Performance Evaluation of Kubernetes as Deployment Platform for IoT Devices

Alessandro Caetano Beltrão[1], Breno Bernard Nicolau de França[2], and Guilherme Horta Travassos[3]

[1,3] Universidade Federal do Rio de Janeiro, COPPE, Rio de Janeiro, Brasil
[2] University of Campinas, UNICAMP, Campinas, Brazil
{alessandrocb,ght}@cos.ufrj.br,
breno@ic.unicamp.br

Abstract. With the advent of the Internet of Things paradigm, more and more devices and sensors are physically distributed to accomplish many tasks in our daily lives. Edge computing has emerged as a way to migrate part of the storage, data processing, and computation capabilities to the edge devices. Such a migration aims at addressing the challenges regarding communication between IoT devices, pressure on the cloud infrastructure generated by a large amount of data, network complexity, and scalability. As these applications migrate to this new paradigm, issues already addressed on developing traditional software begin to emerge as new challenges. One of these challenges is the deployment process of applications on the edge devices since, in a real scenario, the scale of an IoT platform can be tremendous. In this paper, we discuss the usage of Kubernetes as a platform to enable deployments from the cloud to IoT gateways and conduct a performance evaluation of Kubernetes on two common IoT scenarios. The preliminary results show the potential and feasibility of Kubernetes as a platform for deploying applications on edge devices.

Keywords: Internet of Things, Edge Computing, Performance Evaluation, Continuous Deployment

1 Introduction

Recently the Internet of Things (IoT) paradigm has changed the way applications, people, and devices interact with each other [1]. More and more devices are getting integrated into our personal lives and activities to accomplish this new level of interaction with the end-user. They may collect data, act as hubs for communication between other devices, and work collaboratively. As IoT becomes widespread, challenges regarding security, scalability, communication, networking also surfaces, creating a favorable scenario for more investigation and research.

To deal with some of these challenges, one of the approaches that have surfaced is the concept of *edge computing* [2]. It refers to the enablement of edge devices and technologies to perform computation at the edge of a network [2][3][4]. That ap-

proach differs from the already established *cloud computing* approach where the cloud is responsible for all the data processing that comes from devices. *Edge computing* also potentially provides some benefits over conventional *cloud computing*. Yi *et al.* described in [5] indicate that a platform for face-recognition had its response time reduced from *900ms* to *169ms* by moving the computation to the edge. Studies [6] and [7] also show positive results regarding response times, energy-consumption by applying concepts of partitioning, merging, migration, and on-demand instantiation from cloud to edge devices.

In IoT or even edge systems, a common challenge is the provisioning and deployments of the application on the devices. In a toy application, it is relatively easy to provision the devices manually. However, in a real-world application, the number and the geographical distribution of the devices can turn this task arduous and complicated. Moreover, to perform such deployments continuously [8], i.e., on a regular and frequent basis, is an even harder activity in the IoT context. Some studies as Rufino *et al*. [9], Morabito [10], Lwakatare *et al.* [11], and Moore *et al.* [12] explore the possibility of utilizing *Linux* container technologies, especially *Docker*, as a way to enable scalable deployments on edge devices, since *Docker*, and containers, generally speaking, are widely used for deployments of microservices into the cloud.

Lwakatare *et al.* [11] collected evidence on the usage of *DevOps* practices on the embedded systems domain, by identifying the challenges as hardware dependency and compatibility with software versions and the lack of the technology to automatically, reliably and repeatedly deploy new features in customer-specific environments among others [11]. Morabito conducted performance testing of virtualization technologies, including *Docker* on boards from different vendors, as *Raspberry* and *ODROID* [10]. The work by Moore *et al.* focuses on the creation of a proof of concept application that uses *Docker* to bootstrap applications on IoT devices [12]. At last, Rufino *et al*. use a proof of concept to propose the usage of *Docker* as an enabler for deployments on Edge Gateways [9].

Our approach differs from the ones stated as it focuses on the performance evaluation of the container orchestration tool *Kubernetes* on the scenario of *Edge Computing*. To evaluate *Kubernetes*, we created a synchronous and asynchronous communication scenario that aimed to reflect the usage of this technology as a means to deploy, schedule, and maintain containers on Edge Gateway nodes. Using those scenarios, we then proceeded to perform a group of performance tests to evaluate the limits of a *Raspberry PI* as a worker node in an architecture where the workers act as Gateways for other devices.

In this paper, we explore the possibility of using *Kubernetes*, a container cluster technology, to enable deployments of IoT-relevant components. To determine the feasibility of our approach, we set up a *Kubernetes* cluster on a group of *Raspberry PI* devices, general-purpose hardware widely available on the market. Then, we

executed performance tests on two common IoT scenarios, using both the HTTP and AMQP protocols for synchronous and asynchronous scenarios, respectively. We used the performance tests to determine whether the cluster could keep the IoT components healthy even in a demanding scenario having the *Raspberry PIs* as gateways for other devices like sensors and actuators. The preliminary results for the tests show that while there are still challenges to overcome in the usage of container technologies on Edge Gateways. However, these technologies may be used for specific scenarios considering a determined number of worker nodes. The main contribution of this paper is the evaluation of the performance of Kubernetes in the IoT context. Nonetheless, it is not within our scope to compare this solution to other already established solutions in the market as KubeEdge, Amazon IoT Core, or K3S as this comparison can be explored in other studies.

Besides this introduction, this paper has six more sections. Section 2 aims to familiarize the reader with concepts related to *Edge Computing*, *DevOps*, and the relevance of containers and Kubernetes in this context. Section 3 presents our evaluation method; it shows how we set up the cluster architecture and test parametrization for each scenario. Section 4 presents the preliminary results of the tests and all the discussion related to these results. In Section 5, we discuss the threats to validity for our study, and finally, in Section 6, we give our final remarks regarding our approach.

2 Related Concepts

Usually, devices used on IoT environments are considered *resource-constrained*, a term that, according to Pisani *et al.* [14], is used to describe devices that are limited in terms of memory, power, energy, and processing capabilities. These devices are often equipped with sensors and actuators that allow them to interact with the real world. They may process and parse data streams coming from sensors, and sometimes act accordingly through actuators. However, the amount of data and the transformations required in specific scenarios may become an overhead for the limited resources of IoT devices.

In a typical *client-server* application, the cloud is in charge of processing data generated from the less capable clients. Nevertheless, in an IoT scenario, processing all the data in the cloud may not be feasible because of the order of the magnitude of the data and latency of requests. *Edge computing* aims to bring computing capable devices closer to the end-user as a means to address challenges regarding latency, data processing, and quality of service (QoS). Yu et al. [2] define the structure of *edge computing* into three aspects (Figure 1): *front-end*, *near-end*, and *far-end*.

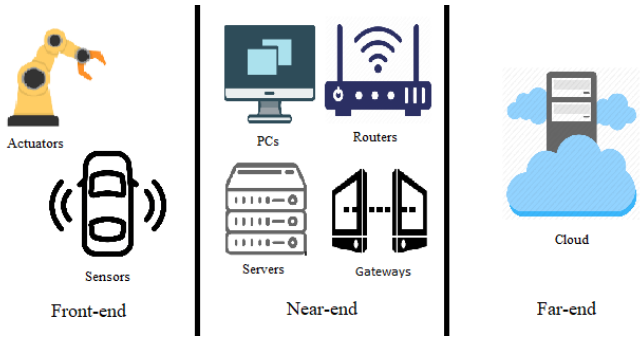

Fig. 1. A typical architecture for *edge computing* networks (adapted from [2]).

In this architecture, the *front-end* devices are usually sensors and actuators. They provide the necessary interaction with the user most of the time using the processing power and capacity from devices nearby and receiving and pushing information from and to IoT gateways, located in the *near-end*. These gateways usually support the flow of information, perform pre-processing, cache data, and perform the offloading, reducing the number of requests made to the cloud servers. The *far-end* devices are cloud servers, proving the necessary computing power and storage for post-processing, data mining, and machine learning [2].

One of the concerns on IoT discussions is the provisioning and deployment of applications into the edge of networks. As the number of devices grows, the number of supporting gateways also increases, and the complexity of deploying and maintaining applications on the devices becomes more and more challenging. Applying *DevOps* practices can help with that task. However, the intrinsic characteristics of IoT devices and applications may not be suitable for all modern *DevOps* solutions.

Studies, as the ones conducted by Rufino *et al*. [9], Morabito [10], Lwakatare *et al.* [11], and Moore *et al.* [12] propose to tackle these challenges using *Linux* Containers, a lightweight form of virtualization, to provide the rapid deployment of microservices into different devices on the *near-end* and *far-end*. Containers allow developers and practitioners to create separate user spaces capable of isolating processes and hardware resources into the same operating system. They started in *Unix* development, in 1979 with the implementation of a group of functionalities that would later be merged into a *system call* named *chroot* [15]. Today, there are many container technologies available for general use; some examples are *LxC*, *Docker*, and *runC.*

However, none of these studies explores the use of container orchestration tools for deployment on an IoT scenario. These tools can help by allowing practitioners to deploy, monitor, and schedule containers across a group of devices that, in most cases, is a server. Although practitioners already use these tools on a conventional

software development process, their usage of IoT is still limited and cares for more research.

One of the most used container orchestration tools is *Kubernetes*; it provides a set of functionalities that can be useful to an IoT scenario. *Kubernetes* allows the fast deployment of services by assigning containers to devices in the cluster. It helps by managing node and container communications, providing high availability by migrating failing services into backup nodes, and taking care of updating the nodes in a controlled way.

The usage of Kubernetes as a driving technology for IoT is an idea that is already being explored on the market. Technologies such as *K3S* and *KubeEdge* are being developed to adapt this technology to IoT. *K3S* specifically tries to optimize *Kubernetes* binaries for the *ARM64* and *ARMv7* architecture while also providing optimizations as an *SQLite* as a replacer for the default *etcd* datastore [20]. *KubeEdge* extends *Kubernetes* default controllers to create a device synchronization mechanism while also using *SQLite* as a lightweight database instead of the default, *etcd*, on the edge devices [21].

Some cloud providers, such as Amazon AWS and Azure, are also providing services for the IoT in the cloud. *Amazon IoT Core* aims to provide communication and synchronization between the devices and the cloud, while also adding compatibility with other common AWS infrastructures, such as *S3*, *EC2*, and *SageMaker* [22]. Azure is providing a similar service, called *Azure IoT Hub*, that aims to provide infrastructure to IoT devices on the Edge [23].

3 Evaluation Method

As the research on the usage of *Kubernetes* in the context of IoT is still incipient, we felt the need to evaluate whether this technology could be used in this context. To do that, we started by assessing the performance of *Kubernetes* worker nodes deployed into *Raspberry PIs*. The tests had two main objectives, and the first is to check whether an ARM-based device (as a *Raspberry PI*) with low resources could fulfill the requirements to act as a worker node. The second objective is to evaluate the performance of such nodes in two common IoT scenarios and determine the limits of the container technologies into these nodes.

To execute the performance tests, the first issue was to set up a *Kubernetes* cluster using the *Raspberry PIs* as worker nodes. Using the available *Raspberry PIs*, we provisioned a four-node cluster, with one master and three workers, the master being a workstation computer with four cores and eight threads *x64* Intel processor and 16 gigabytes of *DDR4* ram. We used this setting because the *Raspberry PI* cannot handle all the components of the *Kubernetes* master. The workers were composed of *Raspberry PI Model 3 B+*, which until the time of the test, were the

latest ones. Figure 2 represents the architecture of scenario 1. We also put them under a Demilitarized Zone on a router. The use of *Docker* as Container Runtime Environment (CRE) led to disabling the bridge that *Docker* uses to map ports from containers to the device network interface. The usage of the bridge could represent another confounding factor in the test, as the study conducted by Felter *et al*. confirmed that using *Docker NAT* and port mapping represents a loss on performance regarding *Network IO* while Docker without those configurations has the approximated performance of bare-metal applications [17]. We implemented each test using Apache JMeter, a tool for load testing and measuring the performance of applications and servers. Moreover, after each test, the deployment of the application under test was destroyed and recreated.

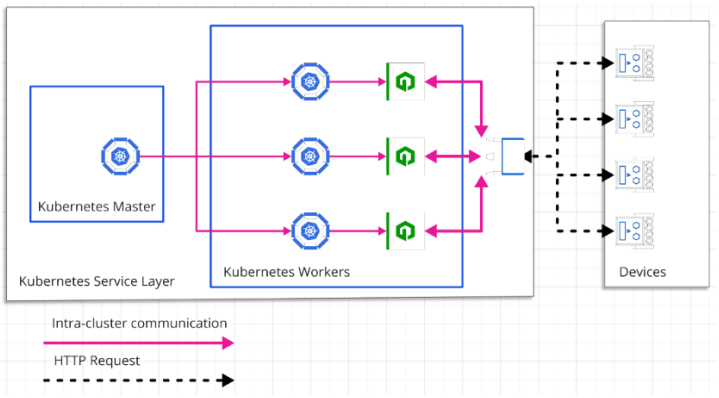

Fig. 2. Cluster Topology for the Synchronous Scenario

The first scenario represents the synchronous communication from IoT Devices (like sensors and actuators) with the IoT Gateway (deployed as Kubernetes Workers), implemented as a simple *HTTP* request that should return response code 200. We set this test up using the official *NGINX*, a widely used application server, and reverse-proxy, Image from *DockerHub*[1]. On this test, the *Raspberry PI* worker node runs a pod with one container that should be able to respond to requests from a client. Each request sent has 114 bytes in data and should get as a response a web page with 850 bytes, being 238 for Header and 612 for Body.

To evaluate this scenario, we designed nine test cases establishing maximum response times of the application being *100ms*, *1000ms*, and *10000ms* based on the work published by Jakob Nielsen in [13]. These times represent thresholds for the attention span of a user in web applications, being *100ms* perceived as real-time, *1000ms* the limit for a user flow of thought, and *10000ms* the limit for keeping user attention. We also established the number of requests per second (RPS) as being powers of 10, as the goal is to stress the cluster to maximum, so the RPS values

[1] https://hub.docker.com/_/nginx

used are 10^1, $10^{2,}$ and 10^3, these requests are applied to the system, and not just to each of the nodes. Using the values for max response time and RPS, we applied them into Equation 1 [16] to calculate the total number of threads needed for each test. Table 1 describes each one of the test scenarios, and below, we present the equation used to calculate the thread number used in each execution.

$$\frac{RPS \times Max\ Response\ Time}{1000} = Thread\ Number \quad (1)$$

Test ID	RPS	Max Response Time (*ms*)	Thread Number
0	10	100	1
1	10	1000	10
2	10	10000	100
3	100	100	10
4	100	1000	100
5	100	10000	1000
6	1000	100	100
7	1000	1000	1000
8	1000	10000	10000

Table 1. Test Execution Parameters for HTTP Protocol Scenario

The second scenario aimed to represent an asynchronous communication (like a publisher-subscriber architecture) using *RabbitMQ*[2] as a broker, and the *AMQP* protocol, in an IoT application, as shown in Figure 3. The AMQP protocol was chosen because it can be used over mobile and unstable networks and is more robust than the MQTT [19]. In this scenario, we set up the tests to allow the *Raspberry PI* to work as an IoT Gateway. This way, we could simulate tests in which publishers sent data to the gateway; a small number of subscribers took that data to perform another action. Using *JMeter* and *Apache AMQP Client*, we set up two different test configurations (Table 2). For each test, the publishers have to deliver a message that has a hexadecimal body of 850 bytes. The delivery of the message is considered a success when an ACK signal is received. On *RabbitMQ*, an exchange is configured to broadcast all messages to queues, as each test is composed of only one queue.

[2] https://hub.docker.com/_/rabbitmq

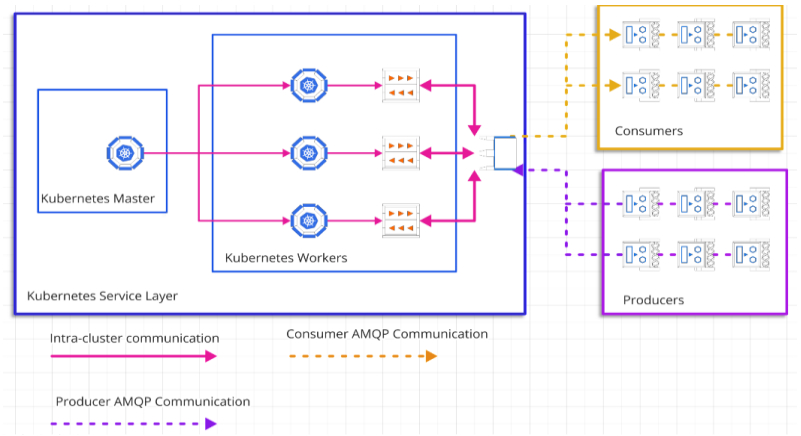

Fig. 3. Cluster Topology for the Asynchronous Scenario

Test ID	Subscribers	Publishers	RPS Limit on Publishers
0	1	10	100
1	10	100	1000

Table 2. Test Execution Parameters for the AMQP Protocol Scenario

Similarly, as in the *HTTP* scenario, on the *AMQP* tests, we also scaled the subscribers and publishers by powers of ten from one test to another. Typically, in an Edge Computing scenario, the number of publishers tends to be higher than the number of subscribers. As the number of devices on the *Front-End*, the sensors and actuators are, in general, higher than the number of devices in the *Near-End*. In a real publisher scenario, each publisher sends messages from time to time, and, to simulate that behavior, we established a limit on the number of requests per second the publishers can make. In both scenarios, the cluster had to support the load of the tests for three minutes and twenty seconds, in which the ten starting seconds is ramp-up, and the last ten is ramp-down time. We consider that the time given for each test scenario was sufficient to show the shortcomings of the architecture and the *Raspberry Pi 3B+* as the number of requests on each test was high and already capable of disrupting the cluster in some scenarios.

4 Preliminary Results

After setting up the cluster and creating the test execution scripts using *JMeter*, we executed the scenarios. The first tests performed are from the *HTTP* scenario. As expected, the cluster could easily handle all the requests from the first five tests with a low standard deviation from the mean response time (Table 3). Test 5 and 6 show that the standard deviation for the response time started to grow exponential-

ly along with the number of threads. Still, the number of failures reported by *JMeter* on tests 0 to 5 were mostly insignificant, as to consider the sample failure, the server had to respond with any other status code than 200, and the request-response time should be higher than the max response time defined in the test execution plan. Table 3 shows the results of the tests for all the HTTP test scenarios.

Test ID	Samples	Mean (*ms*)	Min (*ms*)	Max (*ms*)	SD (*ms*)	Error %	Throughput
0	1824	26	13	188	12,93	0,66%	9,6/sec
1	1663	34	14	364	25,23	0,00%	8,3/sec
2	1387	33	13	265	20,43	0,00%	6,9/sec
3	18926	25	12	230	19,60	1,87%	95,1/sec
4	18873	25	11	203	19,03	0,00%	94,4/sec
5	17715	32	12	1028	41,08	0,00%	88,9/sec
6	53701	350	8	21025	785,43	63,30%	263,0/sec
7	54242	3496	8	70138	4359,41	83,42%	253,0/sec
8	78340	23887	0	198230	37907,06	51,80%	371,8/sec

Table 3. Test Execution Results from the *HTTP* Scenario

Table 3 shows that the cluster operated well until the number of threads reached close to one hundred, and the number of requests per second was close to one thousand. The pod could not serve all the static files promptly as *NGINX* started creating new threads to serve requests. Some of the threads needed to wait until it could be processed, creating a queue that generated high response times. On Test 6, 7, and 8, the major problem was the limited number of resources from the *Raspberry PI* Worker node to process all the requests simultaneously. This phenomenon can be observed by looking at the maximum throughput generated, and the *RPM* expected from the tests.

Regarding the results of the tests from the synchronous scenarios, the results show that the maximum throughput that *NGINX* from one pod with only one replica is 371,8/sec. The *Kubernetes* documentation points out ways to achieve higher throughput by horizontally escalating the architecture and allowing more replicas to coexist. Another way to improve the performance of the pods is fine-tuning the resources allocated per pod, as *Kubernetes* enables users to control the number of allocated *CPU shares* and memory. The usage of monitoring tools, together with Kubernetes, also may be useful to detect eviction signals sent by Kubernetes controllers on the *Worker Nodes* [18].

In the second scenario, the use of a more lightweight protocol like *AMQP* greatly favors the *Raspberry PI* Worker nodes. In both executed tests, the results were significantly better. The first test shows that a *Raspberry PI*, as an IoT Gateway, is perfectly capable of handling ten other devices publishing messages at a rate of

100 *m/s*. Also, it could still keep the broker healthy without any failure, as it is possible to see in Figure 4.

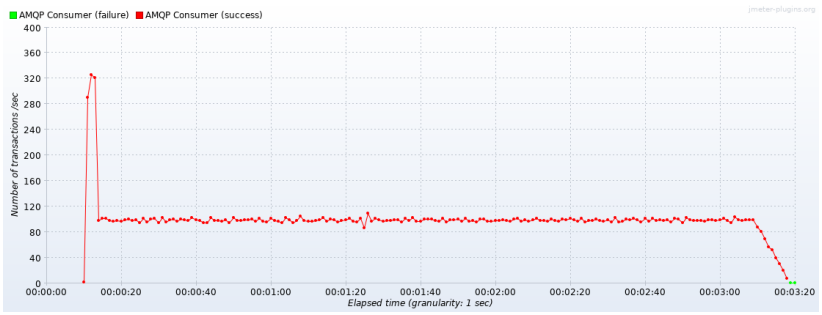

Fig. 4. Subscriber Transactions for Test 0 on the AMQP Scenario

On the second test of the publisher-subscriber scenario, the limits of the *Raspberry PI* as a platform is more apparent. During the test execution, *RabbitMQ* reported that the cluster reached the Memory High Watermark, a configuration that starts to block publishers' queues until the messages are clear and the memory usage lowers. On the cluster, the flag Memory High Watermark was set to be 40% of the node total memory. On the curve generated by the publisher's transactions, we can observe these effects by analyzing the dips in Figure 5. Besides, the green line on the bottom of the graph represents the requests that received a *NACK* from the broker.

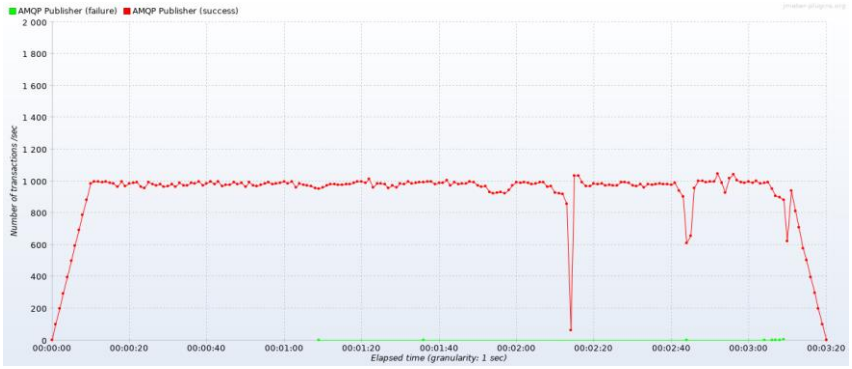

Fig. 5. Publisher Transactions for Test 1 on the *AMQP* Scenario

However, even activating the memory limits, the number of simultaneous publishers connections on a pod got too high. The large connection pool ended up causing the *RabbitMQ* to stop broadcasting messages for the subscriber, as we can see in Figure 6. The graph shows that *RabbitMQ* was prioritizing publisher threads, as

they were in a proportion of 10:1 from subscribers. The high peaks characterize that when a subscriber thread could finally be served, it consumed a high number of messages simultaneously and then stopped to consume. At the end of this test, the subscribers left a total of 118,086 messages hanging in the queue.

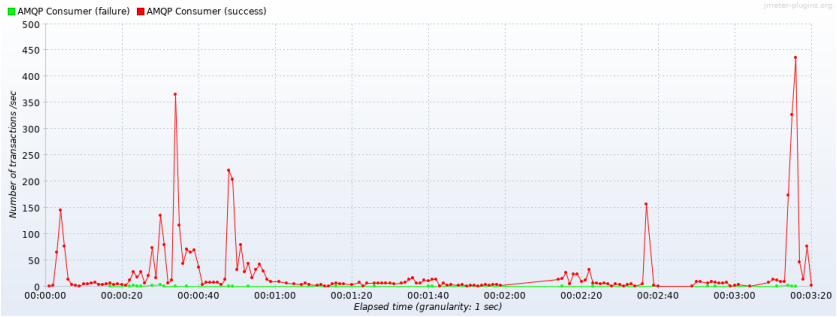

Fig. 6. Subscriber Transactions for Test 1 on the *AMQP* Scenario

In this test, we can see that scaling the publishers and subscribers by powers of ten was not the ideal. The relation between the number of publishers and subscribers also needs a proper tuning as a way to avoid publishers from clogging the broker, a vulnerability for this architecture that can be explored in future research.

5 Limitations and Threats to Validity

Regarding external validity, we acknowledge that our tests are still limited to a small set of devices and technologies, imposing barriers to a broader generalization regarding the applicability of *Kubernetes* on IoT. However, the need to understand the limits of this technology and the particularities of *Edge Computing* sustain the need for further research. Currently, the usage of *Kubernetes* is heavily limited by the processor architecture and nodes resources. However, we believe that as time goes on, the trend is that embedded devices get more powerful resources.

On internal validity, we consider that all the executed tests were adequately set up to reduce noise and interference from any other factor. The environment where the tests executed was a clean *Raspbian Linux* operating system, with a newly instantiated *Kubernetes* cluster and disabling any network configuration made by *Docker*. After each test, the environment was destroyed and recreated automatically. Therefore, any execution of one test could not influence another.

To deal with construct validity, we tried to set up the cluster as close to what a real production environment would be. All the configurations were done by following the official *Kubernetes* documentation [18] and each of the respective documentations for the container images. We executed all the tests from an isolated environ-

ment, and they were also parametrized to reflect the usage intensity of a real IoT scenario.

About the evaluation process, we consider that the automated tests generated alongside the configuration files necessary to set-up the cluster should enable any person to execute the tests at any time. Our implementation source code (including cluster setup configurations, compiled binaries for *Kubernetes* on *armv6l* platform, test scripts, and tutorials) are available on an instance of Gitlab[3] and can be accessed by anyone. However, as we did not conduct any statistical tests of the results, this represents a conclusion validity threat to our study.

6 Final Remarks

In this paper, we presented the application of the *Kubernetes* Engine as a deployment platform for IoT devices.

The results shown in this paper presents the applicability of *Kubernetes* for at least two scenarios, one using synchronous (*HTTP*) and one using asynchronous (*AMQP*) protocols. During the tests, the cluster kept the applications alive and handled the high load in the *Raspberry PI*. Furthermore, we believe that container-based technologies may prove as a way to enable deployments on highly distributed environments as envisioned by the IoT paradigm.

The adoption of the *Kubernetes* Engine as a platform for IoT still needs more research, as it still imposes barriers regarding the requirements of the cluster worker and master nodes. The cluster implementation may not be suitable for all constrained devices built for IoT purposes in the market. Nevertheless, on platforms that are capable of handling a *Kubernetes* cluster, the gain from being able to deploy and maintaining those applications can outweigh the complexity and overhead of bootstrapping the cluster on those environments.

As future works, we envision improving our tests by making use of simulations to evaluate how *Kubernetes* would perform in a large-scale scenario. For that, we aim to make use of Smart City simulations tools and specify a configuration that should enable *Kubernetes* to provision IoT Gateways on-demand.

Acknowledgments

The CNPq (Brazilian National Council for Scientific and Technological Development) and CAPES support this research. Prof. Guilherme Horta Travassos is a CNPq researcher (304234/2018-4).

[3] https://www.mrdevops-gitlab.com/plataforma-de-desenvolvimento-continuo-para-iot

7 References

1. Atzori L, Iera A, Morabito G (2010). The Internet of Things: A survey. Computer Networks 54:2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010
2. Yu W, Liang F, He X, et al. (2018). A Survey on the Edge Computing for the Internet of Things. IEEE Access 6:6900–6919. https://doi.org/10.1109/ACCESS.2017.2778504
3. Varghese B, Wang N, Barbhuiya S, et al. (2016). Challenges and Opportunities in Edge Computing. In: 2016 IEEE International Conference on Smart Cloud (SmartCloud). IEEE, New York, NY, USA, pp 20–26
4. Shi W, Cao J, Zhang Q, et al. (2016). Edge Computing: Vision and Challenges. IEEE Internet Things J 3:637–646. https://doi.org/10.1109/JIOT.2016.2579198
5. Yi S, Hao Z, Qin Z, Li Q (2015). Fog Computing: Platform and Applications. In: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE, Washington DC, DC, USA, pp 73–78
6. Ha K, Chen Z, Hu W, et al. (2014). Towards wearable cognitive assistance. In: Proceedings of the 12th annual international conference on Mobile systems, applications, and services - MobiSys '14. ACM Press, Bretton Woods, New Hampshire, USA, pp 68–81
7. Chun B-G, Ihm S, Maniatis P, et al. (2011). CloneCloud: elastic execution between mobile devices and cloud. In: Proceedings of the sixth conference on Computer systems - EuroSys '11. ACM Press, Salzburg, Austria, p 301
8. Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Signature Series (Fowler). Pearson Education.
9. Rufino J, Alam M, Ferreira J, et al. (2017). Orchestration of containerized microservices for IIoT using Docker. In: 2017 IEEE International Conference on Industrial Technology (ICIT). IEEE, Toronto, ON, pp 1532–1536
10. Morabito, R (2017). Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. IEEE Access 5:8835–8850. https://doi.org/10.1109/ACCESS.2017.2704444
11. Lwakatare LE, Karvonen T, Sauvola T, et al. (2016). Towards DevOps in the Embedded Systems Domain: Why is It So Hard? In: 2016 49th Hawaii International Conference on System Sciences (HICSS). IEEE, Koloa, HI, pp 5437–5446
12. Moore J, Kortuem G, Smith A, et al. (2016). DevOps for the Urban IoT. In: Proceedings of the Second International Conference on IoT in Urban Space - Urb-IoT '16. ACM Press, Tokyo, Japan, pp 78–81
13. NIELSEN J (1993). Chapter 5 - Usability Heuristics. In: NIELSEN J (ed) Usability Engineering. Morgan Kaufmann, San Diego, pp 115–163
14. Pisani F, Immich R, Bittencourt LF, Borin E (2019). Fog Computing on Constrained Devices: Paving the Way for the Future IoT. 37
15. Cheswick, B., n.d. An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied 11.
16. JMeter Throughput Shaping Timer, https://jmeter-plugins.org/wiki/ThroughputShapingTimer/, Last accessed 2019/11/12
17. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp 171–172
18. Kubernetes Docs, https://kubernetes.io/docs, Last accessed 2019/12/13

19. J. Luzuriaga, P. Boronat, M. Perez, C. Calafate, J.-C. Cano, e P. Manzoni, “A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks,” 2015, doi: 10.1109/CCNC.2015.7158101.
20. K3s - 5 less than K8s, https://rancher.com/docs/k3s/latest/en/, last accessed 2020/02/26.
21. What is KubeEdge — KubeEdge Documentation 0.1 documentation, http://docs.kubeedge.io/en/latest/modules/kubeedge.html#advantages, last accessed 2020/02/26.
22. Visão geral do AWS IoT Core – Amazon Web Services, https://aws.amazon.com/pt/iot-core/, last accessed 2020/02/26.
23. Hub IoT | Microsoft Azure, https://azure.microsoft.com/pt-br/services/iot-hub/, last accessed 2020/02/26.