adapters is to interact with different monitoring entities to collect metrics. Each adapter communicates with a single monitoring entity and intends to establish real-time monitoring of resources. The monitoring entity is, in turn, a monitoring tool (e.g., Prometheus, Zabbix, Netdata, and Nagios) that runs within the slice part and is responsible for collecting on-site resource metrics.

During the slice provision, important artifacts are deployed in each slice part, such as the VIMs/WIMs and the monitoring entities responsible for gathering the metrics for that infrastructure. The adapters are instantiated after the slice provision by the interaction between the *CNS-AOM Orchestration* and the *Monitoring Controller*. By using the information of each monitoring entity, the *Monitoring Controller* is capable of instantiating the corresponding adapters present in that slice. Then, the R&S Monitoring components will abstract the different infrastructures and technologies to store the metrics in terms of slice and slice parts. More details will be explained in Section 4.

An important characteristic of CNS-AOM is that the components are deployed for each managed and monitored slice. This is depicted in Fig. 2, with blue rectangles indicating the individual slices. For the *R&S Management*, each slice has their own *resource adapters*. Similarly, for *the R&S Monitoring*, each slice has their own *adapters*, *agents*, and an *aggregator & collector*.

Communication between the monitoring subsystem and external components is an important point in this proposal. To satisfy this need, two communication interfaces are created. These interfaces enable communication between the monitoring subsystem and the *SRO*. The *monitoring control interface* is responsible for receiving the start, update, and delete requests to manage the monitoring elements. The *monitoring report interface* allows the *SRO* to access the metrics stored in the database to orchestrate the resources and/or services. Therefore, the *SRO* can pool or push the metrics to check them against the SLOs/SLAs defined by the tenant in order to trigger elasticity operations. In this paper, we implement the SRO to pool the metrics since our focus at this moment is not on optimizing the performance. However, we intend to implement and test the pushing mechanism to check the SLOs/SLAs in future work. The monitored metrics can be accessed by the tenant also; however, a discussion on this is not in scope of this paper.

4 Implementation Details

In this section, we present the implementation details of each CNS-AOM subsystem (management, monitoring, and orchestration)[11] as well as two workflows explaining the four most important methods. These include the methods to start the components for infrastructure and service monitoring, the method to start the components for management of the CNS, and the method to (re)deploy services. Aspects such as the management and orchestration of network elements and security are not the focus of this paper and will be explored as future work. The description will start with the Resource & Service components, since the SRO is responsible for invoking all the methods of those components.

4.1 Resource & Service Management Implementation

The main scope of R&S management is to provide an abstraction layer for multiple VIMs/WIMs deployed across a multi-domain environment, which mainly includes the deployment and reconfiguration of services. All the components are implemented in Python, but given the generic purpose of the architecture, it is possible to add adapters written in any other language.

The *Resource Controllers and Monitoring Controllers* are responsible for starting, configuring, and managing relevant *adapters* in the *management & monitoring abstraction layer*, to hide implementation details of a specific VIM/WIM or monitoring entity. The *CNS-AOM orchestration* component interacts with the *resource controller* through the *resource control interface* to provide methods for each VIM/WIM present in a slice. Five main methods are implemented in the *resource controller* to enable management operations and are all available in the *CNS-AOM orchestration* component, namely:

- ***start_management***: responsible for instantiating specific *adapters* for each VIM/WIM present in the slice. The parameter to call this function is a YAML file containing information about the slice parts and VIMs/WIMs and can be seen in Fig. 3;
- ***stop_management***: responsible for removing all the *resource adapters* instantiated for a specific slice. The input parameter for this method is a *Slice Id*, provided by the tenant;
- ***update_management***: responsible for instantiating/removing *resource adapters* on demand, as the result of elasticity operations such as adding or removing slice part(s) or resources;
- ***deploy_service***: responsible for instantiating a service requested by the tenant. The parameter for this method changes according to the set of VIMs/WIMs used. More specifically, it changes for each service, depending on the slice resources

[11] All the source code used in this paper is available at https://gitlab.com/necos/demos/musts.

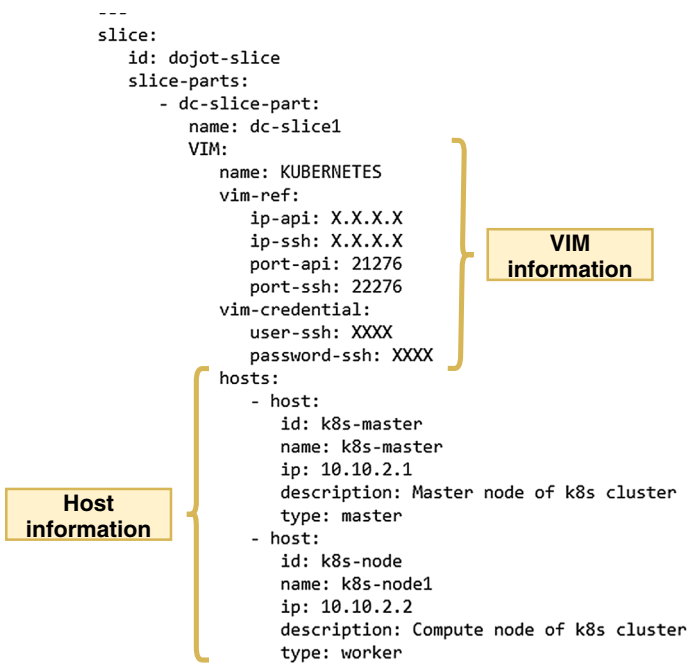


Fig. 3 Example of a YAML file used for starting the components for management

requested by the tenant and the VIMs/WIMs deployed by the resource providers. An example of the YAML file used for the tenant to request the service deployment is shown in Fig. 5;

– ***delete_service***: responsible for removing a service running in a slice. Again, the input parameter for this method changes according to the set of VIMs/WIMs used.

Figure 3 illustrates an example of a YAML file received by the CNS-AOM orchestration to call the *start_management* method. This YAML contains all the data needed to start the components properly such as the VIM and host information. For the VIM, the following parameters must be defined: the VIM name, the entry points (IP address and ports), and the credentials needed to call the VIM methods. The host information is specific to each VIM. In Kubernetes, for example, we need to define the role type (master or worker) of each host instantiated in the slice part and the corresponding IP addresses. This structure can be expanded to adjust for any VIM/WIM. All information is filled by the Slice Builder component [2] right after the slice provision phase. The Slice Builder handles all the slice provision phase and interacts with the SRO after the slice provision to provide information such as the details about the VIMs/WIMs, monitoring entities, and the slice parts for both R&S subsystems.

Adapters are wrappers for different VIM/WIM implementations, providing a basic service agnostic-platform for common CNS-AOM functions. In practice, the

adapters should translate generic calls into commands or methods for specific VIMs and WIMs. Given the context of CNSs, five main types of adapters can be listed: cloud adapters (e.g., OpenStack, Kubernetes, Docker Swarm, and VLSP), transport adapters (SDN controllers such as Opendaylight and Floodlight), VNF adaptors (e.g., OpenMano, Open Baton, and OPNFV), RAN adapters, and edge adapters.

Currently, three cloud adapters were developed in the PoCs: *Kubernetes*, *Docker Swarm*, and *SSH adapter*. The Kubernetes and Docker Swarm adapters are chosen because the container technology is an enabler for several use cases in mobile networks (e.g., 5G) [51] and it is widely used in different verticals. Those adapters support most of the functionalities provided by the Kubernetes and Docker Swarm APIs, such as *deploy service, get pods, list services, get service, and stop service*. Conversely, the SSH adapter is based on the SSH protocol and abstracts the execution of commands remotely on physical hosts that belong to a specific slice part.

4.2 Resource & Service Monitoring Implementation

In order to monitor a multi-domain CNS environment, some challenges need to be addressed, such as the monitoring of infrastructure and services, the monitoring of different resource types (e.g., hosts, virtual machines (VMs), and containers, network elements), and the abstraction for different monitoring entities (e.g., Prometheus and Netdata). To overcome these challenges, we present in this paper the R&S Monitoring as described in Sect. 3.3. The components are represented in Fig. 2 and are detailed in the remainder of this section, from an implementation perspective.

The *monitoring controller*, similar to the management subsystem, is the component that implements the functionalities for starting, configuring, and controlling relevant end-to-end slice components, such as agents and adapters, responsible for gathering measurements from the slice parts and forwarding them to the *aggregator & collector* component. These components are instantiated on-demand by the *monitoring controller*, which acts as a measurement aggregator for an end-to-end slice.

The *CNS-AOM orchestration* sends a request to the *monitoring controller* with a YAML descriptor as a parameter through a REST API. The *monitoring controller* receives the requests through the *monitoring control interface* and wraps the actions to start the monitoring components correctly. The content of the YAML file is converted to JSON for parsing and extracting the information needed (e.g. name, IPs and port numbers of the monitoring entities, metrics to be monitored/stored) to create slice monitoring components. A control table is used to store the IDs of each monitoring component that had been started. The management of these components is possible in the case of requests to update or delete the monitoring environment.

The *monitoring control interface* allows the *CNS-AOM orchestration* to trigger the on-demand instantiation of the monitoring subsystem components that deploy the abstraction mechanisms required for collecting measurements from end-to-end slice infrastructure and services. In addition, this interface implements the lifecycle management of all the monitoring subsystem components, including the

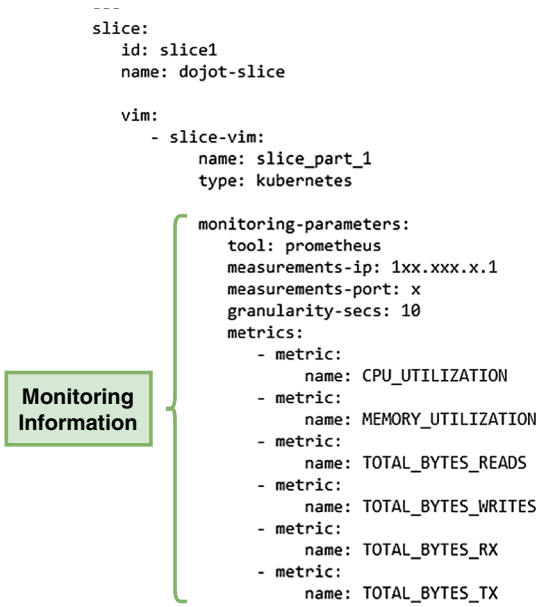


Fig. 4 Example of a YAML file to start the infrastructure monitoring components

infrastructure and service monitoring. The *CNS-AOM Orchestration* communicates using this interface, which supports the following API methods:

- ***start_monitoring***: responsible for instantiating an *aggregator & collector* component, a specific *agent* for each slice part, and a specific *adapter* for each monitoring entity. The parameter for this function is a YAML file (Fig. 4) containing monitoring information about all slice parts (e.g., name, IP address and port numbers of each monitoring entity). Also this YAML includes parameters, such as the metrics that will be monitored, represented by the field *"metrics"* (e.g., CPU, memory, and network) and the polling frequency to collect those metrics, represented by the field *"granularity-secs"*;
- ***start_service_monitoring***: responsible for instantiating an *adapter*-and-*agent* pair capable of collecting the service metrics for each slice part. The input to this function is a YAML file that contains monitoring information about the service, such as the service metrics to be monitored (frame per seconds, latency, etc.) and the polling frequency to collect those metrics;
- ***delete_monitoring***: responsible for removing the infrastructure and service monitoring components instantiated for a specific slice. The input parameter is a YAML file that contains a slice identifier;
- ***update_monitoring***: responsible for instantiating or removing end-to-end monitoring components on demand, as the result of elasticity operations such as adding or removing slice part(s).

Figure 4 represents an example of a YAML file used by the CNS-AOM orchestration to call the *start_monitoring* method. In this file, the tenant must define

the monitoring parameters for each slice part. The required parameters are the name of the monitoring entity (*"tool"* in Fig. 4) used for collecting the resources metrics, the IP address (*"measurements-ip"* in Fig. 4) and port number (*"measurements-port"* in Fig. 4) to interact with the monitoring entity, the interval for pooling the desired metrics (*"granularity-secs"* in Fig. 4), and the names of the desired metrics (*"metrics"* in Fig. 4). The YAML file is used for triggering the *start_service_monitoring* method. In comparison to the YAML file presented in Fig. 4, this file contains the desired metrics related to the service instead of the infrastructure. Examples of service metrics include frames per second, latency, HTTP requests per second, and transactions per second.

We implement two *adapters*, allowing the abstraction for the open-source monitoring tools, Prometheus and Netdata. Those tools are under the Cloud Native Computing Foundation[12] initiative and are being used widely in the community and cloud projects, such as shown in the works presented in [16, 52]. If different monitoring tools or VIM/WIMs technologies are required by the tenant, new adapters or resource adapters need to be implemented by the slice provider.

The Prometheus adapter is responsible for communicating with the tools to monitor the resources of a slice part through REST API calls. To be effective, this call must contain the Prometheus interface address (IP and port) and the appropriate search query for the desired metric. A database is created to link the desired metric with the query used in the search. The Prometheus entity receives a call with a query to collect metrics and then replies in a JSON format, which is converted by the *adapter* to a standard model as presented next in Listing 1 and Listing 2. The formats shown in Listing 1 and Listing 2 show how all the metrics are stored following an information model capable of supporting different monitoring entities, and the resource technologies namely, cloud computing, networking and storage.

Listing 1 Common format to store all the metrics.

```
Measurement A:
    timestamp: <Integer>,
    resource_id: <Integer>,
    slice_id: <Integer>,
    slice_part_id: <Integer>,
    value: <Float>
```

Listing 2 Example of common format to store all the metrics.

```
Measurement CPU_UTILIZATION:
    timestamp: 1599921848,
    resource_id: 5,
    slice_id: 1,
    slice_part_id: 1,
    value: 49.5
```

[12] https://www.cncf.io.

Similarly, the Netdata adapter uses a REST API to communicate with the monitoring entity. The call must contain the Netdata host address (IP and port number) as well as the appropriate query. As in the previous case, a set of queries linked to the requested metrics are created. The measures returned by the Netdata REST API are in a JSON format, which are converted to the format presented in Listing 1 and Listing 2. Both adapters are implemented to collect service metrics in containerized environments (e.g., Docker and Kubernetes) as well as well-known infrastructure metrics (e.g., CPU, memory, and disk usage).

More examples of metrics can be added, however, the tenant must be aware of which metrics can be collected for each monitoring entity and select the best suitable option of the monitoring entities to improve the SLOs checking. An ongoing work in this line is to make use of NLP (Natural Language Processing) so that the tenant may be able to define the requirements using flat text and these would be converted into infrastructure and service metrics through well-defined algorithms.

The *aggregator & collector* component is responsible for writing the metrics converted as time-series measurements to a time-series database (InfluxDB instance), from where the *infrastructure orchestration* could access the metrics through a REST API. For each end-to-end slice, a suitable and isolated adaptation layer is created to collect and send per-tenant measurements to a time-series database, while ensuring the proper degree of isolation between different slices and tenants.

4.3 Slicing Resource Orchestrator

This subsection presents the implementation details of the CNS-AOM's orchestration and infrastructure orchestration components. The components are minimally implemented to perform the proofs of concept (PoCs) of this paper. Most of the CNS-AOM's orchestration responsibilities are already covered in the previous subsections. Therefore, we will present the service operations and the elasticity algorithms here. Both components are implemented in Python from scratch, and the interfaces are implemented using REST architecture.

The SRO can delegate the deployment and interactions with the service to the tenant. It should be capable of deploying, deleting, and updating a service by calling *CNS-AOM orchestration* through the tenant interface. In Fig. 5, we show a YAML file needed to deploy a specific service through the CNS-AOM orchestration. Such an operation has already been described in the *deploy_service* method presented in Sect. 4.1; here we focus on the communication between the SRO and the tenant.

When the CNS-AOM orchestrator receives a YAML file, it must be able to identify information such as the slice part the service must be instantiated (parameter "name" in "dc-slice-part"), the host and VIM that will be instantiated (parameters "host" and "VIM"), and the commands (parameter "commands") that must be executed by the resource adapter in the slice part. After identifying all these parameters, the CNS-AOM orchestration calls the resource controller to deploy the requested services. In Sect. 4.4, we present the workflows describing the main methods of our solution, including the *deploy_service* method.

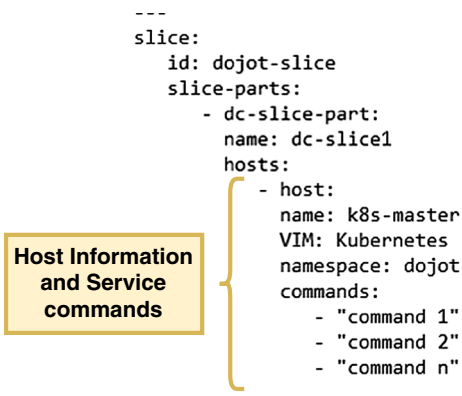


Fig. 5 Deploy service YAML file

As mentioned before, the brain of the SRO is the infrastructure orchestration component that has four main responsibilities: (i) interpreting the policies sent by the tenant; (ii) periodically checking the metrics being monitored; (iii) triggering the elasticity operations based on elasticity algorithms; and (iv) adapting the CNS-AOM subsystems and services after an elasticity operation. The tenant can define infrastructure policies for their slice. To do this, the infrastructure orchestration provides a simple interface to retrieve policies and information such as metrics to be monitored periodically, slice parts to be monitored, operations to be triggered when needed, elasticity algorithm to be used (threshold, learning-based, reinforcement learning-based, statistic), and thresholds needed to trigger the elasticity operation when using the threshold algorithm. With this information, the infrastructure orchestration is able to gather the monitored metrics through the monitoring report interface and check if the policy requirements are being satisfied or not.

If the policy requirements are not satisfied, the infrastructure orchestration performs the elasticity operation previously defined in the policy. After the elasticity operation is performed, the infrastructure orchestration requests the CNS-AOM orchestration to check whether it is necessary to adapt the CNS-AOM subsystems or the services. The CNS-AOM decides on whether or not to adapt the subsystems/services based on the type of elasticity operation. The *infrastructure orchestration* triggers this adaptation in the *CNS-AOM orchestration*, calling the *update_management* and *update_monitoring* methods.

We will present the elasticity operations in Sect. 5.2, where we explain the IoT slice PoC. In this work, after a vertical elasticity operation is performed, no adaptation of the subsystems and services is needed. Conversely, when a horizontal elasticity operation is performed, the subsystems and the services are adapted or (re) deployed. We assume this because horizontal elasticity always occurs on a slice-part level. Because the CNS-AOM components (resource adapters and adapters) have a 1:1 relationship with the slice part's technologies such as VIMs/WIMs and monitoring entities, adding or removing a slice part would always result in adaptation or (re) deployment of the components and services.

Next, we describe potential elasticity algorithms for the infrastructure orchestration component. Elasticity algorithms are responsible for triggering the elasticity operations based on the monitoring metrics. The following is a description of the elasticity algorithms that can be implemented in the infrastructure orchestration:

- For **threshold elasticity algorithms**, the tenant defines a threshold policy for infrastructure orchestration to periodically check the monitoring metrics. In this algorithm, an elasticity operation is triggered when the threshold is breached a pre-defined number of times in sequence. The IoT demonstration considered in this paper uses this algorithm activating a trigger after three threshold violations. For instance, after three consecutive measures of CPU usage above an 80% threshold, the infrastructure orchestration will trigger the applicable elasticity operation;
- For **learning-based elasticity algorithms**, the infrastructure orchestration benefits from machine learning algorithms to predict the threshold value that must be used to trigger the elasticity operations. One SRO implementation example that uses a learning-based elasticity algorithm was presented as a demonstration in the NECOS project.[13] The demonstration shows a recurrent neural network (RNN) responsible for performing the key performance indicator (KPi) estimation, the SLA prediction, the slice resources optimization, and the enforcement of slice modifications.
- For **reinforcement learning-based algorithms**, the infrastructure orchestration implements reinforcement/deep learning algorithms to trigger the elasticity operations. The purpose of these algorithms is to maximize a reward based on the decisions previously made by an agent to achieve the best choice of elasticity operation to be realized for specific scenarios.
- For **time window-based approach**, elasticity operations are triggered when a threshold crossing is registered, and such threshold crossing is kept for a given period of time (time window). The purpose of this approach is to avoid unnecessary elasticity operations due to statistical fluctuations of the infrastructures supporting the slices.

In this paper, we leverage elasticity based on the time window elasticity approach. A detailed description of the elasticity algorithms that have been validated in the NECOS project is available for the interested reader in reference [53].

4.4 Workflows

This section describes the workflows considering the following methods:(i) *start_monitoring*, (ii) *start_management*, (iii) *deploy_service*, and (iv) *start_service_monitoring*. Methods (i) and (ii) occur immediately after the slice instantiation and are responsible for starting the components of both monitoring and management subsystems, on behalf of a specific CNS. Thus, with those components running in the

[13] http://www.maps.upc.edu/public/MLO_demo_video_with_audio.mp4.

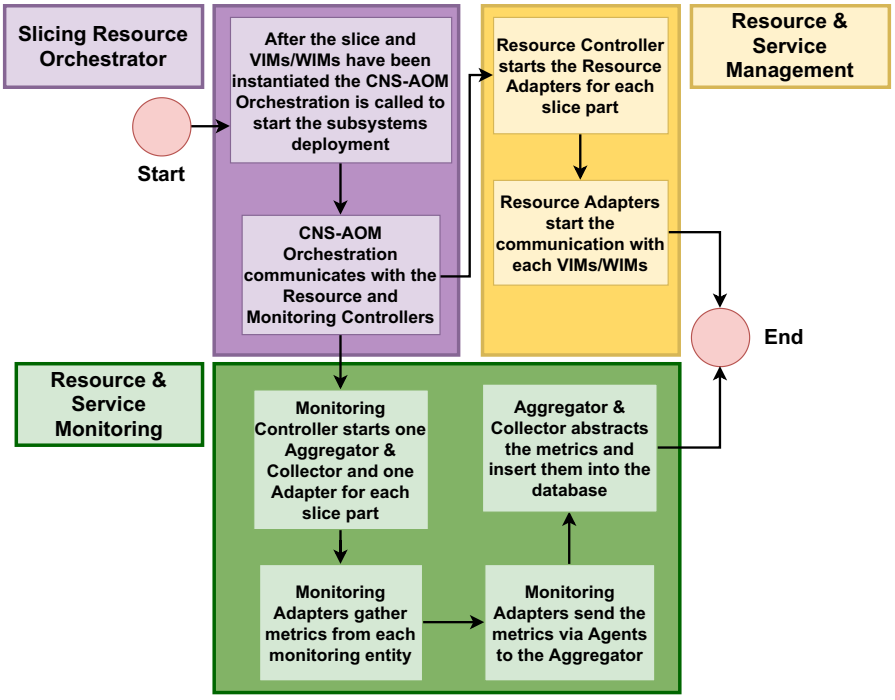


Fig. 6 The workflow responsible for *start_management* (purple-to-yellow) and *start_monitoring* (purple-to-green)

slice provider, the tenant can deploy the service, (iii), and start the monitoring components, (iv), responsible for collecting the established metrics.

Figure 6 illustrates the steps needed to start the monitoring and management subsystems. Assuming that the slice infrastructure is instantiated and the last two subsystems are deployed appropriately, the tenant can request a service deployment and monitoring from the CNS-AOM. The workflow of these tenant requested operations is depicted in Fig. 7.

The workflow depicted in Fig. 6 starts after CNS instantiation. The CNS-AOM orchestration component instructs both controllers to deploy each subsystem (monitoring and management). In terms of management, the resource controller is called through the method *start_management*, instantiating the resource adapters for communication with the VIMs/WIMs of that CNS. In terms of monitoring, several components need to be instantiated/configured as described in Subsection 3.3. Thus, the *start_monitoring* method requests the deployment of one aggregator & collector as well as a set of agents and adapters (one pair for each slice part comprising the CNS). Finally, after all instantiation processes are complete, the CNS-AOM orchestration receives a response indicating that all components are created correctly and the subsystems' information is returned to the tenant.

The workflow illustrated in Fig. 7 starts after the deployment of the R&S monitoring and management subsystems. The tenant can request a service deployment by

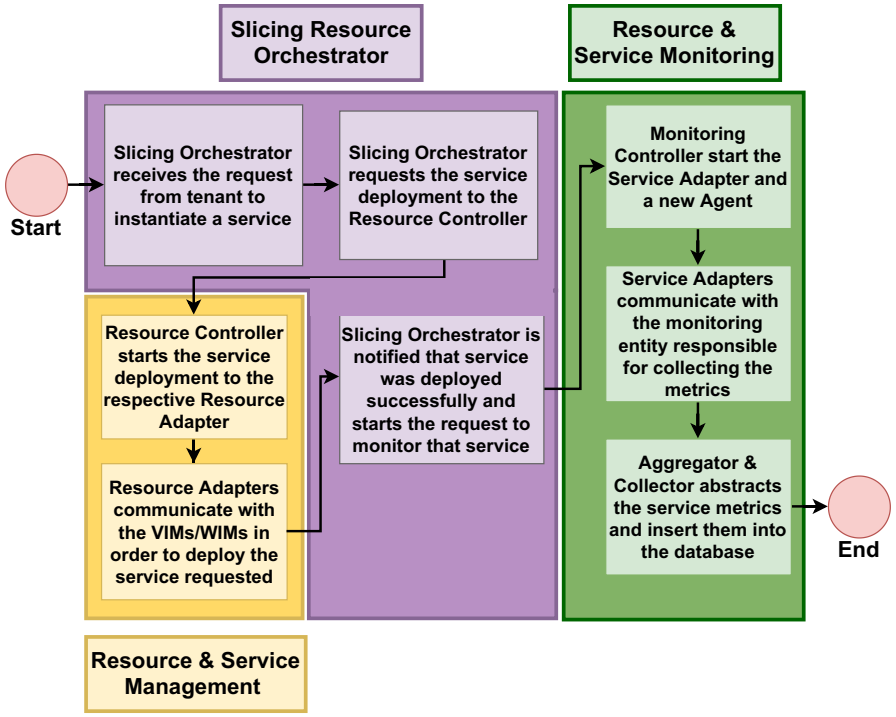


Fig. 7 The workflow responsible for *deploy_service* (purple-to-yellow) and *start_service_monitoring* (purple-to-green)

sending a detailed service description to the CNS-AOM orchestration. Upon receiving the request, the latter invokes the resource controller's *deploy_service* method that instructs the resource adapters to deploy the service in the VIMs and WIMs. Next, the orchestrator calls the monitoring controller to deploy the respective adapters responsible for collecting metrics from the service. As the other monitoring components are already deployed, the *start_service_monitoring* method is responsible for instantiating only the service's agent(s) and adapter(s). As the aggregator & collector is already instantiated, the new service metrics will be aggregated and inserted into the database according to the format described in Listing 1.

5 Proof of Concept Validation

In this section, we describe two services instantiated over distinct CNSs, aiming to validate the proposal through the PoC implementations. The two different slices are monitored, managed, and orchestrated by the CNS-AOM subsystems. The first slice allocates a content distribution network (CDN) service, which is deployed among three cities in the state of São Paulo, Brazil. The second slice instantiates the IoT platform called *dojot* [54], comprising an architecture of microservices, in which the development and offering of IoT services are supported. The CNSs presented in

this section are deployed across different cities in the case of the CDN and different countries (Brazil and Spain) in the case of the IoT platform. Here we highlight the important characteristics of CNSs, such as multiple administrative and technology domains as well as the elasticity operations.

The purpose of the PoCs discussed in this section is to verify whether the proposed architecture is able to provide monitoring, management, and orchestration of CNSs. First, the CDN PoC will demonstrate the service functionalities, deployment of CNS-AOM subsystems through the SRO, the service deployment through R&S management and multiple VIMs, and the service metrics being monitored through R&S monitoring from multiple monitoring entities. Second, the *dojot* PoC will evaluate the slice elasticity operations by using a tenant-defined policy and periodically monitoring the CNS through infrastructure orchestration component. If the monitored metrics exceeds a predefined threshold, the infrastructure orchestration triggers an elasticity operation and adapts the subsystems when necessary.

In the next two subsections, each of the services will be presented in detail, specifically in terms of the configuration setup and analysis of the obtained results.

5.1 CDN Service Description

This service is responsible for providing multimedia content (photos, videos, etc.) of tourist areas to end-users. Based on the user's location, the CDN is capable of providing content from the closest server and minimizing the delay to download the content. The CDN is composed of a core cloud, responsible for processing the end-users' requests, and edge clouds instantiated on demand, responsible for delivering the content close to the requester. The reason behind this service is that users (tourists) located in high-profile tourist areas are more likely to request content about local sites and consume videos on limited-resource devices (e.g., tablets and smartphones with battery and bandwidth limitations). Therefore, we use CNS-AOM to promptly instantiate the service on local edge clouds and deliver the content from a closer server, seeking to reduce the delay of downloading the consumed multimedia. The functionalities of the core and edge cloud components are as follows.

- The `core cloud` hosts a central service page and a video streaming content repository related to tourist attractions around the world. A DNS lookup service is also deployed, which is capable of redirecting a user's request to the most appropriate server (edge or core) based on the requester's location;
- The `edge cloud` provides web and video servers hosting cached content geographically distributed in predetermined areas around the world. These servers are responsible for replying to redirected requests to where each edge cloud service is located. Assuming that the edge cloud infrastructure is already deployed, the video service will be instantiated upon a request to the core cloud from a user located closer to the edge servers.

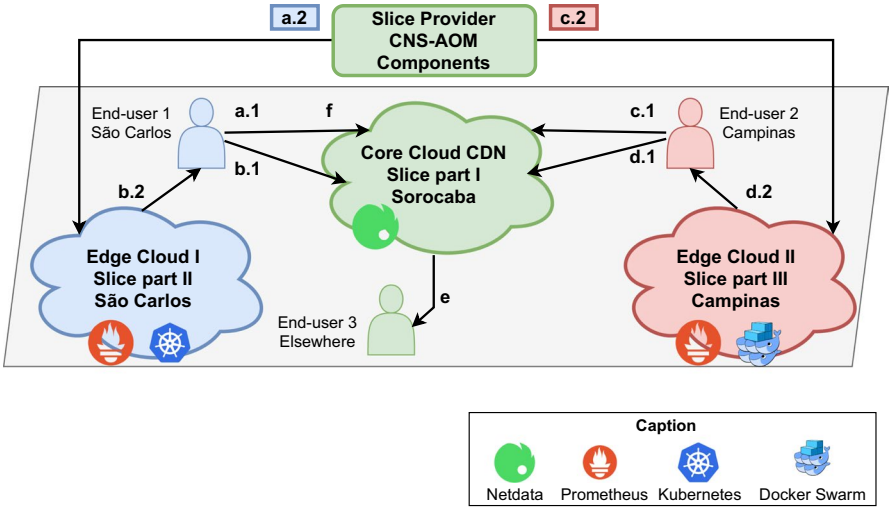


Fig. 8 CDN service across São Paulo cities

5.1.1 Configuration setup

Based on the proposed CDN service, we consider important aspects of CNSs, such as geo-localization, multiple VIMs, diverse monitoring entities, and different administrative domains. The objective of this PoC is to show that the CNS-AOM subsystems are capable of monitoring and managing the CDN slices, and validating these important aspects through a real scenario following a set of steps, which are detailed next.

Figure 8 illustrates the steps performed in the evaluation of the CDN, as well as the slice deployment. Three slice parts are instantiated across different cities of São Paulo state, Brazil. The slice provider contains the CNS-AOM components and is deployed in Sorocaba city. The core cloud CDN is also located in Sorocaba. The first edge cloud is deployed in the city of São Carlos and the second edge cloud is located in Campinas. In addition, to represent the technological heterogeneity supported by the CNS-AOM, different VIMs and monitoring entities are chosen for each slice part.

The steps shown in Fig. 8 demonstrate the monitoring and management operations performed. For the sake of simplicity, a single video ($\approx$ 2.4 MB) is present in each slice part and is transmitted at 20 kBps to the three end-users requesting video content. In this PoC, requests do not occur simultaneously since the goal of it is to validate the CNS-AOM management (including monitoring the infrastructure). To identify the location of the end-user, we define and implement different subnets for each city (São Carlos, Sorocaba, Campinas, and others). We also assume that there are enough resources to deploy and start the edge services closer to the user. The network connectivity between the slice parts is simplified using a VXLAN tunnel through the Internet. The following steps are executed to collect the metrics from the offered service:

(a.1) `end-user 1`, located in `São Carlos`, requests a `video` from the `core cloud` service, which delivers the requested video;
(a.2) after identifying one request from São Carlos, the `core cloud` service deploys `dge cloud 1` through R&S management components;
(b.1) `end-user 1` requests the `video` from the `core cloud` service again;
(b.2) the video is delivered by `edge cloud 1`, since it is geographically closer to `end-user 1`;
(c.1) `end-user 2`, located in `Campinas`, requests a `video` from the `core cloud` service, which delivers the requested content (likewise to step `a.1`);
(c.2) after identifying one request from Campinas, `edge cloud 2` is deployed through R&S management components;
(d.1) `end-user 2` requests the `video` from the core cloud service again;
(d.2) the video is delivered by `edge cloud 2`, as it is geographically closer to `end-user 2`;
(e) `end-user 3`, located elsewhere, requests a `video` from the `core cloud`, which delivers the requested content;
obs: after a period of inactivity, both edge cloud services are decommissioned using the R&S management components;
(f) `end-user 1` requests the `video` for the third time, which is again delivered by the `core cloud` service as `edge cloud 1` was decommissioned.

For the evaluation, we assumed that the core cloud service is already deployed by the CNS-AOM and is capable of distributing web and video content to end-users. In the same way, the edge clouds are previously instantiated, but the edge services are activated only after a core cloud request, depending on the end-user's location. The services and monitoring/management subsystems are deployed by the CNS-AOM according to the workflows described in Sect. 4.4. To perform the video redirection, the service checks the end-user location based on the IP address. If the closest edge service is not available, the infrastructure orchestration component starts the closest available edge service and gets the video from the core cloud. On the other hand, the end-user gets the video from the edge service closer to the end-user. The experimental setup used for this assessment comprises the following configuration:

- **Slice provider**
  - **Slice provider configuration:** Supermicro model X10SDV-TP8F, with OS Linux Ubuntu 18.04 LTS, Kernel 4.15.0-58-generic x86_64, CPU Quad core Intel Xeon D-1518 2.2GHz (-MT-MCP-) with 8 threads, RAM 64GB DDR4 and HD 2TB.
  - **Localization:** Sorocaba - São Paulo, Brazil.
- **Core Cloud - Slice Part 1**
  - **Resource provider configuration**: Supermicro model X10SDV-TP8F, with OS: Linux Ubuntu 18.04 LTS, Kernel: 4.15.0-58-generic x86_64, CPU Quad core Intel Xeon D-1518 2.2GHz (-MT-MCP-) with 8 threads, RAM 64GB DDR4 and HD 2TB.

  - **Localization:** Sorocaba - São Paulo, Brazil.
  - **VIM:** We use an SSH adapter to interact/deploy the service.
  - **Monitoring Entity:** Netdata.
- **Edge Cloud - Slice Part 2**
  - **Resource provider configuration:** One virtual machine with 8gb de RAM, OS Linux Ubuntu server 18.04 LTS, 4 virtual CPUs and 100 GB of storage.
  - **Localization:** São Carlos - São Paulo, Brazil.
  - **VIM:** Kubernetes.
  - **Monitoring Entity:** Prometheus.
- **Edge Cloud - Slice Part 3**
  - **Resource provider configuration:** Supermicro model X8DT3-LN4F, OS: Linux Debian 9, CPU Intel Xeon E-5520 2.26GHz with 8 MB cache, RAM 8GB DDR3 1066, HD 1TB.
  - **Localization:** Campinas - São Paulo, Brazil.
  - **VIM:** Docker Swarm.
  - **Monitoring Entity:** Prometheus.

5.1.2 CDN Service Evaluation

Figure 9 presents the amount of bytes transmitted by each service over time. In the case of the *core cloud*, the CDN service runs as an application on a physical host. The edge services run inside containers for both `edge cloud 1` and `edge cloud 2`. The labels **a**, **b**, **c**, **d**, **e**, and **f**) represent the steps described in the previous section. For instance, label **a** illustrates the transmitted bytes for the `core cloud` service considering steps **a.1** and **a.2**.

Based on the plot in Fig. 9, we present two important observations:

The black lines represent the `core cloud` service located in Sorocaba and show the time between receiving the request and delivering the video to the end-user. A short time interval between steps indicates that all operations executed by R&S management, R&S monitoring, and SRO subsystems are performed

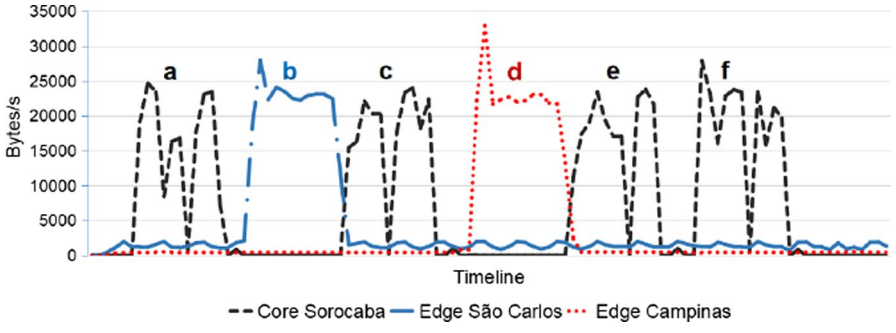


Fig. 9 Bytes/s transmitted by core cloud and edge cloud instances over time

Fig. 10 IoT service big picture

successfully as expected. These operations include the methods: *start_management*, *start_service_monitoring*, and *deploy_service*.

The containers running `edge cloud` instances are deployed shortly before steps **b** and **d**; more precisely, during steps **a.2** and **c.2**, respectively. This indicates that the R&S monitoring subsystem was able to collect the metrics from the service being monitored as soon as the edge instances were instantiated. The indicated behavior is illustrated in the figure using blue and red lines, representing metrics collected from `edge cloud 1` and `edge cloud 2`, respectively.

Finally, step **e** shows that *end-user 3*, who is located elsewhere, receives the video from the *core cloud* service, which in turn is located geographically closer to it. After a period of inactivity, the edge services are decommissioned. The last request from *end-user 1* is delivered by the *core cloud* service (step **f**) because *edge cloud 1* is no longer in service. However, as expected, after instantiation of the edge services by R&S management, the content is delivered to the requester from the closest server.

5.2 IoT Service Description

Figure 10 illustrates the IoT service "big picture", which consists of a real-time cargo monitoring and tracking solution. There are monitoring devices with multiple

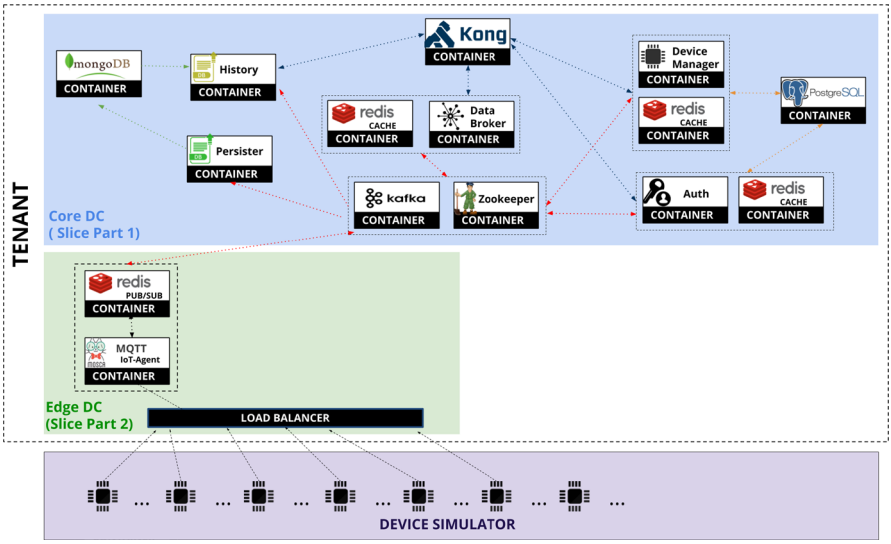


Fig. 11 Microservices of the *dojot* platform

sensors (temperature, humidity, light, and GPS) communicating through wireless networks. Those sensors travel in cargo containers during their journey. Their purpose is to provide monitored data periodically from the sensors to a centralized system located in the core cloud, allowing customers to have up-to-date information about their cargo during transportation. The sensors send the retrieved information, such as door openings, extreme temperature shifts, localization, and humidity, to the core cloud instance.

To test the CNS-AOM subsystems, the considered IoT service is instantiated in a real CNS setup, built across three cities in Brasil (Goiânia, Campinas, and Sorocaba) and one in Spain (Madrid). The objective is to show the elasticity operations being performed by the SRO (infrastructure orchestration) based on the monitored metrics from the CNS infrastructure.

An open-source IoT platform is used to implement the aforementioned service. The development of *dojot* is led by CPqD and is composed of several services such as Kong, Redis, Kafka, Zookeeper, MongoDB, and PostgreSQL. All these services are configured and deployed to provide IoT service validation. In addition, a load testing tool is developed for message queuing telemetry transport (MQTT) of the IoT devices, which is used to increase the number of requests.

Figure 11 illustrates how the *dojot* platform services are allocated in two different slice parts, one representing the core cloud and another representing the edge cloud. The former includes all the main services running, whereas the latter uses an MQTT IoT-agent to monitor the devices and a Redis microservice to publish the monitored metrics to the core when deployed. The main responsibilities of the core cloud are managing the IoT device lifecycle, storing the device's measurements, and providing a REST interface to retrieve historical and real-time data about the devices. One

MQTT IoT-agent, responsible for interacting with the IoT devices and sending the collected metrics to the core cloud, is instantiated for each edge cloud.

5.2.1 Configuration Setup

The *dojot* slice is composed of three slice parts, one core cloud instance and two edge cloud instances. The objective is to show the slice being monitored, managed, and orchestrated by the CNS-AOM subsystems. As shown in Fig. 12, the CNS-AOM subsystems are located in the city of Goiânia, representing the slice provider. The core cloud is located in the city of Campinas and comprises all the *dojot* microservices. The edge cloud instances are deployed in the city of Madrid as well as in the city of Sorocaba. The connectivity between the different slice parts is provided by VXLAN tunnels to connect the cities between Brazil and Spain. The VXLAN tunnels are created during the slice creation phase.

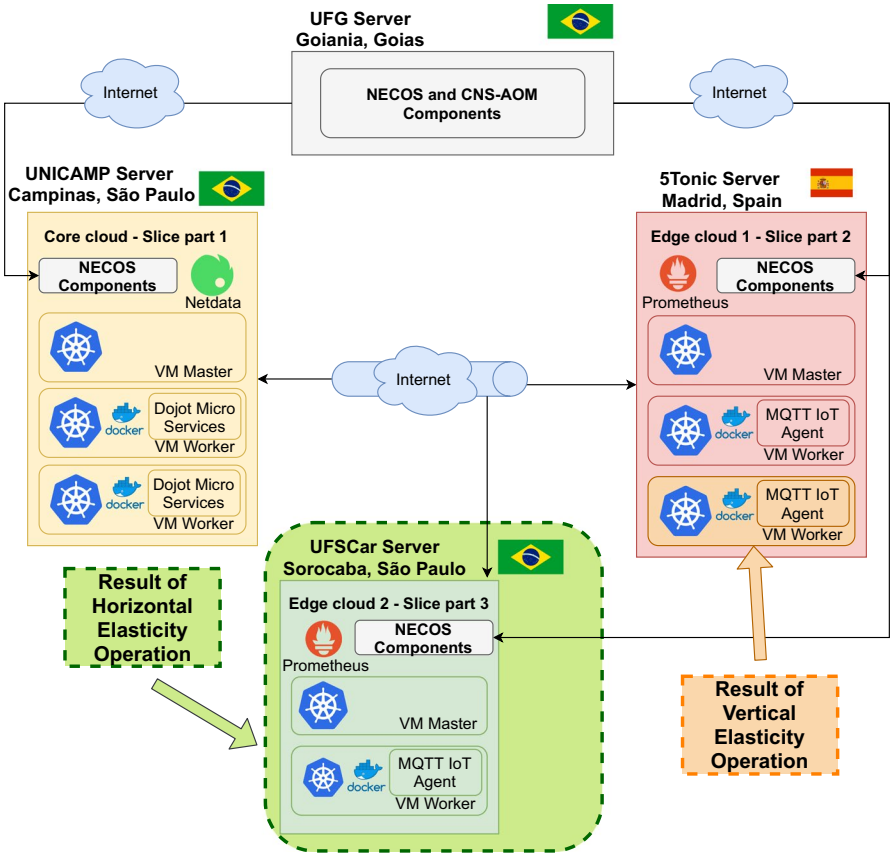


Fig. 12 *Dojot* slice PoC deployment

The complete description of the configuration setup used in this experiment follows.

- **NECOS slice provider**
  - **Slice provider configuration**: Dell EMC PowerEdge R740 server, equipped with two Intel Xeon Silver 4114 processor, 128 GB (8x 16GB RDIMM, 2666MT/s, Dual Rank) of RAM, and 12 TB of HD.
  - **Localization:** Goiânia - Goiás, Brazil.
- **Core Cloud - Slice Part 1**
  - **Resource provider configuration**: Dell PowerEdge R740, with OS: Linux Ubuntu 18.04 LTS, Kernel: 4.15.0-51-generic x86_64, two Intel Xeon Silver 4114 CPU 2.20GHzD-1518 processors, RAM 64GB DDR4 and HD 2TB.
  - **Localization:** Campinas - São Paulo, Brazil.
  - **VIM:** Kubernetes.
  - **Monitoring Entity:** Netdata.
- **Edge Cloud - Slice Part 2**
  - **Resource provider configuration:** Dell EMC PowerEdge R720 with 2x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz, 6 cores per socket (for a total of 12 cores or 24 vCPU), 128GB of RAM, and 2TB as HDD.
  - **Localization:** Madrid, Spain.
  - **VIM:** Kubernetes.
  - **Monitoring Entity:** Prometheus.
- **Edge Cloud - Slice Part 3**
  - **Resource provider configuration:** Supermicro model X10SDV-TP8F, with OS: Linux Ubuntu 18.04 LTS, Kernel: 4.15.0-58-generic x86_64, CPU Quad core Intel Xeon D-1518 2.2GHz (-MT-MCP-) with 8 threads, RAM 64GB DDR4 and HD 2TB.
  - **Localization:** Sorocaba - São Paulo, Brazil.
  - **VIM:** Kubernetes.
  - **Monitoring Entity:** Prometheus.

In this work, we consider two different types of elasticity, namely vertical and horizontal elasticity. Vertical elasticity means the addition of a new resource (in this case, a new VM worker) in one slice part already instantiated. Horizontal elasticity means adding a new slice part with the service running inside it. For more detailed information about elasticity operations in the context of CNSs, we recommend the work presented in [16]. The purpose of this PoC experiment is to demonstrate the CNS-AOM's capacity to execute both elasticity operations fully in a real environment. Seeking to show these operations, we start the *dojot* slice with only the core cloud and edge cloud 1, allowing for edge cloud 2 to be added after an elasticity operation.

The experiment attempts to increase the requests to IoT devices located at edge cloud 1 using a tool developed by the CPqD development team. Based on a policy

in the infrastructure orchestration defined by the tenant, the CPU usage of the infrastructure needs to be monitored and an elasticity operation has to be triggered if this metric exceeds an 80% threshold value three times, consecutively. We use a time window of 5 s. After interpreting this information, the SRO (infrastructure orchestration) has to periodically check the metrics being monitored by the R&S monitoring component. An elasticity operation is triggered upon detecting an increase in the CPU usage due to a large number of requests. Upon completion, two different elasticity scenarios are exercised; a vertical elasticity operation adds a new worker VM to edge cloud 1 and a horizontal elasticity operation adds a new slice part to edge cloud 2, as shown in Fig. 12.

The vertical elasticity using Kubernetes as a VIM does not require the (re)deployment of the service, as it is handled automatically by the VIM. This also happens with the monitoring entities Prometheus and Netdata. Therefore, after the occurrence of a vertical elasticity operation in the slice part already being monitored, these entities automatically recognize the new resource and the R&S monitoring component starts to store the collected metrics. However, after executing a horizontal elasticity operation, the service needs to be instantiated in the new slice part, considering that a new VIM is also instantiated. The R&S management component is responsible for the service's (re)deployment and communicating with the VIMs to distribute the requests to the IoT devices. Additionally, after a horizontal elasticity operation, the R&S monitoring component is responsible for instantiating a new adapter and a new agent to monitor the infrastructure metrics. The next subsection details the elasticity operations and the obtained results.

5.2.2 IoT Service Evaluation

We aim to show the results and describe the most important challenges addressed in this evaluation. More specifically, we present the results in terms of the time required to: (i) instantiate the whole slice infrastructure and VIMs; (ii) deploy the *dojot* service; and (iii) perform a horizontal elasticity operation. In addition, we discuss the timeline of the CNS being monitored before and after the elasticity operations are performed. We identify three important moments when analyzing the monitored metrics; first, when the infrastructure orchestration triggers the elasticity operation; second, when monitoring of the new resources (VM or slice part) starts; and third, when a decrease in CPU usage occurs after the addition of a new resource, as this load-balanced the requests.

Table 2 presents the instantiating times of the *dojot* slice service deployed in a real setup environment between Brazil and Europe. First, we observe that the slice infrastructure instantiation takes 1917.16 s (about 32 min) to be performed, from the time a request is made to the slice provider (NECOS platform) until all resources, management, and monitoring components are up and running. During this time, the task that takes more time to complete is the VIMs/WIMs deployment, being completed in 1718.74 s (about 28 min). Second, the reservation time, represented by the time elapsed to choose the best resource allocation options, takes 9.58 s. After slice creation and component deployment, the R&S

Table 2 The *dojot* slice deployment times

Slice instantiation time (s)	1917.2
Reservation time (s)	9.6
Dojot servicedeploy time (s)	505.8
Total	2432.6

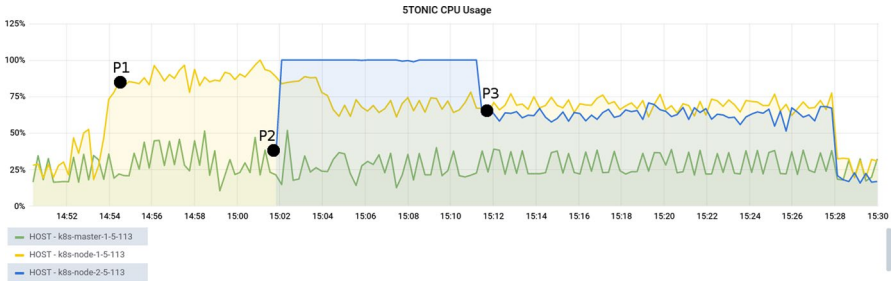
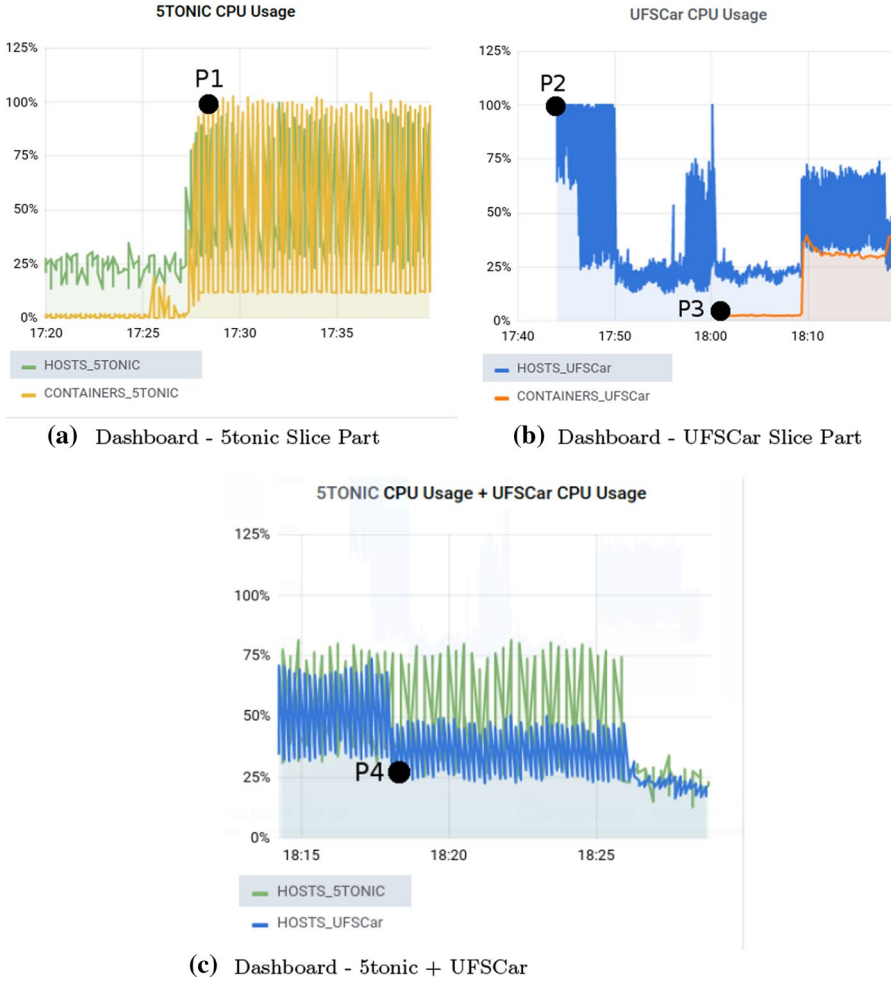


Fig. 13 Vertical elasticity operation taking place

management takes 505.81 s (about 8 min) to deploy the *dojot* platform using Kubernetes as VIMs in the core cloud (Brazil) and in the edge cloud (Spain). This is the time taken from a tenant service deployment request until all the microservices are running properly. The total time elapsed to deploy the slice and to have the services operational is 2432.55 s (about 40 min).

Next, we discuss the monitoring moments when the elasticity operations are triggered. In Fig. 13, we can see the CPU usage for each VM inside the edge cloud in 5Tonic (Spain) and the effects of the vertical elasticity implemented during the test in the first scenario. We highlight three important moments marked as **P1**, **P2**, and **P3**. Point **P1** depicts the time that the infrastructure orchestration first detects an increase in CPU usage as a result of an increased number of requests for the IoT devices. At this moment, the SRO starts the vertical elasticity operation, which aims to deploy a new worker VM in edge cloud 1. After a few minutes, the new VM is deployed and can be seen in the figure as **P2**. Finally, **P3** shows the moment when the requests are equally distributed between the two worker VMs, decreasing the CPU usage of the first worker VM (lines in yellow and blue).

The horizontal elasticity operations follow a similar procedure. However, here we observe a new slice part deployment and distribution of the IoT device requests, as illustrated in Fig. 14a. For the 5tonic CPU usage monitoring, **P1** highlights the moment when the infrastructure orchestration triggers the horizontal elasticity operation, as a result of the high volume of device requests. Figure 14b shows that the edge cloud 2 deployment starts at **P2**, activating the monitoring of the new infrastructure. A few minutes later, the monitoring of the *dojot* microservices by the R&S monitoring is activated, as indicated by **P3**. Finally, Fig. 14c presents the splitting of requests for IoT devices between both edge clouds 1 and 2 (5tonic and UFSCar), as highlighted by **P4**. In this PoC experiment, we validate the following important aspects of the proposed solution:



(a) Dashboard - 5tonic Slice Part

(b) Dashboard - UFSCar Slice Part

(c) Dashboard - 5tonic + UFSCar

Fig. 14 Horizontal elasticity operations taking place

- The instantiation of a real IoT scenario overseas, ensuring the communication between the microservices of the *dojot* platform;
- The execution of management and monitoring tasks for the IoT service, by handling the elasticity operations;
- The presentation of the CNS-AOM functionalities as well as the accuracy of monitored metrics reflecting the actual state of the infrastructure;
- The interoperability capacity in a multi-domain environment and the abstraction of working with different monitoring entities;
- The collection of metrics for the infrastructure and services from two different monitoring entities (Prometheus for the edges and Netdata for the core cloud).

6 Conclusions and Future Work

This paper presented the design and implementation of the CNS-AOM architecture for monitoring, managing, and orchestrating CNSs. The proposed architecture considered the following challenges that arose from this new concept: (i) the heterogeneity of VIMS/WIMs and monitoring entities used in different slice parts; (ii) the monitoring, management, and orchestration of infrastructure and services on demand, considering multiple administrative and technological domains; and (iii) the elasticity operations being performed by the CNS-AOM subsystems in a real overseas CNS instantiation. As far as we know, the proposed implementation is the first attempt to encompass these characteristics, taking a step forward toward the adoption of CNSs. Additionally, we deployed real CNSs hosting two services. First, a CDN slice instantiated across three different cities showed the feasibility of the architecture. The components and the CDN service were presented across different VIMs/WIMs and used different monitoring tools, proving the functionalities of the two proposed components. In summary, after instantiating the CNS using the slice provider platform, the tenant could fully monitor, manage, and orchestrate the CNS and the service through the CNS-AOM subsystems designed and implemented in this research. Notably, the IoT service was deployed among far-away cities in Brazil and Spain, although a considerable amount of time elapsed between the slice request and the *dojot* service deployment. Through the service provided by the *dojot* platform, we successfully tested one of the most important characteristics of CNSs, which is the elasticity operation. The ability to change the amount of resources when detecting a change in usage in the monitored metrics is highly desirable in such environments. In addition, the CNS-AOM capability of handling all the proposed processes was proven, including the adaptation of components after an elasticity operation and service redeployment to take advantage of the new slice infrastructure.

We intend to advance the proposed architecture further by considering the following important aspects as future work: the implementation of other elasticity algorithms (e.g., learning-based), the improvement of the elasticity operation adding different mechanisms to check the SLOs/SLAs, the inclusion of network slice parts with network elements and WIMs, and the security concerns about the CNS management and orchestration.

Acknowledgements Work funded through the H2020 4th EU-BR Collaborative Call, under the grant agreement no. 777067 (NECOS - Novel Enablers for Cloud Slicing). This work was financed also by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

References

1. Silva, F. S. D., Lemos, F. S. D., Medeiros, A., Neto, A. V., Pasquini, R., Moura, D., Rothenberg, C., Mamatas, L., Correa, S. L., Cardoso, K. V., Marcondes, C., Abelem, A., Nascimento, M., Galis, A., Contreras, L., Serrat, J., Papadimitriou, P.: Necos project: Towards lightweight slicing of cloud

federated infrastructures. In *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 406–414, (2018)
2. Clayman, Stuart, Neto, Augusto, Verdi, Fábio., Correa, Sand, Sampaio, Silvio, Sakelariou, Ilias, Mamatas, Lefteris, Pasquini, Rafael, Cardoso, Kleber, Tusa, Francesco, Rothenberg, Christian, Serrat, Joan: The necos approach to end-to-end cloud-network slicing as a service. IEEE Commun. Maga. **59**(3), 91–97 (2021)
3. Galis, A., Makhijani, K., Dong, J., Bryant, S., Boucadair, M., Martinez-Julia, P.: Network Slicing - Introductory Document and Revised Problem Statement draft-gdmb-netslices-intro-and-ps-02. https://datatracker.ietf.org/doc/html/draft-gdmb-netslices-intro-and-ps-02, (2017). Accessed 10 May 2021
4. Geng, L., Galis, A., Dong, J., Makhijani, K., Bryant, S., Galis, A., de Foy, X., Kuklinsk, S.: Network Slicing Architecture draft-geng-netslices-architecture-02. https://datatracker.ietf.org/doc/html/draft-geng-netslices-architecture-02, (2018). Accessed 14 May 2021
5. 3GPP. 3GPP SA5 - Study on management and orchestration of network slicing/Network slice management (3GPP TR 28.801), (2016)
6. ETSI. Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework. ETSI GR NFV-EVE 012 V3.1.1. https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf, (2017). Accessed 12 May 2021
7. ETSI. Next Generation Protocols (NGP); E2E Network Slicing Reference Framework and Information Model. ETSI GR NGP 011 V1.1.1. https://www.etsi.org/deliver/etsi_gr/NGP/001_099/011/01.01.01_60/gr_ngp011v010101p.pdf, (2018). Accessed 12 May 2021
8. Galis, A., Kiran M.: Network Slicing Landscape: A holistic architectural approach, orchestration and management with applicability in mobile and fixed networks and clouds (2018)
9. Galis, A.: Perspectives on Network Slicing—Towards the New 'Bread and Butter' of Networking and Servicing. https://sdn.ieee.org/newsletter/january-2018/perspectives-on-network-slicing-towards-the-new-bread-and-butter-of-networking-and-servicing. Accessed 23 Nov 2020
10. Pasquini, R., Baliosian, J., Serrat, J., Gorricho, J., Neto, A., Verdi, F.: Inferring cloud-network slice's requirements from non-structured service description. In: NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–5 (2020)
11. Medeiros, A., Neto, A., Sampaio, S., Pasquini, R., Baliosian, J.: Enabling elasticity control functions for cloud-network slice-defined domains. In: NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7 (2020)
12. Medeiros, Alisson, Neto, Augusto, Sampaio, Silvio, Pasquini, Rafael, Baliosian, Javier: End-to-end elasticity control of cloud-network slices. Internet Technol. Lett. **2**(4), e106 (2019)
13. de Foy, X., Rahman, A.: Network Slicing - 3GPP Use Case. http://www.ietf.org/internet-drafts/draft-defoy-netslices-3gpp-network-slicing-02.txt, October (2017). Accessed 23 Nov 2020
14. Yousaf, Z., et al.: GR NFV-IFA 022 - V3.1.1—Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on Management and Connectivity for Multi-Site Services. https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/022/03.01.01_60/gr_NFV-IFA022v030101p.pdf, (2018). Accessed 23 Nov 2020
15. Thalanany, S., et al.: 5g end-to-end architecture framework v3.0.8. https://www.ngmn.org/wp-content/uploads/Publications/2019/190916-NGMN_E2EArchFramework_v3.0.8.pdf, (2017). Accessed 22 Nov 2020
16. Beltrami, A., Maciel, P.D., Tusa, F., Cesila, C., Rothenberg, C., Pasquini, R., Verdi, F.L.: Design and implementation of an elastic monitoring architecture for cloud network slices. In: NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7 (2020)
17. Galis, A., Tusa, F., Clayman, S., Rothenberg, C., Serrat, J.: Slicing 5G Networks: An Architectural Survey. In: Wiley 5G Ref: The Essential 5G Reference, pp. 1–41. Wiley, New York (2020)
18. Farrel, A., Gray, E., Drake, J., Rokui, R., Homma, S., Makhijani, K., Contreras, L.M., Tantsura, J.: Framework for IETF Network Slices. Internet-Draft draft-ietf-teas-ietf-network-slices-04, Internet Engineering Task Force. Work in Progress (August 2021)
19. Swapna, A.I., Rosa, R.V., Rothenberg, C.E., Sakellariou, I., Mamatas, L., Papadimitriou, P.: Towards a marketplace for multi-domain cloud network slicing: use cases. In: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 1–4 (2019)
20. Maciel Jr., P.D., Verdi, F.L., Valsamas, P., Sakellariou, I., Mamatas, L., Petridou, S., Papadimitriou, P., Moura, D., Swapna, A.I., Pinheiro, B., Clayman, S.: A marketplace-based approach to cloud

network slice composition across multiple domains. In: 2019 IEEE Conference on Network Softwarization (NetSoft), pp. 480–488 (2019)
21. ITU-T. Y.3001 : Future networks: Objectives and design goals. https://www.itu.int/rec/T-REC-Y.3001-201105-I, 2012. Accessed 12 May 2021
22. ITU-T Focus Group. IMT-2020 Deliverables. https://www.itu.int/en/publications/Documents/tsb/2017-IMT2020-deliverables/mobile/index.html#p=1. Accessed 12 May 2021 (2017)
23. NGMN. NGMN 5G White Paper v1.0. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf. Accessed 12 May 2021 (2015)
24. Hedman, P, NGMN P1 WS1 E2E Architecture Team.: Description of Network Slicing Concept 160113. https://www.ngmn.org/wp-content/uploads/160113_NGMN_Network_Slicing_v1_0.pdf. Accessed 12 May 2021 (2016)
25. Homma, S., Nishihara, H., Miyasaka, T., Galis, A., Ram OV, V., Lopez, D., Contreras, LM., Martinez-Julia, P., Qiang, L., Rokui, R., Ciavaglia, L., de Foy, X.: Network Slice Provision Models draft-homma-slice-provision-models-02. https://datatracker.ietf.org/doc/html/draft-homma-slice-provision-models-02. Accessed 14 May 2021 (2019)
26. Galis, A..: Network Slicing - Revised Problem Statement draft-galis-netslices-revised-problem-statement-01. https://datatracker.ietf.org/doc/html/draft-galis-netslices-revised-problem-statement-01. Accessed 14 May 2021 (2017)
27. Geng, L., Qiang, L., Ordonez, J., Adamuz-Hinojosa, O., Ameigeiras, P., Lopez, D., Contreras, L.: COMS Architecture draft-geng-coms-architecture-02. https://datatracker.ietf.org/doc/html/draft-geng-coms-architecture-02. Accessed 14 May 2021 (2018)
28. Makhijani, K., Qin, J., Ravindran, R., Geng, L., Qiang, L., Peng, S., de Foy, X., Rahman, A., Galis, A., Fioccola, G.: Network Slicing Use Cases: Network Customization and Differentiated Services draft-netslices-usecases-02. https://datatracker.ietf.org/doc/html/draft-netslices-usecases-02. Accessed 14 May 2021 (2017)
29. Qiang, L., Geng, L., Makhijani, K., de Foy, X., Galis, A.: The Use Cases of Common Operation and Management of Network Slicing draft-qiang-coms-use-cases-00. https://datatracker.ietf.org/doc/html/draft-qiang-coms-use-cases-00. Accessed 14 May 2021 (2018)
30. Qiang, L., Galis, A., Geng, L., Makhijani, K., Martinez-Julia, P., Flinck, H., de Foy, X.: Technology Independent Information Model for Network Slicing draft-qiang-coms-netslicing-information-model-02. https://datatracker.ietf.org/doc/html/draft-qiang-coms-netslicing-information-model-02. Accessed 15 May 2021 (2018)
31. Galis, A., Makhijani, K., Yu, D., Liu, B.: Autonomic Slice Networking draft-galis-anima-autonomic-slice-networking-05. https://datatracker.ietf.org/doc/html/draft-galis-anima-autonomic-slice-networking-05. Accessed 10 May 2021 (2018)
32. 3GPP. 3GPP SA2—Study on Architecture for Next Generation System /Network slice related functionality (3GPP TR 23.799) (2015)
33. 3GPP. 3GPP SA2—System Architecture for the 5G System /Network slice related functionality (3GPP TS 23.501) (2016)
34. 3GPP. 3GPP SA2—Procedures for the 5G System: Procedures and flows of the architectural elements/Network slice related procedures (3GPP TS 23.502) (2016)
35. 3GPP. 3GPP SA3—Study on the security aspects of the next generation system/ Network slice related security (3GPP TR 33.899) (2016)
36. 3GPP. 3GPP SA5—Provisioning of network slicing for 5G networks and services: Detailed specification of network slice provisioning/Network slice management (3GPP TS 28.531) (2017)
37. 3GPP. 3GPP SA5—Management of network slicing in mobile networks - concepts, use cases and requirements (3GPP TS 28.530) (2017)
38. ETSI.: Network Functions Virtualisation—White Paper on NFV priorities for 5G. https://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf. Accessed 12 May 2021 (2017)
39. Open Network Foundation.: TR-526 Applying SDN Architecture to 5G Slicing. https://opennetworking.org/wp-content/uploads/2014/10/Applying_SDN_Architecture_to_5G_Slicing_TR-526.pdf. Accessed 12 May 2021 (2016)
40. NECOS.: D3.1: NECOS System Architecture and Platform Specification. V1 Deliverable. http://www.maps.upc.edu/public/NECOS%20D3.1%20final.pdf. Accessed 10 Oct 2020 (2019)
41. Almeida, L., Maciel Jr. P.D., Verdi, F.L.: Cloud Network Slicing: A systematic mapping study from scientific publications. https://doi.org/10.21203/rs.3.rs-34336/v1 (2020)

42. Bonnet, J., et al.: D4.2 service platform first operational release and documentation. https://bscw.5g-ppp.eu/pub/bscw.cgi/d137267/SONATA%20D4.2%20Service%20platform%20first%20operational%20release%20and%20documentation.pdf. Accessed 10 Oct 2020 (2016)
43. Tusa, F., Clayman, S., Galis, A.: Real-time management and control of monitoring elements in dynamic cloud network systems. In: 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pp. 1–7 (2018)
44. Tusa, F., Clayman, S., Valocchi, D., Galis, A.: Multi-domain orchestration for the deployment and management of services on a slice enabled nfvi. In: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1–5 (2018)
45. Mamatas, L., Clayman, S., Galis, A.: A service-aware virtualized software-defined infrastructure. IEEE Commun. Mag. **53**(4), 166–174 (2015)
46. Manvi, Sunilkumar S., Shyam, Gopal Krishna: Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. J. Network Comput. Appl. **41**, 424–440 (2014)
47. Jennings, Brendan, Stadler, Rolf: Resource management in clouds: survey and research challenges. J. Network Syst. Manag. **23**, 03 (2014)
48. Montero, Rafael, Agraz, Fernando, Pagès, Albert, Spadaro, Salvatore: Enabling multi-segment 5g service provisioning and maintenance through network slicing. J. Network Syst. Manag. **28**(2), 340–366 (2020)
49. Chiha, Asma, Van der Wee, Marlies, Colle, Didier, Verbrugge, Sofie: Network slicing cost allocation model. J. Network Syst. Manag. **28**(3), 627–659 (2020)
50. Valera-Muros, Barbara, Panizo, Laura, Rios, Alvaro, Merino-Gomez, Pedro: An architecture for creating slices to experiment on wireless networks. J. Network Syst. Manag. **29**(1), 1 (2020)
51. Luong, D.-H., Thieu, H.-T., Outtagarts, A., Ghamri-Doudane, Y.: Cloudification and autoscaling orchestration for container-based mobile networks toward 5g: experimentation, challenges and perspectives. In: 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), pp. 1–7 (2018)
52. Sukhija, N., Bautista, E.: Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In: 2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 257–262 (2019)
53. NECOS.: D5.2: Intelligent Management and Orchestration. http://www.maps.upc.edu/public/D5.2%20final.pdf. Accessed 28 Nov 2020 (2019)
54. CPqD.: Dojot documentation. https://dojotdocs.readthedocs.io/en/latest/. Accessed 10 November 2020 (2020)

André Luiz Beltrami Rocha received an M.Sc. degree in Computer Science from Federal University of São Carlos in December 2016. He is currently working towards a Ph.D. in Computer Science at Federal University of São Carlos, in the scope of orchestration and management of cloud-network slices. In January 2018, he joined the NECOS Project. His research interests include 5G networks, network virtualization, network slicing, cloud-network slicing, monitoring, orchestration, data centers and cloud computing.

Celso Henrique Cesila holds the Computer Engineering degree from the São Francisco University (USF), and an M.Sc. degree in Computer Engineering from UNICAMP, 2020. He is currently a Ph.D. candidate at the Department of Computer Engineering and Industrial Automation of the School of Electrical and Computer Engineering - UNICAMP). He is involved as a researcher in academy-industry collaboration projects related to networked systems (e.g., SDN, NFV, IBN). Since 2013 he has been a Computer Analyst at the Institute of Mathematics - UNICAMP. His research interests span SDN, NFV, IBN, networking monitoring, and networking slice.

Paulo Ditarso Maciel Jr. holds a doctorate's degree in Computer Science from the Federal University of Campina Grande (UFCG), a master's degree in Systems and Computing Engineering from the Federal

University of Rio de Janeiro (UFRJ), and a bachelor's degree in Computer Science from the Federal University of Paraíba (UFPB). He also held a postdoc position at the Computing Department of the Federal University of São Carlos (UFSCar-Sorocaba) for two years. As Associate Professor in the Federal Institute of Education, Science and Technology of Paraíba (IFPB), he works in the undergraduate and graduate courses of the Academic Unit of Informatics, and his research interests include computer networks, network management, cloud computing, and virtualization.

Sand Luz Correa is an associate professor at the Institute of Informatics - Universidade Federal de Goiás (UFG), where she has been a professor and researcher since 2010. She holds a degree in Computer Science from Universidade Federal de Goiás (1994), has MSc (1997) in Computer Sciences from University of Campinas (UNICAMP) and PhD (2011) in Informatics from Pontifical Catholic University of Rio de Janeiro (PUC-Rio). In 2015, she spent her sabbatical at Virginia Tech (in the USA) and, in 2020, at Inria Saclay Research Centre (in France). Professor Sand has participated in some international research projects (including two from joint calls BR-EU). Her research interests include cloud computing, SDN, data plane programmability, and sensor systems and smart city platforms.

Javier Rubio-Loyola received the Engineering degree in communications and electronics and the M.Sc. degree in digital systems from the National Polytechnic Institute of Mexico and the Ph.D. degree in telecommunications from the Universitat Politécnica de Catalunya, Barcelona. He is a Research Scientist with the Center for Research and Advanced Studies, Mexico. He has participated in a number of Spanish and IST-European research projects mainly in the network management area. His research interests focus on network management, network functions virtualization, and software-defined networks.

Christian R. Esteve Rothenberg is an Assistant Professor in the Faculty of Electrical & Computer Engineering (FEEC) at University of Campinas (UNICAMP), Brazil, where he received his Ph.D. and currently leads the Information & Networking Technologies Research & Innovation Group (INTRIG). His research spans all layers of distributed systems and network architectures and are often carried in collaboration with industry, resulting in multiple open source projects among other scientific results.

Fábio Luciano Verdi is an Associate Professor in the Computer Science Department at Federal University of São Carlos (UFSCar). He has been working with data centers, cloud computing, SDN, and dataplane programmability. He is a member of the LERIS Research Group and has been leading projects in the area of computer network monitoring, virtual resources and cloud infrastructures. He is currently a research visitor at KTH, Sweden.

Authors and Affiliations

André Luiz Beltrami Rocha[1] · Celso Henrique Cesila[2] · Paulo Ditarso Maciel Jr.[3] · Sand Luz Correa[4] · Javier Rubio-Loyola[5] · Christian Esteve Rothenberg[2] · Fábio Luciano Verdi[1]

✉ André Luiz Beltrami Rocha
andrebeltrami@estudante.ufscar.br

Celso Henrique Cesila
ccesila@dca.fee.unicamp.br

Paulo Ditarso Maciel Jr.
paulo.maciel@ifpb.edu.br

Sand Luz Correa
sandluz@ufg.br

Javier Rubio-Loyola
javier.rubio@cinvestav.mx

Christian Esteve Rothenberg
chesteve@dca.fee.unicamp.br

Fábio Luciano Verdi
verdi@ufscar.br

[1] Computing Department (DCOMP), Federal University of São Carlos (UFSCar), Sorocaba, Brazil

[2] School of Electrical and Computer Engineering (FEEC), University of Campinas (Unicamp), Campinas, Brazil

[3] IFPB, João Pessoa, Brazil

[4] UFG, Goiânia, Brazil

[5] Cinvestav, Ciudad de Mexico, Mexico