Alpaquita Linux
Using Firecracker and QEMU microVMs

Alpaquita Linux
Revision 1.0
March 2025

bell/soft

Contents

1. Introduction

Firecracker VM

Firecracker is an open-source virtualization technology that is specifically designed for creating and managing secure, lightweight virtual machines (microVMs). Originally developed by Amazon Web Services (AWS) for their serverless computing platform, AWS Lambda and AWS Fargate, Firecracker has gained popularity due to its focus on speed, security, and efficiency. It is built on top of the Linux Kernel-based Virtual Machine (KVM) and is optimized for running transient and short-lived workloads, such as serverless functions, microservices in multi-tenant environments.

Reference Links

For more information on Firecracker VM, refer to the following resources:

- Firecracker GitHub. This is the official repository for Firecracker, containing documentation, source code, and examples.
- Quick Start Guide. A step-by-step guide to setting up and running Firecracker.
- Firecracker API Documentation. Learn how to interact with Firecracker programmatically using its API.
- CPU Templates Documentation. This document provides detailed information on CPU templates and supported CPU features.
- Firecracker Jailer Documentation.

Alpaquita Linux

Alpaquita Linux is designed to work seamlessly with Firecracker VM, offering a streamlined setup process that eliminates the need for custom kernel compilation or complex configurations:

- **Alpaquita Linux kernel (vmlinux)** - pre-built and optimized for Firecracker VM, ensuring compatibility and performance out of the box.

- **Root Filesystem (rootfs)** - even though the root filesystem can be easily created using Alpaquita's base Docker images, we also provide a **ready-to-use microVM rootfs** for your convenience. This rootfs can be downloaded and customized to meet your specific needs. Additionally, you can check and download the **Dockerfile** used to build the rootfs and adjust it to meet your specific requirements.

2. Performance and Resource Usage

While Firecracker VM provides stronger security than containers, it is also more resource-efficient than traditional VMs like QEMU. Here’s how Firecracker VM compares to containers and QEMU in terms of performance and resource usage:

- Startup Time - Firecracker VM with Alpaquita Linux is fully initialized in less than 380ms (Intel Skylake machine), which is slower than containers but significantly faster than traditional VMs.
- Resource Usage - Firecracker VM uses more memory than containers but less than QEMU. In our tests, Firecracker VM with Alpaquita Linux used half the memory consumed by QEMU.

3. Enhanced Security and Isolation

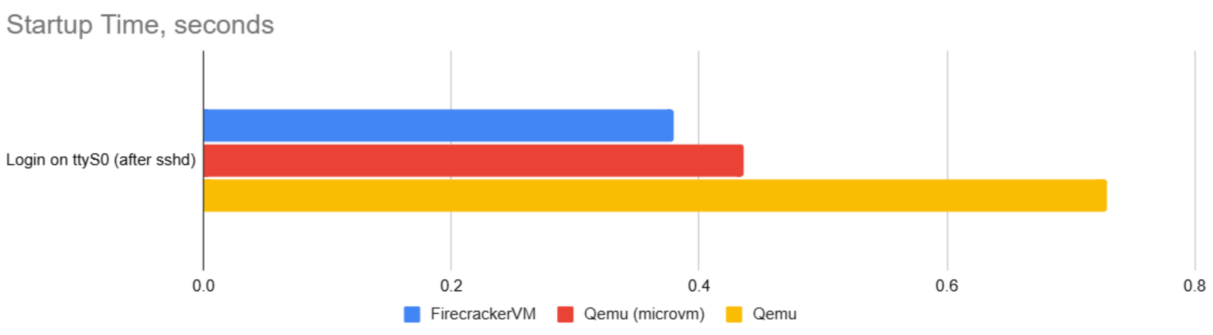
Firecracker VM is designed to provide strong isolation while maintaining lightweight and fast performance. The following list contains the key advantages of Firecracker VM over containers and QEMU:

- **Hardware-Level Isolation** - Firecracker VM uses KVM to create microVMs that are isolated from the host and other VMs. Each microVM has its own virtualized hardware, including CPU and I/O devices. Configuring CPU features in Firecracker VM helps you optimize the microVM for your specific workload requirements.
- **Minimal Attack Surface:**
    - Firecracker VM emulates only a minimal set of devices, such as virtio-net for networking and virtio-blk for storage, further reducing the attack surface compared to QEMU, which emulates a wide range of devices.
    - Containers rely on the host kernel and its extensive set of system calls, which increases the attack surface.
    - The Firecracker and process is statically built and can be easily jailed using cgroups and seccomp.
- **Multi-Tenant Security:** Firecracker VM is ideal for multi-tenant environments, where multiple users or applications share the same physical hardware. Its strong isolation guarantees that one tenant cannot access or interfere with another tenant’s resources.
- **CI/CD Pipelines**: Firecracker VM can be used to create ephemeral environments for testing and deployment with specific CPU capabilities, reducing the time required to spin up and tear down environments.

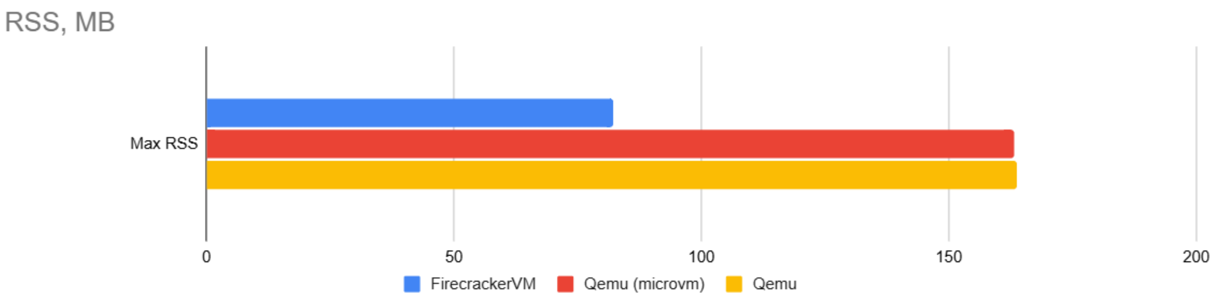
4. Performance Comparison: Firecracker VM vs QEMU

To demonstrate the performance benefits of Firecracker VM, we conducted a series of tests on Alpaquita Linux, comparing **Firecracker VM** with **QEMU**. All tests were performed on the Intel Core i5-6600 bare-metal machine, ensuring a consistent environment for both virtualization technologies. Each guest VM was allocated 1 CPU core and 128MB of RAM, with Alpaquita Linux serving as the operating system for both the host and the guest VMs. Below, we present the average results of our performance evaluation.

- Login time: it is the final mark in /etc/inittab so it means all the services, including sshd, are already running by this time, the network is up and functional. This time point includes firecracker/qemu process startup too since we measured it via external socket started right before microVM launch.



- Resource Usage: Firecracker VM uses significantly less memory than QEMU, with a maximum RSS (Resident Set Size) of 82 MB compared to 163 MB for QEMU. This makes Firecracker VM more resource-efficient while still providing the security benefits of a full VM.



For benchmarking, we used two types of the QEMU options to boot `vmlinux` and `rootfs`, ensuring a fair comparison with Firecracker:

- default machine type:

```
qemu-system-x86_64 \
  -cpu host -enable-kvm -smp 1 -m 128m \
  -kernel vmlinux \
  -append "console=ttyS0 quiet trace_clock=local root=/dev/vda" \
  -drive file=rootfs.ext4,if=virtio \
  -nographic -serial mon:stdio \
  -device virtio-net-pci,netdev=net0,mac=06:00:AC:10:00:02 \
  -netdev tap,id=net0,ifname=tap0,script=no,downscript=no \
  -no-reboot
```

- `microvm` machine type (was inspired by firecracker)

```
qemu-system-x86_64 -M microvm \
  -cpu host -enable-kvm -smp 1 -m 128m \
  -kernel vmlinux \
  -append "console=ttyS0 pci=off quiet trace_clock=local root=/dev/vda" \
  -drive file=rootfs.ext4,format=raw,id=blk0 \
  -device virtio-blk-device,drive=blk0 \
  -nographic -serial mon:stdio \
  -device virtio-net-device,netdev=net0,mac=06:00:AC:10:00:02 \
  -netdev tap,id=net0,ifname=tap0,script=no,downscript=no \
  -no-reboot
```

However, at first, it resulted in increased kernel boot time due to a failed `rtc_cmos` probe. To resolve this, we explicitly enabled the RTC: `-M microvm,rtc=on`. While this significantly improved boot times, unfortunately the memory usage remained consistent with the default machine type.

5. Deploying Alpaquita with Firecracker

Setting up networking for the Guest VM

If your use case does not require network connectivity for the guest VM, you can skip this section and proceed directly to preparing the vmlinux kernel and rootfs filesystem.

To enable network connectivity for the guest VM, configure a **TUN/TAP device** and set up **nftables rules** to forward traffic between the guest VM and the host's default gateway. The following script automates this process:

```
#!/bin/sh -e

# Install required packages
sudo apk update
sudo apk add cmd:jq iproute2 nftables uuidgen

# Load the TUN/TAP kernel module
sudo modprobe tun tap

# Define network settings
TAP_DEV="tap0"
TAP_HOST_IP="172.16.0.1"
TAP_GUEST_IP="172.16.0.2"
MASK_SHORT="/30"  # Subnet mask for the TAP interface
# Host interface for outbound traffic
HOST_IFACE=$(ip -j route list default | jq -r '.[0].dev')

# Set up the TAP device
# Delete existing TAP device (if any)
sudo ip link del "$TAP_DEV" 2> /dev/null || true
sudo ip tuntap add dev "$TAP_DEV" mode tap # Create a new TAP device
# Assign an IP address to the TAP device
sudo ip addr add "${TAP_HOST_IP}${MASK_SHORT}" dev "$TAP_DEV"
# Bring the TAP device online
sudo ip link set dev "$TAP_DEV" up
```

```
# Enable IP forwarding
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"

# Configure nftables for NAT and traffic forwarding
sudo nft add table firecracker
sudo nft 'add chain firecracker postrouting { type nat hook postrouting
priority srcnat; policy accept; }'
sudo nft 'add chain firecracker filter { type filter hook forward priority
filter; policy accept; }'
sudo nft add rule firecracker postrouting ip saddr $TAP_GUEST_IP oifname
$HOST_IFACE counter masquerade
sudo nft add rule firecracker filter iifname tap0 oifname eth0 accept
```

Configuration details:

- The host is assigned the `172.16.0.1` IP address, and the guest will use `172.16.0.2`.
- A NAT rule is added to masquerade traffic from the guest VM (`172.16.0.2`) to the host's default gateway.
- If the subnet `172.16.0.0/30` is already in use on your host system, you can modify the `TAP_HOST_IP` and `TAP_GUEST_IP` variables. Ensure that the Firecracker configuration file (alpaquita.json) is updated accordingly .

Verifying network configuration

Check nftables rules:

```
sudo nft list table firecracker
```

Expected output:

```
table ip firecracker {
   chain postrouting {
       type nat hook postrouting priority srcnat; policy accept;
       ip saddr 172.16.0.2 oifname "eth0" counter packets 0 bytes 0 masquerade
   }

   chain filter {
       type filter hook forward priority filter; policy accept;
       iifname "tap0" oifname "eth0" accept
   }
}
```

Check TAP Device:

```
ip addr show tap0
```

Expected output:

```
38: tap0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state
DOWN group default qlen 1000
    link/ether e6:38:b2:53:2f:c0 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.1/30 scope global tap0
       valid_lft forever preferred_lft forever
    inet6 fe80::e438:b2ff:fe53:2fc0/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

Setting up vmlinux and rootfs

Download Alpaquita Linux root filesystem and kernel

To set up Alpaquita Linux for Firecracker, download the root filesystem (`rootfs`) and the kernel (`vmlinux`). Use the following commands to fetch these components:

```
# Define system architecture and libc variant (glibc or musl)
ARCH="$(uname -m)"
LIBC=glibc  # Replace with "musl" if using musl libc

# Create a directory for the root filesystem
mkdir rootfs

# Download and extract the Alpaquita Linux root filesystem.
# Note that sudo is required to preserve file ownership
# (alternatively use fakeroot or container environment).
wget -q -O - https://packages.bell-
sw.com/alpaquita/${LIBC}/stream/releases/${ARCH}/alpaquita-microvm-rootfs-
stream-latest-${LIBC}-${ARCH}.tar.gz | sudo tar -C rootfs -xz

# Download the Alpaquita Linux kernel
wget -q -O - https://packages.bell-
sw.com/alpaquita/${LIBC}/stream/releases/${ARCH}/alpaquita-microvm-vmlinux-
stream-latest-${LIBC}-${ARCH}.xz | unxz > vmlinux
```

Configure SSH access for the Guest

To enable SSH login to the guest system, generate an SSH key pair and copy the public key to the root filesystem:

```
# Generate an SSH key pair (skip if you already have one)
ssh-keygen -f id_rsa -N ""

# Copy the public key to the root filesystem for SSH authentication
sudo cp id_rsa.pub rootfs/.ssh/authorized_keys
```

Configure DNS resolution for the Guest

To ensure the guest system can resolve domain names, copy the host's DNS configuration (`resolv.conf`) to the guest root filesystem:

```
sudo cp /etc/resolv.conf rootfs/etc/
```

Fine-tune the Alpaquita rootfs

- `chroot` command

  If you want to further customize the Alpaquita `rootfs` after it has been created and adjusted in the previous steps, you can run the `chroot` command. This helps you enter the rootfs environment and make changes directly:

  ```
  sudo chroot rootfs /bin/sh
  ```

  For example, once inside the `chroot` environment, we can install JDK:

  ```
  apk update
  apk add openjdk23
  ```

  After adding modifications, exit the `chroot` environment:

  ```
  exit
  ```

- `docker`/`podman` commands

  For users who need to create the rootfs from scratch, we provide the `Dockerfile.rootfs` file used to build the original Alpaquita microVM tarball. This approach is more complex, but offers complete control over the rootfs creation process. The following steps will help you get started:

Download the `Dockerfile.rootfs` from the official Alpaquita GitHub repository so you have something to start with:

```
wget https://raw.githubusercontent.com/bell-
sw/Alpaquita/refs/heads/master/microvm/\
Dockerfile.rootfs
```

Edit the `Dockerfile.rootfs` file to your specific needs. For example:

```
COPY resolv.conf /etc/resolv.conf
RUN apk add <extra-package-name>
```

After that, rebuild the rootfs with your customizations, specifying the desired variant (`stream-glibc` or `stream-musl`) using the `TAG` build argument:

```
docker build --no-cache \
   --output type=tar,dest=rootfs.tar \
   --progress plain \
   --build-arg TAG=stream-glibc \
   -f Dockerfile.rootfs .
```

Extract the contents of the generated tarball into a newly created rootfs directory:

```
mkdir rootfs && sudo tar -C rootfs -xf rootfs.tar
```

Creating an ext4 Root Filesystem Image

Finally, create an ext4-formatted root filesystem image from the extracted root filesystem:

```
# Define the size of the root filesystem image
ROOTFS_SIZE="100M"
NAME=rootfs

# Create an empty file of the specified size
truncate -s $ROOTFS_SIZE $NAME.ext4

# Format the file as an ext4 filesystem and populate it with the root
filesystem
mkfs.ext4 -d $NAME -F $NAME.ext4
```

Downloading official Firecracker VM binaries

```
ARCH="$(uname -m)"

release_url="https://github.com/firecracker-microvm/firecracker/releases"
latest=$(basename $(curl -fsSLI -o /dev/null -w  %{url_effective}
${release_url}/latest))
version="${latest}-${ARCH}"
wget -q -O - ${release_url}/download/${latest}/firecracker-${version}.tgz | tar
-xz

# Rename the binaries to "firecracker" and "jailer"
mv release-${version}/firecracker-${version} firecracker
mv release-${version}/jailer-${version} jailer
```

Preparing Alpaquita microVM configuration

Firecracker VM provides a rich HTTP API for managing microVMs, which makes it easy to integrate with orchestration tools and automation pipelines.

For demonstration purposes, we'll use the `alpaquita.json` configuration file instead:

```
{
  "boot-source": {
    "kernel_image_path": "vmlinux",
    "boot_args": "console=ttyS0 reboot=k panic=1 pci=off quiet
trace_clock=local"
  },
  "drives": [
    {
    "drive_id": "rootfs",
    "path_on_host": "rootfs.ext4",
    "is_root_device": true,
    "is_read_only": false
    }
  ],
  "network-interfaces": [
    {
        "iface_id": "eth0",
        "guest_mac": "06:00:AC:10:00:02",
        "host_dev_name": "tap0"
    }
  ],
  "machine-config": {
    "vcpu_count": 1,
    "mem_size_mib": 128
  }
}
```

The guest IP address is converted automatically inside the guest VM (**fcnet** service) from the hardware address ("guest_mac"), using the last 4 bytes, such as `"06:00:AC:10:00:02" → "172.16.0.2"`.

Also, if you skip the host network setup, remove the **"network-interfaces"** section from `alpaquita.json`.

Launching Firecracker VM

Now you can start your microVM. You can start your microVM directly or using the Jailer with a more secure environment.

Start Firecracker directly

```
./firecracker --no-api --config-file alpaquita.json
```

Expected output:

```
2025-03-18T11:02:29.432333074 [anonymous-instance:main] Running Firecracker
v1.10.1
[...]
2025-03-18T11:02:29.478294133 [anonymous-instance:main] Successfully started
microvm that was configured from one single json

   OpenRC 0.60 is starting up Alpaquita Linux (x86_64)

ssh-keygen: generating new host keys: RSA ECDSA ED25519

Welcome to BellSoft Alpaquita Linux!

login[855]: root login on 'ttyS0'
microvm:~#
```

You can also connect via ssh:

```
ssh -i id_rsa root@172.16.0.2
```

Start Firecracker using the Jailer

Jailer is a companion tool for Firecracker that enforces additional security measures by running Firecracker in a chroot environment, dropping privileges, setting resource limits and namespacing.

To streamline the process of setting up and running a Firecracker microVM using the **Jailer** tool, we'll create a wrapper script named `jailer.sh`. This script automates the configuration, file setup, and execution of the microVM in a secure, isolated environment.

The following is an example of the `jailer.sh` script:

```
# Define the configuration file (default: alpaquita.json)
CONFIG="${1:-alpaquita.json}"

# Generate a unique ID for the microVM
UNIQ_VM_ID="aq-$(uuidgen)"

# Retrieve the current user's UID and GID
USER_ID="$(id -u)"
GROUP_ID="$(id -g)"

# Define the base and root directories for Jailer
JAILER_BASE_DIR="/srv/jailer"
JAILER_ROOT_DIR="$JAILER_BASE_DIR/firecracker/$UNIQ_VM_ID/root"

# Create the Jailer root directory and copy necessary files
sudo mkdir -p $JAILER_ROOT_DIR
sudo cp $CONFIG vmlinux rootfs.ext4 $JAILER_ROOT_DIR/

# Set ownership of the root filesystem to the current user
sudo chown $USER_ID:$GROUP_ID $JAILER_ROOT_DIR/rootfs.ext4

# Run Jailer to start the Firecracker microVM
sudo ./jailer \
  --id $UNIQ_VM_ID \
  --uid $USER_ID \
  --gid $GROUP_ID \
  --chroot-base-dir /srv/jailer \
  --exec-file firecracker \
  -- \
  --no-api --config-file $CONFIG
```

Once the `jailer.sh` script is prepared, you can start the Firecracker microVM with the following commands:

```
# Make the script executable
chmod +x jailer.sh

# Run the script to start the microVM
./jailer.sh
```

Example of the expected output:

```
2025-03-18T14:31:56.555642425 [aq-60ed995f...:main] Running Firecracker v1.10.1
[...]
2025-03-18T14:31:56.593874464 [aq-60ed995f...:main] Successfully started
microvm that was configured from one single json

   OpenRC 0.60 is starting up Alpaquita Linux (x86_64)


Welcome to BellSoft Alpaquita Linux!

login[886]: root login on 'ttyS0'
microvm:~#
```

To confirm that Firecracker has successfully transitioned to a restricted environment (chroot, dropped privileges, and seccomp filters) when using the Jailer tool, we can inspect the process's attributes. The following commands help you verify the setup.

Use the `ls` command to list the contents of the chroot directory:

```
ls -l /proc/$(pgrep firecracker)/root/
```

Output:

```
total 155176
-rw-r--r-- 1 bellsoft bellsoft 104857600 Mar 18 17:32 alpaquita-stream-
glibc.ext4
drwx------ 3 bellsoft bellsoft      4096 Mar 18 17:31 dev
-rwxr-xr-x 1 root   root    2707976 Mar 18 17:31 firecracker
-rw-r--r-- 1 root   root          4 Mar 18 17:31 firecracker.pid
drwx------ 2 bellsoft bellsoft      4096 Mar 18 17:31 run
-rw-r--r-- 1 root   root        555 Mar 18 17:31 alpaquita.json
-rw-r--r-- 1 root   root    51306496 Mar 18 17:31 vmlinux-6.6.82-1-lts
```

Inspect the dev directory within the chroot environment to verify that only essential devices are exposed:

```
ls -l /proc/$(pgrep firecracker)/root/dev/
```

Output:

```
total 4
crw------- 1 bellsoft bellsoft 10, 232 Mar 18 17:31 kvm
drwx------ 2 bellsoft bellsoft  4096 Mar 18 17:31 net
crw------- 1 bellsoft bellsoft  1,   9 Mar 18 17:31 urandom
```

Check the `Uid` and `Gid` fields in the process status to confirm that Firecracker is running with dropped privileges:

```
grep -E '([UG]id|Seccomp)' /proc/$(pgrep firecracker)/status
```

Output:

```
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
Seccomp:    2
Seccomp_filters:    1
```

Alpaquita Linux

Using Firecracker and QEMU microVMs

bell/soft