A Technical Review of AWS Firecracker

Index

[I. Introduction 2](#_Toc57672645)

[II. Overview 2](#_Toc57672646)

[III. Open-Source Community 4](#_Toc57672647)

[IV. Firecracker vs QEMU 5](#_Toc57672648)

[V. Bibliography 6](#_Toc57672649)

# Introduction

The microservice architecture is an architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable, organised around business capabilities, and owned by a small team.

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organisation to evolve its technology stack. A service is said to be scalable if, when we increase the resources in a system, it results in increased performance in a manner proportional to resources added. Increasing performance in general means serving more units of work, but it can also be to handle larger units of work, such as when datasets grow.

Firecracker is a lightweight virtualisation solution for serverless applications developed by engineers at Amazon Web Services (AWS). Written in Rust, it is an open-source virtual machine monitor (VMM), that was purpose-built for the creation and management of secure, multi-tenant container and function-based services. It is built on top of the Linux kernel virtual machine (KVM), a feature in the Linux kernel that allows the building of virtual machines using a standard application programming interface (API). According to the project’s GitHub page [2], Firecracker currently supports Intel CPUs, with AMD and Arm support in developer preview.

Distributed computing studies the models, architectures, and algorithms used for building and managing distributed systems. A distributed system is a collection of independent computers that appears to its users as a single coherent system. Since distributed systems are composed of more than one computer that collaborate together, it is necessary to provide some sort of data and information exchange between them, which generally occurs through the network.

One of the main advantages of using distributed computing is that efficient scalable programs can be designed so that independent processes are scheduled on different nodes, and they communicate only occasionally to exchange results. In comparison, supercomputers consist of thousands of processors that are housed in a rack, each of which communicate through shared memory, with multiple simultaneous accesses to a common memory.

# Overview

At the base of Firecracker is AWS’s concept of microVMs: lightweight virtual machines that enable the deployment of vast numbers of workloads. These microVMs prove even more efficient than traditional virtual machines (VMs) for many use cases, providing enhanced security and workload isolation. At the same time, they retain the speed and resource efficiency of containers.

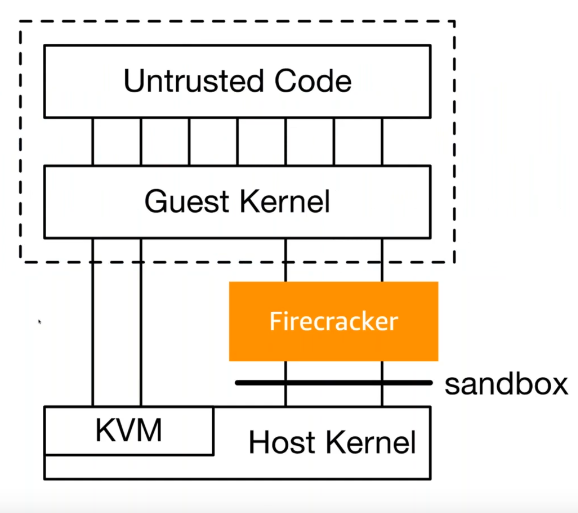


Figure 1 – Showing location of Firecracker [5]

Firecracker itself is a VMM, used to create and manage microVMs. It sits next to the KVM, running atop the host kernel. It takes care of the setup of the KVM, instructing the kernel to setup the VMs as needed. Unlike tradition VMs, in order to reduce required resources as much as possible, Firecracker excludes unnecessary devices and superfluous guest functionality for each microVM. This has the added benefits of improving security, by removing unwanted features, as well as decreasing the VM start-up times, and increasing hardware utilisation. It also handles the I/O virtualisation, such as the block I/O, and networking. Firecracker’s design uses a simplified device model. It does not have a BIOS, instead it boots from the Linux kernel directly. It does not have any PCI, instead the kernel talks to these protocols directly. This greatly simplifies both the guest kernel, as well as the VMM. The kernel handles the heavy lifting of virtual on platform, the hardware details, the page tables, as well as some other more low-level details.

Firecracker is currently in production in AWS Lambda, Amazon’s serverless function service, wherein it handles millions of concurrent workloads, meaning trillions of requests each month. One of the primary objectives for AWS engineers in the conception and development of Firecracker was economics. The minimum requirements of an AWS Lambda function is a meek 128MB of RAM, compared to Amazon’s Elastic Compute Cloud (EC2)’s 384GB of RAM, an immensely significant difference. As an aside, even Raspberry Pi’s have more than 128MB of RAM. The developers claim that as many as 4,000 small functions can be loaded up, running at the same time, in thanks to its small overhead and required resources. The Firecracker process is statically linked, which means all the libraries it needs to run are included in its executable code. This makes new Firecracker environments safer by eliminating outside libraries.

Firecracker is capable of running thousands of functions on a single machine, with acceptable overheads, and acceptable densities. This was what inspired the Amazon engineers to work on customising the special needs of virtualisation workloads. This solution has the perks of virtualisation’s isolation, along with the overheads and densities of container technology. A stipulation set by the engineers, the microVMs provide enhanced security over the traditional VMs. From a security perspective, it is safe for multiple functions to be running on the same hardware. Workloads from different end customers are isolated, yet can run safely on the same machine.

Firecracker allows for soft allocation. This means that it is possible to over-commit CPU, memory, and other computing recourse when establishing microVMs needed for a particular task. This feature facilitates the ability to run a vast amount of workloads on a box that didn’t commit the worst-case amount of computing resources. Instead, for economics, the engineers wanted to take the actual amount of CPU and memory that each workload was using at a given moment, and consume that from the machine. Instead of allocating a specific amount of memory, you can get memory as needed. In the same vein, instead of allocating a single core, you can use extra cores as needed.

Firecracker offers its users many compatibility benefits. It allows customers to bring arbitrary Linux binaries and libraries into AWS Lambda and Firecracker, meaning customers can use their existing code bases, code that perhaps they have been building for years, even decades, that the developers are familiar with and understand, without needing to make changes, and in some cases, without needing to recompile it.

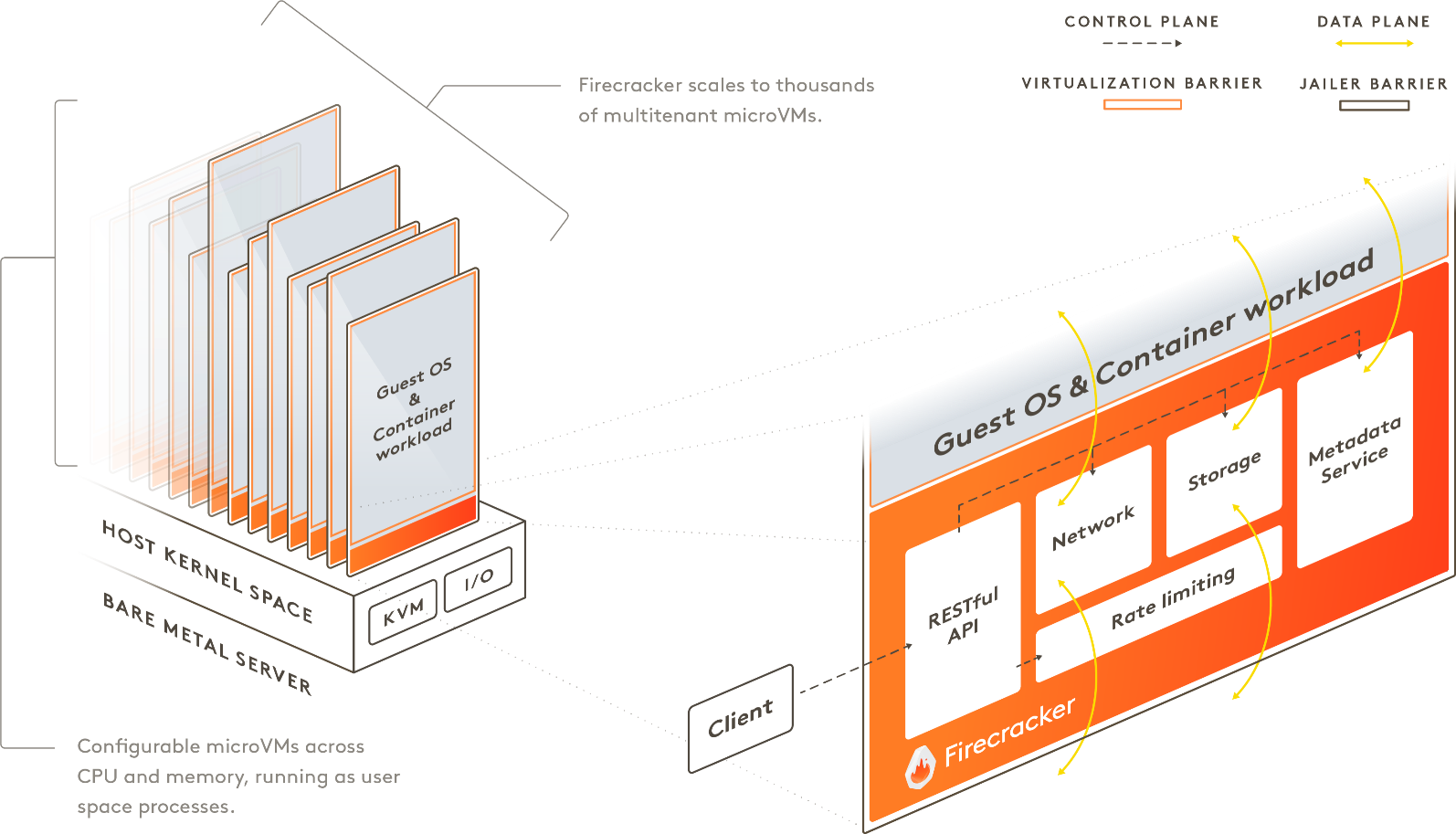


Figure 2 – An example host running Firecracker microVMs [2]

# Open-Source Community

Firecracker is open-source, published on GitHub, and ready for contribution, changes to be reviewed by Amazon before implementation. AWS engineers worked with the open-source community to integrate Firecracker with the current container system. One such integration is with Kubernetes, called Firekube. Firecracker allows you to replace the standard Linux container type isolation mechanism with a stronger isolation mechanism provided by virtualisation. As the platform is open-source, the codebase is available on GitHub under the Apache 2.0 license.

One such open-source distribution that uses Firekube is Weave Firekube. As described here [15], it enables secure clouds anywhere. Firekube uses Weave Ignite to run Kubernetes on Firecracker by default. As such it is a portable and secure alternative to Project Pacific in that Kubernetes is integrated with a VM stack. The whole stack is managed using GitOps which simplifies correct installation and management. Firekube may also be seen as an alternative to KIND using Ignite and GitOps. Firekube pulls everything from Git, detects your operating system and can boot up a secure cluster from nothing in 2.5 minutes.

# Firecracker vs QEMU

As understood by the AWS engineers involved in the Firecracker project, customers of AWS don’t want to lose performance when using a serverless solution. They want the serverless solution to perform just as well as their container solutions or legacy service were doing. Firecracker uses the basis of serverless computing, and gives the ability to create a microservice as secure microVM. It was specially designed for lightweight cloud workloads.

Prior to Firecracker, the alternative was to use QEMU with KVM. This technology was known to experience density and overhead challenges. QEMU (Quick EMUlator), is a generic, open-source hardware emulator and virtualisation suite. It is frequently used in conjunction with acceleration in the form of a Type-I hypervisor, such as a KVM. If no accelerator is used, QEMU will run entirely in user-space using its built-in binary translator TCG (Tiny Code Generator). Using QEMU without an accelerator is relatively inefficient and slow. In comparison with QEMU, Firecracker boasts 96% fewer lines than QEMU, equating to Firecracker coming up as a mere 4% the size of QEMU.

# Bibliography

1. [nsdi20-paper-agache.pdf](https://cit.instructure.com/courses/49160/files/855157/download?wrap=1)
2. <https://firecracker-microvm.github.io/>
3. <https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/>
4. <https://www.techrepublic.com/article/aws-firecracker-10-things-every-tech-pro-should-know/>
5. <https://www.youtube.com/watch?v=BIRv2FnHJAg>
6. <https://www.youtube.com/watch?v=PAEMGa-i2lU>
7. <https://www.youtube.com/watch?v=jUeQAQkQpXU>
8. <https://www.youtube.com/watch?v=DrrvPqX_Qr8>
9. <https://www.sciencedirect.com/topics/computer-science/distributed-computing-systems>
10. <https://blog.stackpath.com/distributed-system/>
11. <https://microservices.io/>
12. <http://highscalability.com/amazon-architecture>
13. <https://wiki.gentoo.org/wiki/QEMU>
14. <https://github.com/weaveworks/wks-quickstart-firekube>
15. <https://www.weave.works/blog/firekube-fast-and-secure-kubernetes-clusters-using-weave-ignite>