Firecracker

Slides by Prof. Yijing Zhang,
With edits from Anze Xie and Chirag Dasannacharya
Announcements

- Project progress report extended to 11/9 (Wed)
- Last 20 min today switched to Joe Hellerstein’s talk
- Quiz 2 at the end of 11/9 lecture
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?
Use Case - Serverless Applications

Serverless loads are -

- Often tiny
- Need quick startup

While still requiring -

- Security guarantees
- Performance and isolation
Use Case - Serverless Applications

Serverless loads do NOT need -

- Support for different kernels (AWS uses Linux)
- Instruction emulation
- Support for legacy components
- VM migration (AWS Lambda - 15 minute execution limit)
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:**
  - Helps provide multi-tenancy.
  - It must be safe for multiple functions to run on the same hardware

- **Overhead & Density:**
  - Thousands of functions on a single machine.
  - Firecracker overhead - 5MB per Lambda

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

- **Compatibility:**
  - Arbitrary Linux binaries and libraries. No code changes or reccompilation

- **Fast Switching:**
  - Start quickly, clean up quickly

- **Soft Allocation:**
  - It must be possible to over-commit CPU, memory, and other resources
Existing Virtualization Solutions - Linux Containers

- Cgroups to provide isolation
- Seccomp-bpf to limit syscalls
- Chroot to control filesystem access

Problems:

Compatibility issues (seccomp-bpf)
Side channel attacks and isolation (rich interfaces like /proc)
Existing Virtualization Solutions - Language based Isolation

Use JVM, with multiple Lambdas sharing a process (Isolation and compatibility issues)

Use one process per Lambda, where each Lambda written in a safe language or runtime (eg Chromium websites) - Compatibility issues.
Existing Virtualization Solutions - Current hypervisors/VMs

- Slow startup (can speed up with unikernels)
- Unnecessarily feature rich with high overhead (QEMU on KVM)

Unikernels - Compatibility issues
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.
Origins

- Started with a branch of Chrome OS’s open source crosvm
  - crosvm works on KVM, used to protect Chrome OS apps
  - Removed >50% code (drivers, etc.) and added more
- Written in Rust - strict types, strong memory protections, good performance
Firecracker Overview

- A VMM that uses KVM (hardware virtualization) and virtIO (paravirtualization) to provide minimal VMs (MicroVMs)

- Supports Linux and OSv guest OS

- Rely on Linux whenever possible
  - Saving implementation efforts
  - Fits what Amazon’s operators are familiar with
KVM

KVM is -

- A Linux kernel module, that
- Uses hardware virtualization (Intel VT-x, AMD-V) to
- Turn the host kernel into a hypervisor.
- Guests are run in non-root mode with VT-x enabled
- Guest VMs appear to the host as processes (can use commands like top)
- Uses host Linux kernel functionality as VMM functionality (CPU scheduling, memory management, etc)
- Fork of QEMU (but does not provide instruction emulation, etc)
Firecracker - Architecture

Firecracker scales to thousands of multitenant microVMs.

Configurable microVMs across CPU and memory, running as user space processes.
Security

Firecracker has two layers of security -

- Between firecracker or the host kernel and the microVM (virtualization)
- Between the host and firecracker-microVM combination (‘jailer’)
  - Uses cgroup, chroot, seccomp-bpf, with 24 allowed calls, pid and network namespaces, etc

Question: Why do we need the Jailer as an extra layer of security?
Firecracker Functions

- 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
Lambda is a compute service which runs functions in response to events.

Typical use-cases:

- Backends for mobile and web applications
- Request-response and event sourced microservices
- Real-time streaming data processing

E.g., Lambda as a backend for web applications
High-Level Architecture

Frontend

- Receives invoke traffic from *Invoke REST API*
- Authenticates requests and checks for authorization
- Loads function metadata

Worker Manager

- High-volume (millions of requests per second) low-latency (<10ms) stateful router
- Sticky routing
- Concurrency control protocol with workers

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

Worker fleet

- Slots: pre-loaded execution environments for each function

Placement Service

- Is called when no available slot for a function
- Optimizes placements of slots for a single function
- Time-based lease protocol

Question: What benefit is brought by the time-based lease protocol?
The Lambda Worker

● Each worker runs 100s or 1000s of MicroVMs
● One Firecracker process launched per MicroVM
  ○ Creates and manages the MicroVM
  ○ Provides device emulation
  ○ Handles VM exits

Figure 3: Architecture of the Lambda worker
The Lambda Worker

Micro Manager

- One per worker
- Communicates with shim process via TCP/IP socket
- Responsible for
  - Managing the Firecracker processes
  - Event invoke API to the Frontend
  - Slot management and lock API to placement
  - Return payload or error details to Frontend on completion

Question: Is there any benefits of communicating into and out of the MicroVM over TCP/IP?
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
Evaluation Setup

- EC2 m5d.metal instance
- 2 Intel Xeon Platinum 8175M processors (48 cores)
- 384 GB of RAM
- 4 * 840 GB local NVMe disks
MicroVM start latency (serial)
MicroVM start latency (50 parallel)
QD32 IO Throughput vs Bare Metal

<table>
<thead>
<tr>
<th></th>
<th>4k read</th>
<th>4k write</th>
<th>128k read</th>
<th>128k write</th>
</tr>
</thead>
<tbody>
<tr>
<td>metal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud HV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bandwidth (MB/s)

© 2020, Amazon Web Services, Inc. or its Affiliates.
QD1 IO Latency vs Bare Metal
Discussion

● As many of the recent virtualization systems are moving away from traditional, full-scale VMs, do you still think VMs can have a good market? Is it still worthwhile spending efforts into improving full-scale VMs?

● Unikernel is from academia, and Firecracker is from the industry. What do you think are some of the different focuses?

● In terms of virtualization, do you think the industry is leading the academia or the other way round? Which way do you think it should be? Give some examples.
Firecracker + Lambda

- “Sticky” routing to few workers
- Each worker runs 100s to 1000s of slots (MicroVMs)
- If no available slot, executes “Placement” service
- Shim process in MicroVM communicates with MicroManager

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