Amazon Firecracker

Yiying Zhang
Firecracker
Lightweight Virtualization for Serverless Applications

Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Diana-Maria Popa, and Phil Piwonka

February 2020
Why Firecracker?
EC2 m5.metal instance
384GB of RAM

Smallest Lambda Function
128MB of RAM
AWS Lambda before Firecracker

- Linux containers on VM
  - One container per function
  - One VM per customer (per machine)
- Containers: trading off security and compatibility
- VMs: difficulties of efficiently packing workloads
Requirements for Amazon

• **Isolation**: It must be safe for multiple functions to run on the same hardware

• **Overhead & Density**: Thousands of functions on a single machine

• **Performance**: Functions must perform similarly to running natively

• **Compatibility**: Arbitrary Linux binaries and libraries. No code changes or recompilation

• **Soft Allocation**: It must be possible to over commit CPU, memory, and other resources
What about Existing Virtualization Solutions?

- QEMU/KVM: density and overhead challenges
- Linux *containers*: isolation and compatibility challenges
- LibOS approaches: compatibility challenges
- Language VM isolation: compatibility and isolation challenges
Firecracker is an open source VMM that is purpose-built for creating and managing secure, multi-tenant container and function-based services.
Firecracker Overview

- A VMM that uses KVM to provide minimal VMs (MicroVMs)
- Supports Linux and OSv guest OS
- Rely on Linux for many functionalities
  - Saving implementation efforts
  - Fits what Amazon’s operators are familiar with
- Started with a branch of Google Chrom’s crosvm
  - Removed >50% code (drivers, etc.) and added more
- Written in Rust, open source
Firecracker Design Details

- Device model
  - Limited emulated devices, virtio for network and block devices
- REST API for configure, manage, start, and stop MicroVMs
- Rate limiters in block and network devices
- Security: apply defenses of hardware-based attacks when deploying
- Jailer: a wrapper around Firecracker to sandbox it

let MicroVMs use guest OS’s file system (better security)
Firecracker + Lambda

Figure 2: High-level architecture of AWS Lambda event path, showing control path (light lines) and data path (heavy lines)

Figure 3: Architecture of the Lambda worker

Figure 4: State transitions for a single slots on a Lambda worker
Operational Lessons
• In production in AWS Lambda
  • Millions of workloads
  • Trillions of requests/month
Lesson #1: Compatibility is Hard

Just disabling Hyperthreading revealed two bugs in Apache Commons HTTP Client, and one in our own code.

Re-implementing OS components would have been worse.

Performance compatibility too!
Lesson #2: Immutable, Time-Limited Machines

Common systems-administration tools like `rpm` and `dpkg` are non-deterministic.

Limiting max fleet life helps operational hygiene.
Lesson #3: The Job is Never Done

Changing customer needs means that there are always improvements to be made.
Performance
MicroVM start latency (serial)
MicroVM start latency (50 parallel)
QD1 IO Latency vs Bare Metal

![Graph showing latency comparison between metal, FC, Cloud HV, and QEMU for 4k read, 4k write, 128k read, and 128k write operations.](https://example.com/graph.png)
QD32 IO Throughput vs Bare Metal

Bandwidth (MB/s)

- metal
- FC
- Cloud HV
- QEMU

4k read 4k write 128k read 128k write

© 2020, Amazon Web Services, Inc. or its Affiliates.
The Future
Opportunities

IO performance and scalability (offload)
Scheduling, especially for tail latency
Formal correctness proofs

Features like snapshots, ballooning, etc.

rust-vmm, and the container community.