Cloud Computing With Kubernetes Cluster Elastic Scaling

Brandon Thurgood
Dept. of Computing
Letterkenny Institute of Technology
Letterkenny, Co. Donegal, Ireland
Brandon.Thurgood@outlook.com

Ruth G. Lennon
Dept. of Computing
Letterkenny Institute of Technology
Letterkenny, Co. Donegal, Ireland
Ruth.Lennon@lyit.ie

***Abstract*—Cloud computing and artificial intelligence (AI) technologies are becoming increasingly prevalent in the industry, necessitating the requirement for advanced platforms to support their workloads through parallel and distributed architectures. Kubernetes provides an ideal platform for hosting various workloads, including dynamic workloads based on AI applications that support ubiquitous computing devices leveraging parallel and distributed architectures. The rationale is that Kubernetes can be used to support backend services running on parallel and distributed architectures, hosting ubiquitous cloud computing workloads. These applications support smart homes and concerts, providing an environment that automatically scales based on demand. While Kubernetes does offer support for auto scaling of Pods to support these workloads, automated scaling of the cluster itself is not currently offered. In this paper we introduce a Free and Open Source Software (FOSS) solution for autoscaling Kubernetes (K8s) worker nodes within a cluster to support dynamic workloads. We go on to discuss scalability issues and security concerns both on the platform and within the hosted AI applications.**

Keywords—Autoscaling, Kubernetes, Artificial Intelligence, parallel and distributed architectures, Infrastructure as a Service, Container as a Service

I. INTRODUCTION

Devices that are compatible with ubiquitous computing are typically small in order to allow them to remain unobtrusive, which generally limits their processing power and ability to run Artificial Intelligence (AI) based applications. AI applications processing the information from devices on the sensor network allow the devices to adapt to the environment efficiently as more data becomes available. Transferring the data to and from the sensor network can be achieved by leveraging technologies such as Radio Frequency Identification (RFID) technology, Wireless Sensor Networks (WSN) or Near Field Communication (NFC) devices [1]. Another method that can be used to overcome this limitation is to leverage a single device within the sensor network capable of internet connectivity, such as a smartphone or tablet device. A single device with internet connectivity within the sensor network could allow ubiquitous devices to communicate with cloud-based systems by leveraging the device as a proxy. The cloud-based system can then perform more complex computations of the data and communicate results with the devices and surrounding architecture, while also improving user experience.

Ubiquitous computing devices such as Wi-Fi, RFID or WSN enabled armbands sold in the form of concert tickets can communicate directly with containerized AI applications hosted on the Kubernetes cloud platforms, providing valuable information that can be used to both track and predict crowd movement. Crowd movement detection can be achieved through several methods such as video-based or signal-based identification methods. Video-based methods such as Mid Based Foreground Segmentation and Head-Shoulder Detection [2] are costly to implement as they require both cameras and vast amounts of storage to host the video files. Signal-based methods for detecting crowd movement typically function upon radio frequency identification (RFID) [3] tags requiring dedicated sensing equipment to be placed at the venue. At present, there is a promising method for detecting the number of people in a queue by utilizing the widespread Wi-Fi signal to extract the received signal strength (RSS) or channel state information (CSI) [4], however these methods are unlikely suited for dense crowd counting and movement detection within confined spaces. While a combination of these crowd movement methods could possibly be leveraged to communicate with applications hosted on cloud architecture, this paper focuses on the elastic scaling of a Kubernetes cluster based on demand.

Allowing interconnected systems to respond to certain types of crowd movements by changing the environment hosts a plethora of possibilities. Should an environment be able to adjust to certain types of crowd movements, for instance by widening or opening additional doorways, not only could user experience be improved, overcrowding hazards could also be prevented.

Smart homes connected via sensors to containerized AI-based applications running on Kubernetes could improve living experiences of users by automatically adjusting the environment within the home. Adjusting elements such as lighting, temperature and music to the needs of the user could vastly improve experience in an unobtrusive manner. Ubiquitous computing within the home powered by Kubernetes and its ability to auto scale provides endless possibilities to automating and improving living experiences.

As more devices join the network, the cloud-based system in turn is required to scale in accordance with load in order to maintain stability and provide the best user experience in a sustainable method. While commercial cloud-based solutions such as Google Kubernetes Engine (GKE) provide this functionality in public clouds, as of the time of writing no free and open source solution existed for elastically scaling private cloud or on-premise K8s clusters. This paper introduces a Free / Open-Source Software (FOSS) solution based on Kubernetes (K8s). The Infrastructure as a Service (IaaS) layer of this

ICFNDS '19, July 1–2, 2019, Paris, France

ACM ISBN 978-1-4503-7163-6/19/07…$15.00
https://doi.org/10.1145/3341325.3341995

solution is currently based on VMware vSphere technology in a private cloud model which is both proprietary and costly, however the solution can easily be adapted to leverage platforms such as oVirt or libvirt to make it entirely FOSS, with only the likes of server and networking hardware incurring cost. The solution is also adaptable to elastically scale into hybrid cloud architectures and leverage edge computing.

II. Related Work

Cloud computing can take many forms, supporting various devices which are backed by a myriad of supporting infrastructure.

A. Ubiquitous Computing

While ubiquitous computing is intended to work transparently to the user [5], advanced levels of computation are required for the solution to remain effective, which the sensor devices are typically incapable of. Having a single device on the sensor network capable of internet communication will allow the sensor data to be uploaded to the cloud and analysed before instructions are made available for the devices to download and execute. In order to achieve these advanced levels of computation and AI processing a containerized cloud-based system [6] running on Kubernetes is proposed. This solution will need to auto-scale based on load, as it would be difficult to predict load on the system as users interact more and move in and out of network coverage areas, as is the norm in ubiquitous computing.

B. Edge and Fog Computing

Edge computing, commonly referred to as just "edge", brings processing close to the data source, eliminating the need for the data to be sent to a remote cloud or other centralized system for processing. Elimination of the distance and time it takes to transport data to centralized sources improves the speed and performance of data transport which in turn improves applications performance on the edge. Edge computing can potentially address the concerns of response time and bandwidth constraints inherent with cloud computing [7].

Fog computing is a defined standard of how edge computing should work. It facilitates the operation of compute, storage and networking services between edge devices and cloud computing hosted in the datacentre.

C. Infrastructure as a Service

Cloud-based systems capable of elastically scaling [8] and interacting with ubiquitous computing sensor networks require an Infrastructure as a service component such as VMware vSphere to run the workloads. This layer provides computational abilities far beyond that of individual ubiquitous computing devices on the sensor network and provides an environment for the Kubernetes cluster nodes to both run and scale. Thus, IaaS can be exploited to support Ubiquitous Computing.

D. Container as a Service

Artificial intelligence components of ubiquitous computing systems should run in containerized environments following 12-factor designs. This design will allow them to scale as required and to accommodate the unpredictable load sensor networks will place on the system [9]. Kubernetes provides an ideal platform for this type of workload. While K8s does provide a scaling service known as the Horizontal Pod Autoscaler (HPA), functionality to elastically scale the number of cluster worker nodes is not currently offered within the platform itself. The clusters ability to automatically scale would enable scaling support for ubiquitous computing beyond the limits of the cluster, not only on the user facing components but on the operational and supporting services as well. In order to successfully support ubiquitous computing, it is proposed that all elements of the system elastically scale vertically and horizontally on demand. Horizontal scaling will be implemented in the form of increasing compute resources through additional worker nodes rather than vertical scaling which entails adding resources to existing nodes. Horizontal scaling was selected as it requires zero downtime as apposed to vertical scaling which requires hosts to be powered down before adding additional resources.

E. Distributed Architecture

Distributed architecture is a software system with interconnections between a collection of independent systems. Coordination and communication is established between the systems through API calls or message passing, with the intention of achieving a common goal. This type of architecture can be leveraged extensively in various designs including but not limited to application, infrastructure and network design [10].

F. Artificial Intelligence

Artificial intelligence technologies are becoming increasingly prevalent, with their impact on individuals and societies varying widely [11]. While AI has no generally accepted definition, the term obscures the actual mechanism, with the possibility of hiding untrustworthy methods [11]. Implementation of AI methods without rigorous integrity can lead to devices and systems that are untrustworthy and sometimes dangerous [12]. Systems that are aware of the location of dense crowds of people and which control mechanisms such as opening and closing of doors can have profound negative impact if not implemented in a failsafe trustworthy manner.

III. Proposed System and its Parameters

Cloud and ubiquitous computing in the context of this paper may take the form of a smart home with interconnected devices throughout, consisting of the user wearing a smart watch that interacts with distributed sensors, all communicating via Wi-Fi with the containerized AI application hosted on Kubernetes. The sensor network could not only turn lights on when a room in entered, a variety of other functions could be performed based on the constantly uploaded sensor data to the Kubernetes cloud platform which processes the data and can provide constant feedback to the sensor network. The AI applications could trigger actions such

as setting of ambient lighting or relaxing music based on mood, posture or a variety of other factors.

Having the ubiquitous computing sensor network respond to both physical motions, number of inhabitants and various other inputs would allow for a fully interactive experience in an unobtrusive manner. As the number of users interacting with the platform are likely to fluctuate, as family and guests come and go, or new sensor networks are onboarded, the platform is able to automatically scale both in the form of containers spinning up as required within the cluster, and the cluster itself scaling new worker nodes as the number of containers consume the capacity of the cluster.

Another scenario in which ubiquitous computing devices can leverage the cloud platform would be through the form of sensor network connected armbands sold as concert tickets. As users enter the arena, the platform could be used to track the number of concertgoers entering the stadium, which entrances were used, compressed areas within the arena that require attention as well as several other use cases, particularly in emergency situations should they arise. Having a flood of devices either join or leave the network requires a platform that can elastically expand and contract, which is one of the key focus areas of the proposed solution.

The proposed solution was primarily tested with on-premise private cloud infrastructure; however, the design could theoretically be adapted to run in hybrid cloud or edge computing designs as well. An in-depth discussion of the solution and the need to scale beyond the cluster boundary can be found in [13].

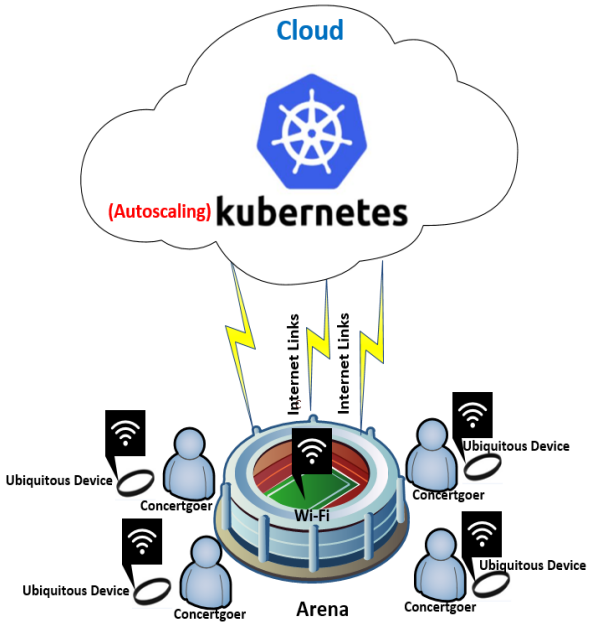

Figure 1. Ubiquitous computing powered by Kubernetes cluster autoscaling

The solution consists of virtual machines that make up the Kubernetes platform running on ESXi hosts, managed by VMware vCenter. This design choice was made due to market penetration analysis of virtualization in private clouds, according to Smart Profile's analysis in 2017, VMware held seventy five percent of the server virtualization market [14]. The selection of Kubernetes as the container orchestration platform was based on its widespread adoption in the market and the fact that it has become commonly known as the standard for container orchestration. Within the Kubernetes virtual machines are Linux operating systems based on Ubuntu 16.04.5 LTS, which in turn have Docker runtime 17.03.3-ce installed, providing the container execution environment. Ubuntu was selected as it is the standard platform for K8s, while Docker was selected due to its tight integration with K8s and wide industry adoption. In order to manage the container-based workloads, Kubernetes v1.13.0 provided a container orchestration platform which consisted of 3 master nodes and 3 worker nodes, which are the base of the unscaled cluster configuration. A minimum of three master nodes are required to establish a redundant control plane, as this is required for etcd to maintain quorum should a single master node fail. The VM scaling solution was based on Foreman Version 1.19.1. Foreman is a complete lifecycle management tool for physical and virtual servers. The Foreman implementation was deployed as a virtual machine based on CentOS Linux 7 (Core). Foreman was selected as it is completely FOSS as opposed to many market contenders, as well as its tight integration with VMware products and adaptability to other platforms. The installation of Foreman utilized a collection of Puppet modules and configured the Puppet master at version 5.5.8 to control both Foreman and the scaled Kubernetes worker node VMs from the same server.

The ingress solution, which was based on HA-Proxy version 1.6.3 was run on additional virtual machines based on Ubuntu 16.04.5 LTS. The HA-Proxy design choice was based on it being FOSS and its wide industry adoption. In addition to these servers there was a VM used to manage the Kubernetes cluster and act as the CA (Certificate Authority). The Public Key Infrastructure (PKI) server used was CFSSL, CloudFlare's PKI/TLS toolkit and was selected due to widespread usage on K8s and available documentation.

The virtual machines in which the Kubernetes nodes run should be distributed across a minimum of three physical ESXi nodes, with anti-affinity rules configured on the vCenter to separate the VMs in a single-master and single-worker node per physical ESXi host configuration. This design is intended to provide redundancy to support the ubiquitous computing devices allowing the solution to remain entirely functional in the event of virtual machine or physical host failure.

The container network was implemented using Weave Net. Weave Net implements industry standard VXLAN encapsulation between hosts to create a virtual overlay network that connects Docker containers across multiple hosts and enables their automatic discovery.

Communication with the containerized applications running within the Kubernetes environment was established by initiating connections through separate virtual machines configured to run HA-Proxy version 1.6.3 in a Virtual Router Redundancy Protocol (VRRP) Active/Active Cluster configuration. HA-Proxy was then configured to relay connections to the Kubernetes ingress API, which manages external access to services within the cluster. This design contributed to the level of redundancy required for the solution to support the dynamic workloads. The HA-Proxy VMs should be governed by anti-affinity rules in the vCenter environment forcing them to run on separate physical ESXi hosts to increase their redundancy.

The ability for this solution to elastically scale virtual machines on the vCenter managed IaaS platform was provided by the lifecycle management tool named Foreman. Foreman interfaces with the vCenter API to trigger creation of additional virtual machines, based on preconfigured templates. The templates were hosted on the vCenter platform and contained the base Linux OS based on Ubuntu 16.04.5 LTS, with the Kubelet package and configuration scripts preinstalled.

The Foreman tool managed both the Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) solutions. DHCP was based on Internet Systems Consortium, Inc. (ISC) DHCP, and DNS was based on ISC DNS which is based on Berkeley Internet Name Domain (BIND) version 9.

Scaling worker nodes was initiated via a vCenter alarm which triggered once a specified Central Processing Unit (CPU) or Memory threshold was reached and maintained for a defined number of minutes within a control VM, which was one of the worker nodes in the base configuration of the Kubernetes cluster. Configuration of the vCenter alarm can be seen in figure 2. The control VM selection can be any worker node in the cluster however only a single VM should be monitored in order to avoid scaling multiple VMs simultaneously. The reason any worker node can be selected is due to the Horizontal Pod Autoscaler (HPA) distributing Pods onto all available worker nodes at the time of initial scaling. The alarm was configured to execute a bash script hosted on the vCenter server when it fires. The performance metrics affected by the synthetic application load can be seen in figure 3.

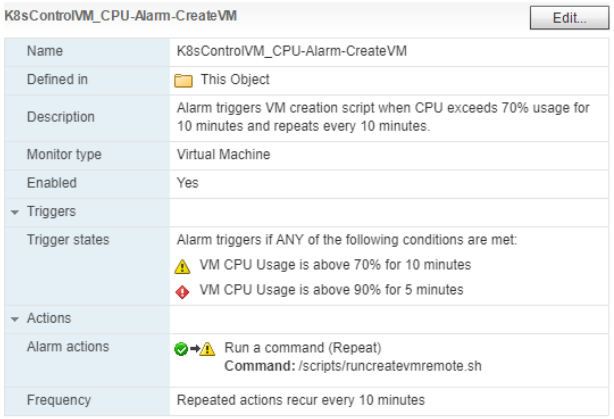

Figure 2. The vCenter create-VM alarm

Synthetic application load that triggered elastic scaling was generated using a tool named Locust, this tool simulated users accessing a web app hosted in a Pod within the Kubernetes cluster. This page performed CPU intensive calculations purely to simulate load. In a real-world scenario this application would be based on AI code that interacts with the relevant ubiquitous computing devices. The synthetic load generated by Locust can be seen in figure 4.

Figure 3. Control VM's CPU performance metrics with load.

Figure 4. The Locust tool generating load.

The bash script was configured to execute a separate script on the remote Foreman server which performed multiple functions. First a hostname with a random unique integer appended was generated, both stored as a variable and written to a text file. The script then called the Command Line Interface (CLI) tool for Foreman, named Hammer. Switches were passed to the Hammer CLI tool, which include the unique hostname which was stored as a variable, as well as switches that instruct Foreman to create a VM from template, based on preconfigured values within Foreman, such as which template to instruct the vCenter to clone, number of vCPUs etc. Various preconfigured values exist within the Foreman tool which allow Hammer to trigger a preconfigured VM build process. The process the script follows to create the scaled Kubernetes worker nodes can be seen in figure 5.

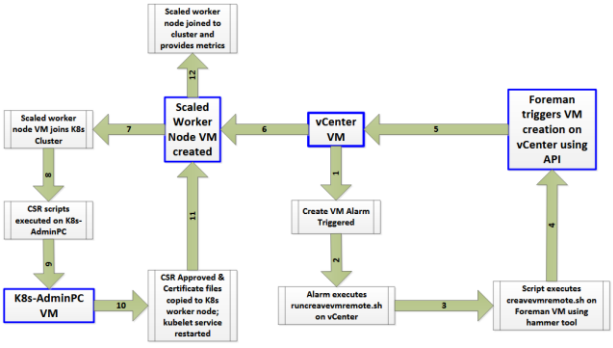

Figure 5. The create VM script process

When the Hammer CLI tool executes the API call to Foreman, several tasks are initiated. First an API call to the vCenter server is made calling for a clone to be created from the preconfigured template, with the same name passed as the hostname switch via the CLI. Next a DHCP reservation is created based on the Media Access Control (MAC) address returned from the vCenter server during the API call, which is the configured MAC address of the new VM, using an available Internet Protocol (IP) address from the configured DHCP pool of addresses. A DNS A and PTR record are then added to the BIND zone file listing the hostname previously passed as a switch to the Hammer tool, with the same IP address configured in the DHCP reservation based on the newly created VMs MAC address. This allows the VM to boot with an expected IP address and hostname, allowing Puppet to connect to the VM once booted and complete configuration.

Configuration of the scaled VMs once booted is controlled by the Puppet tool, which Foreman interacts with via its API. A preconfigured Puppet script which is hosted within Foreman is executed on the booted VM via the Puppet master and is used to perform various functions, including an update of installed packages, installation of new packages if required and perform any other defined configuration tasks. As part of the Puppet script, the hostname is configured to be the same as that registered in DNS, then a command is executed to join the scaled VM to the Kubernetes cluster as a worker node. For the command to function indefinitely a non-expiring bootstrap token was generated on the Kubernetes platform.

For the solution to automatically downward scale both VMs and K8s worker nodes, a separate vCenter alarm was configured to monitor CPU or Memory and trigger when the configured threshold is below the defined threshold for a defined number of minutes. The alarm configuration can be seen in figure 6. When the alarm is triggered a script is executed on the vCenter server to remove the worker node from the Kubernetes cluster and power the VM down and delete it. This activity can be seen in figure 7. The script process to delete Kubernetes worker nodes and their associated VMs can be seen in figure 8.

ScaleVMdown — Edit...

Name	ScaleVMdown
Defined in	This Object
Description	
Monitor type	Virtual Machine
Enabled	Yes
Triggers	
Trigger states	Alarm triggers if ANY of the following conditions are met: VM CPU Usage is below 20% for 10 minutes VM CPU Usage is below 10% for 30 minutes
Actions	
Alarm actions	Run a command (Repeat) **Command:** /root/script.sh
Frequency	Repeated actions recur every 5 minutes

Figure 6. The vCenter delete-VM alarm.

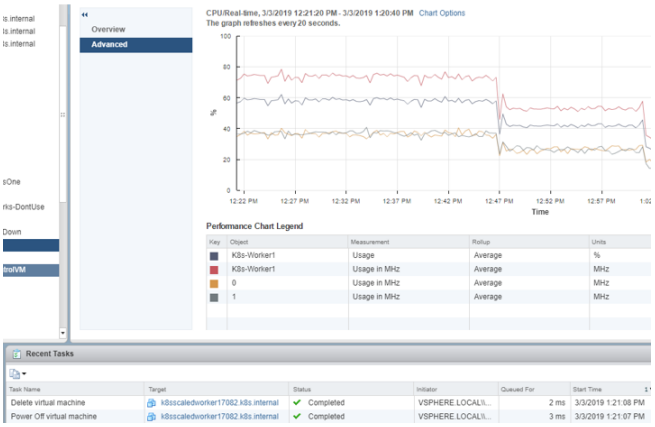

Figure 7. VM deletion triggered by load reduction.

The bash script used to scale inwards or remove worker nodes and VMs called a separate script which was run on the remote Foreman server where several commands and additional scripts were also executed.

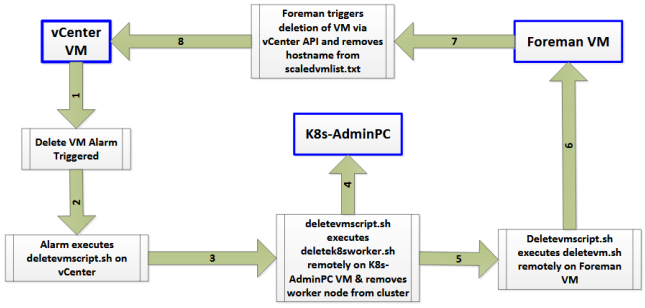

Figure 8. The delete VM script process

First, Secure Copy (SCP) was called to copy the text file containing the scaled VM names created during execution of the upward-scaling script to the VM used to manage the K8s cluster. SCP was then called again to copy the same text file to the Foreman VM. A separate bash script was then called from within the script to execute on the remote K8s management VM and remove the scaled worker node from the cluster. The script used the Linux sed command to parse the top line of the text file for the relevant hostname and passed that into the kubectl delete node command as a switch. Once the remote script used to remove the K8s node completed, a second script was called from within the original script to execute on the remote Foreman server, again parsing the top line of the text file containing scaled hostnames using the Linux sed command, then passing that as a variable to the Hammer CLI command to delete the specified VM. The Hammer CLI tool then called the vCenter API to power the VM down and delete it. The sed command was then used to remove the first line from the file containing hostnames, this allowed the process to complete on any additional scaled VMs.

The process proved to be a robust solution for automatically and elastically scaling K8s worker nodes hosted on the vSphere IaaS platform. As application load increased within the Kubernetes cluster, available resources were consumed on all available worker nodes. This triggered elastic scale-out which introduced more resources into the cluster, making them available to serve existing and further increased load. As application load was either decreased or removed

entirely the solution triggered scale-in activity removing all unnecessary worker nodes in the cluster.

The cost savings gained through this form of elastic cluster scaling, based on public cloud VMs referenced in [15], are US$0.10 per virtual machine/hour. Based on these figures, increasing the cluster size with three additional nodes will cost approximately $219 per month, whereas elastically scaling out as needed within the month may cost US$50.40 based on typical usage*. This reduction in costs could equate to as much as a 76.99% saving.

*Typical usage is defined as intermittent bursts with a maximum of 1 week per month.

IV. Discussion

This solution brings with it a vast number of use cases. In addition to the two scenarios listed, it could be used to support medical, law enforcement, agriculture, traffic and a myriad of other use cases.

Possibly one of the most crucial aspects of this solution is security. Any system that is aware of the location and movement of people is likely to be a target for nefarious individuals and systems to exploit the data. Not only will securing the data be paramount, how the system uses that data could lead to dangerous situations. For example, should the AI be implemented in an untrustworthy or unsafe manner, crowds fleeing towards an exit could be obstructed by doors closing rather than opening. Threat agents who gain access to the system or data and can manipulate its functionality could in theory force the system to act in a dangerous manner.

During testing the HPA was found to distribute workload unevenly amongst available worker nodes. Under load across 3 nodes and 20 scaled Pods, a variance of as much as a 20% CPU activity was witnessed. Based on this the control VM selection should be based on analysed workload distribution within the environment.

While the solution can horizontally scale by adding Pods to existing worker nodes very fast through the HPA, the much slower speed at which it can scale the cluster by adding new worker nodes should be taken into consideration when configuring alarm thresholds. Under testing conditions scaled worker nodes were only active in the Kubernetes cluster 6-8 minutes after threshold alarms fired and triggered VM builds. This is due to several factors such as the hardware type and configuration, amount of time it takes to clone the VM from template, boot the operating system, configure it and join it to the Kubernetes cluster.

Based on the amount of time it takes to complete the elastic scaling activity, the solution should be configured to only respond to reasonable periods of increased load on the system. Should the alarm threshold be configured to fire after short periods of increased load, VMs may still be in the process of creating while the load drops below the requirement for that added worker node. Should the alarm thresholds be configured to fire only after extended periods of time, user experience may be impacted. Careful consideration should be taken to determine alarm threshold parameters to cater for each of these factors.

While the base system is configured with three worker nodes, which the solution is not able to automatically reduce past that, additional worker nodes can manually be added to the base configuration in which case the solution would only scale once usage across all existing base nodes exceeds the configured threshold. Therefore, when the solution is deployed, a performance baseline should be taken to determine the optimal number of worker nodes required for the system to run optimally and the base number of worker nodes adjusted accordingly.

Configuration of the MaxPods value in the HPA should be carefully evaluated as it could inhibit the efficacy of the solution. Once the MaxPods value is reached, the Pods will no longer scale within the cluster unless this value is set large enough or until it is increased once worker nodes are added. Dynamically adjusting the HPA MaxPods value as part of the elastic scaling activity would be effortless to implement.

Testing of the solution was conducted using synthetic load based on a containerized web application that performed a CPU intensive mathematical calculation every time the web page was hit, based on the K8s HPA example Pod. Load was generated by using a tool named Locust, which is an open source load testing tool [16]. In order to generate load that proved enough to maintain above 70% CPU utilization on the control VM, 300 users were simulated at a hatch rate of 300.

This solution could yield positive results by hosting the base cluster on private cloud architecture for security reasons, while elastically scaling either into a hybrid cloud design or onto edge devices to increased accessibility and reduced latency and bandwidth consumption.

V. Conclusion and Future Work

While this research was based on a proprietary IaaS solution, additional research could produce an entirely FOSS solution. Foreman has built-in support for oVirt and libvirt which can be leveraged; however, the alarming solution will also need to be adapted as it is currently based on vCenter performance alarms. Use of Prometheus and Alertmanager would likely yield positive results in triggering VM builds through the Foreman API. This solution provides a dynamically scaling support infrastructure for ubiquitous computing which can be used in a variety of different use cases. Running AI, although not a requirement, is likely to yield advances in in the field.

This research provides a discussion on scalability issues but the issue of security within such devices remains a concern as previously mentioned. Whilst many solutions have been put forward [17, 18] it is clear that attacks on ubiquitous devices and their associated AI applications hosted on cloud infrastructure can range from issues regarding privacy to endangering lives [19]. Further issues in ubiquitous and cloud computing include the human aspects [20] including interaction and design and contextual usage. It is clear that much research in this field is yet to be done.

Acknowledgment

The authors would like to thank Letterkenny Institute of Technology for their funding of this research work.

References

[1] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M., 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), pp.1645-1660.

[2] Li, M., Zhang, Z., Huang, K. and Tan, T., 2008, December. Estimating the number of people in crowded scenes by mid based foreground segmentation and head-shoulder detection. In *2008 19th International Conference on Pattern Recognition* (pp. 1-4). IEEE.

[3] F. Xiao et al., "One More Tag Enables Fine-Grained RFID Localization and Tracking," IEEE/ACM Trans. Networking, vol. 26, no. 1, Jan. 2018, pp. 161–74

[4] Xiao, F., Guo, Z., Ni, Y., Xie, X., Maharjan, S. and Zhang, Y., 2019. Artificial Intelligence Empowered Mobile Sensing for Human Flow Detection. *IEEE Network*, *33*(1), pp.78-83..

[5] Satyanarayanan, M., 2001. Pervasive computing: Vision and challenges. *IEEE Personal communications*, *8*(4), pp.10-17.

[6] Aguilera, X.M., Otero, C., Ridley, M. and Elliott, D., 2018, July. Managed Containers: A Framework for Resilient Containerized Mission Critical Systems. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 946-949). IEEE.

[7] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L., 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), pp.637-646.

[8] Lorido-Botrán, T., Miguel-Alonso, J. and Lozano, J.A., 2012. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, *12*, p.2012.

[9] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J., 2016. Borg, omega, and kubernetes.

[10] Guingo, P., Mouilleron, V., Jansen, A. and Damm, G., Alcatel-Lucent SAS, 2012. Distributed architecture for real-time flow measurement at the network domain level. U.S. Patent 8,095,640.

[11] Grosz, B.J. and Stone, P., 2018. A Century Long Commitment to Assessing Artificial Intelligence and its Impact on Society. *arXiv preprint arXiv:1808.07899*.

[12] Parnas, D.L., 2017. The real risks of artificial intelligence. *Communications of the ACM*, *60*(10), pp.27-31.

[13] Thurgood B., Lennon R. G., Elastic Scaling of Kubernetes Cluster Nodes on Private Cloud Infrastructure, MSc in Cloud Computing, Letterkenny Institute of Technology, 2019.

[14] Smart Profile. (2017). VMware by far the largest in the server virtualisation market. Available: https://www.smartprofile.io/analytics-papers/vmware-far-largest-server-virtualisation-market/. Last accessed 8th May 2019.

[15] De Assunção, M.D., Di Costanzo, A. and Buyya, R., 2009, June. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In Proceedings of the 18th ACM international symposium on High performance distributed computing (pp. 141-150). ACM.

[16] Locust.io. Locust – A modern load testing frame-work 2014. Available from: http://locust.io/

[17] Astorga, J, Matías, J., Sáiz, P. and Jacob, E., 2009. Security for Heterogeneous and Ubiquitous Environments Consiting of Resource-Limited Devices: An Approach to Authorization Using Kerberos. Lecture Notes of the Institute for Computer Sciences, Social-Informatiics and Telecommunications Engineering, 42, pp. 65-77

[18] Shen, J., Liu, D, Shen, J., Liu, Q. and Xingming, S, 2017. A secure cloud-assisted urban data sharing framework for ubiquitous-cities, Pervasive and Mobile Computing, 41, pp 219-230.

[19] Kusen, E. and Strembeck, M, 2016. A decade of security research in ubiquitous computing: results of a systematic literature review. International Journal of Pervasive Compuitng and Communications, 12, pp. 216-259.

[20] López, G., Marín, G. and Calderón, M., 2016, Human aspects of ubiquitous computing: a study addressing willingness to use it and privacy issues, Journal of Ambient Intelligence and Humanized Computing, 8(4), pp. 497-511.