Orchestrating State at Scale

A. McClernon*
*The University of Melbourne, School of Computing and Information Systems

2184 Words (+-10% of 2000)

***Index Terms*—Container, Container Orchestration, Cloud Computing, Stateful Orchestration, Minecraft, Elasticity**

I. Introduction

Cloud computing has seen widespread adoption over the past decade, giving rise to increased attention to how resources are virtualized and brokered to applications. From the first public cloud in 2007 by AWS, the default method for brokering resources to the client has been *hypervisors*. Each server has an hypervisor, which acts to simulate hardware and execute requests from *virtual machines* (VMs). Each VM contains it's own operating system and user software, isolating it from other VMs running on the same physical hardware. Hypervisors incur substantial overhead cost [1] and despite being the incumbent resource virtualization technology, there has been a significant shift to *container virtualization* [2]. Container based virtualization provides less cumbersome process isolation, with recent studies demonstrating measurable decreases in resource overhead, startup and shutdown times over hypervisor virtualization (VMs) [3] [4]. Containers have also demonstrated improvement in distributed application management, enhancing application portability across operating systems [5] [4].

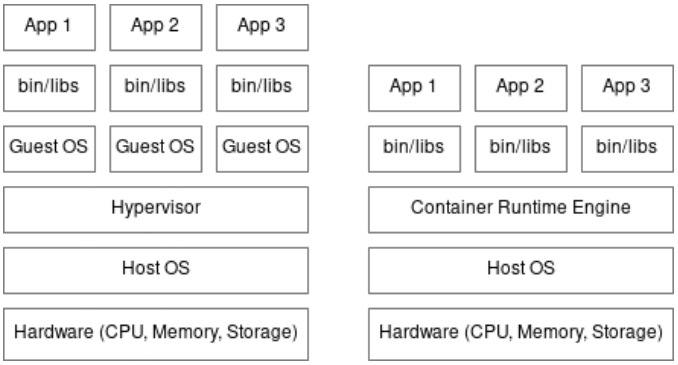

Fig. 1. Hypervisor Virtualization (left). Container Virtualization (right)

Containers are isolated by design, requiring configuration for inter-container communication; modern applications deploy hundreds to thousands of containers [6], necessitating the use of an organized framework to manage complexity. *Container orchestrators* manage resources, scheduling, load balancing, health checking, fault tolerance and auto scaling of containers. Container Orchestration first inception was Borg [7], developed by engineers at Google manage their containers. Following it's success, Kubernetes was released by Google - currently the most popular orchestration framework in use [8].

Stateless applications, which store application data **solely** in persistent storage (database, file-system) are easily orchestrated. An orchestrator can load balance traffic from one stateless service to another as they are functionally interchangeable, holding homogeneous (no) state [9]. Stateful applications however cannot, as each applications state depends upon previous requests it has received. This makes orchestrating stateful services uniquely challenging, requiring additional coordination and complexity.

II. State of the Art

The earliest published research into container orchestration began in 2015 [10], since then there has been accelerating rate of research; several taxonomies and literature reviews have also been conducted to establish the state of the art over that period to now [11] [12] Upon performing a broad synthesis of the current literature, the author in [9] notes that only as recently as 2017 was attention directed towards examining the role of container orchestration under stateful requirements in [5]. Further, since 2017 there has only been three publications on this topic [5] [8] [13].

Netto et al has examined two novel methods for state replication, DORADO and Koordinator [5] [8]. In both methods, the authors investigated the efficacy of stateful replication in an orchestrated environment, using an integrated and external approach. In both papers, the experiment environments were identical, consisting of only 4 VMs for testing. Vayghan et al propose an integrated method for achieving high availability stateful orchestration. The authors establish a Kubernetes integrated method, reporting an improvement in recovery time by 55% over a non-orchestrated solution. However, much like [5] [8], only 4 VMs underlie the deployment used in the experiment environment; detailing an unrealistically low level of distribution and scale, when compared to deployments in broadly related orchestration research [14] and in industry [15].

III. Identification

Stateful container orchestration is relatively new and has many unexplored areas of research in comparison to the more broadly defined research area of cloud computing [12]. There are several open research problems (**RP**), identified in earlier work [9]. They are summarized below.

- **RP1:** Stateful orchestration under elastic demand requirements.
- **RP2:** Comparison of the existing platforms for stateful application management as opposed to orchestration with respect to cost under Quality of Service (QoS) requirements.

- **RP3:** Applicability of well established autonomic-planning methods in cloud computing as applied to stateful orchestration.

Upon evaluating the above, **RP1** has the potential for greatest benefit to science. **RP2, RP3** both spawn novel and important research questions, however **RP2** would require a broad survey and subsequent preliminary evaluations of each candidate platform. The implications of the broad scope in **RP2** make it difficult to investigate in sufficient detail under the time constraints and scope of a Master's Thesis. **RP3** poses a nuanced problem, drawing on of past literature in autonomic-computing [16], better approached by a researcher with past publications in autonomic-computing.

The following research questions **(RQ)** are proposed in order to address **RP1**

- **RQ1:** What is the degree of elasticity provided by Kubernetes for stateful applications solely through its default auto-scaler?
- **RQ2:** What is the impact of augmenting Kubernetes with a state-aware auto-scaler on the elasticity of stateful applications?
- **RQ3:** What are the impacts on application QoS under elastic demand in: Kubernetes, the augmented Kuberenetes proposed in RQ2, and a non-orchestrated application manager.
- **RQ4:** How does the degree of elasticity achievable with an augmented Kubernetes solution for stateful applications compare to non-orchestrated solution?

These research questions aim to address a current gap in stateful orchestration literature. Namely, the ability to scale up resources in order to meet an increasing demand; or scale down in order to avoid an over-allocation of resources under decreasing application demand. Further, the research questions require the incorporation of more than the existing standard of 4 VMs in any experiment environment; remedying the unrealistic level of distribution and scale in the state of the art previously discussed [9].

As noted by Herbst et al [17], elasticity has been defined several different ways under various contexts within cloud computing. To further aid this confusion, cloud providers adopted this term in marketing and product naming (ElasticSearch, ElasticBeanstalk, ElasticKubernetesService) [18]. This paper establishes the following definition of elasticity with regards to orchestration [17]:

> Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.

IV. Method

This section synthesizes relevant research methodology, discussing the trade-offs that exist with respect to answering RQ1-4. Accordingly, a suitable experiment environment, application, workload, measurement and interpretation method is described and justified.

A. Experiment Environment

As recommended by Papadopoulos' review of methodological standards in cloud computing [19], this paper proposes a dual cloud platform in order to control for variations in performance due to the shared resource model offered by modern cloud providers.

The proposed cloud platforms are:

1) NecTar Research Cloud, used previously in orchestration research [20] [14].
2) Amazon Web Services (AWS) EC2, among the most popular servers for orchestration [2].

TABLE I
Specification of VMs

	NeCTar	EC2 (m4.xlarge)
Operating System	CentOS 8.2	CentOS 8.2
vCPU	4 Cores	4 Cores
Memory	16GB	16GB
Kubernetes	1.19	1.19

Table I provides the specifications of VMs that will be used in the experiment environment. *Table II* presents three environment configurations, each comprised of the available number of VMs, starting container deployment size and peak containers that can be supported. These configurations follow similar studies on container elasticity for stateless orchestration [21] [22].

TABLE II
Experiment Environment Configurations

	Pilot	Medium	Large
VMs	8	64	128
Start Containers	1	8	16
Peak Containers	12	96	172

B. Experiment Workload

Two applications have been used previously in experiments for stateful orchestration research. A logging application [5] [8] and a streaming application [13]. In the former, there lacked a real-time requirement that imposed state being stored in the containers volatile memory. In the latter, there is an adequate argument for keeping fine grained session state in memory. In pursuance of a suitable application for experiment testing, the following criteria was used to evaluate possible candidates.

- **REQ1:** The application must be required to retain some degree of state in volatile memory, outside of a database or file system. To be deemed stateful.

- **REQ2:** The application must be publicly available, to reduce friction in reproducing results. This is in line with strict requirements for publishing at the *ACM Conference on Performance Engineering Conference* [19].
- **REQ3:** The application should support horizontal scaling in order to avoid bottlenecks that prevent testing the underlying performance of each platform.
- **REQ4:** The application must have existing literature that models resource consumption w.r.t application demand.

TABLE III
EXPERIMENT APPLICATION CRITERIA

	REQ1	REQ2	REQ3	REQ4
Caiopo Logger	-	✓	✓	-
VLC Streaming	✓	✓	✓	-
Minecraft Server (Game)	✓	✓	✓	✓
Second Life (Game)	✓	✓	-	✓

To that end, an array of applications including streaming were considered, including: streaming, logging and game servers as shown in *Table III*. Ultimately, only Minecraft fulfilled all necessary criteria; having the added bonus of a large player base and active developer community. Minecraft's dedicated server system is relatively advanced and allows for meshing hundreds to even thousands of servers together using open source software [23]. Furthermore, Minecraft has a strong history of academic interest in computer science - providing ample reference [24] [25] [26].

In order to evaluate the system under test, the application must be put under stress. There are several models for generating realistic user load for Minecraft Servers [27] [28] [23]. Yardstick [23] is the most rigorous model, providing more parameters than [27] [28] and extensive validation; seeing adoption in related Minecraft cloud computing research [29]. In addition, Yardstick identifies player actions per minute (APM) and concurrent player connections as indicators of game server resource consumption [30].

A similar process was conducted in selecting the non-orchestrated platform for stateful application management. Zookeeper was most appropriate, as the predecessor to orchestration and being employed extensively in related cloud research [31] [32].

C. Description

Below details the experiment process for Kubernetes (**RQ1,3**), augmented Kubernetes (**RQ2,3**) and Zookeeper (**RQ2,4**).

1) Start the stateful application manager (Zookeeper, Kubernetes) and application, following the configuration in *Table II*.
2) Generate user load from [23], according to *Table IV*.
3) Repeat this process for each stateful application manager, sampling the system under test as outlined in *D. Measurement*

Each configuration experiment shall last 9 hours, where the workload generated will move between Baseline, Mid and Peak levels as shown in *Table IV*. Movement between each level shall be according to a noisy model proposed by Herbst et al [17]. This duration allows for the examination of short to longer term impacts on elasticity and QoS each platform exhibits as recommended in [19]. Additionally, each experiment will be run three separate times for each cloud provider, at each configuration level (pilot, medium, large) and at varying times. This experiment setup is common among experimental cloud computing research [19], to control for the impact of testing on shared computing infrastructure (NeCTar, AWS).

TABLE IV
WORKLOAD GENERATION

	Pilot	Medium	Large
Baseline Players	0	0	0
Baseline APM	0	0	0
Mid Players	75	500	1000
Mid APM	60	60	60
Peak Players	150	1000	2000
Peak APM	120	120	120

D. Measurement

This plan proposes sampling the infrastructure metrics (CPU) of the resources listed in *Table II*. Additionally, measuring request latency and the game server ticks-per-second (TPS) for QoS in **RQ2**. In both cases, measurements can be taken using the reference model Yardstick [23]. The sampling frequency for CPU in Table V is 10Hz. The sampling frequency for the QoS measurements follows from Yardstick: request latency is *per request* whilst TPS (Hz) is measured *once per second*.

E. Metrics

This plan adopts the metrics established in [17] for determining elasticity. This shall be used in answering **(RQ1,2,4)**.

Let the matching function, which defines the number of resources r, that is needed to satisfy the system's performance requirements at a given workload intensity w be:

$$m(w) = r$$

Further, let the workload intensity w_t be equal to the number of user requests present in the system at the same time (concurrently).

$$w_t = \sum request_t$$

Lastly, given the workload intensity w_t and currently allocated resources r_t at some time t, let the following define three states: *underprovisioned*, *overprovisioned* and *optimal*.

$$State(w_t, r_t) = \begin{cases} underprovisoned & : m(w_t) > r_t \\ optimal & : m(w_t) = r_t \\ overprovisioned & : m(w_t) < r_t \end{cases}$$

Given the definitions above, this paper considers the following metrics as indicators of elasticity:

- $\overline{A}$ The mean time to switch from an underprovisioned state to and optimal or overprovisioned state, corresponding to the average speed of scaling up.
- $\sum A$ The accumulated time in underprovisoned state.
- $\overline{U}$ The mean amount of underprovisioned resources during an underprovisioned period.
- $\sum U$ The accumulated amount of underprovisioned resources.
- $\overline{B}$, $\sum B$, $\overline{O}$ and $\sum O$ are defined likewise for overprovisoned states

Let the *average precision* of scaling up P_u be $P_u = \frac{\sum U}{T}$ where T is the total duration of the evaluation period [17]. Similarly, define $P_d = \frac{\sum O}{T}$ as the average precision of scaling down.

Elasticity can now be defined as $E_u = \frac{1}{\overline{A} \cdot \overline{U}}$ for scaling up and $E_d = \frac{1}{\overline{B} \cdot \overline{O}}$ for scaling down [17]. This can be summarized as the elasticity of a *system under test* (SUT) s, which can be captured in a matrix M_s, where each vector represents an elasticity dimension d_i.

The QoS metrics that will be used to answer **RQ3** will be the mean and upper $95th$/$99th$ percentile for both request latency and TPS. The summary of metrics discussed are shown in Table V.

TABLE V
MEASUREMENTS, METRICS AND RQS

	Derived Metrics From Measurements	RQ
CPU	$\overline{A_{cpu}}, \overline{B_{cpu}}, E_{cpu}, P_{CPU}$	*1, 2, 4*
TPS	$\overline{TPS}, TPS_{99}, TPS_{95}$	*3*
Requests	$R_{latency}, RPS$	*1, 2, 3, 4*

F. Interpretation

RQ1, RQ2, RQ4

By examining the elasticity of a system under test (SUT) $s = \{E, \overline{A}, \overline{B}, P\}$, we can consider an overview of each application manager's elasticity. The baseline method provided by Kubernetes alone in **RQ1** can then be evaluated with respect to **RQ2** and **RQ3**; as there are currently no similar experiment results in publication for stateful elasticity. It is important that when evaluating **RQ1,2,4** that metrics are considered relative in terms of performance [17]. Additionally, considering the upper tail $(A_{95}, A_{99}, B_{95}, B_{99})$, $95th$ and $99th$ percentile for each respective metric. As tail performance remains an important part of service level indicators (SLIs), widely adopted in industry [19].

RQ3

In determining the application impacts of elastic demand upon QoS, it was proposed above to consider TPS and latency per request as the representative metrics. It would be appropriate in analyzing the results to consider for each application manager independently: the relative level of QoS achieved under *Baseline Players* and *Baseline APM* as opposed to the Peak workload for each (shown in *Table IV*).

The analysis method discussed is consistent with similar studies [8] [33] and is structured around an established model [23].

V. INTENDED CONTRIBUTION

This paper has identified three open research problems in stateful orchestration, determining that an investigation of elasticity provides scope for the greatest contribution to the state of the art. The contributions of research planned in this paper will advance the state of the art by: (1) Conducting the first investigation into stateful orchestration without static resource constraints. (2) Provide a future model by which stateful orchestration can be examined under elasticity requirements. (3) Lastly, determine the degree of elasticity Kubernetes provides to stateful applications by default and if a state-aware auto-scaler can improve upon it.

REFERENCES

1 Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A., "Performance evaluation of container-based virtualization for high performance computing environments," in *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013*, 2013.

2 Casalicchio, E., "Container Orchestration: A Survey," in *Systems Modelling: Methodologies and Tools*. Springer, Cham, 2019, pp. 221–235.

3 Casalicchio, E. and Perciballi, V., "Measuring Docker performance: What a mess!!!" in *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, 2017.

4 Saha, P., Uminski, P., Beltre, A., and Govindaraju, M., "Evaluation of docker containers for scientific workloads in the cloud," in *ACM International Conference Proceeding Series*, 2018.

5 Netto, H. V., Lung, L. C., Correia, M., Luiz, A. F., and Sá de Souza, L. M., "State machine replication in containers managed by Kubernetes," *Journal of Systems Architecture*, 2017.

6 Kleppmann, M., *Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Inc., 2017.

7 Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J., "Large-scale cluster management at Google with Borg," in *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*, 2015.

8 Netto, H. V., Luiz, A. F., Correia, M., De Oliveira Rech, L., and Oliveira, C. P., "Koordinator: A Service Approach for Replicating Docker Containers in Kubernetes," in *Proceedings - IEEE Symposium on Computers and Communications*, 2018.

9 Mcclernon, A., "The State of The Art in Stateful Orchestration," in *COMP90044 Literature Review*, 2020.

10 Tosatto, A., Ruiu, P., and Attanasio, A., "Container-Based Orchestration in Cloud: State of the Art and Challenges," in *Proceedings - 2015 9th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2015*, 2015.

11 Casalicchio, E. and Iannucci, S., "The state-of-the-art in container technologies: Application, orchestration and security," in *Concurrency Computation*, 2020.

12 Rodriguez, M. A. and Buyya, R., "Container-based cluster orchestration systems: A taxonomy and future directions," *Software: Practice and Experience*, vol. 49, no. 5, pp. 698–719, 5 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2660

13 Abdollahi Vayghan, L., Saied, M. A., Toeroe, M., and Khendek, F., "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes," in *Proceedings - 19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019*, 2019.

14 Rodriguez, M. and Buyya, R., "Container Orchestration With Cost-Efficient Autoscaling in Cloud Computing Environments," 12 2020, pp. 190–213. [Online]. Available: http://arxiv.org/abs/1812.00300

15 Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. C., "Containers and Virtual Machines at Scale," *Proceedings of the 17th International Middleware Conference on - Middleware '16*, 2016.

16 Casalicchio, E., "Autonomic orchestration of containers: Problem definition and research challenges," in *ValueTools 2016 - 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2017.
17 Herbst, N. R., Kounev, S., and Reussner, R., "Elasticity in Cloud Computing : What It Is , and What It Is Not," *Presented as part of the 10th International Conference on Autonomic Computing*, 2013.
18 Amazon Web Services, "Operational Excellence Pillar. AWS Well-Architected Framework," *AWS Well-Architected Framework*, 2020.
19 Papadopoulos, A. V., Versluis, L., Bauer, A., Herbst, N., Von Kistowski, J., Ali-eldin, A., Abad, C., Amaral, J. N., Tuma, P., and Iosup, A., "Methodological Principles for Reproducible Performance Evaluation in Cloud Computing," *IEEE Transactions on Software Engineering*, 2019.
20 Kozhirbayev, Z. and Sinnott, R. O., "A performance comparison of container-based technologies for the Cloud," *Future Generation Computer Systems*, 2017.
21 Warke, A., Mohamed, M., Engel, R., Ludwig, H., Sawdon, W., and Liu, L., "Storage service orchestration with container elasticity," in *Proceedings - 4th IEEE International Conference on Collaboration and Internet Computing, CIC 2018*, 2018.
22 Luong, D. H., Thieu, H. T., Outtagarts, A., and Ghamri-Doudane, Y., "Predictive Autoscaling Orchestration for Cloud-native Telecom Microservices," in *IEEE 5G World Forum, 5GWF 2018 - Conference Proceedings*, 2018.
23 Van Der Sar, J., Donkervliet, J., and Iosup, A., "Yardstick: A benchmark for minecraft-like services," in *ICPE 2019 - Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019.
24 Engelbrecht, H. A. and Schiele, G., "Koekepan: Minecraft as a research platform," in *Annual Workshop on Network and Systems Support for Games*, 2013.
25 Engelbrecht, H., "Transforming Minecraft into a research platform," in *2014 IEEE 11th Consumer Communications and Networking Conference, CCNC 2014*, 2014.
26 Taylor, D. C. J., Mwiki, H., Dehghantanha, A., Akibini, A., Choo, K. K. R., Hammoudeh, M., and Parizi, R., "Forensic investigation of cross platform massively multiplayer online games: Minecraft as a case study," *Science and Justice*, 2019.
27 Alstad, T., Riley Dunkin, J., Detlor, S., French, B., Caswell, H., Ouimet, Z., Khmelevsky, Y., and Hains, G., "Game network traffic simulation by a custom bot," in *9th Annual IEEE International Systems Conference, SysCon 2015 - Proceedings*, 2015.
28 Cocar, M., Harris, R., and Khmelevsky, Y., "Utilizing Minecraft bots to optimize game server performance and deployment," in *Canadian Conference on Electrical and Computer Engineering*, 2017.
29 Donkervliet, J., Trivedi, A., and Iosup, A., "Towards supporting millions of users in modifiable virtual environments by redesigning minecraft-like games as serverless systems," in *HotCloud 2020 - 12th USENIX Workshop on Hot Topics in Cloud Computing, co-located with USENIX ATC 2020*, 2020.
30 Truyen, E., Van Landuyt, D., Reniers, V., Rafique, A., Lagaisse, B., and Joosen, W., "Towards a container-based architecture for multi-tenant SaaS applications," in *ARM 2016 - 15th Workshop on Adaptive and Reflective Middleware, colocated with ACM/IFIP/USENIX Middleware 2016*, 2016.
31 Hunt, P., Konar, M., Junqueira, F. P., and Reed, B., "ZooKeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX Annual Technical Conference, USENIX ATC 2010*, 2019.
32 Oliver, J., *ZooKeeper: Distributed Process Coordination*, 2013.
33 Nardelli, M., Hochreiner, C., and Schulte, S., "Elastic provisioning of virtual machines for container deployment," in *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, 2017.