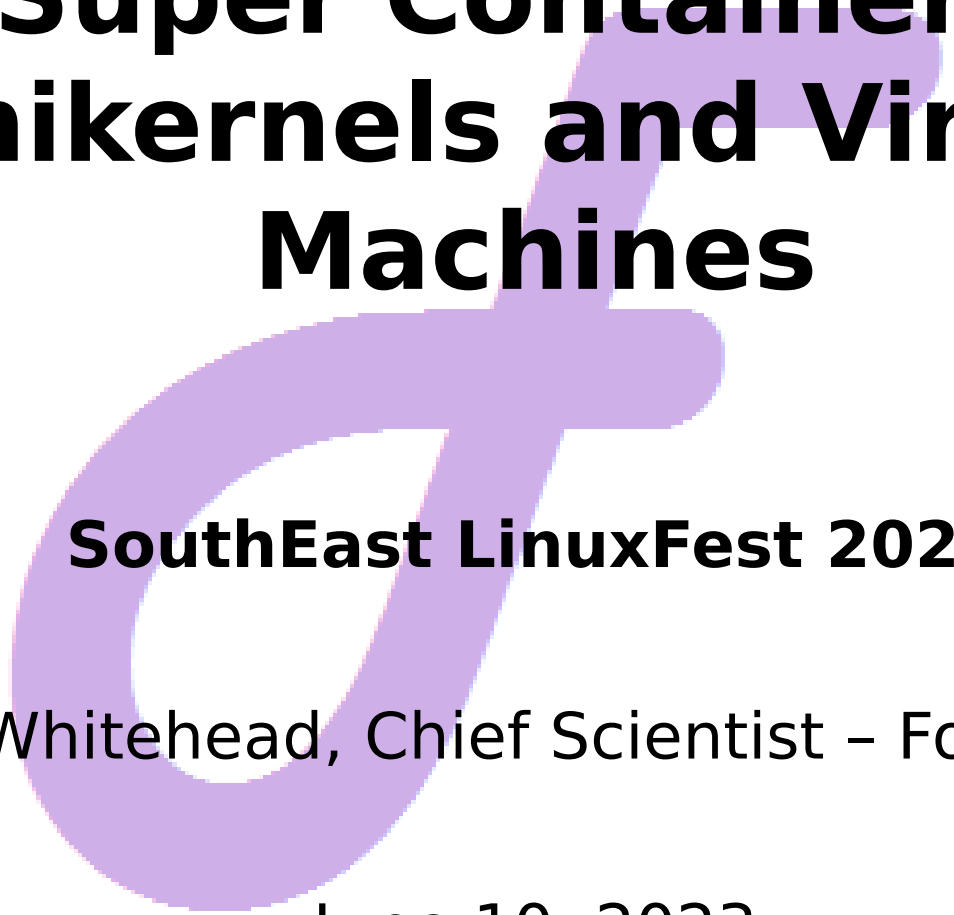
Super Containers: Unikernels and Virtual Machines



SouthEast LinuxFest 2023

Brad Whitehead, Chief Scientist – Formularity

June 10, 2023

“Super Container”

- 1400 Times More Secure Than A Well-configured Docker Container
- Boots 37 Times Faster Than That Docker Container
- Can Run 10 Times More Microservices On The Same Physical Hardware
- Can Be Managed By Kubernetes Or Apache Mesos

What Is This Incredible New Technology?

It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor

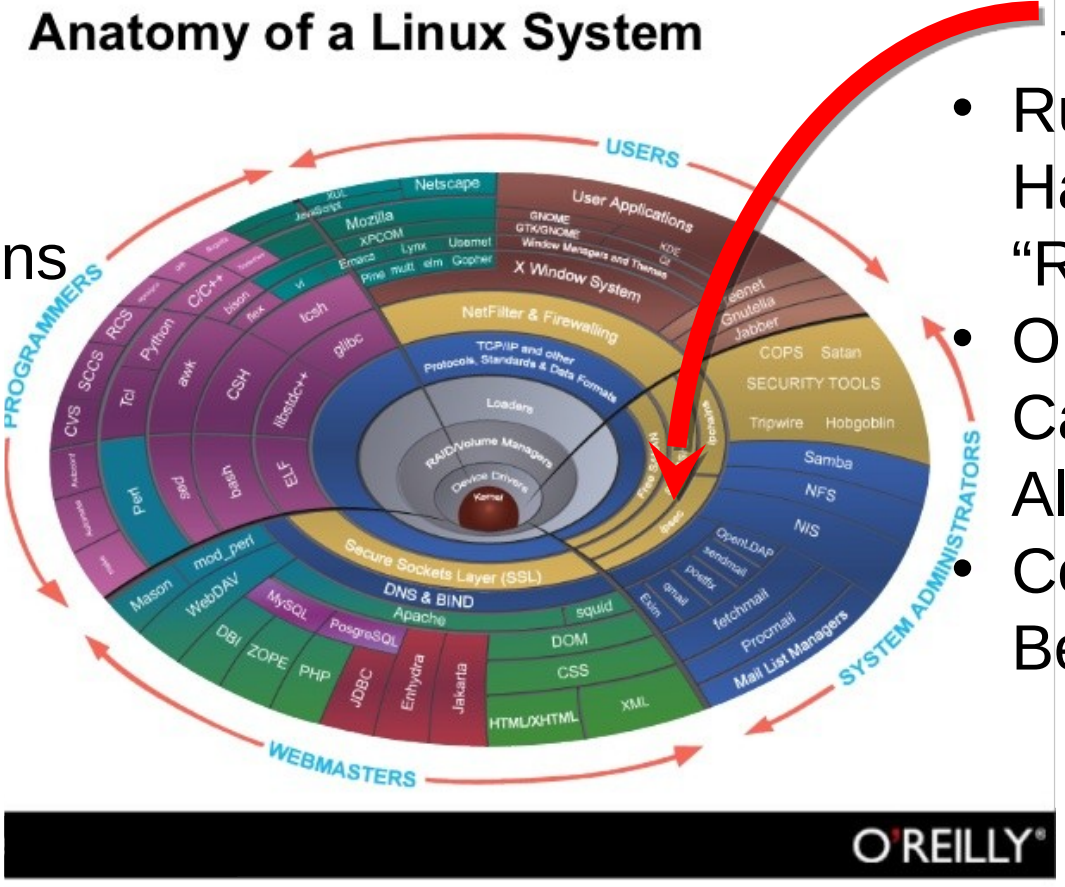
What’s A Unikernel?

- To Answer That Question, We Have To Take A Look At The Structure Of A Modern Operating System
- Doesn’t Matter If It’s Microsoft Windows, Linux, UNIX, ~~Mac OS X~~ ~~OS X~~ macOS, etc.
- All The Mainstream Operating Systems Have The Same Fundamental Anatomy

The Anatomy of An Operating System

"UserLand"

- Where Applications Run
- No Privileges
- Can Not Access Resources
- Can Not "Talk" to Other Programs



"The Kernel"

- Runs in Special Hardware Mode – "Ring 0"
- Only Program That Can Access or Allocate Resources
- Copies Data Between Programs

Growth of Operating Systems

- Linux Kernel Is Now 22 Million Lines Of Code!
- Windows Is Estimated At 50 Million Lines Of Code!
- With An Industry Average Of 15-50 Defects Per 1000 Lines Of Code*:
  - Linux(Just The Kernel) = 330,000 To 1.1 Million Defects
  - Windows = 750,000 To 2.5 Million Defects

*(Steve McConnell, “Code Complete 2”, 2005)

It Gets Worse!

- The “Userland” Support Software Is Often 10 To 20 Times Larger Than The Kernel!
- Red Hat Enterprise Linux (RHEL) Userland Is Approximately 420 Million Lines Of Code
  - Try Not To Think About The 6.3 To 21 Million Defects Running On Your Bank’s Server!

Can It Get Even Worse?!?!

- The Kernel Is Full Of Junk!
- A Large Number Of Device Drivers Are Routinely Compiled Into The Kernel, Regardless Of The Actual Hardware In The Computer
  - There Are Device Drivers For Hardware That No Longer Exists
  - Amazon Ami Images ~~Have~~ Had Drivers For Floppy Disks And Audio Cards
    - In 2015, The Venom Vulnerability (CVE-2015-3456) Used A Flaw In The Floppy Disk Controller (FDC) Driver To Compromise Both Physical And Virtual Machines

Can It Get Even Worse?!?! (Continued)

- Likewise, There Are Thousands Of Storage And Communications Protocols In The Kernel That Will Not Be Used In Your Application
- Linux Recognizes 7 Different Executable Formats, Even Though The Vast Majority Of Applications (Including Yours) Are In ELF Format
- **Each Of These Extra, Unused Chunks Of Code (With Its 15-50 Defects/1000 Sloc) Is A Potential Hack Waiting To Happen!**

What If We Cut Out All The Parts We Don’t Need?

- Code Traces Show That The Average Application Uses Less Than 0.08% Of The Total Code In The Kernel!
- Take The Standard C Library As An Example
  - The C Library Contains Thousands Of Functions, But A Modern Linker Only Includes The Actual Functions (And Code) That An Application Uses
- Could We Do The Same With Our Operating System?

What About Actors (Microservices)?

- Run A Single Application
- As A Single User
- Known Set Of Hardware Drivers
- 1 Or 2 Communications Protocols
- Speed (Startup And Latency)
- Reliability
- Security (From Unauthorized Access - “Hacking”)
- Repeatability (Multiple Identical Servers)

Keeping Only The Parts of the Operating System We Actual Use

- What Does It Buy Us?:
  - Let’s Start With Security:
    - Greatly Reduced Attack Surface (99.92% Reduction)
    - Potentially A Small Enough Subset To Be Mathematically Verifiable
    - We Don’t Need Any Userland Applications (Bye-bye 410 Million Lines Of Potentially Flawed Code!)
    - No Ability To Run Malicious Or Hacking Tools On Our Server Or IoT Device

More Benefits

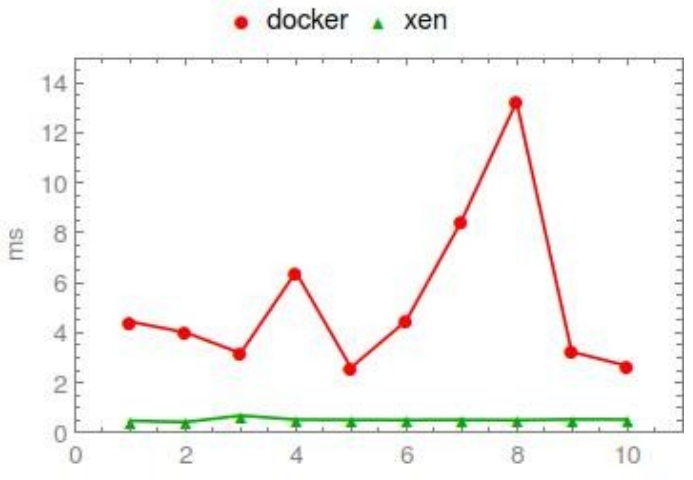
We Can Statically Link Everything (Including The Kernel Functions) And Our Software Becomes Immutable

- No Injection Attacks
- No Re-configuration Attacks
- Vastly Reduced “Return Oriented Programming” (ROP) Vulnerability

Increased Reliability And Improved Security Means Reduced Devops Costs!

Increased Performance

- Smaller, Less Memory Intensive Images Mean More Virtual Machines Per Hardware Server
  - 5 Megabyte Virtual Machines = 10,000 VMs Per Physical Server
  - Smaller Than Most Docker Containers
- 6 Millisecond Boot Times
  - Jitsu – Boot-On-Demand
- 45 Microsecond Throughput Times
  - No Context Switches
  - No Information Copying
  - Single Address Space



How To Include Only The Needed Code?

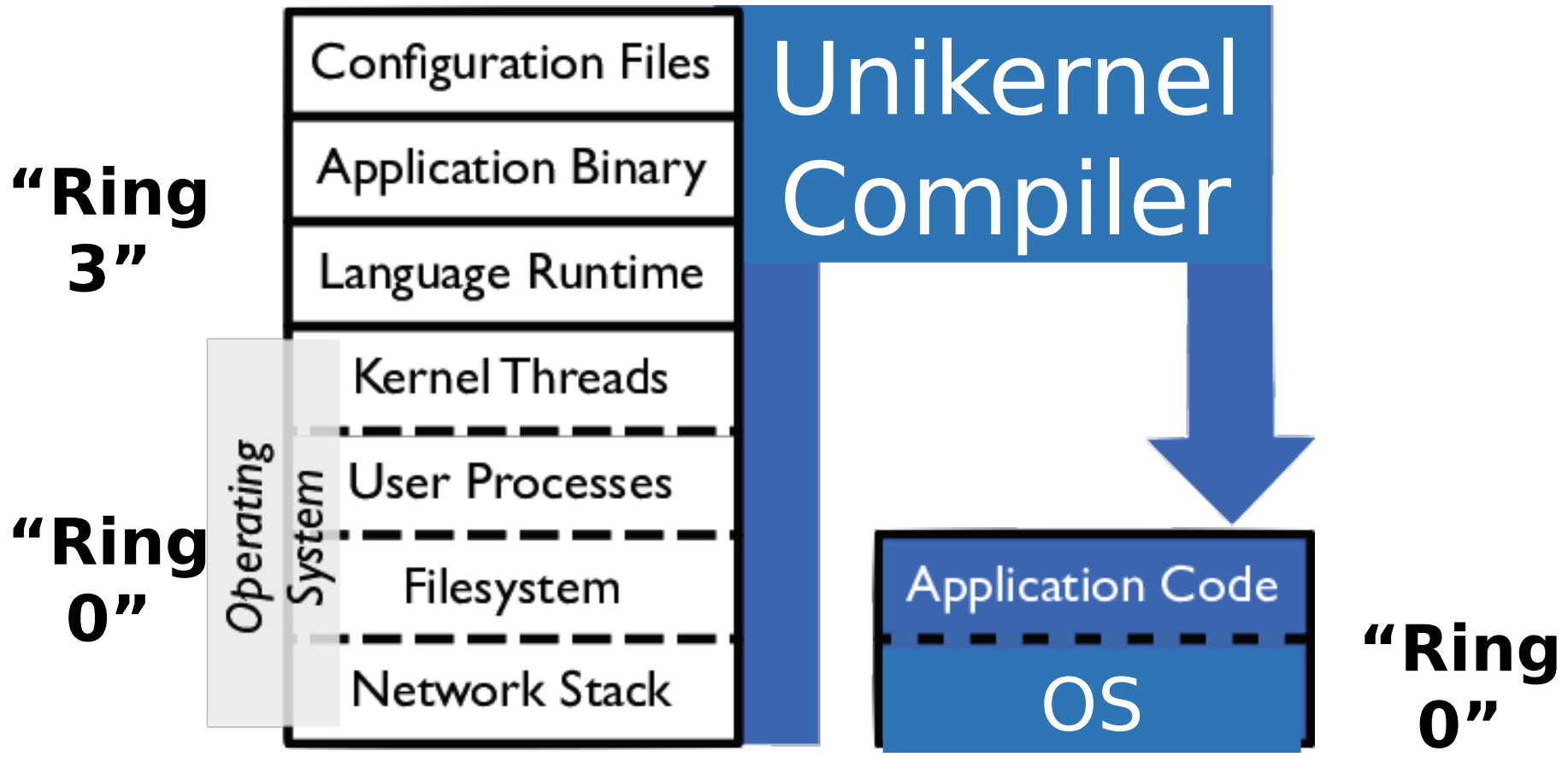
- Again, The C Library Analogy Is The Key
  - The C Library Is Actually A “Middle Ware Layer”
  - It Converts Standard C Function Calls Into Equivalent Kernel System Calls
  - Instead of Handing The Function Call Off As a System Call, What If We Extended the C Library to Include the Appropriate Kernel Code?
    - Instead of the C Library Passing a “Printf()” Call To The Kernel, the Library Can Include the Machine Instructions to Do The Actual I/O

The “Library Operating System”

- Common Operating System Functions, Drivers, And Protocols Are Written As A Library Of Functions
- When You Link These “Library Operating System” Functions To Your Application, You Have A Single Executable That Runs Directly On Hardware Or A Hypervisor…

…You Have A Unikernel!

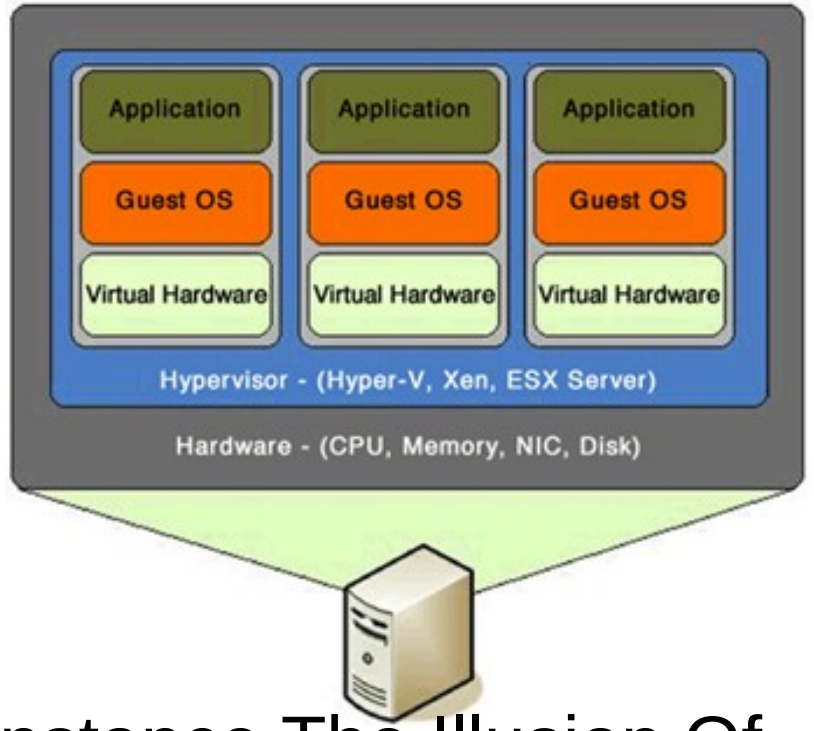
What Does A Unikernel Look Like??



Unikernels Are Only Half The Answer To Small, Fast, Secure Containers

The Other Half Are Lighter Weight Virtual Machines

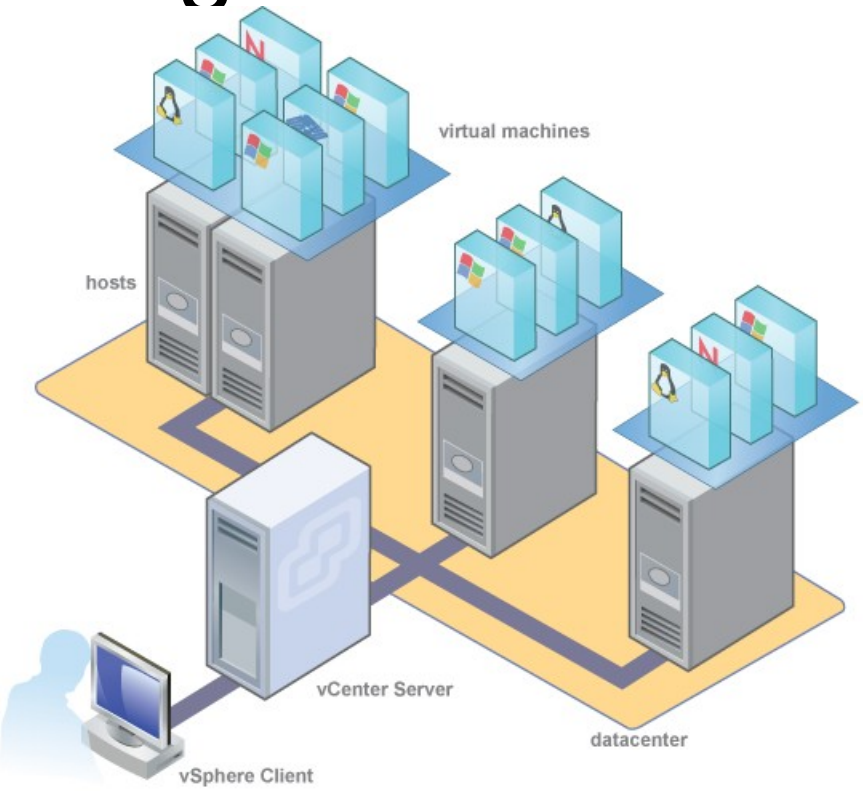
Virtual Machines



- Virtual Machine Monitor (VMM) Or “Hypervisor” Typically Sits Between The Real Hardware And Multiple Operating Systems
- Gives Each Operating System Instance The Illusion Of Running On Its Own Hardware – A “Virtual Machine”
- **Strong** Physical Isolation Between Operating Systems

Cloud Computing

- Virtual Machines Are The “Fuel” Of Cloud Computing
- Multiple “Virtual Machines”, Each With Its Own Operating System
- Each Virtual Machine Isolated And Managed By The Virtual Machine Monitor Or Hypervisor

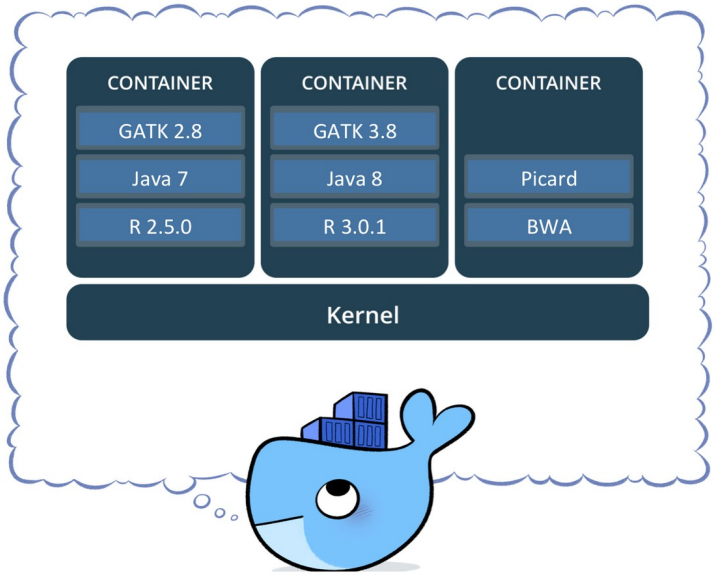


Drawbacks to Current Virtual Machines

- Size - Each VM Requires Its Own Operating System, Userland Software, And A Certain Amount Of Dedicated Memory, Making VMs **BIG**:
  - VMware – Max Of 380 VMs Per Physical Host
  - AWS Xen ~ 10 VMs Per Physical Core
  - AWS Nitro – 6000 VMs On A 36 Core Processor
- Speed - Slow To Startup – Boot Times Measured In Seconds And Minutes

Enter the “Container”

- A Container Is A Package That Bundles Up An Application And All Its Dependent Userland Software (Such As Libraries And Services) Into A Single Image
- A Container Runs Like A Pseudo-Virtual Machine, **Weakly** Isolated From The Host Processes And Other Containers
- All Containers On A Host Use The Host’s Kernel
- While There Are Several Different Container Formats, Docker Is The Most Common



Advantages of Containers

- Neatly Solves The Library Dependency And Versioning Problem (“DLL Hell”)
- Since The Kernel Is Already Running, Containers “Boot” In Milliseconds
- Less Dedicated Memory Is Required For A Container Than A Conventional VM
- “Orchestration” Software Has Been Developed To Deploy And Manage Containers
  - Kubernetes, Apache Mesos, Docker Swarm, et al
  - Google, Netflix

Containers Have Changed The Way We Develop and Deploy Software

- Applications Are Deployed As Complete Images, Ready To Run, Instead Of Being Installed
- Containers Are Replaced, Rather Than Being “Patched”
- Containers Support The Concept Of “Microservices”, Allowing Complex Applications To Be Built From Single-function Services Wired Together Through Orchestration Managers
- Multiple Containers Can Be Started And Stopped In Response To Traffic Loads

Drawbacks of Containers

- Limited Isolation Between Containers – Not A Security Mechanism
- The Container Manager Must Run As Root Or Administrator
- Difficult To Strip Down Userland And Container Images
  - Bloat Consumes Memory And Processing Resources
- Differences In Production And Development Environments

Meanwhile, Virtual Machine Technology Has Not Stood Still

Recent Optimizations To Both The Xen And The Linux “Kernel-based Virtual Machine” (KVM) Hypervisors Have:

- Significantly Reduced The Start-up Time Of A Virtual Machine
- Reduced Performance-Robbing Overhead

This New Generation Of Hypervisors Are Called **“LightVMs”**

Next Generation LightVMs - Speed

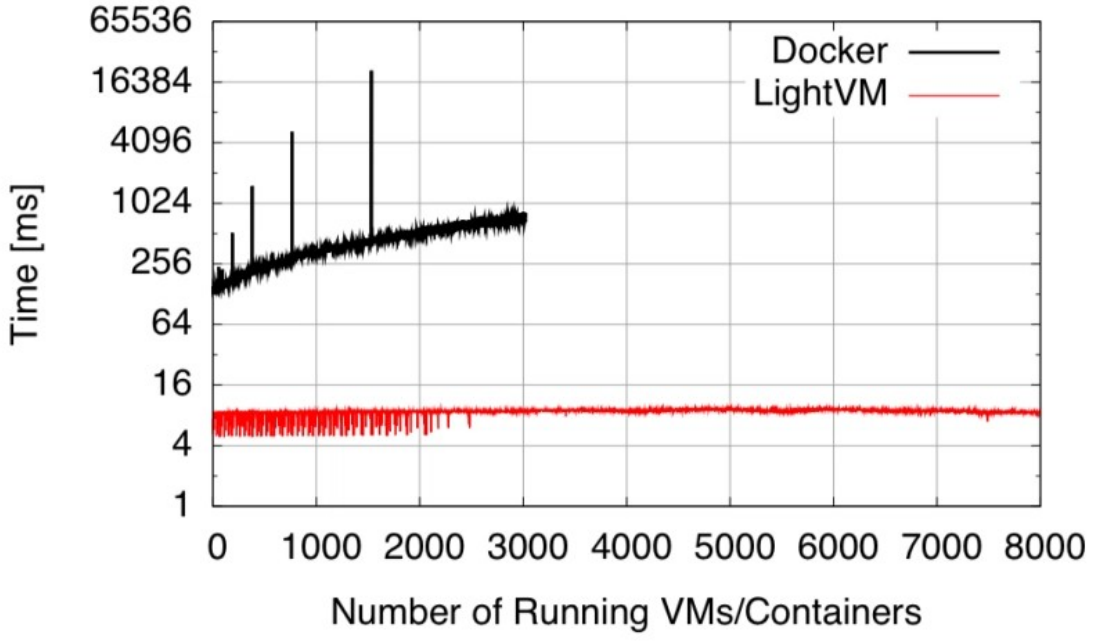
- Combined With Unikernels, These LightVMs Can Launch Microservices In As Little As 4 Milliseconds
- This Is Comparable To The Linux Kernel’s Exec/Fork Times Of Approximately 1 Millisecond And Significantly Faster Than A Docker-type Container’s Start-up Time Of 150 Milliseconds

Next Generation LightVMs - Size

- Additionally, The Reduced Footprint Of The Unikernel Requires Only About 1/10th The Memory Of A Docker-type Container Running On A Debian Kernel

- Since Memory Is Quite Often The Limiting Factor In Properly Designed Microservices, This Means That 10 Times More Unikernel/LightVM Microservice Instances Can Be Run On The Same Physical Hardware.

LightVM Unikernel vs Docker



LightVM Boot Times On A 64-Core Machine With 128GB Memory vs Docker Containers

Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, SimonKuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and FelipeHuici. 2017. My VM is Lighter (and Safer) than your Container. InProceedings of SOSP '17: ACM SIGOPS 26th Symposium on OperatingSystems Principles, Shanghai, China, October 28, 2017 (SOSP '17),16 pages.https://doi.org/10.1145/3132747.313276

Practical Unikernels

Unikernels Have, Until Recently, Been The Province Of Laboratories And Research Projects

This Has Changed As Unikernel Technology Has Matured:

- More Complete Function Libraries
- Mainstream Programming Languages

One Approach – Reuse - AnyKernel

- NetBSD, A Version Of UNIX, Is Famous For Its Ability To Be Ported To New Hardware
- It’s A Monolithic Kernel, But Has Been Internally Structured Into Well Defined Functions And Layers
- A Library Of NetBSD Functions Has Been Created, Called “The Anykernel” Concept
- The Anykernel Concept Allows Existing Application Code, Designed For The Linux Or UNIX (POSIX) Operating Systems To Be Statically Linked With Operating System Functions And Drivers, Forming A Unikernel!

Another Approach – Ground Up - IncludeOS

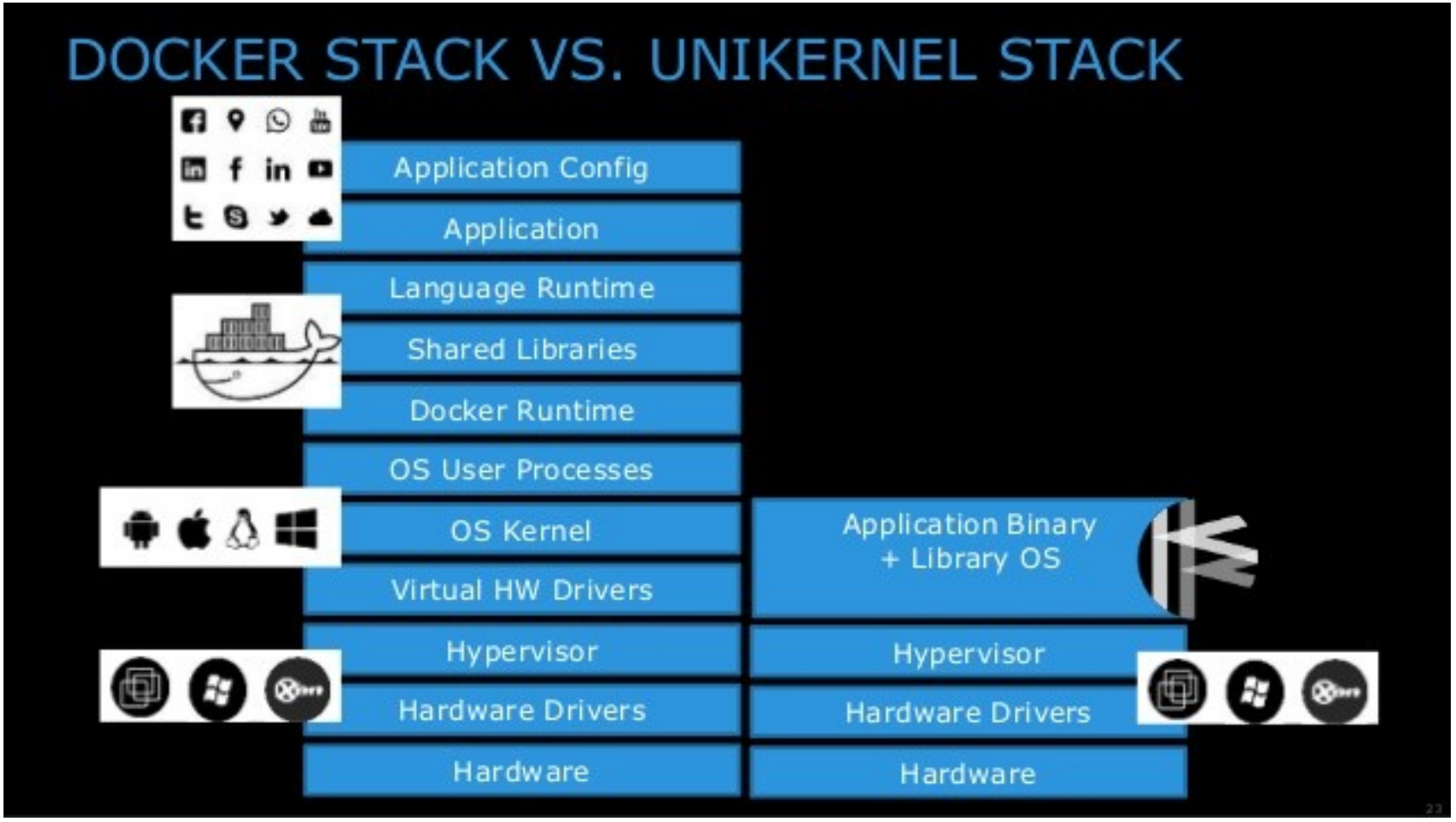
- What Access Does A Modern Cloud Application Require?
  - A Packet Interface For Network Communications
  - A Block Interface For Some Storage
  - A Serial Port To Output Console Data
- IncludeOS Team Wrote These Interfaces (And Other Necessary POSIX Interfaces) From Scratch In C++
- Vast Majority Of Existing C And C++ Applications Will Link Successfully With IncludeOS

Practical Unikernels and Library Operating Systems

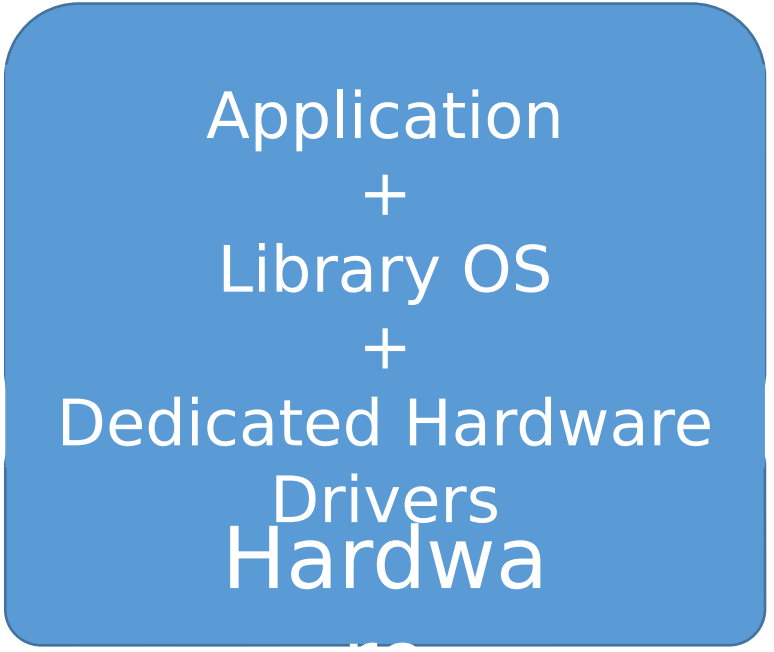
- MirageOS (Written In OCaml)
- ClickOS (Runs Click NFV language)
- HaLVM (Written In Haskell)
- Ling (Written In Erlang)
- RumpKernel (NetBSD AnyKernel - Written In C/C++)
- IncludeOS (Written In C/C++)

With The Last Two, You Can Develop Unikernel Applications In Python, Ruby, Node, Java, etc.

Containers vs Unikernels



For IoT, It’s Even Simpler



VM Unikernel Applications Can Be Managed With The Same Tools As Containers

- CoreOS ‘rkt’ Can Run Unikernel VMs With Docker Swarm, Kubernetes, Or Apache Mesos Management Engines
- Kubernetes:
  - Kubevirt
  - Virtlet
  - RancherVM

Drawbacks
Every Rose Has Its Thorn

- Unikernels in LightVMs Is A New Paradigm
- Lack Of Experience
- Limited Selection Of Libraries And Build Tools
- Existing Applications May Require Modification
- May Be More Difficult To Develop And Debug

Further Resources

- Worried about IoT DDoS? Think Unikernels, Levine, Idit, 4/14/2017 (https://github.com/solo-io/unik/wiki/Worried-about-IoT-DDoS%3F-Think-Unikernels)
- Enterprise IoT Security and Scalability: How Unikernels can Improve the Status Quo, Duncan, Bob; Happe, Andreas; Bratterud, Alfred; IEEE Xplore, 3/20/2107 (https://ieeexplore.ieee.org/document/7881647)
- Unikernels + connected devices, Ryd, Thomas, 9/8/2016 (https://mender.io/blog/unikernels-connected-devices)
- UniK: Build and Run Unikernels with Ease, Levine, Idit 10/26/2016 (https://github.com/solo-io/unik/wiki/UniK:-Build-and-Run-Unikernels-with-Ease)
- What is a unikernel, and why does it matter?, Hewitt Packard Enterprise, 10/2/2017 (https://www.hpe.com/us/en/insights/articles/what-is-a-unikernel-and-why-does-it-matter-1710.html)
- Debunking Unikernel Criticisms, Oliver, Kiran; Jackson, Joab, 10/21/2016 (https://thenewstack.io/utilizing-unikernels-within-internet-things/)
- Making operating systems safer and faster with ‘unikernels’, University of Cambridge, 1/28/2016 (https://www.cam.ac.uk/research/news/making-operating-systems-safer-and-faster-with-unikernels)
- A unikernel experiment: A VM for every URL, Skjegstad, Magnus, 3/25/2015 (http://www.skjegstad.com/blog/2015/03/25/mirageos-vm-per-url-experiment/)
- Unikernel, Wikipedia, 1/5/2018 (https://en.wikipedia.org/wiki/Unikernel)
- My VM is lighter (and safer) than your container, Manco et al., SOSP’17 (http://cnp.neclab.eu/projects/lightvm/lightvm.pdf)
- Unikernel Monitors: Extending Minimalism Outside of the Box , Williams, Koller, 6/20/2016 (https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_williams.pdf)
- UniK: Build and Run Unikernels with Ease, Levine, Idit, 10/26/2016 (https://github.com/solo-io/unik/wiki/UniK:-Build-and-Run-Unikernels-with-Ease)
- Un_GoBack_GoBackiKraft – The Xen Project (https://xenproject.org/developers/teams/unikraft/)

Copies of the Slides May Be Downloaded From the Formularity Website

https://formularity.com

Super Containers: Unikernels and Virtual Machines

SouthEast LinuxFest 2023

Brad Whitehead, Chief Scientist – Formularity

June 10, 2023

Good afternoon. I'm Brad Whitehead. I'm the Chief Scientist at Formularity. Before I get started, I'd like to thank the incredible volunteers that put on the SELF! This conference is outstanding. I'd also like to thank you for choosing to attend this talk. I hope I make it worth your while!

“Super Container”

- 1400 Times More Secure Than A Well-configured Docker Container
- Boots 37 Times Faster Than That Docker Container
- Can Run 10 Times More Microservices On The Same Physical Hardware
- Can Be Managed By Kubernetes Or Apache Mesos

Super Containers - That’s not their real name, but what else would you call a container that’s:

- 1400 Times More Secure Than A Well-configured Docker Container
- Boots 37 Times Faster Than That Docker Container
- Can Run 10 Times More Microservices On The Same Physical Hardware
- Can Be Managed By Kubernetes Or Apache Mesos

What Is This Incredible New Technology?

It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor

What Is This Incredible New Technology?
It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor

What's A Unikernel?

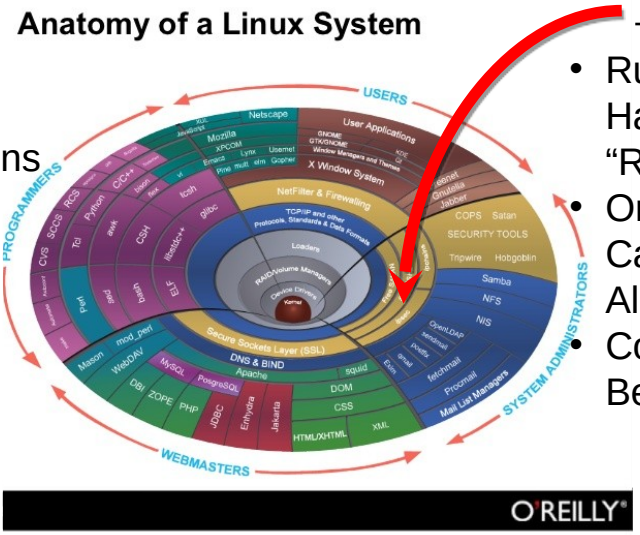
- To Answer That Question, We Have To Take A Look At The Structure Of A Modern Operating System
- Doesn't Matter If It's Microsoft Windows, Linux, UNIX, ~~Mac OS X~~ ~~OS X~~ macOS, etc.
- All The Mainstream Operating Systems Have The Same Fundamental Anatomy

To answer this, let's take a look at a modern operating system
Doesn't matter if we choose Windows, Linux, UNIX, or whatever Apple's calling their desktop operating system these days!
They are all basically the same

The Anatomy of An Operating System

"UserLand"

- Where Applications Run
- No Privileges
- Can Not Access Resources
- Can Not "Talk" to Other Programs



"The Kernel"

- Runs in Special Hardware Mode – "Ring 0"
- Only Program That Can Access or Allocate Resources
- Copies Data Between Programs

Anatomy of an Operating System
User Interface (CLI or GUI)
Standard set of tools and applications ("Userland")
Kernel (privileged, separated from Userland by hardware)
Monolithic (Linux even includes a web server in the kernel!)
Famous flame war between Torvald Linus and Dr. Andrew Tanenbaum, 1992 Usenet
Microkernel (Mach, Minix)
Context switches
data copying

Growth of Operating Systems

- Linux Kernel Is Now 22 Million Lines Of Code!
- Windows Is Estimated At 50 Million Lines Of Code!
- With An Industry Average Of 15-50 Defects Per 1000 Lines Of Code*:
  - Linux(Just The Kernel) = 330,000 To 1.1 Million Defects
  - Windows = 750,000 To 2.5 Million Defects

*(Steve McConnell, “Code Complete 2”, 2005)

Operating system kernels have grown enormously over time!
Linux 22 million SLOC
Windows estimated to be 50 million SLOC
Now, Steve McConnell has conducted a number of studies on the defect rates in modern system software. He estimates that there are between 15 and 50 defects per 1000 lines of code!
Now not all defects result in errors. A defect is anything that does not meet the requirements or is not as intended by the developer. So a defect could be as simple as a misspelled word in an error message. Or as serious as a buffer overrun or a “use after free” pointer error!
Using McConnell’s figures, the Linux kernel - just the kernel - has 330,000 to 1.1 million defects
Windows has 750,000 to 2.5 million defects

It Gets Worse!

- The “Userland” Support Software Is Often 10 To 20 Times Larger Than The Kernel!
- Red Hat Enterprise Linux (RHEL) Userland Is Approximately 420 Million Lines Of Code
  - Try Not To Think About The 6.3 To 21 Million Defects Running On Your Bank’s Server!

That was just the kernel. The kernel needs support software to boot and to perform standard services. This support software, along with all the other applications that come with an operating system distribution, is that userland I referred to a moment ago. Red Hat Enterprise Linux is the leader in Linux software for businesses. Their userland software is around 420 million lines of code. Again, taking McConnell’s numbers, each of your your bank servers could have 6.3 to 21 million defects running!

Can It Get Even Worse?!?!

- The Kernel Is Full Of Junk!
- A Large Number Of Device Drivers Are Routinely Compiled Into The Kernel, Regardless Of The Actual Hardware In The Computer
  - There Are Device Drivers For Hardware That No Longer Exists
  - Amazon Ami Images ~~Have~~ Had Drivers For Floppy Disks And Audio Cards
    - In 2015, The Venom Vulnerability (CVE-2015-3456) Used A Flaw In The Floppy Disk Controller (FDC) Driver To Compromise Both Physical And Virtual Machines

Why is the kernel so big!?!? For one thing, a large number of device drivers are routinely compiled into the kernel. That way, whether you load the operating system on an IBM server or a SuperMicro, it just works - magic! There are device drivers in the kernel for hardware that no longer exists. Amazon's standard Linux image had floppy disk and audio card drivers compiled into it. How many floppy drives are there in an Amazon data center? Who uses those audio cards? I hope Amazon has gone back and recompiled their kernel images, because in 2015, the Venom malware used a defect in the floppy disk driver to compromise both the VM in which it was running, as well as the physical host. How many other device driver defects are out there, waiting to be exploited?

Can It Get Even Worse?!?! (Continued)

- Likewise, There Are Thousands Of Storage And Communications Protocols In The Kernel That Will Not Be Used In Your Application
- Linux Recognizes 7 Different Executable Formats, Even Though The Vast Majority Of Applications (Including Yours) Are In ELF Format
- **Each Of These Extra, Unused Chunks Of Code (With Its 15-50 Defects/1000 Sloc) Is A Potential Hack Waiting To Happen!**

It's not just device drivers. There are a large number of file system drivers and communications protocols compiled into the kernel. Many, most, of these are esoteric and probably won't be needed by your applications. But they are still sitting there, taking up space, processing power, and the reliability budget

What If We Cut Out All The Parts We Don't Need?

- Code Traces Show That The Average Application Uses Less Than 0.08% Of The Total Code In The Kernel!
- Take The Standard C Library As An Example
  - The C Library Contains Thousands Of Functions, But A Modern Linker Only Includes The Actual Functions (And Code) That An Application Uses
- Could We Do The Same With Our Operating System?

When you code trace an average application, you find that it only uses 0.08% of the code in the kernel! Wouldn't it be great if we could jettison the extra 99.92% of the kernel we don't need?

Most of our modern libraries and linkers do this type of jettisoning. The C library has thousands of functions, but when it's linked into an executable, only the functions that are actually used are linked into the code.

Wouldn't it be great if our kernel only contained the functions we needed?

What About Actors (Microservices)?

- Run A Single Application
- As A Single User
- Known Set Of Hardware Drivers
- 1 Or 2 Communications Protocols
- Speed (Startup And Latency)
- Reliability
- Security (From Unauthorized Access - “Hacking”)
- Repeatability (Multiple Identical Servers)

So what are the functions we need?

Let’s make up a list. Let’s assume our application is a microservice or perhaps a Internet of Things application

- Run A Single Application
- As A Single User
- Known Set Of Hardware Drivers
- 1 Or 2 Communications Protocols
- Speed (Startup And Latency)
- Reliability
- Security (From Unauthorized Access - “Hacking”)
- Repeatability (Multiple Identical Servers)

Keeping Only The Parts of the Operating System We Actual Use

- What Does It Buy Us?:
  - Let's Start With Security:
    - Greatly Reduced Attack Surface (99.92% Reduction)
    - Potentially A Small Enough Subset To Be Mathematically Verifiable
    - We Don't Need Any Userland Applications (Bye-bye 410 Million Lines Of Potentially Flawed Code!)
    - No Ability To Run Malicious Or Hacking Tools On Our Server Or IoT Device

So, when we strip away 99.92% of the kernel code, what do we get?
Well, first - Security
Reduced attack surface – .08% of typical
Small enough to possibly be mathematically verified
No tools (no shell, etc.)

More Benefits

We Can Statically Link Everything (Including The Kernel Functions) And Our Software Becomes Immutable

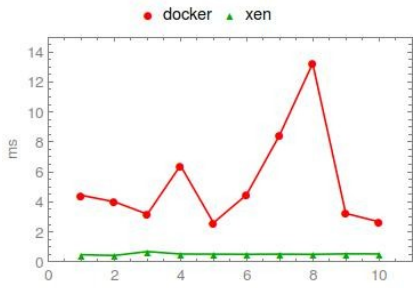
- No Injection Attacks
- No Re-configuration Attacks
- Vastly Reduced “Return Oriented Programming” (ROP) Vulnerability

Increased Reliability And Improved Security Means Reduced Devops Costs!

We can statically link everything together
At that point, our code becomes immutable
Modules and dynamic libraries can’t be added, code can’t be injected

Increased Performance

- Smaller, Less Memory Intensive Images Mean More Virtual Machines Per Hardware Server
  - 5 Megabyte Virtual Machines = 10,000 VMs Per Physical Server
  - Smaller Than Most Docker Containers
- 6 Millisecond Boot Times
  - Jitsu – Boot-On-Demand
- 45 Microsecond Throughput Times
  - No Context Switches
  - No Information Copying
  - Single Address Space



We also get huge performance gains!

Smaller instances or more VMs per instance - 5MB per VM, 10K VMs/hardware server

6 millisecond boot

Since all the code is running in Ring 0 as privileged code, there are no context switches and no need to copy memory between kernel and applications

Server-less Functions (with servers)! – 45 microsecond response

Jitsu

How To Include Only The Needed Code?

- Again, The C Library Analogy Is The Key
  - The C Library Is Actually A “Middle Ware Layer”
  - It Converts Standard C Function Calls Into Equivalent Kernel System Calls
  - Instead of Handing The Function Call Off As a System Call, What If We Extended the C Library to Include the Appropriate Kernel Code?
    - Instead of the C Library Passing a “Printf()” Call To The Kernel, the Library Can Include the Machine Instructions to Do The Actual I/O

How do we include only the required kernel code?

Again, let’s look at the the C Library. The C Library presents a standard POSIX interface to the application’s C code.

When a function in the library is called, the library prepares all the parameters and then executes specific operating system calls.

Why don’t we combine the C library and the kernel code it calls? Then, when we link in the C library code, we get both the POSIX interface code and the underlying kernel functionality

The “Library Operating System”

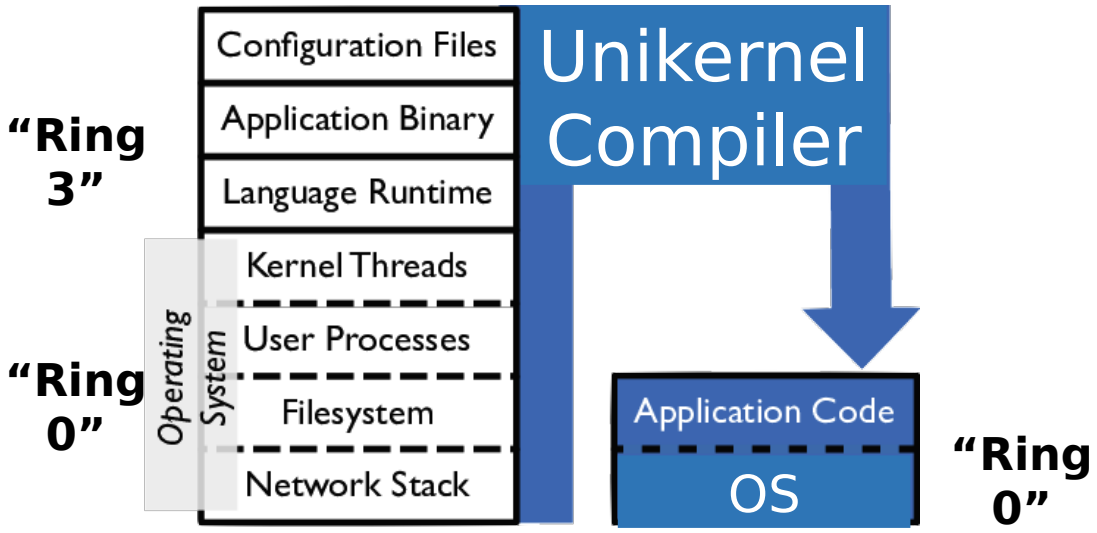
- Common Operating System Functions, Drivers, And Protocols Are Written As A Library Of Functions
- When You Link These “Library Operating System” Functions To Your Application, You Have A Single Executable That Runs Directly On Hardware Or A Hypervisor…

…You Have A Unikernel!

This approach is called, obviously enough, a “library operating system”

- Common Operating System Functions, Drivers, And Protocols Are Written As A Library Of Functions
- When You Link These “Library Operating System” Functions To Your Application, You Have A Single Executable That Runs Directly On Hardware Or A Hypervisor…
- in other words, you have a Unikernel!

What Does A Unikernel Look Like??



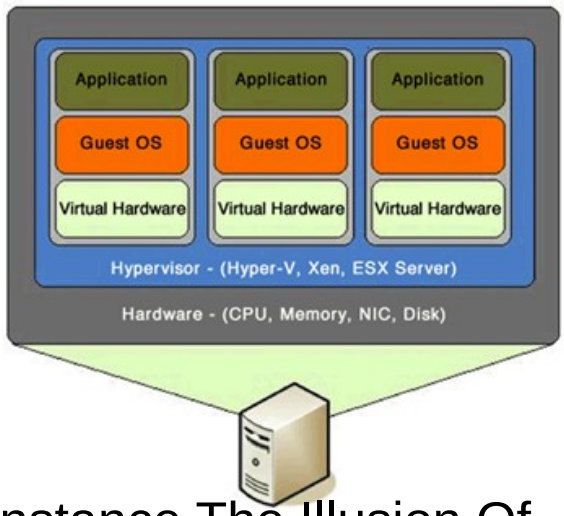
Visually, this is what a unikernel looks like:
On the left, you have the conventional software stack. The bottom four layers are operating system code. They run in Ring 0 and can access the hardware directly. The top three layers are userland and application code. They are unprivledged and run in Ring 3.
On the right hand side is our unikernel. The unikernel compiler has extracted only the operating system functionality we need and combined it with our application code. For the most part, we don't need any userland code. Our new unikernel runs in Ring 0

Unikernels Are Only Half The Answer To Small, Fast, Secure Containers

The Other Half Are Lighter Weight Virtual Machines

Unikernels Are Only Half The Answer To Small, Fast, Secure Containers
The Other Half Are Lighter Weight Virtual Machines

Virtual Machines



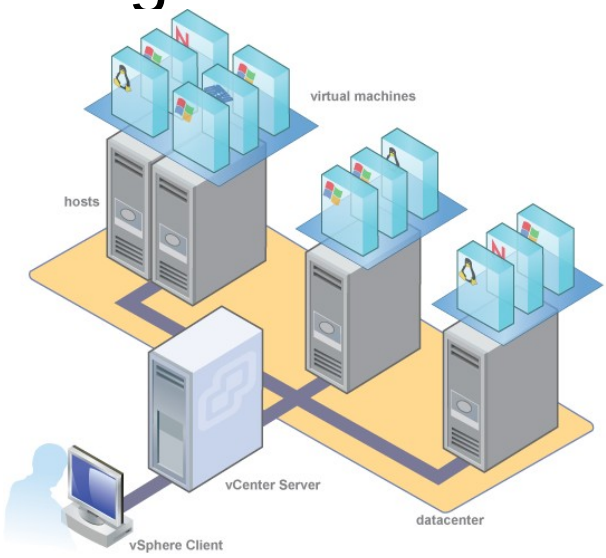
- Virtual Machine Monitor (VMM) Or “Hypervisor” Typically Sits Between The Real Hardware And Multiple Operating Systems
- Gives Each Operating System Instance The Illusion Of Running On Its Own Hardware – A “Virtual Machine”
- **Strong** Physical Isolation Between Operating Systems

I don’t want to spend a lot of time on virtual machines. I assume most of you are familiar with them, at least to the degree we need in order to discuss super containers.

- Virtual Machine Monitor (VMM) Or “Hypervisor” Typically Sits Between The Real Hardware And Multiple Operating Systems
- Gives Each Operating System Instance The Illusion Of Running On Its Own Hardware – A “Virtual Machine”
- Strong Physical Isolation Between Operating Systems

Cloud Computing

- Virtual Machines Are The “Fuel” Of Cloud Computing
- Multiple “Virtual Machines”, Each With Its Own Operating System
- Each Virtual Machine Isolated And Managed By The Virtual Machine Monitor Or Hypervisor



- Virtual Machines Are The “Fuel” Of Cloud Computing
- Multiple “Virtual Machines”, Each With Its Own Operating System
- Each Virtual Machine Isolated And Managed By The Virtual Machine Monitor Or Hypervisor

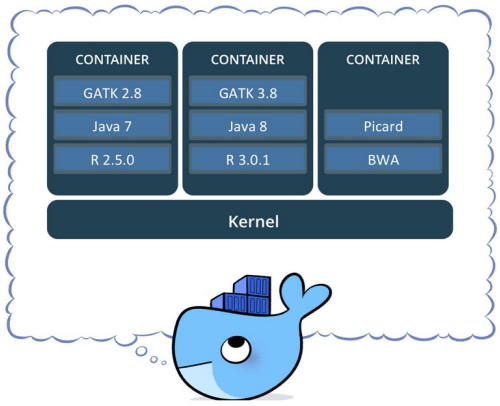
Drawbacks to Current Virtual Machines

- Size - Each VM Requires Its Own Operating System, Userland Software, And A Certain Amount Of Dedicated Memory, Making VMs **BIG**:
  - VMware – Max Of 380 VMs Per Physical Host
  - AWS Xen ~ 10 VMs Per Physical Core
  - AWS Nitro – 6000 VMs On A 36 Core Processor
- Speed - Slow To Startup – Boot Times Measured In Seconds And Minutes

- Size - Each VM Requires Its Own Operating System, Userland Software, And A Certain Amount Of Dedicated Memory, Making VMs BIG:
  - VMware – Max Of 380 VMs Per Physical Host
  - AWS Xen ~ 10 VMs Per Physical Core
  - AWS Nitro – 6000 VMs On A 36 Core Processor
- Speed - Slow To Startup – Boot Times Measured In Seconds And Minutes

Enter the “Container”

- A Container Is A Package That Bundles Up An Application And All Its Dependent Userland Software (Such As Libraries And Services) Into A Single Image
- A Container Runs Like A Pseudo-Virtual Machine, **Weakly** Isolated From The Host Processes And Other Containers
- All Containers On A Host Use The Host’s Kernel
- While There Are Several Different Container Formats, Docker Is The Most Common



- A Container Is A Package That Bundles Up An Application And All Its Dependent Userland Software (Such As Libraries And Services) Into A Single Image
- A Container Runs Like A Pseudo-Virtual Machine, Weakly Isolated From The Host Processes And Other Containers
- All Containers On A Host Use The Host’s Kernel
- While There Are Several Different Container Formats, Docker Is The Most Common

Notice in the diagram how there are two different versions of Java running side-by-side! Dependency isolation is a major feature of containers

Advantages of Containers

- Neatly Solves The Library Dependency And Versioning Problem (“DLL Hell”)
- Since The Kernel Is Already Running, Containers “Boot” In Milliseconds
- Less Dedicated Memory Is Required For A Container Than A Conventional VM
- “Orchestration” Software Has Been Developed To Deploy And Manage Containers
  - Kubernetes, Apache Mesos, Docker Swarm, et al
  - Google, Netflix

- Neatly Solves The Library Dependency And Versioning Problem (“DLL Hell”) - This is a Windows term, but the concept still applies to Linux and other operating systems that support dynamic library loading
- Since The Kernel Is Already Running, Containers “Boot” In Milliseconds
- Less Dedicated Memory Is Required For A Container Than A Conventional VM
- “Orchestration” Software Has Been Developed To Deploy And Manage Containers
  - Kubernetes, Apache Mesos, Docker Swarm, et al

Containers Have Changed The Way We Develop and Deploy Software

- Applications Are Deployed As Complete Images, Ready To Run, Instead Of Being Installed
- Containers Are Replaced, Rather Than Being “Patched”
- Containers Support The Concept Of “Microservices”, Allowing Complex Applications To Be Built From Single-function Services Wired Together Through Orchestration Managers
- Multiple Containers Can Be Started And Stopped In Response To Traffic Loads

- Applications Are Deployed As Complete Images, Ready To Run, Instead Of Being Installed
- Containers Are Replaced, Rather Than Being “Patched”
- Containers Support The Concept Of “Microservices”, Allowing Complex Applications To Be Built From Single-function Services Wired Together Through Orchestration Managers
- Multiple Containers Can Be Started And Stopped In Response To Traffic Loads

Drawbacks of Containers

- Limited Isolation Between Containers – Not A Security Mechanism
- The Container Manager Must Run As Root Or Administrator
- Difficult To Strip Down Userland And Container Images
  - Bloat Consumes Memory And Processing Resources
- Differences In Production And Development Environments

- Limited Isolation Between Containers – Not A Security Mechanism
- The Container Manager Must Run As Root Or Administrator
- Difficult To Strip Down Userland And Container Images
  - Bloat Consumes Memory And Processing Resources

- Differences In Production And Development Environments - Containers are generally “sold” on the fact that development and production are supposed to be the identical environments, but in reality, they never are. You have development and debugging tools in the Development Container that you (better!) remove from Production. Hence the two environments are different. The difference may not make a different (should not make a difference), but then again an accidentally unsatisfied dependency may be discovered in Production at 4AM

Meanwhile, Virtual Machine Technology Has Not Stood Still

Recent Optimizations To Both The Xen And The Linux "Kernel-based Virtual Machine" (KVM) Hypervisors Have:

- Significantly Reduced The Start-up Time Of A Virtual Machine
- Reduced Performance-Robbing Overhead

This New Generation Of Hypervisors Are Called **"LightVMs"**

- Significantly Reduced The Start-up Time Of A Virtual Machine
- Reduced Performance-Robbing Overhead

This New Generation Of Hypervisors Are Called "LightVMs"

Next Generation LightVMs - Speed

- Combined With Unikernels, These LightVMs Can Launch Microservices In As Little As 4 Milliseconds
- This Is Comparable To The Linux Kernel’s Exec/Fork Times Of Approximately 1 Millisecond And Significantly Faster Than A Docker-type Container’s Start-up Time Of 150 Milliseconds

- Combined With Unikernels, These LightVMs Can Launch Microservices In As Little As 4 Milliseconds
- This Is Comparable To The Linux Kernel’s Exec/Fork Times Of Approximately 1 Millisecond And Significantly Faster Than A Docker-type Container’s Start-up Time Of 150 Milliseconds

Next Generation LightVMs - Size

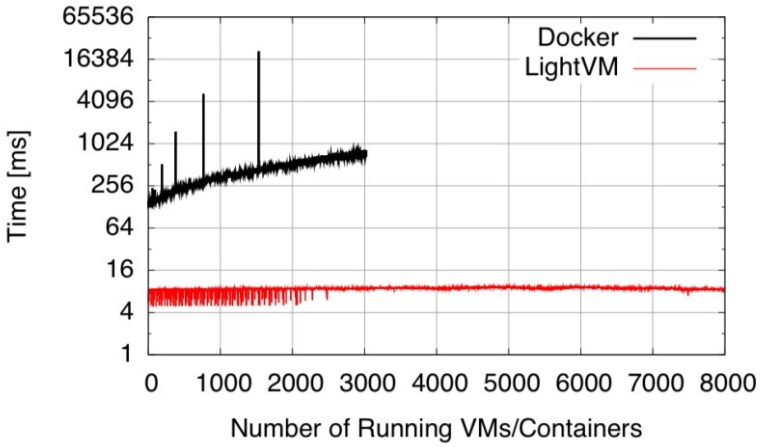
- Additionally, The Reduced Footprint Of The Unikernel Requires Only About 1/10th The Memory Of A Docker-type Container Running On A Debian Kernel

- Since Memory Is Quite Often The Limiting Factor In Properly Designed Microservices, This Means That 10 Times More Unikernel/LightVM Microservice Instances Can Be Run On The Same Physical Hardware.

- Additionally, The Reduced Footprint Of The Unikernel Requires Only About 1/10th The Memory Of A Docker-type Container Running On A Debian Kernel

- Since Memory Is Quite Often The Limiting Factor In Properly Designed Microservices, This Means That 10 Times More Unikernel/LightVM Microservice Instances Can Be Run On The Same Physical Hardware.

LightVM Unikernel vs Docker



LightVM Boot Times On A 64-Core Machine With 128GB Memory vs Docker Containers

Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, SimonKuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and FelipeHuici. 2017. My VM is Lighter (and Safer) than your Container. InProceedings of SOSP '17: ACM SIGOPS 26th Symposium on OperatingSystems Principles, Shanghai, China, October 28, 2017 (SOSP '17),16 pages.https://doi.org/10.1145/3132747.313276

The top line are Docker Containers being launched, starting at 150 milliseconds, going to 1 full second at the 3000th container launched. The line doesn't extend beyond 3000 containers because the physical machine ran out of memory (128GB) at that point.

Now, notice the red line. Those are equivalent functionality unikernels. They all launch at a fairly consistent 4 milliseconds. Memory exhaustion now occurs at north of 8000 unikernel VMs on the same hardware.

Remember the old "C10K challenge" - How many http sessions a single web server could support? Well the new equivalent is "VM100K" - running 100,000 VMs on the same physical host. Think what that does to cloud computing economics. Amazon going from 300 VMs per physical server to 100,000 VMs!

Practical Unikernels

Unikernels Have, Until Recently, Been The Province Of Laboratories And Research Projects

This Has Changed As Unikernel Technology Has Matured:

- More Complete Function Libraries
- Mainstream Programming Languages

- More Complete Function Libraries
- Mainstream Programming Languages

One Approach – Reuse - AnyKernel

- NetBSD, A Version Of UNIX, Is Famous For Its Ability To Be Ported To New Hardware
- It's A Monolithic Kernel, But Has Been Internally Structured Into Well Defined Functions And Layers
- A Library Of NetBSD Functions Has Been Created, Called "The Anykernel" Concept
- The Anykernel Concept Allows Existing Application Code, Designed For The Linux Or UNIX (POSIX) Operating Systems To Be Statically Linked With Operating System Functions And Drivers, Forming A Unikernel!

- NetBSD, A Version Of UNIX, Is Famous For Its Ability To Be Ported To New Hardware
- It's A Monolithic Kernel, But Has Been Internally Structured Into Well Defined Functions And Layers
- A Library Of NetBSD Functions Has Been Created, Called "The Anykernel" Concept
- The Anykernel Concept Allows Existing Application Code, Designed For The Linux Or UNIX (POSIX) Operating Systems To Be Statically Linked With Operating System Functions And Drivers, Forming A Unikernel!

Another Approach – Ground Up - IncludeOS

- What Access Does A Modern Cloud Application Require?
  - A Packet Interface For Network Communications
  - A Block Interface For Some Storage
  - A Serial Port To Output Console Data
- IncludeOS Team Wrote These Interfaces (And Other Necessary POSIX Interfaces) From Scratch In C++
- Vast Majority Of Existing C And C++ Applications Will Link Successfully With IncludeOS

- What Access Does A Modern Cloud Application Require?
  - A Packet Interface For Network Communications
  - A Block Interface For Some Storage
  - A Serial Port To Output Console Data
- IncludeOS Team Wrote These Interfaces (And Other Necessary POSIX Interfaces) From Scratch In C++
- Vast Majority Of Existing C And C++ Applications Will Link Successfully With IncludeOS

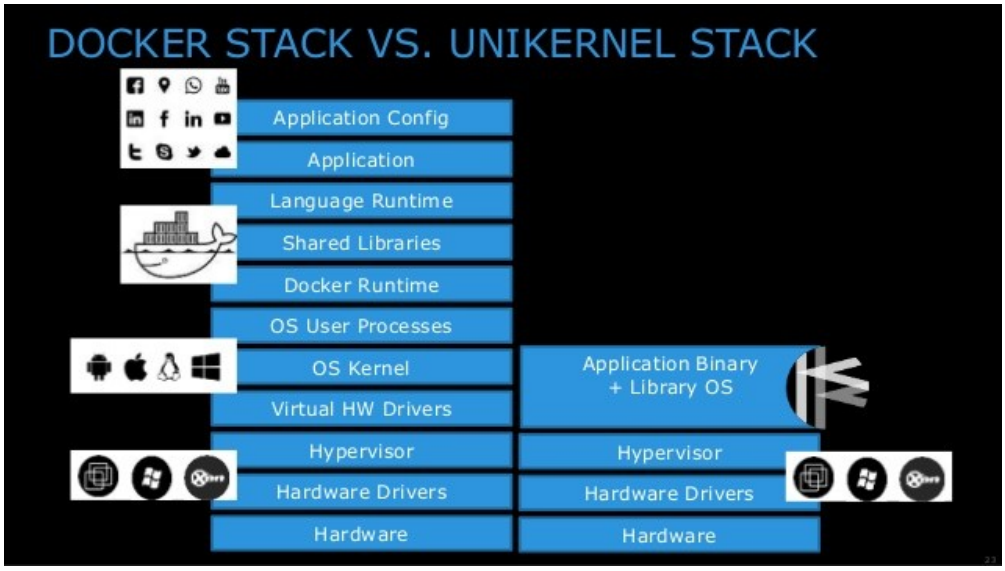
Practical Unikernels and Library Operating Systems

- MirageOS (Written In OCaml)
- ClickOS (Runs Click NFV language)
- HaLVM (Written In Haskell)
- Ling (Written In Erlang)
- RumpKernel (NetBSD AnyKernel - Written In C/C++)
- IncludeOS (Written In C/C++)

With The Last Two, You Can Develop Unikernel Applications In Python, Ruby, Node, Java, etc.

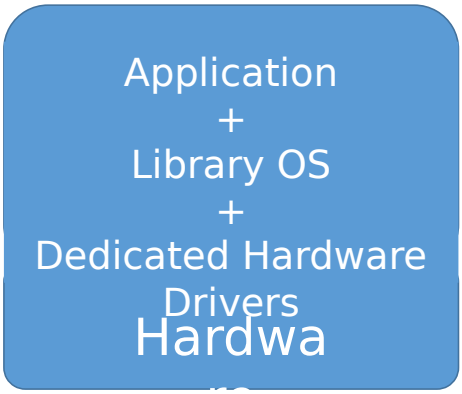
Practical Unikernels and Library Operating Systems
MirageOS
RumpKernel
ClickOS (runs Click NFV language)
HaLVM (Haskell)
HermitCore (C/C++/FORTRAN/Go)
IncludeOS (C/C++)
OSv (C/C++/Java/Ruby/JavaScript)
Runtime.js (JavaScript)

Containers vs Unikernels



So, this is what our microservices computing stack looks like with unikernels. We cut out a lot of unnecessary, defect-ridden layers, and merge other layers. We have a single executable, with no dependencies, ready to run on physical hardware, or under any hypervisor

For IoT, It's Even Simpler



The situation is even better for IoT applications. Especially since we know exactly what hardware we are running on!

VM Unikernel Applications Can Be Managed With The Same Tools As Containers

- CoreOS ‘rkt’ Can Run Unikernel VMs With Docker Swarm, Kubernetes, Or Apache Mesos Management Engines
- Kubernetes:
  - Kubevirt
  - Virtlet
  - RancherVM

The good news is that managing our paradigm-shifting unikernels does not require any further paradigm shift ;-)
We can manage unikernels using the same tools as containers; Kubernetes, Mesos, Swarm, etc., using adapters
These adapters include:
Kubevirt
Virtlet
and RancherVM
CoreOS rkt can natively manage virtual machines, as well as containers

Drawbacks
Every Rose Has Its Thorn

- Unikernels in LightVMs Is A New Paradigm
- Lack Of Experience
- Limited Selection Of Libraries And Build Tools
- Existing Applications May Require Modification
- May Be More Difficult To Develop And Debug

Drawbacks?
Hardware or hypervisor specific drivers
Existing applications may not run correctly in a shared memory model

Further Resources

- Worried about IoT DDoS? Think Unikernels, Levine, Idit, 4/14/2017 (https://github.com/solo-io/unik/wiki/Worried-about-IoT-DDoS%3F-Think-Unikernels)
- Enterprise IoT Security and Scalability: How Unikernels can Improve the Status Quo, Duncan, Bob; Happe, Andreas; Bratterud, Alfred; IEEE Xplore, 3/20/2107 (https://ieeexplore.ieee.org/document/7881647)
- Unikernels + connected devices, Ryd, Thomas, 9/8/2016 (https://mender.io/blog/unikernels-connected-devices)
- UniK: Build and Run Unikernels with Ease, Levine, Idit 10/26/2016 (https://github.com/solo-io/unik/wiki/UniK:-Build-and-Run-Unikernels-with-Ease)
- What is a unikernel, and why does it matter?, Hewitt Packard Enterprise, 10/2/2017 (https://www.hpe.com/us/en/insights/articles/what-is-a-unikernel-and-why-does-it-matter-1710.html)
- Debunking Unikernel Criticisms, Oliver, Kiran; Jackson, Joab, 10/21/2016 (https://thenewstack.io/utilizing-unikernels-within-internet-things/)
- Making operating systems safer and faster with 'unikernels', University of Cambridge, 1/28/2016 (https://www.cam.ac.uk/research/news/making-operating-systems-safer-and-faster-with-unikernels)
- A unikernel experiment: A VM for every URL, Skjegstad, Magnus, 3/25/2015 (http://www.skjegstad.com/blog/2015/03/25/mirageos-vm-per-url-experiment/)
- Unikernel, Wikipedia, 1/5/2018 (https://en.wikipedia.org/wiki/Unikernel)
- My VM is lighter (and safer) than your container, Manco et al., SOSP'17 (http://cnp.neclab.eu/projects/lightvm/lightvm.pdf)
- Unikernel Monitors: Extending Minimalism Outside of the Box , Williams, Koller, 6/20/2016 (https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_williams.pdf)
- UniK: Build and Run Unikernels with Ease, Levine, Idit, 10/26/2016 (https://github.com/solo-io/unik/wiki/UniK:-Build-and-Run-Unikernels-with-Ease)
- Un_GoBack_GoBackiKraft – The Xen Project (https://xenproject.org/developers/teams/unikraft/)

Resources

Copies of the Slides May Be Downloaded From

the

Formularity Website

https://formularity.com

OK, at this point, hopefully I've demonstrated the security, performance, and resource savings of unikernels. Given the security problems of current full operating systems, I truly believe that unikernels are the single most effective base for acceptable business (microservice) and IoT device security. Thank you! Copies of these slides and my talking notes will be available on the Formularity website later today. Are there any questions?...