Will Serverless End the Dominance of Linux in the Cloud?

Ricardo Koller and Dan Williams, IBM Research

Wosc2 2017
The beginning of the end of Linux

- “Throughout the history of computer science there has been a fairly constant opinion that current operating systems are inadequate”
  – Engler and Kaashoek 1995

- Why now?
The beginning of the end of Linux

 “[T]hroughout the history of computer science there has been a fairly constant opinion that current operating systems are inadequate”
  – Engler and Kaashoek 1995

 Why now?
  – Cloud unit of execution shrinking: serverless
  – Complexity of kernel continues to grow
Options and Roadmap

- **Overview**
- Serverless and its demands
- How Linux containers fail to meet these demands
- Rethinking the kernel
Serverless and its demands

- Isolation for multi-tenancy (excludes native processes)
- Performance
Serverless and its demands

- Isolation for multi-tenancy (excludes native processes)
- Performance
Serverless and its demands

- **Latency**
  - **User**: should launch immediately
  - **Provider**: should not require caching complexities

- **Throughput**
  - **Provider**: should at least cover hourly cost of server

**Target**: 100 ms

**Target**: 125 actions/sec
What about caching of warmed containers?

Cache hit ratio

100%

0%

Number of Standby warm containers

100

1000
What about caching of warmed containers?

Cache hit ratio

100%

0%

100 users running python 2.7 with x,y,z packages

Number of Standby warm containers

100

1000
What about caching of warmed containers?

![Graph showing cache hit ratio against number of standby warm containers]

- **Cache hit ratio**: 100%
- **Number of standby warm containers**: 100, 1000
Options Revisited

Fix Linux

Bypass Linux

Replace Linux

Native Serverless OS
Options Revisited

Containers, e.g., runc

libOS approach, e.g., ukvm, unikernels

Replace Linux

Native Serverless OS
Experimental setup

- core1
- core2
- core3
- core16

- regular host process
- runc container
- ukvm unikernel

Linux
Latency

![Graph showing latency for gzip and HTTP GET with different invokers (runc, ukvm, host)]
Throughput

Throughput for gzip:
- Host: 1465 actions per second
- UKVM: 661 actions per second
- Runc: 41 actions per second

Throughput for HTTP GET:
- Host: 807 actions per second
- UKVM: 102 actions per second
- Runc: 6 actions per second
A tradeoff

Impediments due to lack of familiar/useful abstractions

Impediments due to complexity

Native Serverless OS
Can we fix containers?

\[
\text{Cost} = C \times \text{Lines}^{1.3}
\]

If we choose to bypass Linux…

- Linux is just used for setup – “the OS is the control plane”
- We would prefer that the kernel **stop trying to adapt** to cloud workloads
- We should choose to **design** for bypass
If we choose to replace Linux...

- No preemptible scheduling
  - actions are short and run-to-completion

- No process synchronization

- No IPC (inter-process communication)

- Limited set of I/O calls: literally input and output
  - files are unnecessary abstractions
Conclusions?

- Native abstractions in Linux (containers) are not suitable for serverless at this time
- Can they be fixed?
- Should we bypass Linux?
  - Then we should design for bypass
- Is it time to replace Linux? Can that even happen?
Cold starts