Microservices Hosting in The Cloud:
An Infrastructure Comparison of Virtual Machines, Unikernels, and Application Containers

Devin Durham[1], Wes Lloyd[2]
University of Washington
Institute of Technology,
Tacoma, Washington USA
[1]d1durham, [2]wlloyd@uw.edu

***Abstract*—Cloud computing today offers on-demand access to scalable, reliable compute resources with pay-as-you-go pricing. Cloud resources are provided with a variety of levels of resource abstraction including infrastructure (IaaS), platform (PaaS), and service (SaaS), each trading off control vs. simplicity for the end user. Web and microservices hosting at the infrastructure level leverages virtual machines (VMs), or application containers such as Docker, to support service deployment and scalability. In this research, we analyze web and microservice performance and scalability implications for hosting services using traditional Linux-backed VMs, Docker application containers, and Unikernel VMs leveraging the single process OSv operating system. Our objective is to compare the performance and elasticity of these alternate technologies specifically for hosting lightweight REST enabled software services.**

Keywords Resource Management and Performance; Containerization; IaaS; Unikernel; Virtualization; microservices; web services;

I. INTRODUCTION

Cloud applications built using the microservices paradigm require functionality to be disaggregated into discrete small services each hosted by a separate process and communicating using lightweight mechanisms (e.g. HTTP REST). Individual microservices have the advantage of being independently deployable and scalable, providing fine grained elasticity similar to component isolation [1][2]. Given their small size, microservices hosting using traditional cloud-based virtual machines (VMs) requires aggregating the hosting of *many* microservices together onto a single appliance. Dynamic management of monolithic VMs for microservice hosting becomes a bottleneck, as the ability for VM allocations with fixed sizes and extensive launch latencies to rapidly adapt to user demand is limited. Container orchestration frameworks such as Kubernetes and Amazon ECS support the creation of virtual container clusters using public cloud VMs enabling practitioners to deploy and manage microservices deployments [3][4]. Container orchestration frameworks provide infrastructure management capabilities for services and applications akin to management capabilities for VMs offered by infrastructure-as-a-service (IaaS) cloud APIs. These frameworks enable rapid container deployment, as well as repurposing and sharing of hosting resources to simultaneously host many distinct applications.

Virtual machine hypervisors such as XEN already provide low-level operating system features including resource isolation and hardware abstraction (e.g. CPU scheduling). Unikernel operating systems optimized for the cloud including Mirage and OSv can boot directly on a hypervisor, not a physical host, and are designed to leverage hypervisor isolation capabilities considered superior to containers [5][6][7]. Unikernels reinvent the operating system to provide only required libraries for hosting application code. Consequently, they feature much smaller code sizes and reduced overhead in comparison to legacy operating systems (e.g. Linux and Windows). OSv, a new OS, has been designed specifically for the cloud and is able to boot in less than a second, while providing support for unmodified Linux executables and application stacks written in C/C++, Java, Ruby, and Node.js [8]. OSv is a single process operating system, where all operating system services and the hosted application run as separate threads using a shared memory model. This enables CPU context switching (implemented by the hypervisor) to avoid remapping memory as in multi-process OSes such as Linux, theoretically reducing overhead.

In this research, we investigate the performance and elasticity of unikernel VMs, traditional Linux VMs, and application containers for hosting web and microservices. Hosting small, lightweight microservices code demands agile cloud infrastructure to ensure responsiveness to spikes in service demand. Deployments using containers requires organizations to adopt container orchestration frameworks such as Kubernetes to provide infrastructure management and elasticity. Adopting unikernels for hosting avoids an additional abstraction layer between the physical hardware and the application. **While adoption of containers such as Docker has become very popular due to widespread support by many cloud providers, we argue that unikernels, which feature better isolation and potentially less overhead offers a simpler alternative.** Our ultimate goal is to compare and contrast performance and elasticity afforded by alternative cloud hosting resources for lightweight web and microservices.

We investigate the following research questions:

RQ-1: **Scalability**: What is the launch latency (ms) to create new Linux VMs, Unikernel VMs, and containers from the time hosting resources are requested until the infrastructure responds to service requests?

RQ-2: **Performance**: Which hosting alternative (Linux VMs, Unikernel VMs, or containers) provides the best application service throughput (requests/second) and average service execution time for 1:1 request-to-host load balancing? for 1:many request-to-host load balancing?

RQ-3: **Architecture**: What are the performance implications of service architecture (microservices vs. traditional REST services) when hosting using Linux VMs, Unikernels, or containers? What performance differences result for service throughput (requests/second), and average service execution time (ms)?

RQ-4: **Network Performance**: For Linux VMs, Unikernels, and container clients, how does network performance vary for communication between clients? Specifically, how does: (1) network latency (response time and round-trip time), (2) jitter (performance variation), (3) packet loss, and (4) bandwidth utilization *vary* for communication between clients? Do networking performance results statistically correlate with service performance?

To evaluate the potential for unikernels, containers, or VMs to provide better resource elasticity for microservices hosting, we investigate launch latency in **RQ-1**. We explore performance overhead of unikernels, containers, and VMs for single threaded and multi-threaded request processing in **RQ-2**. In **RQ-3**, we explore performance implications of service architecture (microservices vs. traditional web services) for hosting with unikernels, containers, and VMs. Finally, we explore correlations between service and networking performance for unikernels, containers, and VMs by leveraging iPerf [9] in support of **RQ-4**.

II. Experimental Approach

To support this investigation, we leverage 4th generation compute optimized (c4) and general purpose (m4) Amazon EC2 virtual machines running Ubuntu 16.04 and OSv version 0.24. We run Docker containers on EC2 instances with Docker 17.12.0-ce [10]. We leveraged an 8-core c4.2xlarge EC2 instance with "high" (e.g. 1 Gigabit/sec) networking throughput as a REST client to perform single and multithreaded tests.

We implemented Fibonacci as a Spring Boot microservice and Jersey RESTful service in Java. Fibonacci provides an experimental service which is both compute and memory intensive. We profile computation of Fibonacci sequences up to 100,000. Additionally, we implemented a purely compute bound random mathematical calculations service as in [11] using Java Spring Boot, and Jersey REST. We profiled the service at a variety of stress levels from zero math operations, to simply measure the round-trip request latency, up to 10,000,000 random math operations per request using operands stored in large arrays for each service request.

Experiment 1 – Launch Latency Testing

In experiment #1, we investigate launch latency of service hosting resources: containers, unikernel VMs, and full Ubuntu VMs. We do not measure boot time, but rather the time from when resources are requested until they respond to service request(s). We capture the boot time plus application initialization time to investigate **RQ-1**.

Experiment 2 – Service Performance Testing

In experiment #2, we investigate web and microservice performance hosted by containers, unikernel VMs, and full Ubuntu VMs. We profile single-threaded request processing where individual client requests are processed using dedicated resources, as well as multithreaded processing, where resources simultaneously process multiple user requests in parallel to investigate **RQ-2**. Additionally, we compare Java REST microservices implemented using Spring boot [12] vs. traditional RESTful web services implemented using the Jersey framework [13] and hosted by Apache Tomcat [14] in support of **RQ-3**.

Experiment 3 – Network Performance Testing

In experiment #3, we investigate networking performance of containers, unikernel VMs, and full Ubuntu VMs in support of **RQ-4**. We compare network latency, jitter, packet loss, and bandwidth utilization for communicating between two clients, and explore correlations with service performance.

References

[1] Lloyd W, Pallickara S, David O, Lyon J, Arabi M, Rojas K. Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling. Future Generation Computer Systems. July 2013;29(5):1254-64.

[2] Lloyd W, Pallickara S, David O, Lyon J, Arabi M, Rojas K. Service isolation vs. consolidation: Implications for IaaS cloud application deployment. In Proc. of the Int. Conf. on Cloud Engineering (IC2E), March 2013, pp. 21-30.

[3] Kubernetes, Production-Grade Container Orchestration, https://kubernetes.io/

[4] Amazon ECS – run containerized applications in production, https://aws.amazon.com/ecs/

[5] Madhavapeddy A, Scott DJ. Unikernels: the rise of the virtual library operating system. Communications of the ACM. 2014 Jan 1;57(1):61-9.

[6] Madhavapeddy A, Mortier R, Rotsos C, Scott D, Singh B, Gazagnaire T, Smith S, Hand S, Crowcroft J. Unikernels: Library operating systems for the cloud. In Proc. Of ACM SIGPLAN Notices, Mar 2013, Vol. 48, No. 4, pp. 461-472.

[7] Kivity A, Costa DL, Enberg P. OS v—Optimizing the Operating System for Virtual Machines. In Proc. of USENIX ATC'14: 2014 USENIX Annual Technical Conference, June 2014, p. 61.

[8] iPerf – The TCP, UDP, and SCTP network bandwidth measurement tool, https://iperf.fr/

[9] Docker Community Edition, https://docs.docker.com/engine/installation/

[10] Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S. Serverless Computing: An Investigation of Factors Influencing Microservice Performance. To appear in Proc. 2018 IEEE Int. Conf. on Cloud Engineering (IC2E), Apr 2018, 11 pp.

[11] Spring Boot, https://projects.spring.io/spring-boot/

[12] Jersey, https://jersey.github.io/

[13] Apache Tomcat ® - Welcome, http://tomcat.apache.org/

[14] Lloyd W, Pallickara S, David O, Arabi M, Rojas K. Mitigating Resource Contention and Heterogeneity in Public Clouds for Scientific Modeling Services. In Proc. 2017 IEEE Int. Conf. on Cloud Engineering (IC2E), Apr 2017, pp. 159-166.