2022 4th International Workshop on Performance Evaluation of Next Generation Virtualized Environments and
Software-Defined Networks

Orchestration Software for Resource Constrained Datacenters: an Experimental Evaluation

Alexandros Valantasis
University of Thessaly
Volos, Greece
valantas@uth.gr

Nikos Makris
University of Thessaly and CERTH
Volos, Greece
nimakris@uth.gr

Thanasis Korakis
University of Thessaly and CERTH
Volos, Greece
korakis@uth.gr

2022 IEEE 8th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0694-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/NetSoft54395.2022.9844043

***Abstract*—The evolution of the cloud-computing technology has allowed the instantiation of resources almost anywhere. Handheld devices, edge/fog resources, and core cloud datacenters comprise a resource continuum that can be used for hosting almost any service. The rise of micro-services has allowed any application to be hosted over any type of compute resource, regardless of the underlying hardware architecture. In this work, we focus on the far-edge devices that participate in the resource continuum, located at the network access or the fog, and are usually resource constrained. We evaluate two lightweight frameworks which can be used for orchestrating micro-services on top of them. Our evaluation presents experimental evidence in terms of their capabilities for instantiating/tear-down of network services, and their dynamic adaptation to external workloads by using the respective horizontal scaling solutions, when tested under the same experimental environment.**

***Index Terms*—cloud-to-things continuum, orchestration, resource constrained edge, Nomad, K3s, autoscaling**

I. Introduction

Cloud-native networking and applications provide advanced flexibility for the deployment of network services across heterogeneous infrastructure elements, often supporting different system architectures. Cloud-native approaches have gained a lot of attention, as they are driven by the wide adoption of micro-services and provide mechanisms for the overall monitoring and management of the deployed applications/services, based on the observed behavior/metrics. The rise of micro-services and the respective hypervisor software enable the instantiation of different network functions regardless of the specific architecture of the hosting worker node, thus allowing the creation of a resource continuum that spans the user end-devices, the network edge and the core cloud systems.

Since computational power even in handheld devices is continuously rising, an opportunity is created for even more devices to participate in a joint access-fog-edge-cloud continuum of resources; the resource continuum can be used for dynamically deploying/instantiating/migrating services to/from central cloud towards providing higher data security and integrity, and faster algorithm execution without the overhead of transmitting data over long distances. Moreover, network devices and users can benefit from local decisions and analytics with minimal communication overhead, compared to transmitting data to the core cloud. Nonetheless, orchestrating services within such a resource continuum can be challenging, as the devices that are closer to the edge (e.g. the end-devices, IoT infrastructure) might present links with fluctuating performance for connecting to the network (e.g. wireless links) or might have a nomadic behaviour (appearing/disappearing from the network). To this aim, the orchestration and workload sharing among such compute nodes has to be carefully considered in order to meet the requirements in terms of resources per each application/service workload, while the compute nodes are connected to the network.

The evolution of orchestration software is constantly augmenting the network with new capabilities that need to be also considered when dealing with resource-constrained equipment, such as the automatic scaling of services. Most of such frameworks nowadays support the horizontal scaling process; subject to a monitored metric for the deployed services, a new replica can be instantiated when the metric has reached a trigger value, and through proxy services traffic is distributed among the replicas. Similar to the aforementioned challenges, the frameworks need to consider the worker nodes that will host the replicas when autoscaling in the resource continuum.

In this work, we focus on the case of orchestrating services at the far-edge network, which are traditionally resource-constrained, and can be considered as handheld devices/IoT gateway platforms. The limitation in the available resources poses several constraints to the core orchestration software, particularly regarding monitoring performance and service instantiation, in the case that the workloads are not properly balanced among the available devices. Given this behaviour, the orchestration of services in such environments have to meet certain demands in terms of instantiation times, tear-down times, autoscaling and scheduling decisions.

Although there are a plethora of different orchestration frameworks, some of them suitable for resource-constrained devices, they do not support service data management and analytics or horizontal scaling mechanisms, a vital disadvantage for the optimal and efficient usage of compute resources. For this reason we delve into two lightweight frameworks that are the fastest solutions as shown in our prior work [10] and suitable for the orchestration of micro-services over resource-

The research leading to these results has received funding from the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number No 101008468 (H2020 SLICES-SC). The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.

constrained equipment, while providing monitoring and scaling of the instantiated micro-services. The two frameworks we concentrate on are Nomad [1] and K3s [2].

To this aim, we evaluate their performance for the tasks of deploying and tearing down a service setup, under different experimental settings. Since the frameworks provide the capabilities to automatically scale the deployed services based on monitored metrics, we concentrate on determining common metrics and same triggering mechanisms for the autoscaling process and benchmark their operation.

The rest of the paper is organized as follows: Section II presents relevant works in literature which are our motivation. Section III presents our system architecture and experimental setup. Section IV describes our experimental results and discusses the applicability of each framework. Finally, in Section V we conclude our work.

II. Related Work

Orchestration on resource constrained devices has gained lots of attention in relevant literature, as the end-devices continue to gain more computational power. They usually apply more to the IoT sector, where the end-devices (e.g. gateways) can host small workloads of micro-services. In [3], the applicability of the Docker framework as an edge orchestrator framework is evaluated. The authors evaluate Docker Swarm regarding four different requirements namely the deployment and termination of instantiated services, the resource service management, fault tolerance and caching, and conclude that Docker is a viable candidate for orchestrating micro-services in edge and fog computing deployments.

In [4], authors explore the resource continuum between cloud, edge and fog in order to eliminate gaps in resources between the different infrastructure layers in a dynamic IoT environment. They further implement a monitoring system augmented with Artificial Intelligence functionalities that integrates with the orchestrator software which handles the migration of micro-services hosted on edge nodes horizontally (between edge nodes) and vertically (between edge and fog nodes) based on the load conditions of the hosted infrastructure. Similarly, in [5], authors compare three different orchestration tools based on their capabilities to manage cluster nodes, the deployment of the application on a specific host according to its needs and the ability of the orchestrator to ensure mapping of device resources (CPU, RAM, Storage, Serial ports) to the containerized applications. Authors in [6] develop a fog computing framework in order to run applications on resource constrained devices such as Raspberry Pi. In fact, after their comparison between multiple orchestrators (Docker Swarm, Kubernetes, Apache Mesos) they conclude on using Docker Swarm with several extensions in order to create IoT applications on fog infrastructure. In [7], authors create clusters of resource-constrained devices and present their efforts regarding an Edge Cloud Platform as a Service (PaaS). They evaluate the same frameworks and build extensions to the Docker Swarm orchestrator in conjunction with the TOSCA standard, for effectively handling their cluster.

As network conditions for such devices may fluctuate, some works focus on the scheduling decisions for the worker nodes. In [8], the expediency and performance of the Kubernetes framework is evaluated for deploying and orchestrating distributed IoT containerized services over resource-constrained Raspberry-Pi devices. A set of extensions to the core Kubernetes framework is suggested, for making it compatible with fog-computing applications, while their evaluation proves to be more efficient in terms of communication cost and optimal node selection. Authors in [9] introduce a network aware approach for Kubernetes with a more efficient resource allocation scheduling for IoT based services in a fog environment. They provide evidence on the efficiency of their mechanism, introduced as an extension to the Kubernetes scheduler, in terms of service provisioning regarding network latency. In our prior work [10] we provided a deep comparison of the state-of-the art orchestration software regarding the deployment time of different virtual-machines and micro-services. We showed that Nomad and Kubernetes (K8s) are the fastest solutions regarding the other competitors (e.g. the Eclipse Fog05 [11] framework) for the deployment of applications in datacenters.

Most of existing literature focuses on frameworks that have been designed for datacenter operations, and apply/extend them to meet resource constrained resources in the edge domain. In this work, we seek to quantify the differences in instantation and tear-down time on frameworks that are best suited for resource constrained devices. We start from frameworks that are built around this ecosystem, and assume that resources can dynamically appear/disappear from the resource pool, and thus can more efficiently orchestrate services over such equipment. The frameworks are initially evaluated in a similar manner as in [12], for different metrics and varying load of services. We progress beyond such works by evaluating 5G-oriented services, and exploring the autoscaling features of the frameworks, for replicating services according to the load that each one receives. In the following section, we detail our system architecture and the under-study frameworks.

III. System Architecture and Testbed Setup

In this section we describe the overall system architecture in conjunction with the selected tools and methodology used in the experimental process. In order to create our resource-constrained datacenter, we used a 3-node cluster setup hosted in the NITOS testbed [13]. The nodes used are based on the low-cost single board Raspberry Pi (Rpi) 3 Model B+ devices. Such devices offer significantly better response times for the end-user and faster processing speed for low computational tasks than their previous generation, matching the computational power that a contemporary fog device shall have. A complete list of the device and the selected framework releases with respect to the experimental scenario is shown in Table I.

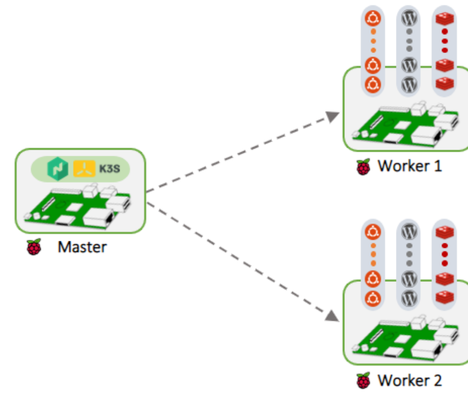
Regarding the experimental architecture the nodes are organized in a 3-node cluster setup fully isolated from the rest of the testbed, as shown in Figure 1. One node is selected to act as a master node, while the remaining two as workers. The former hosts the under-study orchestration frameworks, while

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.

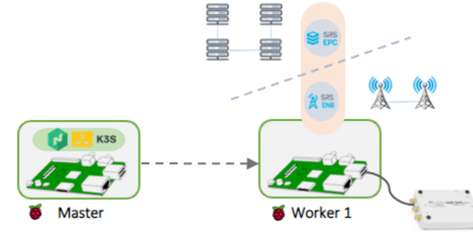
TABLE I: Equipment parameters

Equipment Parameters	Values
Processor	64-bit quad-core ARM Cortex-A53 1.4GHz
RAM	1 GB LPDDR2 SDRAM
Hard Disk	15 GB HDD
Operating System	Raspbian GNU/Linux 10 (buster)
Networking	100 Mbps Ethernet
Docker images	Ubuntu:18.04.03 (bionic), Redis:3.2, Wordpress:latest
K3s version	v.1.18.6+k3s1, commit: 6f56fa1, branch: master[1]
K3s Autoscaler	v2beta2
Nomad version	v.0.12.0, commit: 8f7fbc8, branch: master[2]
Nomad Autoscaler	v.0.1.0[3]
USRP device	USRP B205mini-i

[1] https://github.com/rancher/k3s
[2] https://github.com/hashicorp/nomad/
[3] https://releases.hashicorp.com/nomad-autoscaler/0.1.0/



(a) Deployment of three services on distributed nodes



(b) Deployment of the srsLTE stack on a single node

Fig. 1: Testbed Deployment for measurement collection

the latter are the compute infrastructure where the applications and micro-services for our experiments are deployed.The two under-study frameworks, which have been selected as the optimal approaches for orchestration in resource constrained devices are Hashicorp Nomad [1] and K3s [2].

Nomad has been designed and developed by HaschiCorp, and is based and optimized on the principle that nodes used as the compute infrastructure can present a nomadic behaviour (e.g. a Smartphone that is present in an area), thus making it ideal for fog computing setups. Nomad is optimized for orchestration of applications as docker containers. In addition, Nomad provides an autoscaling daemon, namely *Nomad Autoscaler* [14], equipped with the necessary plugins in order to support the connection with metrics sources and scaling both targets and algorithms. Nomad Autoscaler supports Horizontal application autoscaling, Horizontal Cluster autoscaling and Dynamic application sizing.

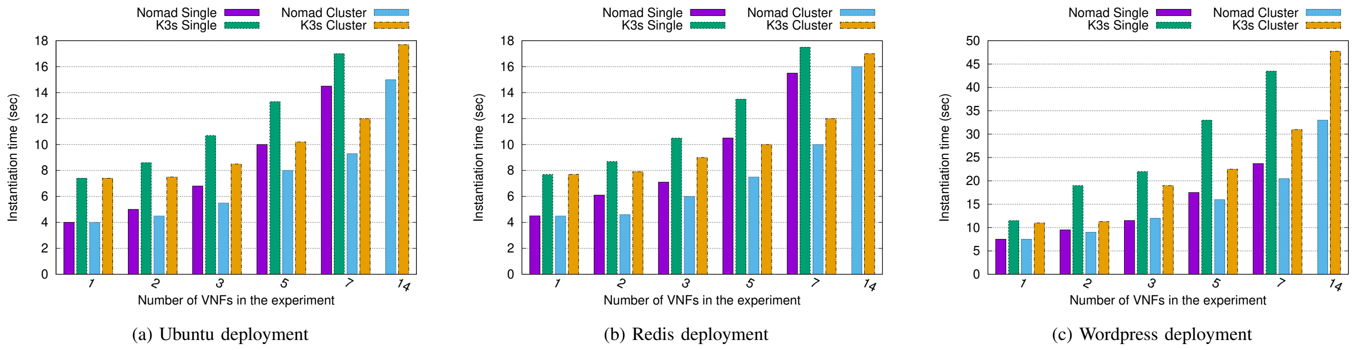
On the other hand, K3s is a light-weight form of the Kubernetes (K8s) framework [15], which is widely considered for edge-based orchestration. Nowadays, K8s claims to be one of the most successful open-source orchestration platform not only compatible with docker containers, but also suitable for the majority of container technologies. K8s provides integrated monitoring functionalities as well, for checking and healing the application's health, efficient resource allocation of resources and storage organization, automated load-balancing and replication of high traffic containers, management application's runtime state and control of the deployments and updates. K3s is a lightweight distribution of the K8s framework, designed and developed especially for IoT and edge computing use cases. Its simplified architecture is infrastructure agnostic, allowing it to run over several platforms, spanning from ARM based edge devices to large datacenters. Regarding the architecture, K8s and K3s consist of a plethora of components offering management of the deployed instances, as well as setup and configuration of the underlying network between the applications, and the manner in which they are exposed, using dedicated services for DNS and proxy entry-points for the applications. In addition, they provide a set of APIs and tools employing both a resource and a controller component in order to perform autoscaling operations. The provided autoscaler functionality provides several functionalities, supporting both pod and node based scaling, namely Horizontal, Vertical and Cluster autoscaling.

Regarding the orchestration process, we present two different types of experiments. Initially, regarding the first type of experiments, we experiment with four different scenarios of variable complexity for the deployed application. Through these experiments, we foresee to test and compare the orchestrator frameworks (Nomad and K3s) for applications with different load characteristics. For the second type of experiments we test the orchestrator engines regarding their autoscaling mechanisms in order to evaluate their behavior under more hardware independent scenarios.

We choose to compare these orchestrator engines due to the claimed fast deployment times and the variety of integrated functionalities offered by K3s and K8s. Since the resources that we use as our fog datacenter are heavily constrained (for the Raspberry-pi 3B+ device, RAM is limited to 1GB), running K8s on top is not a feasible solution. As a result we use the lightweight distribution of K8s, the K3s framework.

The first type of experiments span three different scenarios based on the deployed docker images: 1) a simple Ubuntu 18.04 version bionic (45.8 Mb), 2) a Redis version 3.2 (57.9 Mb) and 3) a WordPress (407 Mb) installation that moves from the very low complexity Ubuntu to a higher complexity WordPress scenario for instantiation. In addition, in order to stress test the frameworks, we employed an Ubuntu based docker image that hosts the srsLTE [16] open source LTE software and creates a real 4G network with a compatible

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.



(a) Ubuntu deployment

(b) Redis deployment

(c) Wordpress deployment

Fig. 2: Deployment time for deployed VNFs

Software Defined Radio (SDR) front-end. The aforementioned image hosts together the eNodeB and Evolved Packet Core (EPC) as the running applications.

Regarding the second type of experiments, we experiment with the horizontal autoscaling mechanisms of the orchestrator platforms. To this need we employed a custom docker image (262 MB) deploying a web server. The respective image serves a PHP webpage which performs very intensive CPU computations whenever a request is received. In order to generate load and test each orchestartor's horizontal autoscaling mechanism we used exactly the same procedure by creating a script that sends an infinite loop of queries to the service. As a result to stress the container's CPU in order to exceed a configured target CPU limit and force the frameworks to spawn a number of new replicas. We configure both solutions with the same set of tools deployed in the 3-node cluster that support the scaling process; *cadvisor*, *node-exporter*, *prometheus* and *Grafana* are deployed as containers in the system, allowing their use from both frameworks for scraping metrics and visualizing them. The metric which we monitor, and the frameworks evaluate for scaling the deployed services is the average CPU utilization of the deployed container regarding the maximum limit of its CPU resources as described by equation 1,

$$\frac{\sum_{i=1}^{N} \mathbf{R(container_CPU_usage_seconds)} * \mathbf{1400}}{\sum_{i=1}^{M} \mathbf{container_CPU_shares}} * 100 \tag{1}$$

where M is the number of deployed containers on the worker node, N is the subset of services changing the specific metric on a container, R is the rate function of the specific metric updated over the last minute, the *container_cpu_usage_seconds* is the time that a monitored container occupies the worker node's CPU, multiplied by the maximum number of MHz supported by our worker nodes (1400 MHz) and *container_CPU_shares* is the maximum shares in CPU power that each container is limited to get from the docker hypervisor (measured in MHz). The metric is further handled to provide us with the percentage of CPU utilization.

IV. Experimental Evaluation

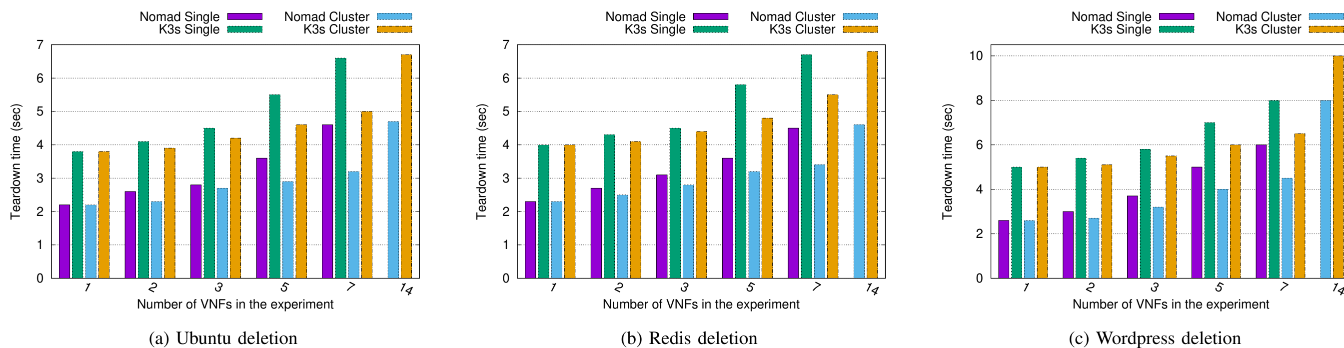
In this section we present our experimental results. As we mentioned we benchmark the orchestrator frameworks (Nomad, K3s) regarding the deployment and tear-down time of a number of services with different load characteristics. We also compare the horizontal autoscaling functionality of the frameworks. In order to present more accurate experiment results we run every case of each scenario 20 different times and present the mean values per each experiment.

For the first three scenarios (Ubuntu, Redis, WordPress) we compare the two orchestrator frameworks regarding the provisioning time needed for the deployment and tear-down of the services. Regarding the deployment time, we measure from the time we request to deploy the services from the orchestration engine until the services become fully functional and the deployed applications have network access. Specifically for the srsLTE scenario, the deployment time is measured from the time we request to deploy the service until the core network (EPC) and eNodeB applications are launched and communicate with each other, so as the LTE network is fully functional. Similarly, for the tear-down time we measure from time we request to destroy the deployed services from the orchestrator until they are actually deleted from the worker nodes, including their attached networks. For each scenario we present values of an increasing number of deployed applications with exactly the same docker image hosted both on a single worker node or distributed equally on different worker nodes. For all the different scenarios, the respective docker images are already present on the worker nodes, in order for us to test the actual deployment time for each of the orchestration frameworks, without the additional time and the fluctuating network overhead to fetch the images from a local or remote registry.

A. Load-based Measurements

1) Ubuntu-Zero load scenario: For the first experimental scenario we measure the provisioning time needed for the instantiation of simple Ubuntu 18.04.03 docker container on a single- and multi-node setup, when workloads are distributed equally on different worker nodes. For each case we have a limitation of up to 7 deployed containers per compute node due to the physical limitations from the Rpi's resources. As we range the number of containers for the different setups (single and distributed setups), Nomad presents better behavior regarding the deployment time than K3s (Figure 2a). K3s is consistently 2-3 sec slower for all the experiment cases regarding the different number of deployed containers. In fact, up to the deployment of seven containers, Nomad proves to be

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.



(a) Ubuntu deletion

(b) Redis deletion

(c) Wordpress deletion

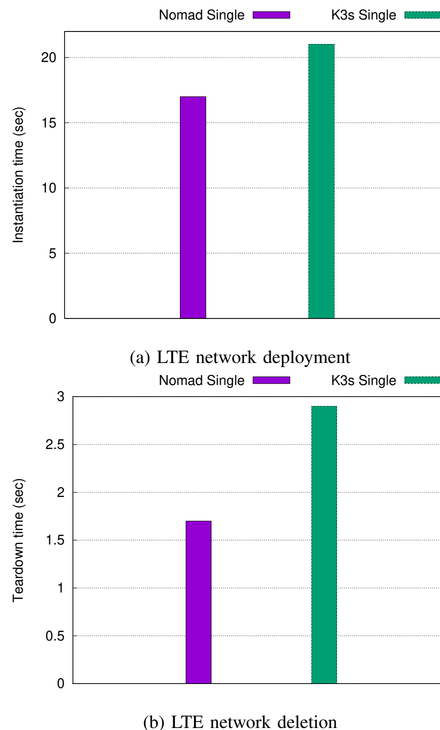
Fig. 3: Deletion time for deployed VNFs

significantly faster for single and distributed nodes cases than K3s. The obtained results are illustrated in Figure 2a. Similarly as we can observe in Figure 3a the tear-down times of Ubuntu based services seems to have the same chronological pattern with the deployment time. Nomad orchestrator engine is approx. 1.6 sec faster for small number of instantiated services, while after the deployment of 5 and 7 Ubuntu containers this time interval between the orchestrators increases up to 2 secs.

2) Redis-Low load scenario: For the second scenario, we orchestrated a low complexity Redis application. The instantiated services provide an open source networked, in-memory, key-value data store that operates as a data structure server. As we can observe in both Figures 2b and 3b there are not significant deviations regarding the deployment and tear down times between the specific application and the previous experiment scenario, due to their similar load levels.

3) Wordpress-High load scenario: Similarly as before, we measure the provisioning time for the deployment and tear-down of a WordPress application. The deployed application provides a content management system (CMS) based on PHP and MySQL. For every case of different number of deployed services, for both orchestrators frameworks, the processes of instantiation and deletion of WordPress applications is slower than before due to the high complexity workload level of the service. Regarding the tear down time of the instantiated WordPress services, K3s is up to 2 sec slower than Nomad for every number of deployed applications. Similarly, focusing on the deployment time, Nomad framework presents again better behavior than K3s for all the deployment cases while in contrast with K3s presents a gradually increment of deployment time regarding the number of instantiated containers. In fact, as we change the number of containers, we observe that K3s framework is affected and presents an exponential increase in the deployment time especially for the cases of 5 and 7 instantiated containers hosted on a single compute node. Similar performance is observed for the case of 14 instantiated containers distributed equally over the two compute nodes. As a result, we can conclude that K3s framework faces performance issues when deploying on the same host a major number of high load demanding services. The obtained results are illustrated in Figures 2c and 3c.

4) srsLTE-Very High load scenario: We conclude the load-based experiments by stress testing the reference architecture



(a) LTE network deployment

(b) LTE network deletion

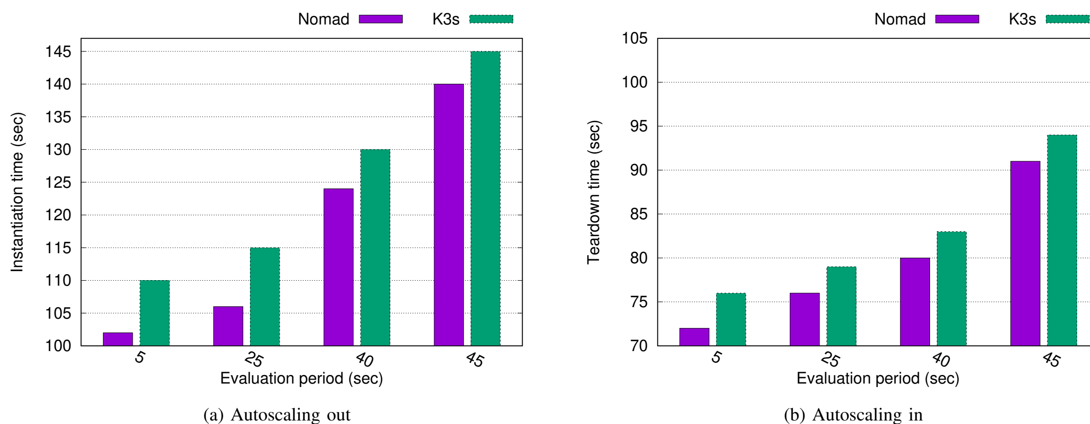
Fig. 4: Deployment and Tear-down time for the srseNB platform

with the deployment of an application that hosts the LTE software and creates a real 4G network. Due to the very high resource demanding workload, as we can observe the deployment time of a single container from both orchestrator platforms is at least twice higher compared to the WordPress scenario. In addition, focusing on the comparison between the two frameworks, Nomad presents better behavior regarding both the deployment and tear-down time of a single service as it is 4 and 1.5 sec faster accordingly. The obtained results are illustrated in Figures 4a and 4b.

B. Autoscaling Measurements

Similarly with the previous scenarios, for the horizontal autoscaling experiment we compare the two orchestrator frameworks regarding the provisioning time needed for scaling out/in the deployments under the same settings. Regarding the deployment time, we measure from the time we start to generate load to the deployed container until the creation of

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.



(a) Autoscaling out

(b) Autoscaling in

Fig. 5: Autoscaling evaluation for the two under-study frameworks

two new replicas (scale out). Similarly, for the tear-down time we measured from time we stop the generated load until the deletion of the two deployed replicas (scale in). For both frameworks, we use the same intervals for scraping metrics from the deployed application with *Prometheus*, intervals for evaluating metrics at the orchestrator side (comparing with the target value), evaluation windows and cool-down timers.

For both frameworks, we set the target limit of the average CPU utilization metric equal to 40 percent for both scale out and scale in operations. As we can observe in Figure 5a, the deployment time for both frameworks are above 100 sec, due to the time waiting until the generated load reach the target of the average CPU utilization metric. In order to examine the autoscaler behavior, we experimented with different evaluation periods. In fact, the autoscalers evaluations periods were tuned to 5, 25, 40, 45 seconds accordingly, which describes how often the autoscaler policy will be evaluated in order to make scaling decisions. For both cases (scale out and scale in), Nomad autoscaler presents a faster behavior than K3s. As shown in Figures 5a and 5b, with respect to the scale out operation, Nomad is constantly 5-9 sec faster than K3s on deploying 2 new replicas for all the cases of different evaluation metrics. Similarly, regarding the scale in operation, Nomad autoscaler is proven to be better, as it is 2-3 sec faster on deleting the 2 new replicas for every case of different evaluation period.

V. Conclusion

In this work, we explored the potential of resource-constrained end devices participating in a resource continuum spanning the end-user access, edge, fog and core network. In this context, we focused on two frameworks that are advertised as ideal for light-weight deployments, namely Nomad and K3s. Nomad shows that in all the under-study cases for deployment/tear-down/autoscaling of services outperforms K3s. This fact and bundled with the existing results from our previous work, places Nomad as the ideal solution for fast deployment of services across the entire resource continuum, especially when considering imminent fluctuations in the end-user connectivity with the rest of the infrastructure. In the future, we foresee to delve into the opportunistic federation of cloud infrastructures, based on the estimated workload, subject to end-user limitations (e.g. battery life, network capacity).

References

[1] "Nomad by HashiCorp," [Online], https://www.nomadproject.io.

[2] "Rancher labs - k3s lightweight kubernetes," [Online] https://k3s.io/.

[3] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*. IEEE, 2015, pp. 130–135.

[4] S. Taherizadeh, V. Stankovski, and M. Grobelnik, "A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers," *Sensors*, vol. 18, no. 9, p. 2938, 2018.

[5] S. Hoque, M. S. De Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 294–299.

[6] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over Raspberrypi," in *Proceedings of the 18th international conference on distributed computing and networking*, 2017, pp. 1–10.

[7] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A container-based edge cloud paas architecture based on raspberry pi clusters," in *2016 IEEE 4th FiCloudW*. IEEE, 2016, pp. 117–124.

[8] P. Kayal, "Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope," in *2020 IEEE 6th WF-IoT*. IEEE, 2020, pp. 1–6.

[9] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in Kubernetes for fog computing applications," in *2019 IEEE NetSoft*. IEEE, 2019, pp. 351–359.

[10] A. Valantasis, N. Makris, C. Zarafetas, and T. Korakis, "Experimental Evaluation of Orchestration Software for Virtual Network Functions," in *IEEE WCNC 2021*, Nanjing, China, Mar. 2021.

[11] "Eclipse Foundation - Eclipse Fog05," [Online], https://projects.eclipse.org/projects/iot.fog05.

[12] I. M. Al Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, and A. Palopoli, "Container orchestration engines: a thorough functional and performance comparison," in *IEEE ICC 2019*. IEEE, 2019, pp. 1–6.

[13] N. Makris, C. Zarafetas, S. Kechagias, T. Korakis, I. Seskar, and L. Tassiulas, "Enabling open access to LTE network components; the NITOS testbed paradigm," in *Proceedings of the 2015 1st IEEE NetSoft*. IEEE, 2015, pp. 1–6.

[14] "The Nomad Autoscaler," [Online], https://www.hashicorp.com/resources/the-nomad-autoscaler.

[15] D. Bernstein, "Containers and cloud: From LXC to docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[16] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An open-source platform for LTE evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 10:28:14 UTC from IEEE Xplore. Restrictions apply.