Lightweight virtualization in the Cloud and at the Edge

Hypervisors gone rogue

Presentation: Anastassios Nanos, Babis Chalios
Research Team: Kostis Papazafeiropoulos, Charalampos Mainas, Orestis Lagkas-Nikolos, Stratos Psomadakis

Contact: info@nubificus.co.uk, @nubificus
Who we are

- Young SME doing research in virtualization systems
- ~7 people involved in various sub-projects:
  - Lightweight FaaS framework targeting low-power devices
  - Efficient storage solutions for microVMs / containers
  - Lightweight hypervisor development
  - Transparent application acceleration in the Cloud and at the Edge
- Based in Sheffield, UK / Athens, GR / Barcelona, ES
Overview

- Setting the scene
  - Microservices-based approaches
  - PaaS, SaaS, FaaS
  - Hybrid sandboxed environments with hypervisors and OS-level virtualization
Overview

- We examine the trade-offs of sandboxed vs fully virtualized deployments of micro-services, focusing on overall systems footprint of the management layers (runtime, hypervisor, I/O handlers).
- We present:
  - a minimal, lightweight Virtual Machine Monitor interfacing directly with KVM.
  - our port of Firecracker (AWS Lambda/Fargate hypervisor) to a Raspberry Pi 4
Setting the scene

- People move slowly to alternative methods of deploying applications:
  - Containers
  - Sandboxes
  - microVMs
  - Unikernels (?)

- The community introduces lower-overhead virtual machine monitors, suited to host the above approaches:
  - OS-level virtualization (containers/sandboxes)
  - Machine-level virtualization (solo5-hvt, firecracker)
Setting the scene

- What about the edge?
  - Small-factor devices, capable of running basic-to-medium sized applications are bloated using generic, conventional OS approaches (Linux variants, fully-blown operating systems) -- have you logged in to a Ubuntu 18.04 running on a RPi4?
  - Going one step further, these devices have a number of hardware accelerators, able to provide power-efficiency when doing compute-intensive tasks like Signal processing (Graphics/Sound), ML inference (Object/Pattern recognition) etc.
How VMs are running today

- Two popular open-source hypervisors:
  - Xen
    - Baremetal hypervisor & VMM, virtual machines running on top, PV drivers running on helper/privileged VMs
  - KVM
    - Linux Kernel patchset + user-space emulator (QEMU)
    - QEMU-lite, NEMU, custom-vmmss: crossvm, rust-vmm, firecracker
    - Lightweight examples: kvmtool (mostly KVM API ref), solo5
How VMs are running today

KVM:

- VMM running in user-space, hypervisor in kernel-space (mode switches)
- Linux device drivers available (and schedulers, and memory mgt optimizations, and...)

However:

- Increased attack surface

Xen:

- Hypervisor & VMM in the same context
- Faster spawn & execution times

However:

- Needs a helper domain for I/O
- No device drivers (need to use linux or something else)
Generic KVM
Generic Xen

Xen

root mode

VMM

Scheduler

Binary Load
Mem/PT setup
Doorbell
Exit Handler

non-root mode

VM

app runtime

PV drivers

MGT toolstack

Network Stack
Block Stack
Device Drivers

Privileged VM

Scheduler

VMenter

PV drivers
How VMs are running today

Running on any of those approaches in the cloud seems OK. What about low-power devices (the Edge)? Resources there are scarce, can we really afford unnecessary software stacks laying around just for the sake of being generic?

To explore options about virtualization at the edge, we tried running some of QEMU’s alternatives on a number of low-power devices:

- Raspberry Pi 4
- Khadas VIM3
- NVIDIA Jetson nano
Lightweight VMMs

We ran simple unikernels (redis / nginx / mirage uDNS) on:

- Solo5-hvt (former ukvm)
- AWS Firecracker

While Solo5 worked out of the box (no interrupt handling whatsoever ;-)), firecracker found it hard to cope with GICv2
Redis Benchmark

Redis benchmark

Requests per second

PING_INLINE, PING_BULK, SET, GET, INCR, LPUSH, RPOP, LPOP, RPOP, SADD, SPOP, LPush, LRANGE_100, LRANGE_300, LRANGE_500, LRANGE_600, MSET
The firecracker way of doing things

Firecracker Virtual Machine Monitor (VMM)

- Powers AWS Fargate & AWS Lambda
- Built on-top of KVM => tenant isolation
- Minimal device model => fast boot times / small attack surface
- Support x86, AMD, aarch64

Firecracker integrates with containerd
Firecracker

- Firecracker v0.19.0 had already support for aarch64 ISA
  - Only for GlCv3 though
  - RPi4 uses the older GlCv2 version
- Patchset for adding GlCv2 support:
  - Make the GlC handling of firecracker generic
  - KVM setup of GlCv2 devices
  - Create correct FDT for microVM
- Patch available at v0.20.0 version
Redis Benchmark
How VMs workloads are should be running today

What if we could take advantage of both worlds?

Following the generic approach, we need:

- a software interface to hardware virtualization extensions
- device drivers to access hardware (network/storage)
- a binary interface for workloads
- a piece of software to ensure isolation and handle VMenter/VMexit
How VMs workloads are should be running today

What if we could take advantage of both worlds?

Following the generic approach, we need:

- a software interface to hardware virtualization extensions → **KVM**
- device drivers to access hardware (network/storage) → **Linux Kernel drivers/**
- a binary interface for workloads execute → **solo5, microvm, even the Linux ABI**
- a piece of software to ensure isolation and handle VMenter/VMexit → **KVMM**
Kernel VMM

Regarding the name, we know... any suggestions? :P

- KVMM is essentially a virtual machine monitor running in the Kernel
- Handles VMenter/VMexits the same way as a user-space VMM
- Interfaces with the network and the block stack directly in the Kernel
Kernel VMM

- Binary Load
- Mem/PT setup
- Exit Handler
- PV drivers
- KVM
- Network Stack
- Block Stack
- Device Drivers
- KVM
- VM
- app
- runtime
- PV drivers
- VMenter
- root mode
- non-root mode
Kernel VMM

KVMM is WiP. Some of the features currently supported:

- x86_64 & aarch64
- the solo5 ABI:
  - solo5 apps / rumprun / mirageOS / unikraft unikernels
- OCI runtime [forked runnc from the Nabla containers team]

You can try it out on your Raspberry Pi:

```bash
# docker run --runtime=runcc cloudkernels/redis-kvm:aarch64
```
Redis Benchmark

Redis benchmark

- qemu
- firecracker
- hvt
- KVMM

Requests per second

PING_INLINE, PING_BULK, SET, GET, INCR, LPush, RPUSH, LPop, RPop, SADD, SPOP, LPush, LRange_100, LRange_300, LRange_500, LRange_600, MSET
Round-trip latency on x86_64

Round-trip latency (ping)

- qemu
- firecracker
- hvt
- KVMM
Round-trip ping latency on Khadas VIM3

- qemu: ~80 us
- fc: ~40 us
- solo5-hvt: ~40 us
- kvmm: ~40 us
Demo for mirageOS DNS
Summary & next steps

- We need to define the necessary requirements for running workloads at the edge.
- We propose KVMM, a minimal VMM in the Linux kernel
- Currently supports the Solo5 ABI (rumprun, mirageOS, unikraft)
- Pre-alpha release (mid Mar)

Next steps:

- Support more ABIs (qemu’s microvm)
- Extend container integration (layered storage)
- Support VM/vCPU pre-allocation / forking
Acknowledgements

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement no 871900 (5G-COMPLETE)
VHPC 2020 -- https://vhpc.org

- 15 Workshop on Virtualization in High-Performance Cloud computing
- Co-located with ISC-HPC 2020, Frankfurt, Germany, **June 21 - 25**, 2020

- Rolling Abstract registration
- Paper Submission Deadline: **Apr 05th**, 2020
- Springer LNCS Proceedings
- Sessions on Container orchestration, Lightweight virtualization, Hardware acceleration, Unikernels etc.
Thanks!

https://nubificus.co.uk
@nubificus
https://github.com/cloudkernels
https://blog.cloudkernels.net