64-bit ARM Unikernels on uKVM

ARM

Wei Chen <Wei.Chen@arm.com>
Senior Software Engineer

Tokyo / Open Source Summit Japan 2017
2017-05-31

Thanks to

- **Dan Williams, Martin Lucina, Anil Madhavapeddy and other Solo5 contributors who give me lots of helps in community.**
- **Shijie Huang and Dennis Chen who are co-working with me to implement ARM64 uKVM monitor and bring up guest.**
- **All my team mates at ARM.**

What are unikernels

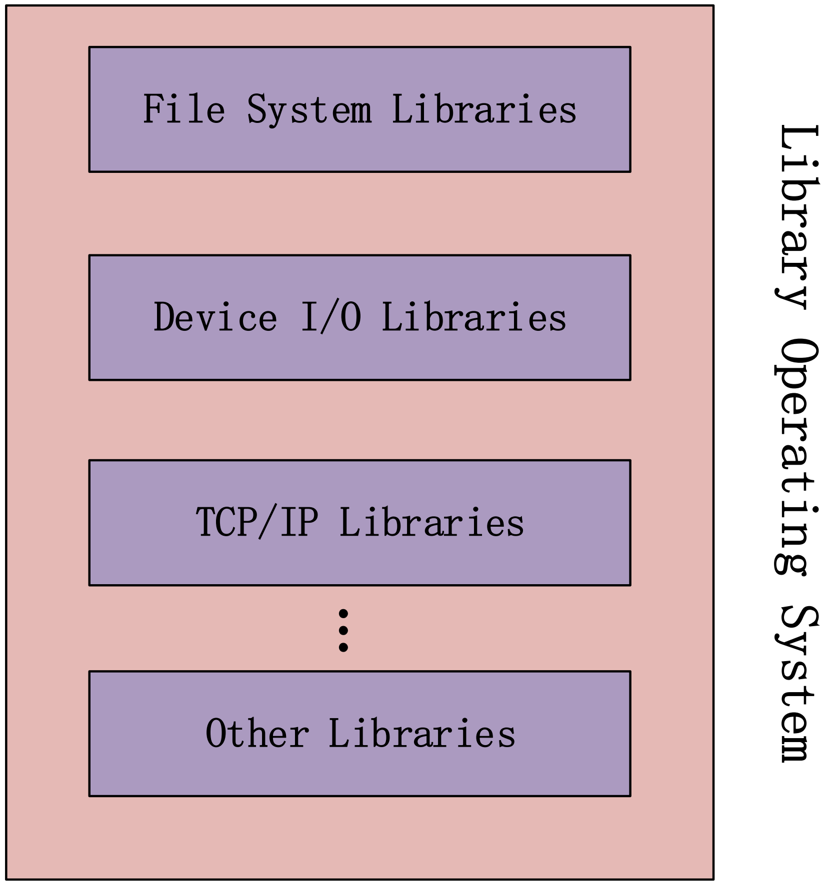
For a functional definition of a unikernel, let's turn to the burgeoning hub of the unikernel community, Unikernel.org, which defines it as follows:

Unikernels are specialized, single-address-space machine images constructed by using *library operating systems*.

In other words, unikernels are small, fast, secure machine images that lack distinction between application and operating systems.

Library operating system

- A special collection of libraries that provides needed operating system functions in a compliable format.
- Most unikernels use a specialized compiling system that compiles the low-level functions libraries into application directly.

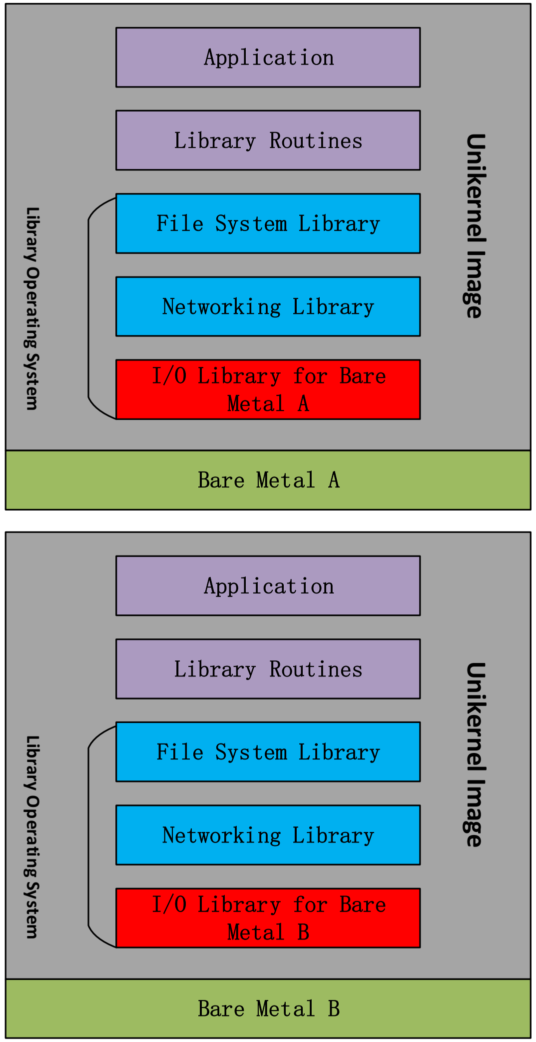


ARM

Unikernels run on bare metal

Unikernels can be designed to run on bare metal directly. But this architecture has two big drawbacks:

- Without a generic operating system, we have to do lots of jobs to support running multiple applications side by side with strong resource isolation on one bare metal.
- Different bare metals may have different devices. We have to rewrite device I/O libraries for these devices. This is a substantial task.

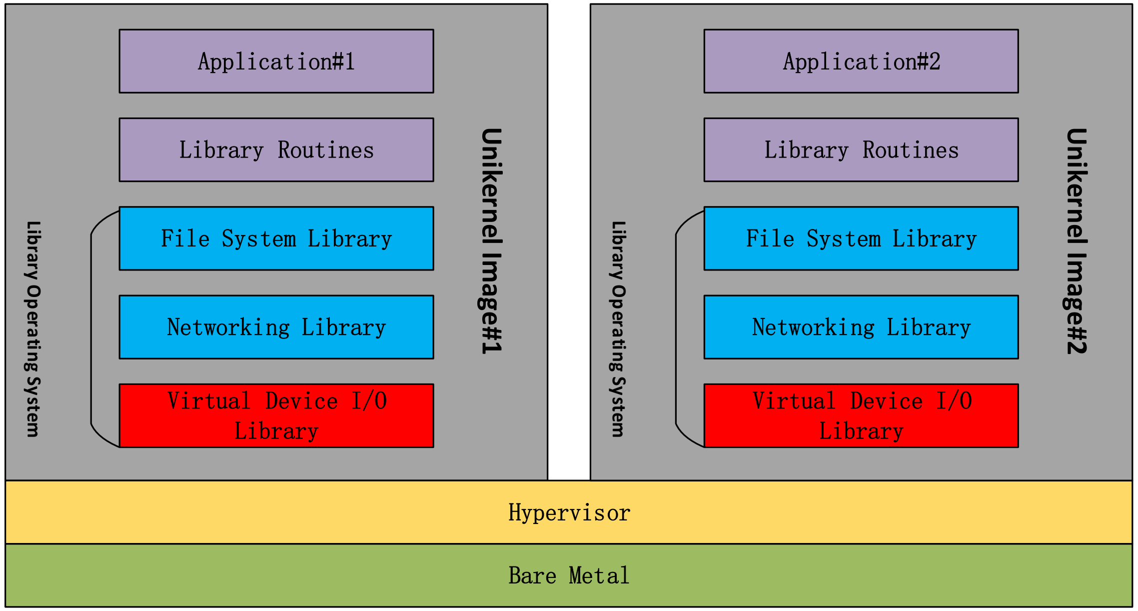


ARM

Unikernels run on hypervisors

Fortunately, modern hypervisors provide virtual machines with:

- **Consistent set of virtual devices**. So a library operating system just need to implement only drivers for these virtual devices.
- **Strong context isolation**. So the isolation between unikernels can be achieved by using hypervisor.



ARM

Why we need Unikernels?

Traditional workloads are large as they are comprised of many components. This can lead to a larger attack surface to exploit as well as a long startup/initialization times.

A unikernel approach allows one to reduce both the attack surface and service complexity

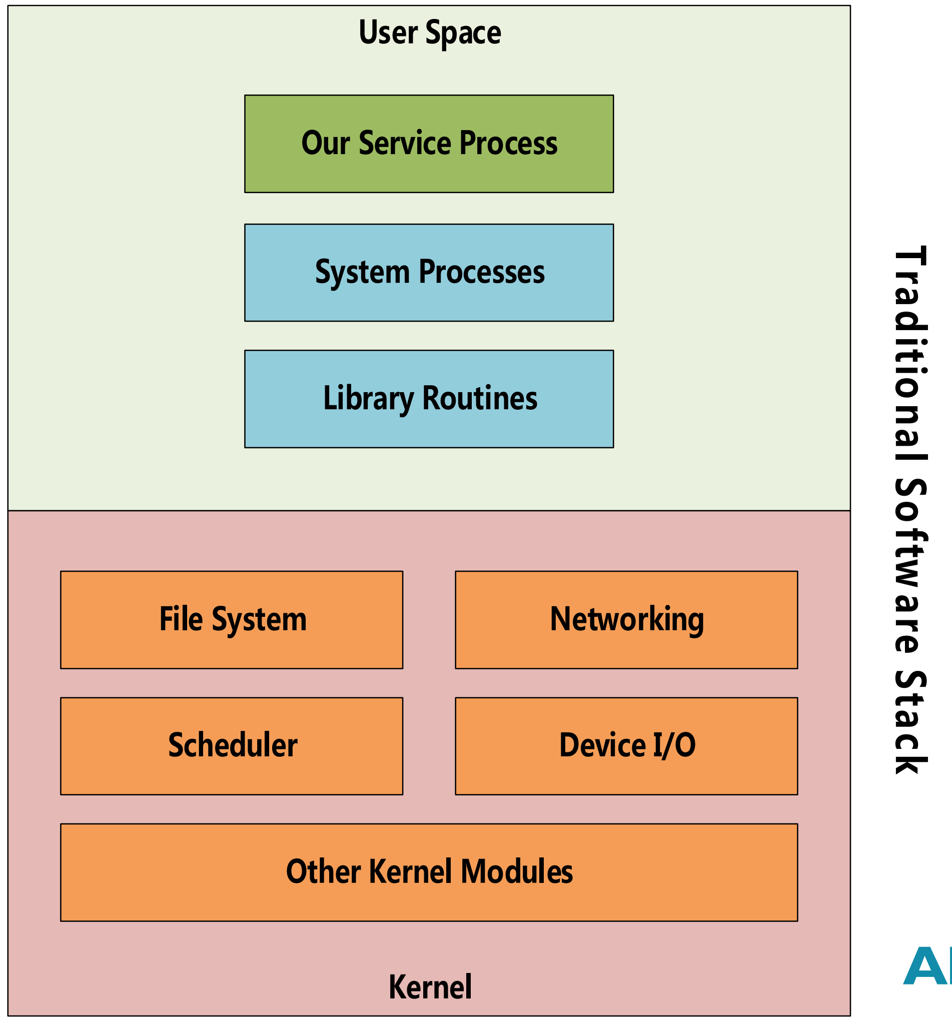
Traditional software stack

In last decade, we have done excellently in transfer every service into cloud. But the software stacks of workloads running on the cloud have remained almost unchanged since the time before cloud.

- Before Our Service Process, we have to startup all needed software before it.
  – Slower initialization.

- Even if it's a simplest service, we still have to spend disk and memory for unused software.
  –More resources used.

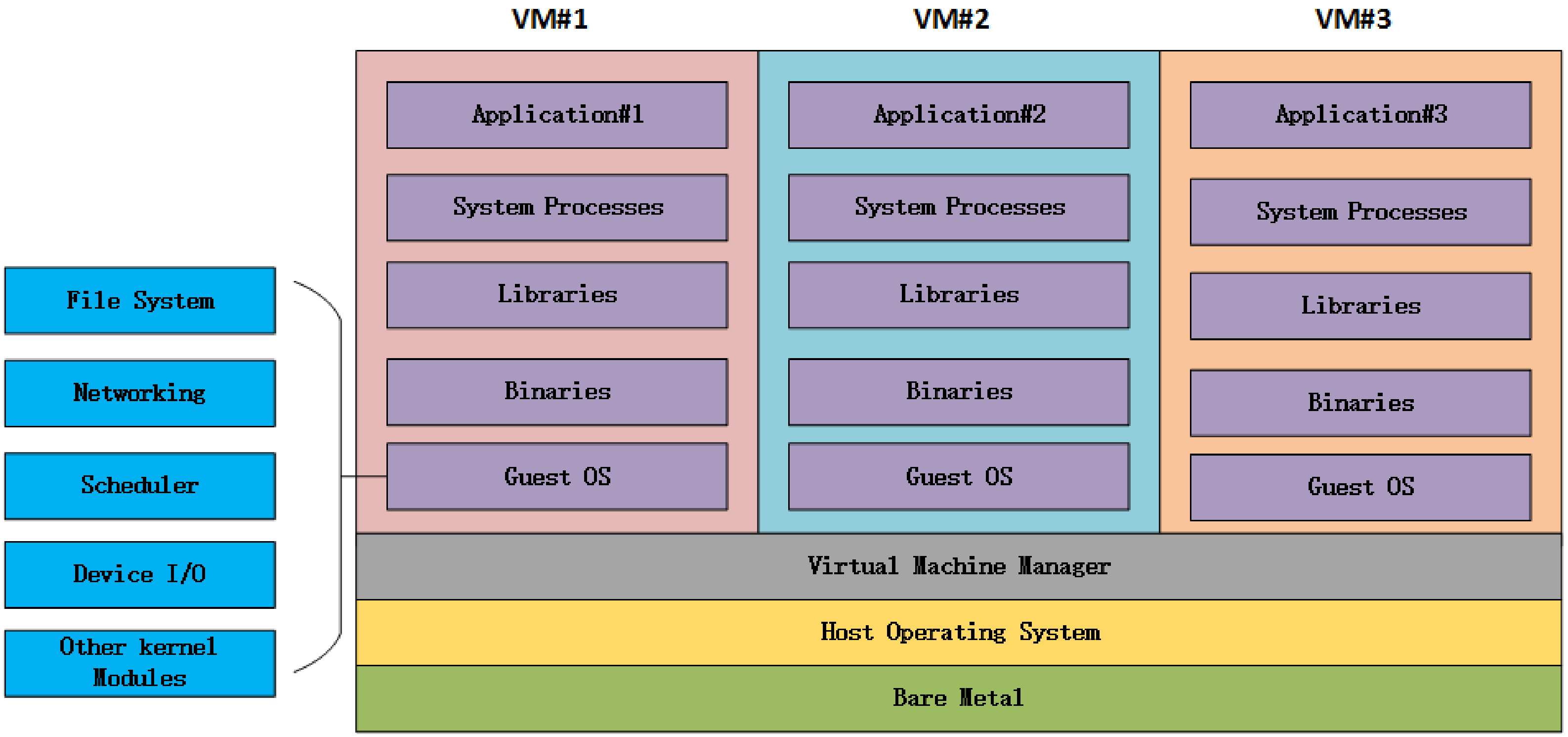
- Big size means big attack surface.
  –More opportunities to exploit.



ARM

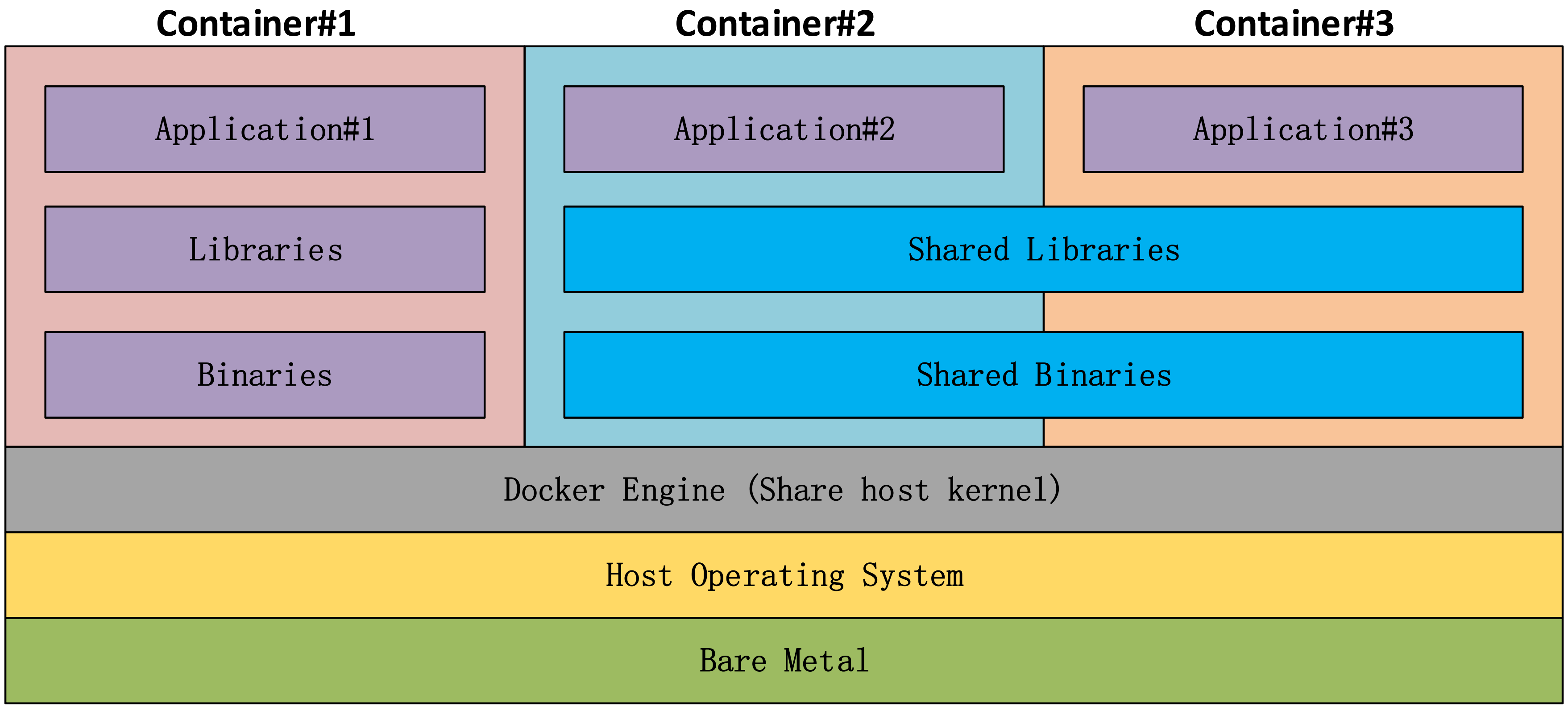
Workloads with Virtual Machine

While we move the workloads into the virtual machine to enjoy the great benefit of context isolation. We still haven't changed the software stacks.



- Every virtual machine image contains separate copies of kernel image, utilities and significant software.
  -- It wastes disk space.
- A virtual machine must boot a separate kernel and normally have a significant number of processes running to provide services. These processes may have already launched during the host system startup.
  -- It wastes CPU and memory resources.
- While starting up the virtual machine, the boot time is spent starting the kernel and support processes.
  -- This can take a long time for many virtual machines.
- Virtual Machines do not reduce the overall attack surface, instead they do a very good job of isolating attack surfaces from each other.

Can container help?



- Containers can share operating system kernel, binaries and libraries with their host system. Eliminating the need for additional copies of them in each container.

  -- Saves disk space.

- Containers can leverage the system processes of their host system. The duplicated processes are not needed to be launched.

  -- Saves memory and CPU resources.

- Relying on the host's kernel and existing system processes, startup of a container is extremely quick.

  -- Faster startup.

ARM

Security is still an issue

Containers are Smaller and Faster, but Security is still an issue.

In fact, unless we do works to make the container be secure before deployment. We may find the container is in a more vulnerable situation than when we were still using a virtual machine to deploy the service. Containers do not provide context isolation to the same extent as virtual machines. Because they share the same kernel, one vulnerable container may expose others to attack.

Container can protect the interfaces to the kernel by seccomp. But we have to know what containers will do. It's difficult for us to make sure what every container does, so it would not be a generic solution.

Are unikernels a better solution?

Unikernel.org lists 4 advantages of unikernels:

- **Improved security**

  Unikernels reduce the amount of code deployed, which reduces the attack surface, improving security.

- **Small footprints**

  Unikernel images are often orders of magnitude smaller than traditional OS deployments.

- **Highly optimized**

  The unikernel compilation model enables whole-system optimization across device drivers and application logic.

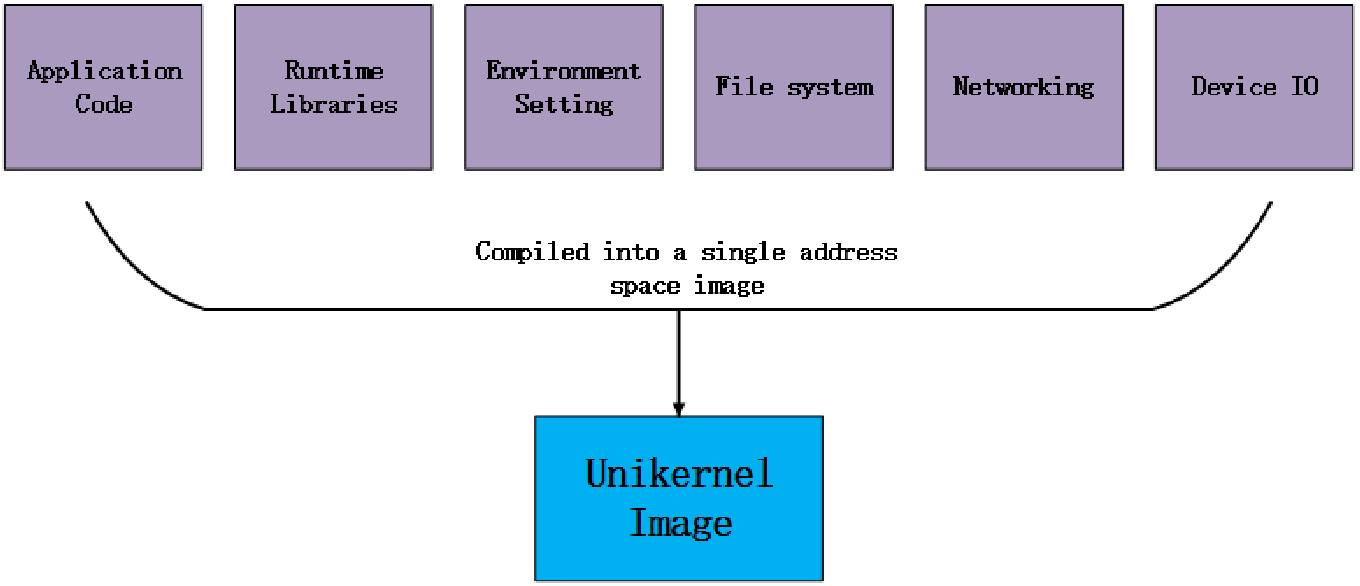
- **Fast Boot**

  Unikernels can boot extremely quickly, with boot times measured in milliseconds.

How can unikernels achieve this?

Compile everything into image:

Most unikernels compile everything needed into an application from library operating system. The result is that, the output unikernel image contains everything a program needed to run, from low-level device I/O functions to high-level logic code.



ARM

Normally, an application only needs a tiny fraction of functions on a generic operating system. One advantage that unikernels supply is the ability to only package what is needed. For Example, if we build a web server unikernel, we may only package:

- Basic architecture initialization functions (timer, console and network).
- TCP/IP stack and HTTP handlers

It requires no generic operating system, no shared libraries, and no system processes. The image size can be orders of a magnitude smaller than traditional web server on generic operating system.

-- *Small footprints, Reduce the attack surface, improving security.*

-- *Boots extremely quickly*

ARM

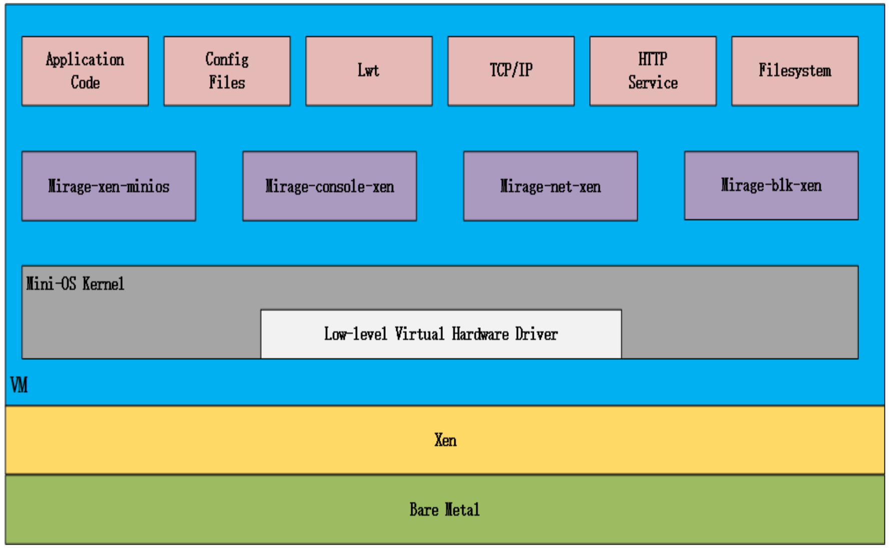
Now, we can see the unikernel satisfies our requirement of new type workload on cloud:

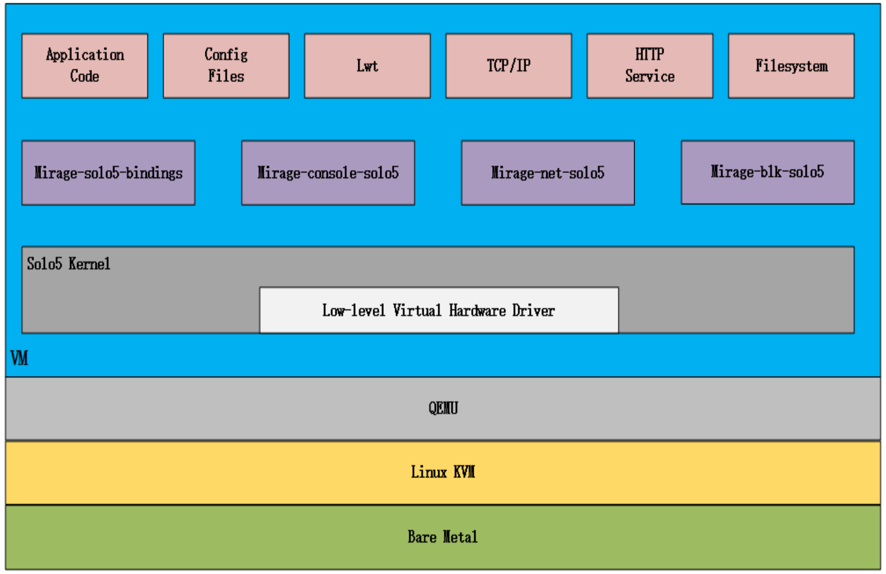
- **Fast**
- **Small**
- **Secure**

But, is it enough? Is there anything we can optimize?

Using MirageOS for example

- Currently, MirageOS unikernel images can run inside Xen and Linux KVM/QEMU hypervisors as a guest.





MirageOS run on generic hypervisors

- By the advantages of unikernels, application with mirage can package only needed functions into the image. So the application image can be very tiny. The application's attack surface has been reduced.
- From previous two samples, we see that two Mirage unikernel images are running on generic hypervisors.
- But these two Mirage unikernels maybe just need a tiny fraction of hypervisor interfaces or complex emulations. An unnecessary interface or emulation can be an additional attack surface.

ARM

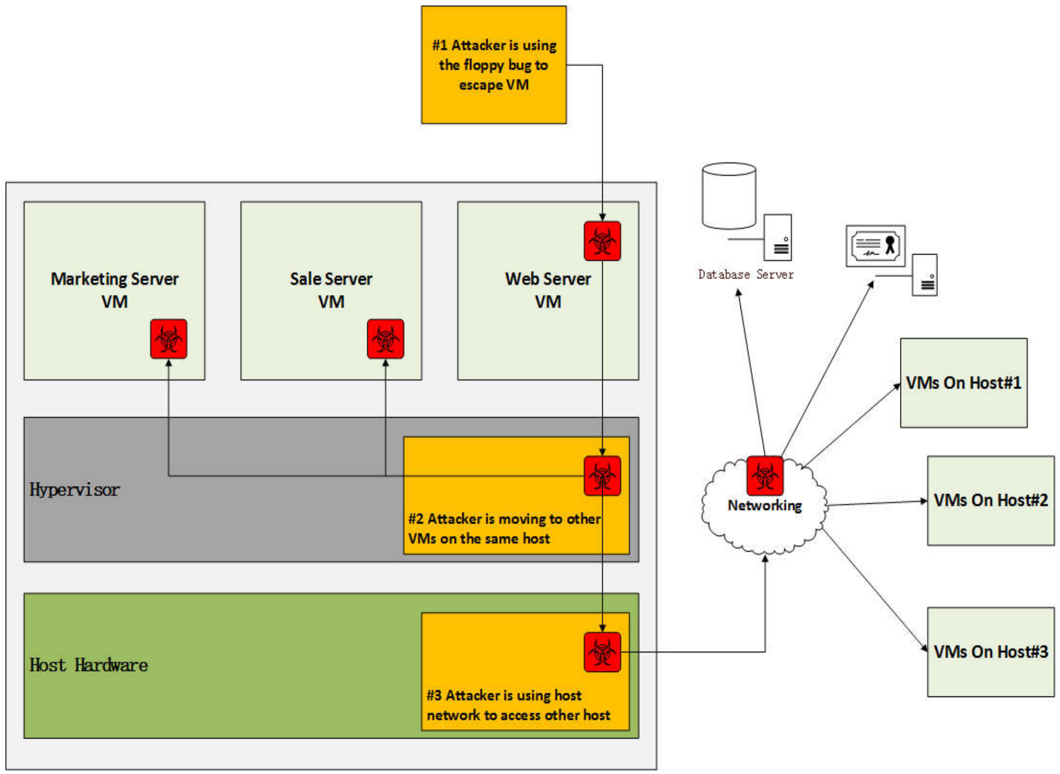
VENOM vulnerability

Origin:

A QEMU virtual device emulation that most virtual machines would not used contains a bug. – Virtual Floppy Device emulation.

Range:

Both the Xen Project and KVM open source hypervisors use QEMU, so all these virtual machines were potentially at risk.



How can we avoid attacks like VENOM?

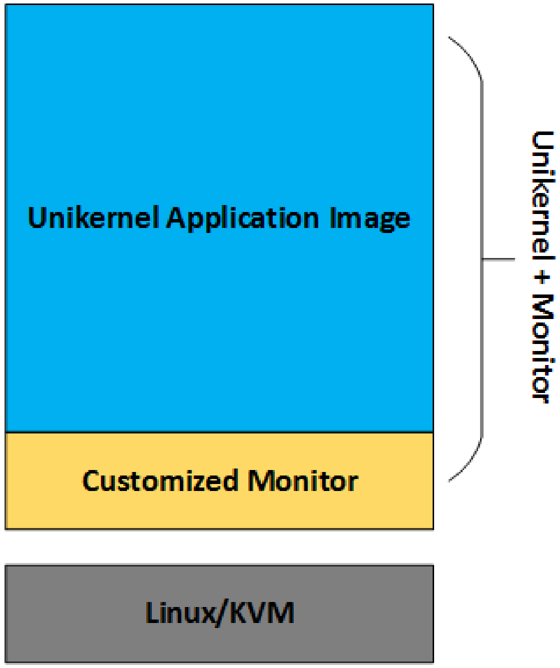
- In monitor layer, package what unikernel applications needed to the monitor. For example, if we want to run a "hello world" unikernel on VM, we could only package console emulation in to the monitor, without network, block and any other modules this application doesn't needed.

- Of course such specialized monitors need to be rigorously audited and security tested to ensure that they are not introducing their own security problems.

uKVM is a specialized unikernel monitor

- Customize and compile the unikernel monitor as application needed.

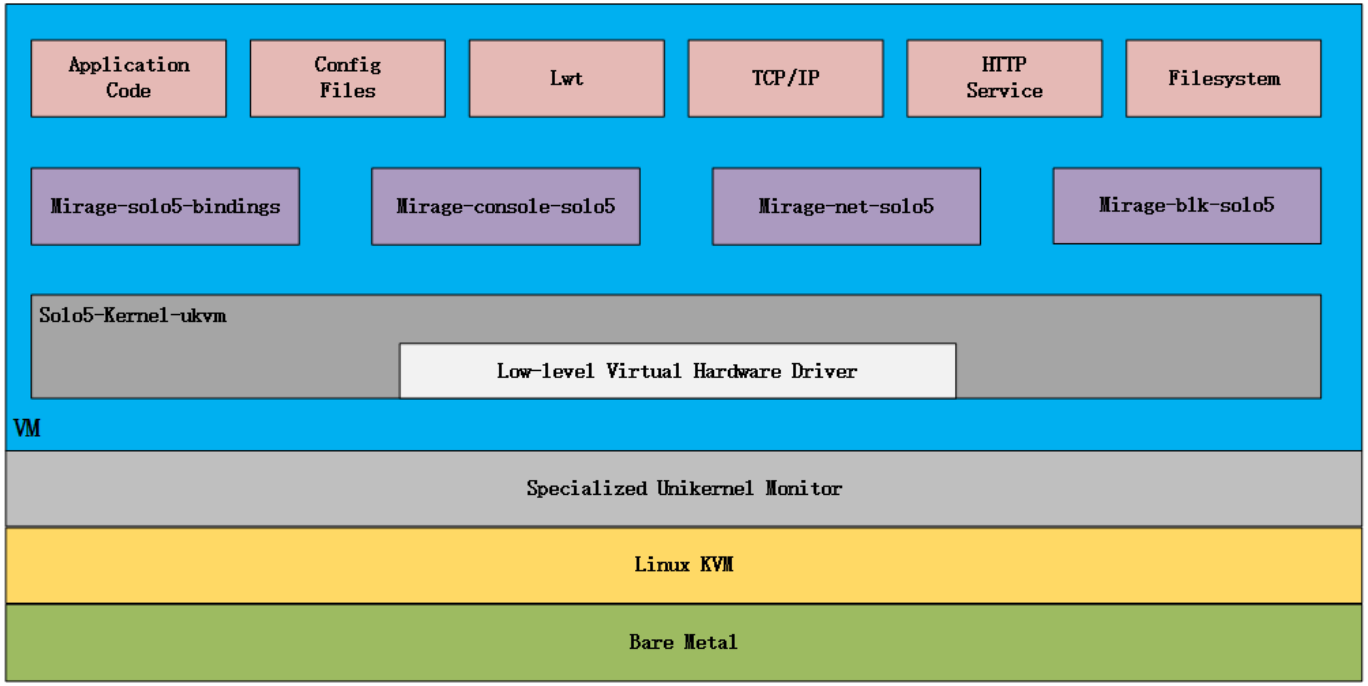
- Provide a VM with minimal set of hypervisor interfaces and emulations.
  - Reduce the VM footprint can help make things more secure
  - Reduce the VM virtual devices can help make monitor initialize faster.



ARM

The changes of the software stack:

- Replace QEMU by a specialized monitor for every unikernel.
- Add specialized monitor supports to library operating system low level functions.



uKVM on AAach64

- We have started to port uKVM on AArch64 at the beginning of this year.

Currently, we have the following working:

- Setup guest CPU
- Setup guest memory
- Setup guest timer
- Setup guest MMU

https://github.com/Weichen81/ukvm-solo5-arm64

And we are working with upstream to get support merged at:

https://github.com/Solo5/solo5

ARM

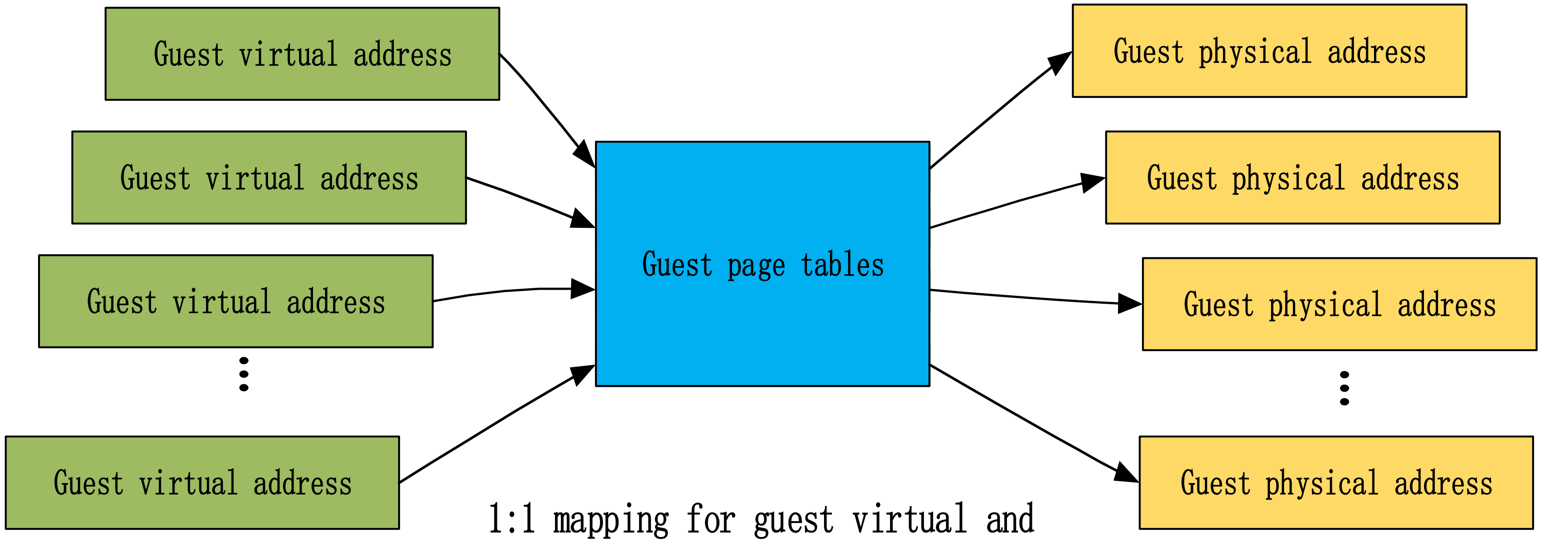
- The solo5 project wants to make the solo5 kernel architecture independent as much as possible. So if the work can be done by solo5 kernel or uKVM, we prefer to do it on uKVM side.

- For example:

Configure CPU vector table register in uKVM. Normally, this work is done by the guest kernel while running guest on the generic hypervisors .

Guest page tables on AArch64

- AArch64 needs to enable MMU for guest to share data for host. Hence the guest will use virtual address to access memory. But x86 guest use physical address. We don't want to make guest on AArch64 be special, so we create page tables for guest to do 1:1 mapping between virtual address and intermediate physical address.



1:1 mapping for guest virtual and physical addresses

ARM

Demo

Hardware Configuration:

- 8 Cortex-A53 2Ghz CPU
- 16 GB memory
- mirage-solo5-ukvm AArch64 Branch:

git checkout –b arm64 https://github.com/Weichen81/ukvm-solo5-arm64

- Testing based commit id:

9d1f576fb41886a7f533375e9d3be7494c3cd7e8

- This tests perform:

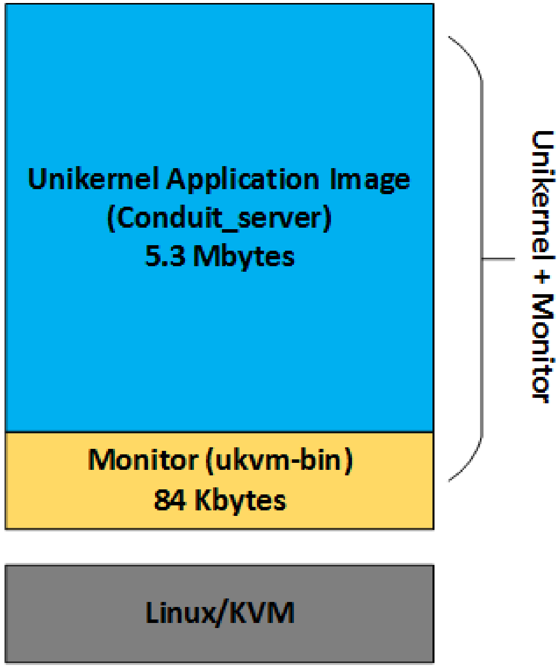
✓ Http server binary size, boot time and memory usage.

✓ How many http servers can run on this host at the same time.

Binary size

Unikernel Monitor:
ukvm-bin, 84Kbytes

Unikernel Application:
Conduit_server.ukvm, 5.3Mbytes



ARM

Boot time

Http Server boot time:

- *Launch to uKVM main entry: ~1ms*
- *uKVM main entry to conduit_server print “SOLO5”: ~50ms*

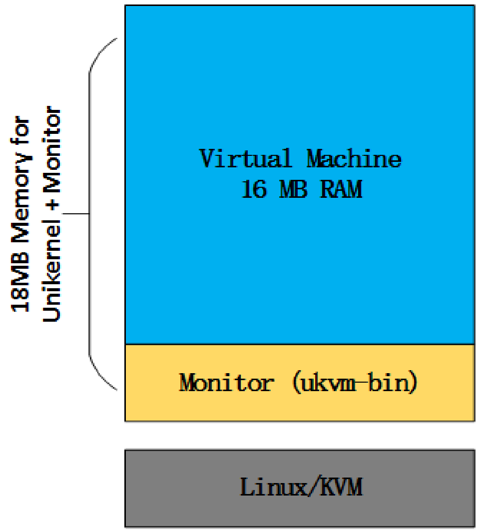
```
            |      __|
  __| _ \ |  _ \ __ \
\__ \ (   | | (   |  ) |
____/\___/ _|\___/____/
Solo5: Memory map: 16 MB addressable:
Solo5:     unused @ (0x0 - 0xfffff)
Solo5:       text @ (0x100000 - 0x306fff)
Solo5:     rodata @ (0x307000 - 0x35ffff)
Solo5:       data @ (0x360000 - 0x471fff)
Solo5:       heap >= 0x472000 < stack < 0x1000000
Solo5: new bindings
STUB: getenv() called
STUB: open() called
STUB: getpid() called
STUB: getppid() called
2017-05-16 09:12:03 -00:00: INF [netif] Plugging into 0 with mac a2:1e:82:57:38:65
2017-05-16 09:12:03 -00:00: INF [ethif] Connected Ethernet interface a2:1e:82:57:38:65
2017-05-16 09:12:03 -00:00: INF [arpv4] Connected arpv4 device on a2:1e:82:57:38:65
2017-05-16 09:12:03 -00:00: INF [udp] UDP interface connected on 10.0.0.2
2017-05-16 09:12:03 -00:00: INF [tcpip-stack-direct] stack assembled: mac=a2:1e:82:57:38:65,ip=10.0.0.2
```

Memory usage

- Http Server memory usage:

In uKVM configuration, we allocate 16MB RAM for VM to run http server.

We use "pmap" to capture the runtime memory of this http server.




```
23318:   ./ukvm-bin --net=tap0 conduit_server.ukvm
Address           Kbytes Mode  Offset           Device    Mapping
0000000000400000      16 r-x-- 0000000000000000 008:00007 ukvm-bin
0000000000413000       4 r---- 0000000000003000 008:00007 ukvm-bin
0000000000414000       4 rw--- 0000000000004000 008:00007 ukvm-bin
000000003ef64000     132 rw--- 0000000000000000 000:00000   [ anon ]
0000ffff8a10d000    1024 rw-s- 0000000000000000 000:00005 zero (deleted)
0000ffff8a20d000    2432 r-xs- 0000000000100000 000:00005 zero (deleted)
0000ffff8a46d000   12928 rw-s- 0000000000360000 000:00005 zero (deleted)
0000ffff8b10d000    1208 r-x-- 0000000000000000 008:00005 libc-2.23.so
0000ffff8b23b000      60 ----- 000000000012e000 008:00005 libc-2.23.so
0000ffff8b24a000      16 r---- 000000000012d000 008:00005 libc-2.23.so
0000ffff8b24e000       8 rw--- 0000000000131000 008:00005 libc-2.23.so
0000ffff8b250000      16 rw--- 0000000000000000 000:00000   [ anon ]
0000ffff8b254000     112 r-x-- 0000000000000000 008:00005 ld-2.23.so
0000ffff8b271000       8 rw--- 0000000000000000 000:00000   [ anon ]
0000ffff8b27a000       8 rw-s- 0000000000000000 000:0000a   [ anon ]
0000ffff8b27c000       8 rw--- 0000000000000000 000:00000   [ anon ]
0000ffff8b27e000       4 r---- 0000000000000000 000:00000   [ anon ]
0000ffff8b27f000       4 r-x-- 0000000000000000 000:00000   [ anon ]
0000ffff8b280000       4 r---- 000000000001c000 008:00005 ld-2.23.so
0000ffff8b281000       8 rw--- 000000000001d000 008:00005 ld-2.23.so
0000ffffc1150000     132 rw--- 0000000000000000 000:00000   [ stack ]
mapped: 18136K    writeable/private: 316K    shared: 16392K
```


ARM

Demo

I have run 256 Conduit Servers on this server at the same time.

```
weic     27175     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap223 conduit_server.ukvm
weic     27176     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap224 conduit_server.ukvm
weic     27177     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap225 conduit_server.ukvm
weic     27178     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap226 conduit_server.ukvm
weic     27179     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap227 conduit_server.ukvm
weic     27180     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap228 conduit_server.ukvm
weic     27181     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap229 conduit_server.ukvm
weic     27182     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap230 conduit_server.ukvm
weic     27183     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap231 conduit_server.ukvm
weic     27184     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap232 conduit_server.ukvm
weic     27185     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap233 conduit_server.ukvm
weic     27186     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap234 conduit_server.ukvm
weic     27187     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap235 conduit_server.ukvm
weic     27188     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap236 conduit_server.ukvm
weic     27189     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap237 conduit_server.ukvm
weic     27190     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap238 conduit_server.ukvm
weic     27191     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap239 conduit_server.ukvm
weic     27192     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap240 conduit_server.ukvm
weic     27193     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap241 conduit_server.ukvm
weic     27194     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap242 conduit_server.ukvm
weic     27195     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap243 conduit_server.ukvm
weic     27196     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap244 conduit_server.ukvm
weic     27197     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap245 conduit_server.ukvm
weic     27198     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap246 conduit_server.ukvm
weic     27199     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap247 conduit_server.ukvm
weic     27200     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap248 conduit_server.ukvm
weic     27201     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap249 conduit_server.ukvm
weic     27202     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap250 conduit_server.ukvm
weic     27203     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap251 conduit_server.ukvm
weic     27204     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap252 conduit_server.ukvm
weic     27205     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap253 conduit_server.ukvm
weic     27206     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap254 conduit_server.ukvm
weic     27207     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap255 conduit_server.ukvm
weic     27208     1  3 04:33 pts/3    00:00:00 ./ukvm-bin --net=tap256 conduit_server.ukvm
```


ARM

Demo

256 Conduit Servers:

- CPU usage: 100%

```
top - 04:35:28 up 18:42,  4 users,  load average: 226.31, 205.80, 232.43
Tasks: 453 total, 257 running, 196 sleeping,   0 stopped,   0 zombie
%Cpu0  : 99.2 us,  0.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 99.6 us,  0.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 99.6 us,  0.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 99.2 us,  0.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 99.2 us,  0.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16361072 total, 11208576 free,   220908 used,  4931588 buff/cache
KiB Swap: 19730428 total, 19730428 free,        0 used. 13928656 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26953 weic      20   0   18144   7840   7756 R   3.5  0.0   0:04.12 ukvm-bin
26956 weic      20   0   18144   8108   8024 R   3.5  0.0   0:04.08 ukvm-bin
26968 weic      20   0   18144   8184   8100 R   3.5  0.1   0:03.99 ukvm-bin
26974 weic      20   0   18144   8092   8008 R   3.5  0.0   0:03.97 ukvm-bin
26975 weic      20   0   18144   8020   7936 R   3.5  0.0   0:04.08 ukvm-bin
26983 weic      20   0   18144   8040   7956 R   3.5  0.0   0:03.96 ukvm-bin
26984 weic      20   0   18144   8124   8040 R   3.5  0.0   0:04.02 ukvm-bin
26988 weic      20   0   18144   8180   8096 R   3.5  0.0   0:03.82 ukvm-bin
26991 weic      20   0   18144   8176   8092 R   3.5  0.0   0:04.08 ukvm-bin
26994 weic      20   0   18144   8112   8028 R   3.5  0.0   0:04.09 ukvm-bin
26997 weic      20   0   18144   8180   8096 R   3.5  0.0   0:04.02 ukvm-bin
26998 weic      20   0   18144   8156   8072 R   3.5  0.0   0:03.97 ukvm-bin
27009 weic      20   0   18144   8040   7956 R   3.5  0.0   0:03.79 ukvm-bin
27013 weic      20   0   18144   8108   8024 R   3.5  0.0   0:04.08 ukvm-bin
27016 weic      20   0   18144   8108   8024 R   3.5  0.0   0:03.97 ukvm-bin
27040 weic      20   0   18144   8072   7988 R   3.5  0.0   0:04.08 ukvm-bin
27044 weic      20   0   18144   8020   7936 R   3.5  0.0   0:04.08 ukvm-bin
27051 weic      20   0   18144   8020   7940 R   3.5  0.0   0:04.02 ukvm-bin
27056 weic      20   0   18144   8156   8072 R   3.5  0.0   0:03.93 ukvm-bin
27083 weic      20   0   18144   8128   8044 R   3.5  0.0   0:03.78 ukvm-bin
27088 weic      20   0   18144   8040   7956 R   3.5  0.0   0:04.01 ukvm-bin
```


ARM

Demo

256 Conduit Servers:

- Memory usage:

	total	used	free	shared	buff/cache	available
Mem:	15G	214M	10G	1.7G	4.7G	13G
Swap:	18G	0B	18G			

ARM

Works still need to be done for AArch64

- Complete the upstream work.
- Add multi-platform supports, currently we only support Linux. If possible, we want to support other platforms like FreeBSD/MacOS.
- Add the VIRTIO support to increase the I/O performance.
- Verify and improve the compatibility of MirageOS libraries on AArch64.

Applications that are appropriate for unikernels

- Initialization needs to be quick.
- Application state does not need to be retained, one can express it as a transient micro-service.
- One wishes to minimize the execution footprint exposed to the internet.
- The application will scale out leading to many instances running in parallel.

Applications that are not suggested for Unikernels

- Multi-processes applications and could not be modified from inter-processes communication to inter-machines communication.

- Multi-user applications. Unikernels are fiercely single user. Multiple users require significant overhead.

- Applications that have lots of functions. Such applications will pull in large libraries, and will lost the advantages such as small footprint or faster boot time.

Running unikernels inside the container?

- As we had mentioned before, the share kernel strategy is the weakness of container security. Benefits by running unikernels inside the container:
- Virtual machine provides context isolation which is more secure than cgroup.
- A shared kernel will not be used any more.
- Breaking up system functionality to modular libraries, applications can package what they need.
- Multi-platform can use the same application image.

ARM

Summary

http://unikernel.org/

https://mirage.io/

https://www.linux-kvm.org/

https://www.xenproject.org/

https://github.com/Solo5/solo5

ARM

Questions?

ARM