Firecracker

Ben, Jack, and Ryan

What is Firecracker?

“Firecracker is an open-source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services that provide serverless operational models.” [1]

Where is Firecracker Useful?

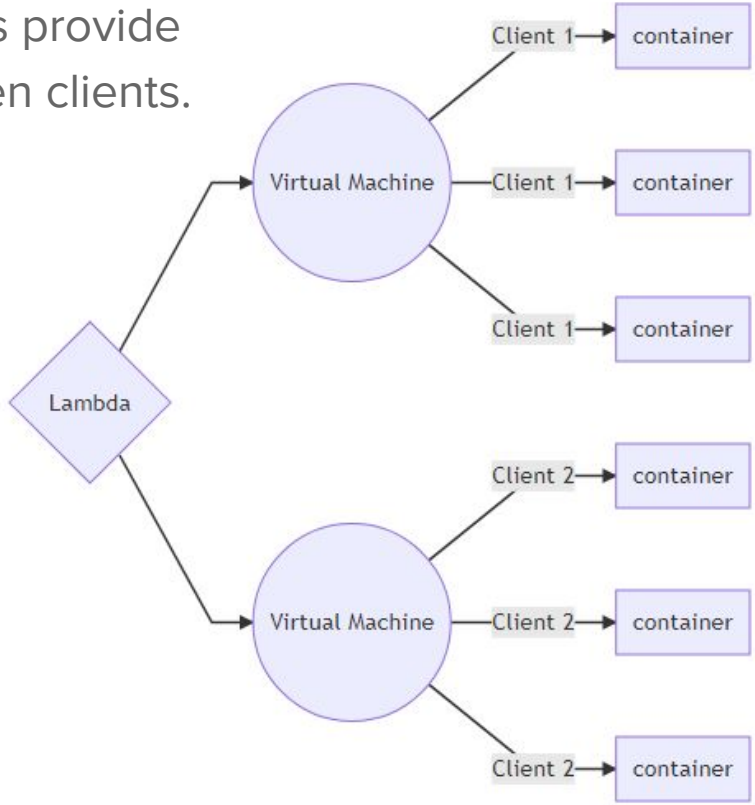
To understand why Firecracker is valuable, we first need to understand the needs of Lambda and what services it is attempting to provide.

AWS Lambda scales applications in response to incoming events. Processes trigger workloads individually and needs to support request scaling from a few per day to thousands per second. [2] Lambda also promises code to not require infrastructure in order to be managed and will scale up and down automatically.

Where is Firecracker NOT Useful?

Firecracker is more suited to be integrated with applications that has an event driven API. Long running and memory intensive workloads, would not be able to benefit greatly from Firecrackers main selling point of being able to scale up and down.

Virtual Machines provide isolation between clients.



Linux containers provide isolation between functions.

Previous Architecture

System Modules

Modules: VMM

The VMM is responsible for handling pre-boot and runtime requests from the firecracker VMM, as well as setting the boot configuration and setting up the VM for boot.

Firecracker is regulated via VmmAction's which provide entry points into VMM logic. Some VmmAction’s include FlushMetrics, Pause, Resume, StartMicroVm, etc. The API exposes different values of request that can be defined from a json request body or over an HTTP request.

Modules: VMM (Contd.)

- The VMM also interacts with the KVM to set up the memory and vCPUs for the microVM.
  - The KvmContext object contains a set of ioctls that the VMM can use to communicate with KVM
  - Examples: AdjustClock, VcpuEvents, ExtCpuid

```
pub(crate) fn setup_kvm_vm(
    guest_memory: &GuestMemoryMmap,
    track_dirty_pages: bool,
) -> std::result::Result<Vm, StartMicrovmError> {
    use self::StartMicrovmError::Internal;
    let kvm = KvmContext::new()
        .map_err(Error::KvmContext)
        .map_err(Internal)?;
    let mut vm = Vm::new(kvm.fd()).map_err(Error::Vm).map_err(Internal)?;
    vm.memory_init(&guest_memory, kvm.max_memslots(), track_dirty_pages)
        .map_err(Error::Vm)
        .map_err(Internal)?;
    Ok(vm)
}
```

Modules: Jailer

The jailer binary is responsible for starting a new Firecracker process.

The jailer initializes system resources that require higher privileges and executes into the Firecracker binary which spawns a new Firecracker process which runs in the microVM as an unprivileged process.

Modules: Rate Limiter

The rate limiter uses a token system to limit the number of operations per second as well as bandwidth. Each type of token has a corresponding bucket with a budget that it has been allocated. If any of the buckets run out of budget, the limiter will enter a blocked state. At this point a timer will then notify the user to retry sending the data.

Module Communication

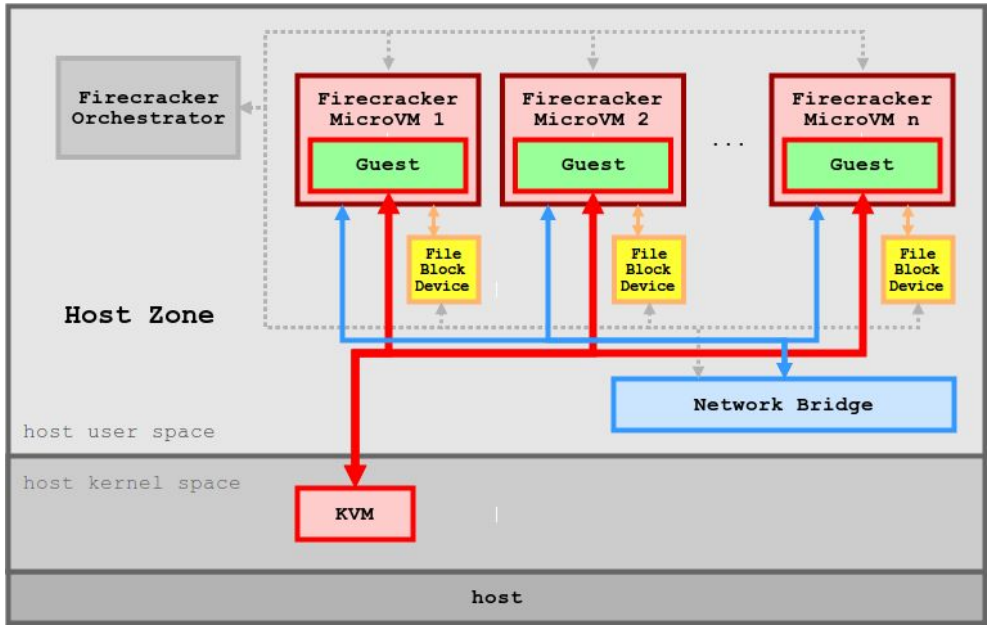
Event Manager:

The event manager provides an abstraction through event manager interface to provide communication between the different modules in the system and support virtualized IO. Epoll monitors multiple file descriptors to see if IO is possible on any of them.

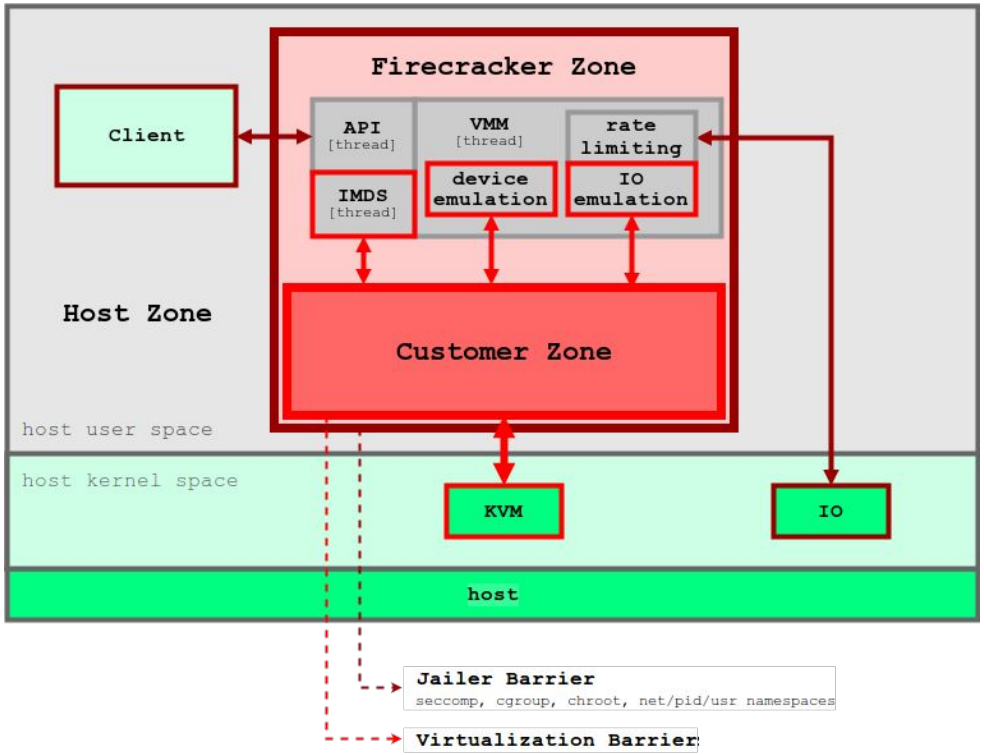
The epoll_wait system call returns file descriptors that are ready for IO. This is then stored inside a buffer. A hashmap contains the list of 'subscribers' that are the processes listening to the file descriptor for IO to be performed.

```
pub struct EventManager {
    epoll: Epoll,
    subscribers: HashMap<RawFd, Arc<Mutex<dyn Subscriber>>>,
    ready_events: Vec<EpollEvent>,
}
```

Security


Firecracker Orchestrator
Firecracker MicroVM 1
Guest
Firecracker MicroVM 2
Guest
...
Firecracker MicroVM n
Guest
File Block Device
File Block Device
File Block Device
Host Zone
Network Bridge
host user space
host kernel space
KVM
host

Overview

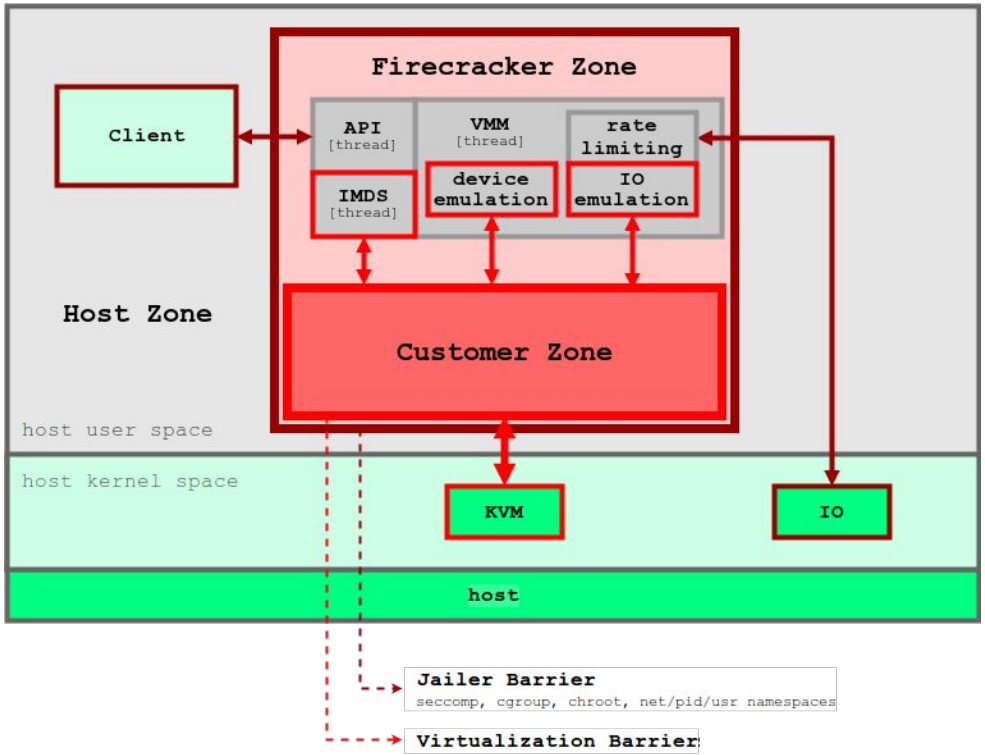


Zoom in on a single microVM

Firecracker Security: Principles

Defense in Depth

Firecracker contains several nested layers of isolation each with different levels of trust.

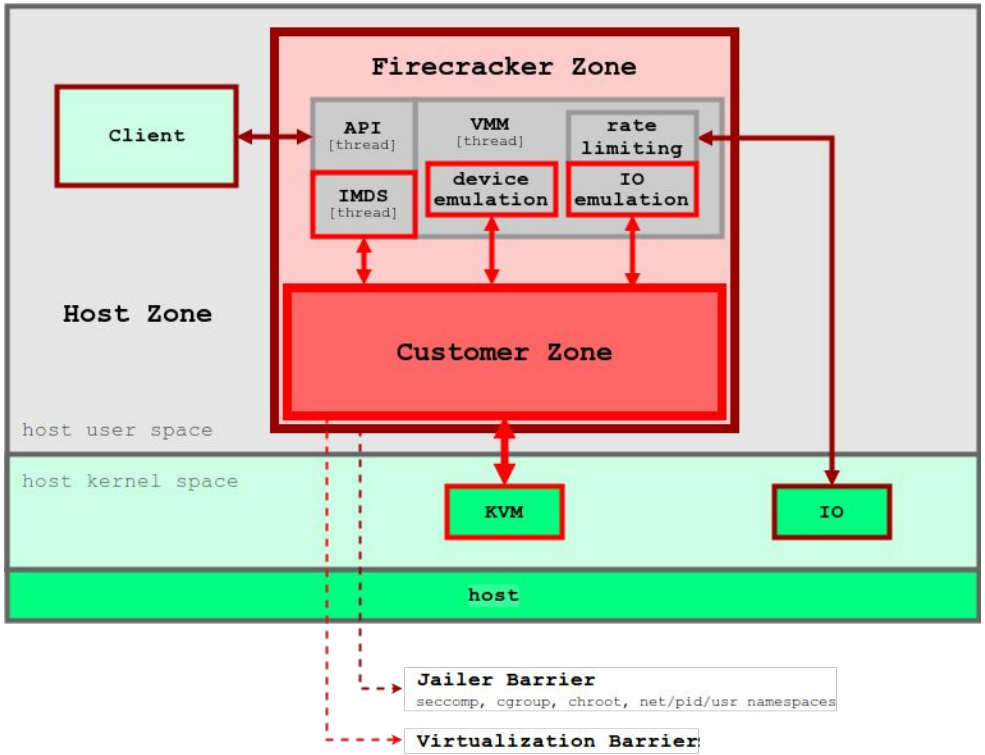


Firecracker Security: Principles

Minimal Trusted Computing Base

Firecracker is only around 50k LoC which is 96% smaller than QEMU.

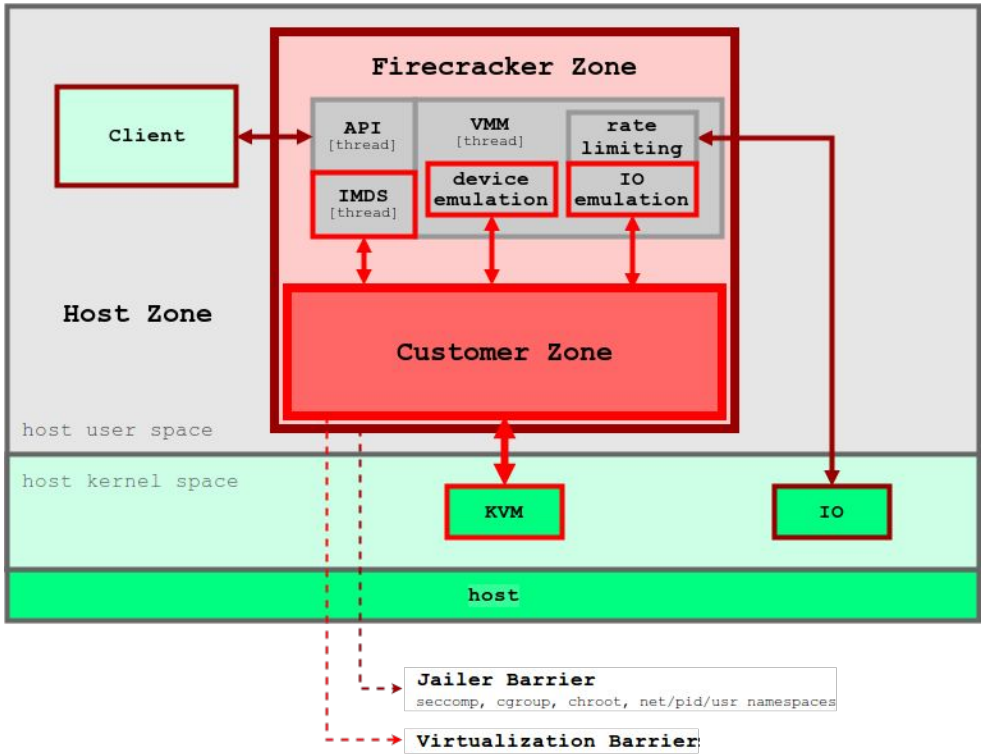
Furthermore, Firecracker implements a minimalist feature set which removes all unnecessary resources/features from the system also minimizes the TCB.



Firecracker Security: Principles

Separation of Privilege

A new firecracker VM is booted up to service each request. These VMs run at a lower privilege level than the EventManager or VMM and only communicate with the VMM through a predefined set of requests.



Firecracker Security: C.I.A

Confidentiality: Requests are kept confidential to the VM they run in, which is created and destroyed when the request enters and exits the system.

Since one firecracker process runs per microVM, we have a pretty clear case for isolation.

Firecracker Security: C.I.A

Integrity: Once a microVM is booted, it is even more limited in how it can interact with the VMM.

```
// Operations not allowed post-boot.
ConfigureBootSource(_)
| ConfigureLogger(_)
| ConfigureMetrics(_)
| InsertBlockDevice(_)
| InsertNetworkDevice(_)
| LoadSnapshot(_)
| SetBalloonDevice(_)
| SetVsockDevice(_)
| SetMmdsConfiguration(_)
| SetVmConfiguration(_)
| StartMicroVm => Err(VmmActionError::OperationNotSupportedPostBoot),
```

Firecracker Security: C.I.A

Accessibility: By having a single process per VM, Firecracker VMs can be pre-configured to consume only the amount of memory and CPU resources required of that process, which prevents a single firecracker VM from using up all of the resources available.

Performance and Optimizations

System Performance: Boot Speed

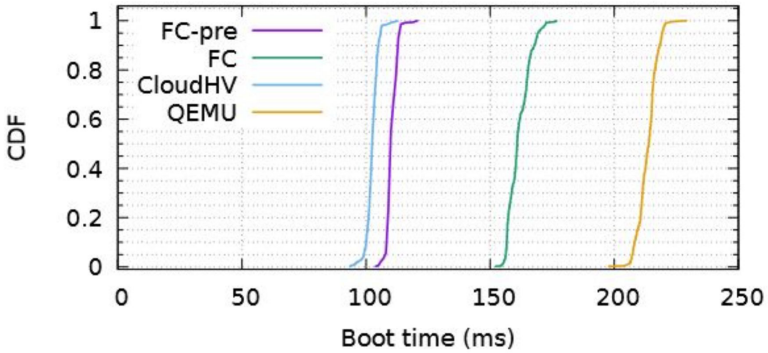
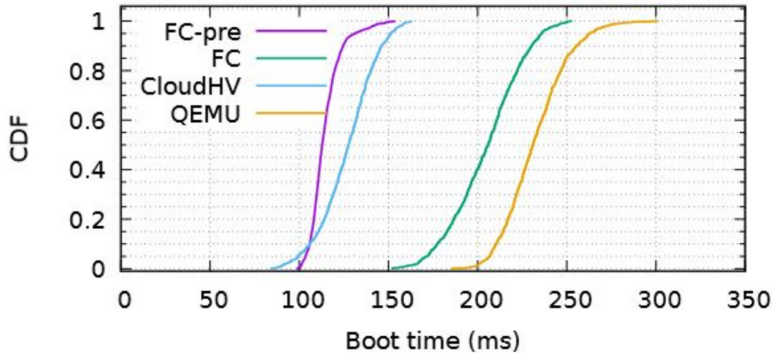


Figure 5: Cumulative distribution of wall-clock times for starting MicroVMs in serial, for pre-configured Firecracker (FC-pre), end-to-end Firecracker, Cloud Hypervisor, and QEMU.



System Performance: I/O Bandwidth

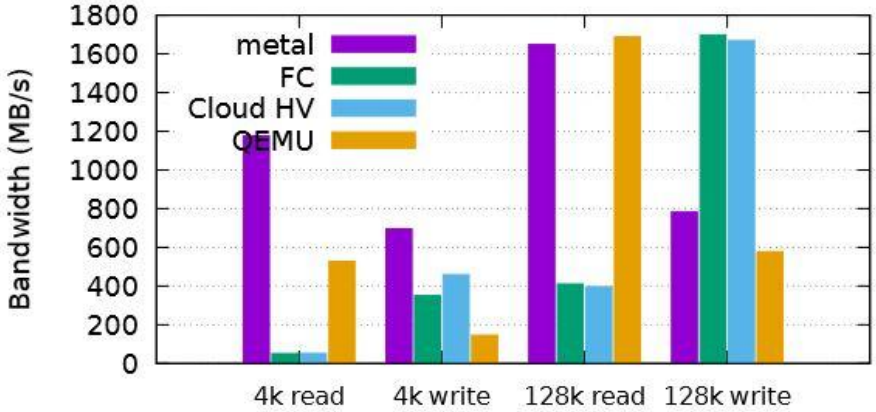


Figure 8: IO throughput on EC2 m5d.metal and running inside various VMs.

Optimizations and Fast Paths

A key optimization to firecracker is the pre-configured boot that results in the fast boot times.

This struct is returned in the function build_microvm_for_boot which sets up the virtual environment to boot the referenced kernel.

```rust
/// Holds the kernel configuration.
#[derive(Debug)]
pub struct BootConfig {
    /// The commandline validated against correctness.
    pub cmdline: kernel::cmdline::Cmdline,
    /// The descriptor to the kernel file.
    pub kernel_file: std::fs::File,
    /// The descriptor to the initrd file, if there is one
    pub initrd_file: Option<std::fs::File>,
}
```

```rust
pub fn build_microvm_for_boot(
    vm_resources: &super::resources::VmResources, // VmResources (IMPORTANT) struct for this VM
    event_manager: &mut EventManager,
    seccomp_filter: BpfProgramRef,
) -> std::result::Result<Arc<Mutex<Vmm>>, StartMicrovmError> {
    use self::StartMicrovmError::*;
    let boot_config = vm_resources.boot_source().ok_or(MissingKernelConfig)?;

    //continued below
```

```rust
/// Holds the kernel configuration.
#[derive(Debug)]
pub struct BootConfig {
    /// The commandline validated against correctness.
    pub cmdline: kernel::cmdline::Cmdline,
    /// The descriptor to the kernel file.
    pub kernel_file: std::fs::File,
    /// The descriptor to the initrd file, if there is one
    pub initrd_file: Option<std::fs::File>,
}
```

```rust
pub struct VmResources {
    /// The vCpu and memory configuration for this m
    vm_config: VmConfig,
    /// The boot configuration for this microVM.
    boot_config: Option<BootConfig>,  // IMPORTANT!!
    /// The block devices.
    pub block: BlockBuilder,
    /// The vsock device.
    pub vsock: VsockBuilder,
    /// The balloon device.
    pub balloon: BalloonBuilder,
    /// The network devices builder.
    pub net_builder: NetBuilder,
    /// The configuration for `MmdsNetworkStack`.
    pub mmds_config: Option<MmdsConfig>,
    /// Whether or not to load boot timer device.
    pub boot_timer: bool,
}
```

References

[1] https://firecracker-microvm.github.io/

[2] https://aws.amazon.com/lambda/

[3] https://www.usenix.org/system/files/nsdi20-paper-agache.pdf

[4] https://man7.org/linux/man-pages/man7/epoll.7.html

[5] https://www.usenix.org/system/files/atc20-ren.pdf