VIRTUALIZATION STACK

The aim is to run single Application with a single user on a single server

Redundancy in the stack – e.g. Isolation
KERNEL COMPLEXITY - PROTECTION

Application safe from application

User safe from user

Application safe from user
INEFFICIENCY

• Needless permission check, it is hard and an updated model from time sharing computer from the 50s, 60s

• Microservices architecture duplicate what Linux did for us

• Kernel include a lot of unnecessary drivers that not being used: floppy

• Update and patches using yum bring a lot of unnecessary components
SECURITY

- Very large attack surface
- A lot of exploits target Linux. It is harder to attack hypervisor - not expose to the internet
- Microservices architecture
  Sharing – Kernel, Memory, filesystem, hardware
  The only thing that makes it safe is kernel extension like: cgroup
HOW DID WE GET HERE? EVOLUTION!

Unix was supported us the entire way!
DECADES OF BACKWARDS COMPATIBILITY

What can Linux run on?

Anything!

What can run on Linux?

Anything!
TRADE OFF

Compatibility

VS

Efficiency
Make it works.

Make it right.

Make it fast.
{uni-} One; having or consisting of one.

{kernel} a bridge between applications and the actual data processing done at the hardware level.
LINUX KERNEL LANGUAGES

AWK
Shell Script
Make
Vim Script
Automake
Scheme
C++
Perl
Autoconf
TeX/LaTeX
HTML
Assembly
XSL Transformation
Objective-C
C
Python
XML
SOURCE LINES OF CODE

Really huge applications: 1Ms

Medium to large applications: 100Ks

Small Applications: 10Ks
Linux Kernel SLOC

<table>
<thead>
<tr>
<th>Linux kernel 2.4.2</th>
<th>Linux kernel 2.6.0</th>
<th>Linux kernel 2.6.29</th>
<th>Linux kernel 2.6.32</th>
<th>Linux kernel 2.6.35</th>
<th>Linux kernel 3.6</th>
<th>Linux kernel pre-4.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>5.2</td>
<td>11</td>
<td>12.6</td>
<td>13.5</td>
<td>15.9</td>
<td>22</td>
</tr>
</tbody>
</table>

© Copyright 2016 EMC Corporation. All rights reserved.
Debian SLOC

<table>
<thead>
<tr>
<th>Debian Version</th>
<th>Year</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian 2.2</td>
<td>2000</td>
<td>59</td>
</tr>
<tr>
<td>Debian 3.0</td>
<td>2002</td>
<td>104</td>
</tr>
<tr>
<td>Debian 3.1</td>
<td>2005</td>
<td>215</td>
</tr>
<tr>
<td>Debian 4.0</td>
<td>2007</td>
<td>283</td>
</tr>
<tr>
<td>Debian 5.0</td>
<td>2009</td>
<td>324</td>
</tr>
<tr>
<td>Debian 7.0</td>
<td>2012</td>
<td>419</td>
</tr>
</tbody>
</table>
TRADITIONAL APPROACH

Application

openGL

iconv    gtk

libz      libgmp    libtlc

libc      Libstd++    libgcc

Kernel
UNIKERNEL APPROACH

Application

openGL

iconv
gtk

libz
libgmp
libtlc

libc
Libstd++
libgcc

Kernel
UNIKERNEL STACK

- Unikernels deploy directly against the hypervisor
- Unikernels have their own network stack
- Unikernels have their own virtualize memory presented as hardware
- Unikernel are completely self contained & ideally immutable

<table>
<thead>
<tr>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
UNIKERNEL STACK

Fewer layers,

less code,

much simpler!
DOCKER STACK VS. UNIKERNEL STACK

- Application Config
- Application
- Language Runtime
- Shared Libraries
- Docker Runtime
- OS User Processes
- OS Kernel
- Virtual HW Drivers
- Hypervisor
- Hardware Drivers
- Hardware

- Application Binary
- Library OS
  (Virt. HW Drivers + Language Runtime)
- Hypervisor
- Hardware Drivers
- Hardware
HOW CAN UNIKERNELS HELP ADDRESS OUR PROBLEMS?

Minimal layers of isolation and abstraction

Includes only what is **really** needed

Less code, fewer bugs, easy to reason about
UNIKERNEL ADVANTAGES

• No other users, no multi-user support

• No permission checks – you can utilize 100% of your hardware

• Isolation at the virtual hardware – only!

• Shared only hardware

• Minimal virtual machine ~1 gb in size, minimal unikernel is tiny, kb in size

• Very short boot time

• A tiny custom surface of attack, less likely to be effected by a public exploit
Backward compatibility

POSIX compliance

Forward compatibility

Language specifics
unik is an open-source tool written in Go for compiling applications into unikernels and deploying those unikernels across a variety of cloud providers, embedded devices (IoT), as well as a developer laptop or workstation.

```bash
Last login: Sun Feb 28 19:57:35 on ttys000
usxxlevinim3:~ leviin$ unik
NAME: unik

USAGE:
  unik [global options] command [command options] [arguments...]

VERSION:
  0.0.0

COMMANDS:
  delete, rm        delete running instances
  delete-unikernel, rmu delete compiled unikernel
  logs, l
  ps
  push, p
  run, r
  target, t
  unikernels, u
  list-volumes, lv
  create-volume, cv
  delete-volume, rmv
  attach-volume, av
  detach-volume, dv
  help, h

GLOBAL OPTIONS:
  --verbose, -V      stream logs from the unik backend
  --help, -h         show help
  --version, -v      print the version

usxxlevinim3:~ leviin$ 
```
UNIK WORKFLOW

unik daemon
unik build -v /my-volume /path-to-source my-unikernel
unik create-volume path-to-data my-volume
unik run -v my-volume:/my-volume -name my-instance my-unikernel
UNIK IS NOT OPINIONATED!

Unikernel types

Cloud providers

Processor architectures

© Copyright 2016 EMC Corporation. All rights reserved.
Docker API can be used to create unikernel via UniK
UNIK INTEGRATION WITH CLOUD FOUNDRY

To provide the user with a seamless PaaS experience, UniK is integrated as a backend to Cloud Foundry runtime.
THE FUTURE IS OURS TO CREATE ...
INTERNET OF THINGS

UniK will Push
Unikernel
To Raspberry Pi

Unikernel will
communicate with
the Panini toaster

We will eat Panini
bread

Toaster will make
Panini

WELCOME TO THE FUTURE!
Because the people who are crazy enough to think they can change the world are the ones who do.

R.I.P. Steve Jobs