Kubernetes or OpenShift? Which Technology Best Suits Eclipse Hono IoT Deployments.

Mohab Aly
SWAT Lab., GIGL-MAGI
Polytechnique Montréal,
Montréal, Canada
mohab.aly@polymtl.ca

Foutse Khomh
SWAT Lab., GIGL
Polytechnique Montréal,
Montréal, Canada
foutse.khomh@polymtl.ca

Soumaya Yacout
MAGI
Polytechnique Montréal,
Montréal, Canada
soumaya.yacout@polymtl.ca

***Abstract*—New verticals within the Internet of Things paradigm, *i.e.,* smart cities, industrie 4.0, etc., require specific platform(s) to allow different components to communicate. The value of the IoT systems often correlates directly with the ability of those platforms to connect different devices efficiently and integrate them into higher-level solutions. Eclipse Hono allows the provisioning of remote service interfaces for connecting devices to a back-end and interacts with them uniformly regardless of their types and communication protocols. Currently, there is a variety of possibilities for using Hono in production; it can be deployed on Kubernetes, OpenShift or Docker Swarms. However, these deployments decisions have important performance implications that the developers are not often aware of. In this paper, we step up loads in Kubernetes and OpenShift to clear out the performance costs of their deployment scenarios, with the aim to provide the practitioners with guidelines to help understand the performance implications of their design and deployment decisions.**

***Index Terms*—IoT, Platforms, Eclipse Hono, Kubernetes, OpenShift, EnMasse, Empirical Study, Performance Evaluation**

I. Introduction

Internet of Things (IoT) systems are now pervasive in our society. As a consequence, resources utilization of those systems and IoT based applications has become an emerging topic in IoT and software engineering research communities [1], [2]. Resource utilization has complex dependencies on the IoT platforms and the various components used by the applications built on top of them, all contribute in raising the resources footprint; making resource optimization a vital and challenging problem.

Eclipse IoT [3], including Eclipse Hono, is an IoT open source framework that is gaining a lot of traction in industry nowadays [4]. It provides a number of protocol implementations, *i.e.,* `HTTP REST`, `MQTT`, etc., Cloud front/back-ends, Machine-to-Machine (M2M) management, devices' authentication and management, Over-The-Air (OTA) update methodologies, security and authorization mechanisms among others. When building, deploying and implementing applications on top of Hono, practitioners, including developers, must seek a compromise between choosing the right deployments, resources utilization and the platform's Quality of Service (QoS) so that the maximum efficiency is achieved. Finding such compromises manually is a daunting task. Practitioners need guidelines to assist them in the selection of efficient design and deployment strategies. Currently, there are many possible deployment options for Hono in a production pipeline; it can be deployed on top of Kubernetes, OpenShift or Docker Swarms. However, those deployments decisions have important performance implications that developers should consider carefully. Moreover, without a good knowledge of performance deviations across different deployment platforms, it is challenging to predict the impact of apps migrations across IoT environments.

In this paper, we conduct an empirical study that aims to assess the performance implications of deploying Eclipse Hono in two different virtual environments, *i.e.,* containers: Kubernetes and OpenShift. We perform performance test comparisons while incorporating EnMasse, *i.e.,* a messaging infrastructure, using the following performance metrics: CPU cores usage, Memory consumption, and Network I/O usage. Our objective is to provide evidence to confirm or refute the efficiency of such technologies and comprehend the interplay between them. We selected Kubernetes and OpenShift for our study because the latter actually distributes Kubernetes so practitioners, including end users, may believe that both technologies offer similar performance.

The rest of this paper is structured as follows. In Section II, we provide some background information about the Eclipse technology in the IoT field. Section III presents the design of our experiments and Section IV discusses the obtained results. Section V explains the possible threats to the validity of our findings. Section VI discusses the related literature, whereas Section VII concludes the paper.

II. Background

In this section, we discuss the motivation behind the Eclipse IoT ecosystem and provide background information about the platforms that constitute it.

A. Eclipse Hono

The Hono project provides a platform for scalable messaging in the IoT. It introduces a middleware layer between both the back-end "micro" services and the devices being registered within the framework. Hence, the communication and networking with the back-end services occur via the

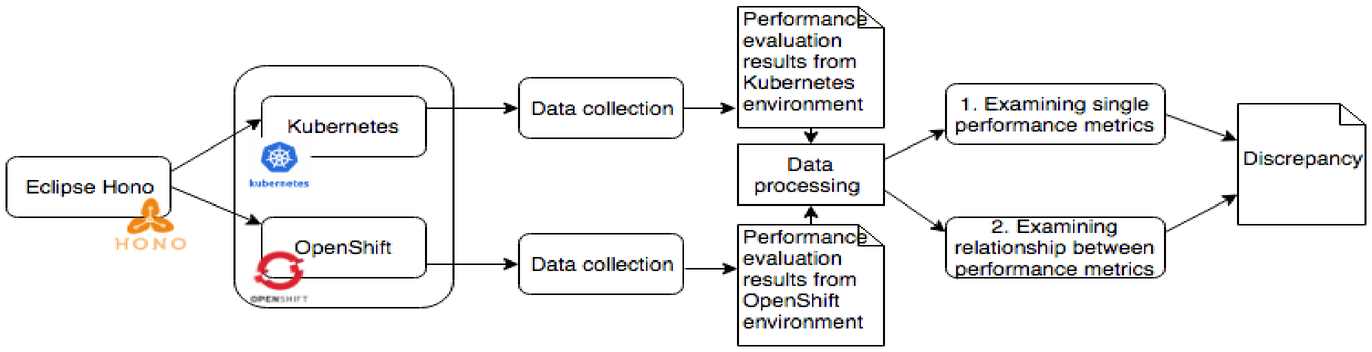

Eclipse Hono
HONO
Kubernetes
kubernetes
OpenShift
OPENSHIFT
Data collection
Data collection
Performance evaluation results from Kubernetes environment
Data processing
Performance evaluation results from OpenShift environment
1. Examining single performance metrics
2. Examining relationship between performance metrics
Discrepancy

Fig. 1: An overview of the study design setup

Advanced Message Queuing Protocol (AMQP). If the participating devices speak this protocol in a direct way, then they can connect to the middleware in a very transparent way; otherwise, Hono provides the so called "protocol adapters" that help to translate messages from the device protocol to the AMQP. Thus, Hono's core services are decoupled from the protocols that specific applications are using. Through the AMQP 1.0 endpoints, the framework provides APIs that depicts two common communications scenarios for devices in the IoT system: (i) Telemetry and Event, (ii) Command & Control, and Registration.

Via Hono's Telemetry and Event API, data flows downstream from devices to the back-end, and to a consumer like a Business Application or Device Management component that usually consists of a small set of discrete values, *e.g.,* sensor readings or status property values. Messages are "one-way" directed where devices send such kind of data and usually do not expect a reply from the back-end. Whereas, Hono's Command & Control API allows the sending of commands "requests messages" to devices and expect a reply reception by the back-end component to such commands from devices asynchronously in a reliable way; messages flowing upstream from back-end components like Business Application often represent invocations of services provided by connected devices, *e.g.,* instructions to download and/or apply a firmware update, setting configuration parameters or querying the current reading sensor. Finally, Hono provides APIs for provisioning and managing both the identities and credentials of connected devices.

The platform "Hono" consists of different building blocks; the first one is the protocol adapters which are required to connect devices that do not have the ability to speak the AMQP natively. In the meantime, Hono comes with two protocol adapters: HTTP-based Representational State Transfer (REST) messages and Message Queuing Telemetry Transport (MQTT). A dispatch router that handles the proper routing of the AMQP messages, between producing and consuming endpoints, within Hono. Such router is based on the Apache Qpid project and is designed with scalability in mind so that it can possibly handle connections from millions of devices. As such, it does not take ownership of the messages being flowing, but rather, passes AMQP packets between different endpoints. This allows a horizontal scaling to achieve reliability and responsiveness. The event and commands messages, that are in a need for a delivery guarantee, can be transmitted and routed through a broker queue; the broker (based on Apache ActiveMQ project) dispatches such messages that need delivery assurance. Conventionally, such messages originate from the Command & Control API. While devices are being connected to the Hono server components, back-end services are connected via subscribing to certain topics at the Qpid server [5].

To enforce security of the routed messages, Hono possesses a device registry that is responsible for the registration, activation of devices, provisioning of credentials and an Auth Server to handle the authentication and authorization of the devices. By using an InfluxDB and a Grafana dashboard "Cloud front-end visualization tool", the platform comes also with some monitoring infrastructure to visualize data via a variety of charts and diagrams configured by the users, *i.e.,* in form of time, series, histogram, bar/line charts, stacks and further customization possibilities. Due to the modularity nature of its design, other AMQP 1.0-compatible message broker than the Apache ActiveMQ Artemis can be used.

B. *Scaling out of Eclipse Hono - EnMasse*

EnMasse is an open source "messaging as a service" platform that simplifies the deployment of a messaging infrastructure on premise and in the Cloud. It provides the scalability and elasticity needed to support the messaging requirements for different IoT use cases. Furthermore, it supports common messaging patterns, including (request/reply, competing consumers and publish/subscribe) in addition to the two main protocols: AMQP 1.0 and MQTT. Nevertheless, HTTP support is coming along the road map of services to be included and supported.

Moreover, this framework provides the possibility of multi-tenancy, meaning, the same infrastructure can be shared between different tenants, but are also isolated from each other. It enforces security with respect to securing connections using Transport Layer Security (TLS) as well as authenticating clients using Keycloak as the Identity Management System (IMS). EnMasse is completely containerized and runs on key container orchestration platforms such as Kubernetes and

OpenShift. This aspect makes it appealing to be used with Eclipse Hono. It is considered to be an excellent complement to Hono's microserivces architecture and deployment models that offers all the features needed to be its messaging infrastructure.

C. *Eclipse Che*

Eclipse Che is an open source Java based developer workspace server and Cloud Integrated Development Environment (IDE), that provides a remote development platform for multi-users purposes. Conventionally, it can be either utilized by its own browser IDE or directly by connecting to the respective workspaces that are realized as customized `Docker` containers which bring their complete runtime environment, *e.g.,* an Ubuntu based installation with Java, Maven and/or a C/C++ tool chain. Different than typical IDEs, the concept of having workspaces alongside with runtime stacks allows skipping the setup times for end-developers by sharing the proper configurations, *e.g.,* with preloaded example projects and tutorials. In Hono, Eclipse Che has proven to be of a valuable asset for designing and developing different applications and projects, such as IoT workload simulators for different requests issued towards the Hono platform.

III. Study Design

This section presents the design of our study which aims to understand the discrepancy of the container technologies while deploying Hono on top of them. An overview of our case study setup is depicted in Figure 1. As previously stated, we select two container technologies (*i.e.,* Kubernetes and OpenShift) which are described as good deployment practices by the Eclipse Community, and address the following research questions:

RQ1. Does Eclipse Hono display similar performance(s) when bare-ly deployed on container technologies?
RQ2. Does EnMasse display similar performance(s) when added up on Hono to scale it up?
RQ3. Do deployed applications display similar performance(s) when being added on top of both EnMasse and container technologies?
RQ4. To what extent does the relationship between the performance metrics change across container environments?

To answer these research questions, we perform a series of experiments with multiple deployments to test and analyze the aforementioned performance metrics and their behaviors. We analyzed three versions of the platform, summarized in table I. Deployments were built from scratch each time a new analysis is performed and the results were collected by performing a series of stress tests on the platform (fixing the number of the issued requests and applications built on top of it) and tracing their executions. The same test sets were used for all experiments to ensure comparable results. The remainder of this section elaborates more the details of our experiments.

TABLE I: Setup Designs

Criteria	**Experimental Designs**	
Deployments	***Kubernetes and OpenShift (KO)***	***Version***
Basic Version	Bare metal deployment of Hono	KO-0
EnMasse	Adding EnMasse on top of Hono	KO-1
EnMasse-App	Deploying apps on top of both Hono and EnMasse	KO-2

A. *Environmental Setup*

The performance evaluation is conducted on a Linux machine, *i.e.,* Ubuntu 17.10, in a lab environment; this machine has an Intel i7-870 Quad-Core 2.93GHz CPU with 16GB of memory, 630GB SATA storage, 8MB Cache and is connected to a local gigabyte Ethernet cable. The hosting machine is configured to have available, Kubernetes and OpenShift, single-node clusters inside a Virtual Machine (VM). Those clusters are created via the below instances.

- Minikube: tool that helps running a single-node Kubernetes cluster inside a VM locally. This makes it easier to try Kubernetes or develop with(in) it.
- Minishift: helps running OpenShift locally by running a single-node OpenShift cluster inside a VM.

Since our goal is to compare the performance metrics in such created clusters, we set up the VM instances using enough resources so that the deployment becomes successful, *i.e.,* 4CPUs, 10GB of Memory and 30GB of disk-size. Default instants' configurations provide a small subset of the host machine resources, in turn, it may not be sufficient to allow the instances to start correctly. That's why it is recommended to scale up the resources whenever possible, "the more resources used, the better", based on the physical machines capabilities.

B. *Design and Procedure*

To assess the benefits and trade-offs of the different deployments considered, the experimentations were orchestrated using two different types of issued requests `(REST HTTP and MQTT)`. For each type, we simulated the registered devices sending both requests simultaneously in a telemetry fashion (*i.e.,* not to expect a response in return). Each experimentation was performed five times (with the number of devices being incremented gradually after each simulation: starting with 10 devices until 100 registered devices – 100 is the limit for this platform by default) to obtain min, max, and average values of the resources been consumed (*i.e.,* a total of 450 readings for each simulation were recorded). We chose to repeat five times to mitigate the effect of variabilities (that are common in virtual environments) on our results. Table I shows the three deployment versions of the platform, the basic version KO-0 don't use any additional overhead, just bare metal deployment of Hono on top of both Kubernetes and OpenShift.

C. *Performance Tests and Evaluations*

Minikube and Minishift are released with `DOCKER_HOST` environment variable to point to the `Docker daemon` running inside the virtual instances; such `daemon` is used to have the final `Docker images` available inside the

TABLE II: p-value of Wilcoxon Test (p-VAL) – *(Median Kubernetes, Median OpenShift) – (Cliff δ Effect Size (ES))*

# Devices	CPU cores usage (*p*-value)			Memory usage (*p*-value)			Network consumption (*p*-value)		
	min	max	avg	min	max	avg	min	max	avg
HONO Bare metal Deployment KO-0									
10	0.6905	0.8413	0.8413	0.5309	0.2948	0.2101	1	0.6752	1
20	0.3095	0.402	0.4206	0.4034	0.4034	0.4206	0.5476	0.6905	0.210075
30	0.2073	0.4206	0.4206	0.2101	0.1437	0.2222	**0.03175** *(70, 33.28)*–(***ES=0.84***)	0.05556 (70, 33.28)–(ES=0.76)	0.06010281
40	0.05556 *(58, 42.991)–(ES=-0.76)*	0.09524	0.09524	**0.007937** *(70, 2.09)*–(***ES=1***)	**0.02157** *(91, 16.62)*–(***ES=0.92***)	**0.01587** *(82, 9.58)*–(***ES=0.92***)	0.09469	0.09524	0.09469294
50	0.09369	0.09524	0.09524	0.09369	0.09469	0.5556 *(68, 45.12)–(ES=0.76)*	**0.02157** *(76, 48.43)*–(***ES=0.92***)	**0.03175** *(95, 63.34)*–(***ES=0.84***)	**0.02157175** *(85, 55.83)*–(***ES=0.92***)
60	0.2222	0.2101	0.2222	0.6761	0.5309	0.834	0.09369	**0.03671** *(103, 80.37)*–(***ES=0.84***)	0.09469294
70	0.6905	0.6905	0.6905	0.1437	0.09469	0.0601	**0.0469** *(85.5, 57.04)*–(***ES=0.68***)	0.1508	0.09524
80	0.09524	1	0.1508	**0.02157** *(80, 40.44)*–(***ES=0.92***)	**0.01219** *(96, 50.1)*–(***ES=1***)	**0.007937** *(88, 45.12)*–(***ES=1***)	**0.01587** *(80, 52.51)*–(***ES=0.92***)	**0.02157** *(97, 75.42)*–(***ES=0.92***)	**0.02157175** *(90.82, 61.03)*–(***ES=0.92***)
90	0.4095	0.09369	0.2222	0.0601	0.0601	**0.03671** *(88, 70.25)*–(***ES=0.84***)	0.2101	0.222	0.210075
100	0.09524	0.3095	0.09524	0.2963	0.2101	0.2222	0.1437	0.4034	0.1436721
HONO-EnMasse KO-1									
10	0.1508	0.09469	0.1436721	0.1508	0.2963	0.210075	**0.01587** *(12.08, 61.69)*–(***ES=-0.92***)	**0.007937** *(27.72, 74)*–(***ES=-1***)	**0.01218578** *(17.981, 69.48)*–(***ES=-1***)
20	**0.03175** *(38.767, 71.787)*–(***ES=-0.84***)	0.1437	0.1508	0.1437	0.2101	0.1436721	**0.02157** *(37.81, 62.18)*–(***ES=-0.92***)	0.09469	**0.03671386** *(46.669, 68.88)*–(***ES=-0.84***)
30	0.222	0.1508	0.1437	0.09524	0.1437	0.09469294	**0.01587** *(44.88, 73.16)*–(***ES=-0.92***)	**0.01219** *(63.17, 88.73)*–(***ES=-1***)	**0.02157175** *(50.839, 81.88)*–(***ES=-0.92***)
40	0.1437	0.09469294	0.9524	0.05556 *(36.99, 75.34)*–(ES=-0.76)	0.09469	0.09524	0.09524	0.9524	0.1436721
50	0.1436721	**0.01219** *(75.719, 99.964)*–(***ES=-1***)	0.9469	0.1508	0.09524	0.1436721	0.2222	0.4034	0.210075
60	0.09369	0.1425	0.09369	0.2222	0.2963	0.4033953	0.09524	0.0601	0.06010281
70	0.1508	0.09469294	0.1437	0.09469	0.09469	0.09524	0.09469	0.09469	0.1436721
80	**0.01587** *(81.704, 92.705)*–(***ES=-0.92***)	0.1437	**0.03671** *(90.789, 102.853)*–(***ES=-0.84***)	0.1437	0.4034	0.1436721	0.09524	0.0601	0.09469294
90	1	0.5309	1	0.09469	0.09524	0.9469294	0.5476	0.210075	0.2962699
100	0.09524	**0.01218578** *(98.387, 110.456)*–(***ES=-1***)	**0.007937** *(89.347, 101.604)*–(***ES=-1***)	0.6905	0.6761	1	0.1425	0.1437	0.1436721
HONO-EnMasse-App KO-2									
10	0.09524	0.09469	0.09469294	0.05556 *(5.06, 53.79)–(ES=-0.76)*	0.09469	0.06010281	0.1508	0.0601	0.09469294
20	0.09469	0.1437	0.9469	0.4206	0.2963	0.4033953	0.8413	0.6761	0.6905
30	1	0.4034	0.4033953	0.8413	1	1	1	0.6905	0.6761033
40	0.6905	0.8345	0.8413	0.5476	0.6761	0.5308683	0.4206	0.2101	0.4033953
50	0.4206	0.4034	0.4206	0.5309	0.8345	0.8345316	0.4034	0.5309	0.6761033
60	0.05556 *(38.253, 74.32)–(ES=-0.76)*	0.1437	0.09469294	0.6905	0.8345	0.8413	0.6905	0.8345	0.6905
70	0.6905	0.5309	0.5476	0.2222	0.2963	0.210075	0.8413	0.6761	0.6761033
80	0.1508	0.09469	0.9524	0.5476	0.4034	0.4033953	0.8345	0.8413	1
90	0.0601	0.09524	**0.03671386** *(49.108, 89.158)*–(***ES=-0.84***)	1	1	1	0.3095	0.4034	0.4033953
100	0.5476	0.8345	0.6761033	0.6761	0.6905	0.5308693	0.6761	0.5309	0.8345316

Minikube/Minishift VMs and make them ready for Hono's deployment. Both deployments provide access to the platform by means of different *services*, the main ones are:

1) dispatch router: router network for business applications to consume data.
2) mqtt-adapter: protocol adapter for publishing telemetry data and events using the `MQTT` protocol.
3) rest-adapter: protocol adapter for publishing telemetry data and events using the `HTTP` protocol.
4) service-device-registry: component for registering and managing devices.

To ensure the consistency between the performance tests, we destroy the environments (VMs) and restart them after every single experiment. This ensures that the formed clusters remain healthy.

D. Data Collection and Preprocessing

Performance Metrics

We used heapster[1] and Prometheus[2] to record the values of the performance metrics. Heapster enables container cluster monitoring as well as performance analysis for Kubernetes. It collects and interprets various signals, such as compute resources utilization, whereas, Prometheus, is a service monitoring system. It collects metrics from configured targets at given time intervals, evaluates, displays the results, and triggers alerts if some conditions are observed to be true. We ran both monitoring tools on each of the clusters constructed then performed statistical analysis on the collected data. Moreover, we recorded the metrics with an interval of "starting the cluster within the VMs until the destroy phase". In total, we recorded about 1350 performance metrics values for all emulations.

System Throughput

We used the collected measurements to calculate the *minimum, maximum and average values* of the systems' resources by measuring the number of telemetry messages, `HTTP` and `MQTT`, sent from each registered device while adopting two applications on top of Eclipse Che – see II-C, *e.g.,* Java and Nodejs apps, on top of the messaging as a service infrastructure, enMasse. The Java app is a "Hello World" app just to warm up the pods, and the latter is about simulating the workload generated from the platforms themselves. The aim is to combine the performance metrics and system throughput while minimizing their gathering noise. The combination is based on the time stamp – on a per minute basis; a similar approach has been applied to address mining performance metrics challenges in [6].

[1]https://github.com/kubernetes/heapster

[2]https://prometheus.io

E. Hypotheses

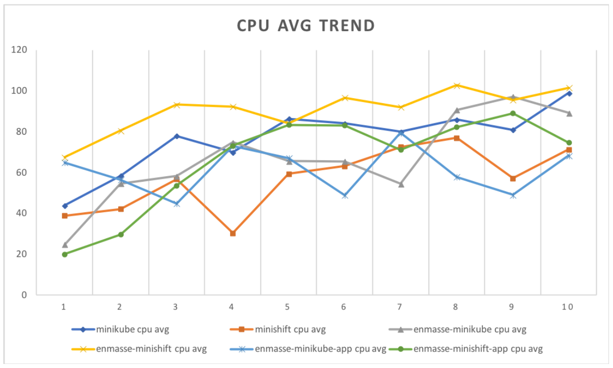
To answer our research questions, we formulate the following null hypotheses, KO-x ($x \in \{1, 2\}$), and KO-0 is the basic version of the platform described in Table I:

- HR_x^1: there is no difference between the amount of CPU/Memory/Network consumed by Hono's design when deployed on top of both Kubernetes and OpenShift.
- HR_x^2: there are no differences in the utilized resources consumed by the messaging infrastructure, *i.e.,* EnMasse when being added on top of Hono.
- HR_x^3: there is no difference between the amount of resources consumed by applications when deployed on top of both Hono and EnMasse's backend.

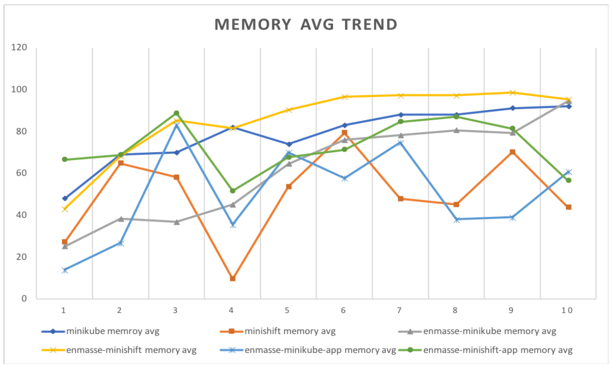
F. Analysis Method

We performed the Wilcoxon test [7] to accept or reject HR_x^1, HR_x^2 and HR_x^3. We also computed the Cliff's δ effect size [8] to quantify the importance of the differences obtained between metrics values. All the tests are performed using a 95% confidence level (*i.e., p*-value $\leq$ 0.05).

A *p*-value that is less than or equal to 0.05 indicates that the obtained results are statistically significant. In this case we reject the null hypothesis (*i.e.,* two populations are from the same distribution) and accept the alternative hypothesis that help stating whether the performance metrics in the Kubernetes and OpenShift environments have the same distribution. We chose the Wilcoxon test because it does not make any assumptions on the distribution of the metrics. *When it happens to have a statistically significant value $\leq$ (or just very close) to the value determined, we depict both its median and effect size values to show which technology performs best and which one is the worst for such kind of deployment.*



(a) Median trend of the CPU avg



(b) Median trend of the Memory avg

Fig. 2: Results obtained for CPU and Memory trends

Wilcoxon test is a non-parametric statistical test that assesses whether two independent distributions and–or trends are the same. Cliff's δ is a non-parametric effect size measure that represents the degree of overlap between two sample trends [8]. It ranges from -1 (when it happens that all selected values in the first group are larger than the second one) to +1 (if all selected values in the first group are smaller than that of the second group). It is zero when the two sample trends are identical [9]. *A Cliff's δ effect size is considered negligible if it is $<$ 0.147 , small if $<$ 0.33, medium if $<$ 0.474, and large if $\geq$ 0.474.*

IV. Case Study Results

This section presents and discusses the results of our research questions. Table II summarizes the results of the Wilcoxon test, median values and Cliff's δ effect size for each performance metric. Significant results are marked in **bold**.

RQ1. *Does Eclipse Hono display similar performance(s) when bare-ly deployed on container technologies?*

Results of Table II show that there is no statistically significant difference between the overall **CPU cores usage** while deploying Hono on top of Kubernetes and OpenShift, hence we cannot reject HR^1_x for KO-x ($x \in \{0..2\}$) for the CPU usage. However, effect size values and Figures 2a, 2b and 3 – (*i.e.,* samples) show that the trend of the CPU, in Kubernetes, is slightly larger than that of the OpenShift in the simulations conducted for the bare metal deployments. The trend tends to fall down towards the statistically significant difference value of 0.05, where Kubernetes is greedy while consuming its CPU to handle Hono's deployment on top of it.

In addition to the previous observation, results also show that there is a statistically significant difference between the **Memory usage** for the container technologies. Furthermore, we obtained statistically significant results with the **Network consumption**, for all Kubernetes deployments in contrast to OpenShift (*i.e.,* all effect sizes are large). Hence we reject HR^1_x for all KO-x ($x \in \{0..2\}$) for Memory and Network usages. We explain such phenomenon by the overhead induced by Kubernetes to be able to cope up with the processes being issued within the system, *i.e.,* setting up the VM, building `DOCKER` images, deploying Hono Platform as well as sending telemetry messages from the participating devices.

RQ2. *Does EnMasse display similar performance(s) when added up on Hono to scale it up?*

Results from table II show that the addition of the messaging infrastructure, EnMasse, on top of the container technologies does affect the overall **CPU cores usage** of the platform as well as the **Network consumption**. This addition is statistically significant for OpenShift, therefore we can reject HR^2_x for KO-x ($x \in \{0..2\}$). Effect size values (*i.e.,* large) as well as Figures 2a, 2b and 3 show the impact of EnMasse on the performance metrics when combined with Hono to scale up the platform.

Regarding the **Memory usage**, we did not obtain significant difference between Kubernetes and OpenShift, when performing the deployment of Hono; hence we cannot reject HR^2_x in this case for the memory usage. However, figures as well as effect size values show that the trend of Memory usage, in OpenShift, is larger than that of Kubernetes in the simulations performed while attempting to add EnMasse on top of Hono. The trend tends to approach the statistically difference value of 0.05, where OpenShift utilizes more memory to sustain the messaging infrastructure on top of it.

RQ3. *Do deployed applications display similar performance(s) when being added on top of both EnMasse and container technologies?*

Results from table II show that the deployment of the two applications on top of the messaging infrastructure, EnMasse, and the container technologies does affect the overall **CPU cores usage** of the platform. Such overhead is statistically significant for OpenShift as well, therefore we can reject HR^3_x for KO-x ($x \in \{0..2\}$). Also, depicted figures and effect size values (*i.e.,* large) show the impact that adding applications to EnMasse and Hono has on the performance metrics.

Regarding the **Memory usage**, the trend tends to also lean towards the statistically significant difference value 0.05, where OpenShift consumes more memory to allow the deployment of applications. On the other hand, for the **Network consumption**, we did not notice any significant difference

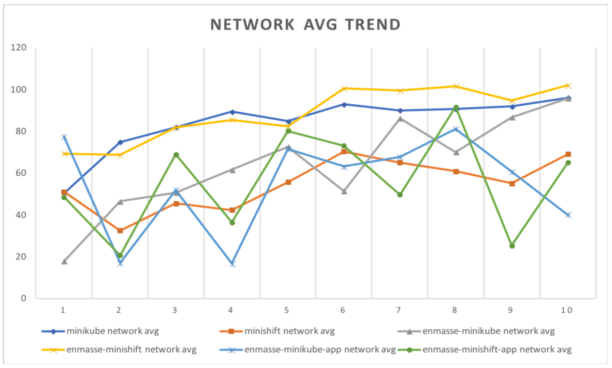


Fig. 3: Median trend of the Network I/O avg

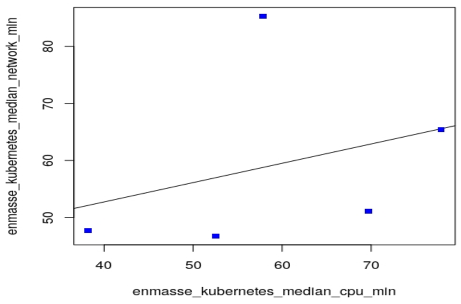


Fig. 4: Spearman's correlation changes for Kubernetes - Hono-EnMasse (CPU–Network) min.

between both container technologies, when allowing the deployment of applications on top of the platform. Hence, in these cases, we cannot reject HR_x^3 for each of the memory and network consumptions.

RQ4. To what extent does the relationship between the performance metrics change across container environments?

The relationship between performance metrics may significantly change and be different between environments, which may be a glimpse of system regression or performance issues. as of [10], combinations of performance metrics are more predictive towards performance issues than a single metric. A change in such combinations can pose discrepancy of performance and help practitioners identify the behavioral changes of a system between different environments. For instance, in one system, the CPU may be correlated to a great extent with network (*e.g.,* when network's operations are high due to the workload being generated, eventually CPU is high to accommodate such increase); on the other hand, on the same system, the correlation between CPU and memory may become low. Such change identified may expose performance issues (*i.e.,* high CPU without memory and–or network I/O operations might be due to a performance failure). For example, in our experiments, we experienced lots of unready pods as they had been running for more than five minutes and had not passed their readiness check, hence, we destroyed the formed clusters and started all over again.

However, if there is a significant difference in correlations simply due to the platform being used, i.e., Kubernetes vs. OpenShift, then practitioners may need to be warned that a correlation discrepancy may be false. Hence, we examined whether the relationship among performance metrics has a discrepancy between both container environments.

Approach

We calculated the Spearman's rank correlation coefficient among all performance metrics in the container environments and studied whether they are different. Spearman's correlation coefficient is a statistical measure of the strength of a monotonic relationship between paired data. In a sample, it is denoted by r_s and constrained to -1 $\leq r_s \leq$1. It is interpreted as, the closer r_s to $\pm$1 the stronger the monotonic relationship.

Correlation is an effect size (ES) measure and the strength of the correlation for the r_s is described as:

0.00-0.19 "very weak", 0.20-0.39 "weak", 0.40-0.59 "moderate", 0.60-0.79 "strong", and 0.80-1.0 "very strong".

Discussion

There exists differences in correlation among the performance metrics in Kubernetes and OpenShift. Tables III, IV and Figures 4 and 5 (*i.e.,* samples) show the changes in the correlation coefficient among the resources utilized in both environments. By closely looking at them, we find that in bare metal deployments, (CPU–Network) in Kubernetes has stronger correlation than that of OpenShift (*i.e.,* noticeable network's operations due to Hono's deployment workload); whereas (Memory–Network) coefficients in OpenShift are stronger than in Kubernetes. Furthermore, adding EnMasse results in stronger coefficients in Kubernetes, but that is not the case in OpenShift where all performance metrics show strong coefficients to handle such addition on top of Hono (*i.e.,* resources have been utilized fiercely where EnMasse has its own additional processes to be run on top of Hono).

For the sake of brevity, we do not show the detailed analysis here which can be consulted in our replication package indicated in the next section, Section V. Box-plots as well as Spearman's correlation trend-lines are depicted there-in to show the correlation changes among related metrics.

V. Threats to Validity

This section briefly discusses the threats to the validity of our conducted study taking into account the guidelines suggested by Wohlin et al. [11].

Construct validity threats concern the relation between both theory and observations, such as the measurements errors in this conducted study. We instrumented the different versions of deployments described in Section III to generate execution readings from which we computed min, max and average values of the performance metrics.

We repeated each experiment five times and computed the median values to mitigate the potential biases that could be induced by perturbations on the network, hardware and our tracing. We are confident that such repeated measurements

TABLE III: Spearman's rank correlation summary of performance metrics in Kubernetes

# Devices	CPU–Memory			CPU–Network			Memory–Network		
	min	max	avg	min	max	avg	min	max	avg
HONO Bare metal Deployment KO-0									
10 –> 100	-0.1025978	-0.1	-0.3077935	**0.6 (strong)**	**0.7 (strong)**	**0.6 (strong)**	0.2051957	0.1	0.2051957
HONO-EnMasse KO-1									
10 –> 100	0.1	0.1	0.2	**0.6 (strong)**	**0.6 (strong)**	-0.1	0	0	-0.1
HONO-EnMasse-App KO-2									
10 –> 100	-0.2	-0.3	-0.2	**0.7 (strong)**	**0.7 (strong)**	**0.6 (strong)**	0.2	0.3	0.5

TABLE IV: Spearman's rank correlation summary of performance metrics in OpenShift

# Devices	CPU–Memory			CPU–Network			Memory–Network		
	min	max	avg	min	max	avg	min	max	avg
HONO Bare metal Deployment KO-0									
10 –> 100	**0.6 (strong)**	0.1	0.3	-0.1	0.1	0.1	-0.3	**0.8 (v. strong)**	**0.6 (strong)**
HONO-EnMasse KO-1									
10 –> 100	**1 (v. strong)**	**0.6 (strong)**	**0.8 (v. strong)**	**1 (v. strong)**	**0.6 (strong)**	**0.9 (v. strong)**	**1 (v. strong)**	**1 (v. strong)**	**0.9 (v. strong)**
HONO-EnMasse-App KO-2									
10 –> 100	0.1	0.4	0.5	**0.6 (strong)**	**0.8 (v. strong)**	**0.6 (strong)**	0.3	-0.1	-0.1

increased the quality of our calibration and measurements. We monitored the performance of the containers since their creation until their destruction and combined the performance metrics for every minute together as a median value.

Internal validity threats concern our analysis method. Our empirical study is based on the performance testing results on subject systems. The way of conducing the performance tests and its quality may introduce threats to the validity of our findings. Particularly, our approach is based on the recorded performance metrics where their quality can have an impact on the internal validity of our study. Our performance evaluations all lasted for a duration of, roughly, four $\rightarrow$ six months, while the length of the evaluations may impact the findings of the conducted case study, we believe that we have observed and analyzed performance readings over a realistic amount of time. Another internal validity threat concerns the statistical analysis that are performed. To mitigate this threat, we paid attention not to violate the assumptions of the statistical tests. Specifically, we applied non-parametric tests that do not require making assumptions on the distribution of our dataset.

External validity threats concern the possibility of generalizing our findings. Further validation with different configuration are desirable to broader our understanding of the impact of Eclipse Hono deployment strategies on the resources utilization, and to provide guidelines to practitioners about the usage of such platform when developing and deploying IoT based applications.

Reliability validity threats concern the possibility to replicate this study. We attempt to provide all the necessary information and details to replicate our study. All the data used in the study are available online in our replication package [3].

Last but not least, the ***conclusion validity*** threats which refer to the relation between the treatment and the outcome do not affect this study since we paid attention to avoid violating the assumptions of the statistical tests used in our analysis. Nevertheless, replications of this study on more complex clusters using different applications are desirable to make our findings more generic.

VI. Related Work

IoT performance assurance activities play a vital role in the development of large IoT systems. Those activities ensure that the IoT, including developed platforms, meets the desired performance requirements [12]. However often, their failures are a result of performance issues rather than functional impairments [6], [13]. Failures can lead to eventual quality refusal of targeted systems with reputational and monetary consequences. To mitigate the resulted performance issues and ensure systems' reliability, practitioners often conduct performance evaluations [12] to be applied to workloads, *e.g.,* mimicking users' behavior in the field, on software systems [14], and monitor the related performance metrics, such as CPU cores usage, that are generated based on those carried tests and evaluations. Conventionally, they use such metrics to gage the performance of the, being worked on, systems and identify potential performance issues, such as memory leaks [15], memory allocated consumption and network throughput bottlenecks [16].

Moreover, practitioners using new systems need guidance on how to build such platforms and–or deploy their applications efficiently on top of them. They need to have the know-how to pick up the right configurations and frameworks since the participating devices are resources constrained; devices are not optimal in terms of resources utilization, *i.e.,* CPU, memory, network, etc., and their misuse is likely to significantly degrade the Quality of Service (QoS) as well as User Experiences (UEs). Prior research has suggested a slew of techniques to analyze performance testing results, *i.e.,* performance metrics. Those techniques, typically, examine the following aspects metrics: (a) single performance metric, and (b) performance metrics relationships.

Single performance metrics Malik et al. [17], [18] suggest approaches that cluster performance metrics using Principal Component Analysis (PCA) where each component generated is mapped to its performance metrics by a weight value. Such value measures how far a metric can contribute to the component. For every performance metric, a comparison is conducted on each component's weight value so that performance regressions are detected.

Relationship between performance metrics Malik et al. [16] leveraged Spearman's rank correlation to capture the relationship that arises between performance metrics. The correlations deviance is measured so that subsystems are pinpointed to take responsibility of the occurring performance deviation.

Analysis of virtual machines overhead Kraft et al. [19] discuss the issues pertaining disk I/O in a virtual environment; they examine disk request–response time performance degradation by recommending a trace-driven approach. They assert on the existing latencies in a virtual machine requests for disk I/O because of the time increments accompanying the request queues. Another work conducted by Huber et al. [20] presents a study on Cloud-like environments. They compare the performance of the virtual environments and conduct analysis on their performance degradation. They further categorize

[3]Complete results, data and scripts are shared online at osf.io scientific data repository: https://tinyurl.com/moaly-ec-hono

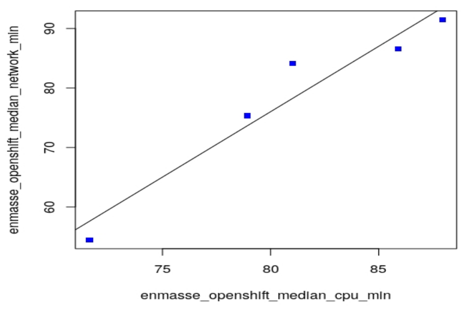


Fig. 5: Spearman's correlation changes for OpenShift - Hono-EnMasse (CPU–Network) min.

the factors that influence the overhead and use regression based models to evaluate them, however, their modeling was only considering CPU and memory.

VII. Conclusion

Performance assurance activities are critical in enforcing software and systems' reliability. The discrepancy between performance testing in Eclipse's IoT container technologies haven't been attempted to be evaluated yet, hence, we aimed to highlight whether there are differences between Kubernetes and OpenShift environments while handling different Hono deployments. In this paper, we step up loads to examine the performance costs related to containers deployment scenarios. By examining the results, we find that there exists discrepancies between performance metrics while considering three different modes:

- Bare metal deployment of Hono on top of Kubernetes and OpenShift.
- Scaling out Hono and incorporating EnMasse (as a messaging as a service) infrastructure to the architecture.
- Adding business applications to the platform so that we have the complete paradigm of the system.

Such results not only provide important guidelines for building and deploying Hono, EnMasse and developing IoT based applications, but also aim to provide understanding of the performance implications of their design so that practitioners, including end users, can make right deployment decisions. The main contribution of this paper includes: (1) Our paper is one of the first attempts to evaluate the discrepancy in the context of analyzing performance testing in Eclipse IoT–Hono. (2) We found unbalanced relationships among related performance metrics (*i.e.,* ranging between strong, moderate and weak) between Kubernetes and OpenShift environments. Therefore, practitioners cannot assume a straightforward overhead from container environments (such as a simple increment of CPU).

Last but not least, our findings highlight the need to be aware of and to reduce the discrepancy between performance testing results in container environments (*i.e.,* especially in open-sourced platform(s)), such as Eclipse-IoT Hono.

References

[1] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 2–11.

[2] N. Nikzad, O. Chipara, and W. G. Griswold, "Ape: an annotation language and middleware for energy-efficient mobile application development," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 515–526.

[3] I. The Eclipse Foundation, "Open source software for industry 4.0, an eclipse iot working group collaboration," Made available under the Eclipse Public License 2.0 (EPL-20), Online: https://iot.eclipse.org/resources/white-papers/Eclipse%20IoT%20White%20Paper%20-%20Open%20Source%20Software%20for%20Industry%204.0.pdf, accessed (2017/10/20), 2017.

[4] B. Cabé, "Key trends from the iot developer survey 2018," Available at https://blogs.eclipse.org/post/benjamin-cab%C3%A9/key-trends-iot-developer-survey-2018, accessed (2018/02/12), 2018.

[5] E. Hono, "Eclipse hono," Available at https://www.eclipse.org/hono/, accessed (2018/02/01), 2017.

[6] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "Mining performance regression testing repositories for automated performance analysis," in *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 2010, pp. 32–41.

[7] J. H. Stapleton, *Models for probability and statistical inference: theory and applications*. John Wiley & Sons, 2007, vol. 652.

[8] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[9] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[10] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control." in *OSDI*, vol. 4, 2004, pp. 16–16.

[11] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[12] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 171–187.

[13] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[14] M. D. Syer, W. Shang, Z. M. Jiang, and A. E. Hassan, "Continuous validation of performance test workloads," *Automated Software Engineering*, vol. 24, no. 1, pp. 189–231, 2017.

[15] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 110–119.

[16] H. Malik, B. Adams, and A. E. Hassan, "Pinpointing the subsystems responsible for the performance deviations in a load test," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 2010, pp. 201–210.

[17] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora, and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. IEEE, 2010, pp. 222–231.

[18] H. Malik, H. Hemmati, and A. E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1012–1021.

[19] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "Io performance prediction in consolidated virtualized environments," in *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5. ACM, 2011, pp. 295–306.

[20] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and modeling virtualization performance overhead for cloud environments." in *CLOSER*, 2011, pp. 563–573.