Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

During the HotCloud '15 workshop, I was invited to join a discussion group. We were supposed to decide which was better, VMs or containers. An hour later, we really hadn't answered the question posed to us, but we did have some answers.

Virtual machine technology has been around since IBM developed VMs as a method for sharing mainframes. That might sound funny, but mainframes in businesses were used to run batch jobs or long-running transaction processing applications, like managing accounts for a bank. Sharing the computer, even if that computer wasn't always busy, was a side issue.

With VMs, customers could run other applications when demand by the main application was low. You could even run IBM's version of UNIX, AIX, and later, Linux, providing the illusion of a time-sharing system that we are most familiar with.

In the early noughts, VM technology really took off. Xen and VMware became popular ways of sharing underused systems. And like the original IBM VMs, you could run applications requiring different operating systems all on the same server.

Containers

Container technology was taking off at about the same time, and the biggest users of containers were companies with clusters all running the same OS. For those uses, running one operating system inside of another, however stripped down, was a waste of processing power. Also, why run an operating system per VM when you could just have a single operating system supporting all of your containers?

For many years, VMs were the prominent technology, with containers being used at Google or hosting companies. And there are both advantages and disadvantages to using containers. While containers were great at improving efficiency and making management easier because there was just one set of system software to manage, containers were not as good as VMs for security. That extra level of separation, eventually supported by special CPU instructions, really did make the combination of a hypervisor and VMs more secure than a system running containers using a single Linux kernel.

And those were, roughly, the results of our discussion group: that VMs were best for running legacy applications and for security, while containers were a packaging framework that is more efficient and easier to manage than VMs. But we did discuss one other technology, one not included in our original remit: unikernels.

The Middle Path

We had two people from Cambridge in our group, and they suggested that we should also consider unikernels, like MirageOS. So let's talk about unikernels.

Where VMs are entire operating systems that happen to be running applications, and containers are namespaces [1] used to isolate just one application, unikernels are more like applications that run directly on top of a hypervisor [2]. Unikernels can be even more efficient than containers because instead of sharing an operating system, like containers, unikernels

don't have an operating system. Unikernels by design run a single-threaded application and rely on the hypervisor for access to hardware resources.

You might just be thinking that being constrained to a single thread can be a serious issue, and you'd be right—for some applications. But for many others, a single, stripped down to bare essentials, dedicated thread is just right. Unikernels jettison almost all of the support found in traditional operating systems in exchange for single-minded efficiency.

The unikernel focus on doing one thing has security benefits as well. While VMs and containers include a whole array of applications, such as shells, administrative commands, and compilers, unikernels have nothing except the application and the support library needed by that application for communication with the hypervisor. Unikernels are a manifestation of least privilege and minimal configuration hard to achieve with VMs or containers.

And it turns out that because the security model of containers is weaker than that of VMs or unikernels, most people who use containers run containers belonging to the same security domain within a VM. I was surprised to learn this because it means that most of the gain in performance over VMs gets tossed for stronger security. That containers are still used at all speaks to how much easier it is to manage applications within containers as compared to entire VMs. Someone who works for a company that runs giant clusters mentioned that they even run VMs from within containers, meaning that they start with a VM that runs containers that run VMs. Sounds silly, but the point is that containers are easier to manage than VMs, and that is actually very important to people who run huge clusters.

You might be wondering why we don't see unikernels everywhere, and you are right to wonder. Unikernels appear to be the best choice when it comes to efficiency and security for many applications. But there are some things that the unikernel people aren't going to tell you.

MirageOS, with its Cambridge and Xen connections, is the best known of unikernels today, but there are others: LING, based on Erlang, and HaLVM, based on Haskell, to name two. MirageOS uses OCaml, a functional programming language. Erlang and Haskell are also functional programming languages. Functional programming languages have real advantages when it comes to security, although OCaml does not require the programmer to write purely functional code. Learning how to write in Haskell, for example, requires serious effort on the part of the programmer: you need to think differently, more like a mathematician, to become a useful functional programmer.

The requirement of needing to be a programmer, familiar with functional languages, is currently a huge impediment to the success of unikernels. Unlike VMs, which provide an environment that appears identical to the one that most people normally work with, and with containers, which focus on packaging, working with a unikernel today means using an application written for a particular unikernel technology. You can certainly do that, but you best be a programmer who can adopt the application of your choice to run in that environment.

Perhaps the easiest unikernel technology to use are rump kernels based on NetBSD, as the environment is POSIX and the language commonly used is C. Antti Kantee, one of the primary creators of rump kernels, has written an article in this issue arguing for the use of unikernels. One of his many points is that much of what operating systems provide us with is support needed by time-sharing systems. Time-sharing was a method designed for sharing mainframes among multiple users; today, most servers run applications that provide services, and their users are other applications, not people. Times have changed, but operating systems have remained the same.

Well, I am exaggerating. Operating systems haven't remained quite the same. They have grown. Enormously. For example, Linux has grown from 123 system calls [3] in version 1 to nearly 400 system calls for the 3.2 kernel. Microsoft Windows Server 2012 has 1144 system calls [4]. Operating systems have become incredibly complex.

While researching how to run legacy code securely within Web browsers, Douceur et al. [5] discovered that they could run some desktop applications with minimal modifications while using just a handful of system calls. Unikernels move us closer to a similarly minimal environment.

The Lineup

We start out this issue with an opinion piece by Antti Kantee. While Kantee certainly has his own axe to grind, he also makes some very good points while being amusing at the same time.

Next we have an article about Grappa (no, not the liquor), a distributed shared memory framework developed by a group at the University of Washington, Nelson et al. The Grappa framework creates an abstraction of a single memory space for programmers seeking to develop software that works like Hadoop, Spark, or GraphLab. Their system also hides the latency of remote memory accesses by taking advantage of the parallelism inherent in processing big data.

We next take a look at a different issue, also caused by non-uniform memory access. Lepers et al. studied how the core interconnects work in server-class AMD processors, and discovered that the bandwidth between cores in AMD chips varies tremendously. They developed and tested software that can determine the best placement for multithreaded applications, and migrate threads to cores with more bandwidth between them.

Both of these articles are based on papers presented at USENIX ATC '15. The next article was related to a FAST '15 paper and presents a novel algorithm for fast inserts, deletes, and updates in B-trees, while providing the same level of read performance. B-epsilon trees trade space used for pivot keys in each node for space used to buffer writes, and the article by Bender et al. explains how the algorithm works, as well as proving it to be faster than B-trees for writes.

Singh et al. present Beam, part of a Microsoft project with a goal of collecting more useful information about certain events from the Internet of Things (IoT). While one type of sensor can provide potentially useful information, having an abstraction for multiple sensors can better answer a query such as "Is someone home?"

Andrea Spadaccini and Kavita Guliani continue the series of articles about the practices of System Resource Engineers (SREs) within Google. They explain how SRE teams handle on-call, one of the many vexing issues facing anyone who supports software services, in a way that has proven to work well and be fair to all participants.

Brendan Burns explains the Kubernetes (pronounced koo-ber-net-tees) project. While Docker has made containers into an easy-to-use packaging system, Kubernetes focuses on managing the services presented by applications running in containers. Kubernetes presents a single IP address for a group of containers, handles load balancing, keeps the configured number of services running, and handles scaling and upgrades.

Andy Seely has more tips for technical managers. In this column, Andy explains how time management is different for managers (compared to sysadmins and other technical staff), and provides advice from his own experience on how to best manage your time.

Dave Beazley's Python column explains some new syntax in Python 3.5. * and ** have been available for use in function arguments, where the function needs to be able to accept a variable number of arguments. Version 3.5 extends how this syntax works, including for specifying keyword-only arguments and conversion of arguments.

David Blank-Edelman explains how you can get Perl to work with WordPress. WordPress currently has a WP-API plugin that might become a standard part of WordPress, and David demonstrates how to get that plugin to work gracefully with the CRUST Web service.

Dave Josephsen wanted to be able to monitor the relative performance of some apps on different laptops. Dave shows how to install and use the Nagios Cross-Platform Agent for Linux and Apple systems.

Dan Geer discusses the denominator of risk: when we attempt to calculate risk, how best to choose the number of systems at risk. When comparing the number of unpatched exploits to the number of potential targets (the denominator), knowing the denominator can make a huge difference.

Robert Ferrell decides to redesign the Internet for better security, working as a non-network non-specialist.

Mark Lamourine has two book reviews this time, on *The Essential Turing* and *Drift into Failure*.

Peter Salus has written another in his series of columns on the history of USENIX, covering the change from having two Annual Tech conferences each year to having many more focused workshops and conferences. Salus also discusses the journal *Computer Systems*.

We conclude this issue with a portion of an interview conducted with Dan Geer in 2000, where he talks about why he became President of the USENIX Board of Directors. We included these statements because Geer explains both where USENIX was at this time (much larger) and his own remarkably insightful projections about the future he imagined 15 years ago.

Speaking of the future, I think we will continue to see both containers and VMs used on the same system. Whether unikernels will become as popular is still up in the air. Containers and VMs provide something familiar, and it is always easier for people to continue dealing with the familiar than to launch into the wilderness of the new. If support for unikernel-based applications continues to grow, these streamlined packages are likely to become just as popular.

Resources

[1] James Bottomley and Pavel Emelyanov, "Containers," *;login:*, vol. 39, no. 5, October 2014: https://www.usenix.org/publications/login/october-2014-vol-39-no-5/containers.

[2] Unikernels: http://wiki.xenproject.org/wiki/Unikernels.

[3] Linux system calls: http://man7.org/linux/man-pages/man2/syscalls.2.html, http://asm.sourceforge.net/syscall.html.

[4] Microsoft, Supported System Calls: https://technet.microsoft.com/en-us/library/Cc754234.aspx.

[5] John R. Douceur, Jeremy Elson, Jon Howell, and Jacob R. Lorch, Microsoft Research, "Leveraging Legacy Code to Deploy Desktop Applications on the Web": http://www.usenix.org/events/osdi08/tech/full_papers/douceur/douceur_html/index.html.

Letters to the Editor

Hello,

I received an issue of *;login:* magazine, "Sysadmin and Distributed Computing" (April 2015) while attending SouthEast LinuxFest (SELF) in June. I was very impressed with your publication and am now thoroughly disgusted with *Wired* magazine.

There was a mention of a Student Programs contact program, and I wanted to ask if you already have a rep on the Virginia Tech campus. If you have a rep, I would like to talk to them; if not, I would be glad to set up a Web site for USENIX info and library, which I can restrict to campus authorization.

I'll also be glad to forward USENIX info to our student Linux Users Group, VTLUG, and the Tech Support and/or Sys Admin campus groups.

Denton Yoder
Computer Systems Engineer
Biological Systems Engineering
Virginia Tech

Rik,

Thank you...USENIX is a great org and *;login:* a great mag. When it arrives, I know there will be an hour coming up shortly where I can put on the headphones, kick back, and read about people and ideas that relax and educate my poor tired computational soul. Good things by good people working for a better Net.

Thanks, and I promise to get on the stick and start submitting. Cyberville here is going 90 mph and just getting warmed up. Look forward to seeing folks out in my neck of the woods for WOOT and then for LISA.

Keep the faith...

Best,
Hal Martin
University of Maryland, Baltimore County