Trabajo - Instalación, Administración y Uso de un servicio en Linux

Gestión de Clúster de contenedores: Containerd y Firecracker.

Manuel Domínguez Montero

Servicios Telemáticos

Departamento de Ingeniería Telemática

ÍNDICE

Herramientas analizadas	Asimiladas a
Firecracker y Containerd	***Servidor***
Api de Firecracker (Open Api) e intéprete Nerdctl	***Cliente***

He cambiado el término de cliente por el de herramienta en los apartados dónde se habla sobre la instalación y uso de los clientes empleados en este Proyecto.

En dichos apartados, se explica por que he tomado esa decision con respecto a una u otra tecnología.

1. Objetivos y Alcance

1.1. Introducción

Actualmente, bien sea por el número de usuarios que requieren servicios distribuidos, como por el número de entidades y empresas que solicitan desplegar sus servicios en la nube, la gestión de recursos físicos y lógicos en red se ha convertido en un pilar fundamental en el ámbito de las telecomunicaciones.

Partiendo en la pasada década de arquitecturas de software monolíticas. Con dependencias fuertemente acopladas, elementos de hardware muy específicos para cada servicio desplegado, así como, la ausencia de tecnologías de distribución estandarizada de configuración. El ámbito de la infraestructura, configuración y aprovisionamiento ha visto nacer multitud de tecnologías enfocadas a solucionar estas necesidades.

Por un lado, tenemos la aparición de los mecanismos estandarizados de virtualización ligera a través de contenedores y sistemas de orquestación.

Por otro lado, productos que ofrecen mecanismos de aprovisionamiento como Ansible[9], o Terraform[6], son hoy día elementos críticos para empresas que trabajan en el ámbito de la infraestructura como servicio.

Sin duda alguna, de entre todos, el hecho más relevante ha sido la aparición de los contenedores de aplicaciones.

Si bien, Docker[2] es por muchos la única tecnología de virtualización ligera conocida, existen muchos paradigmas diferentes de la misma enfocados a diversos casos de uso. Desde los contenedores rootless para el ámbito de la ciberseguridad y el Sandboxing, hasta contenedores basados en KVM como LXC que permiten un alto grado de personalización y modularidad.

En este trabajo se van a estudiar dos productos, uno enfocado a contenedores, Containerd[1]. El otro enfocado a la gestión de máquinas virtuales, Firecracker[10].

1.2. Motivación y funcionalidad de los servicios estudiados.

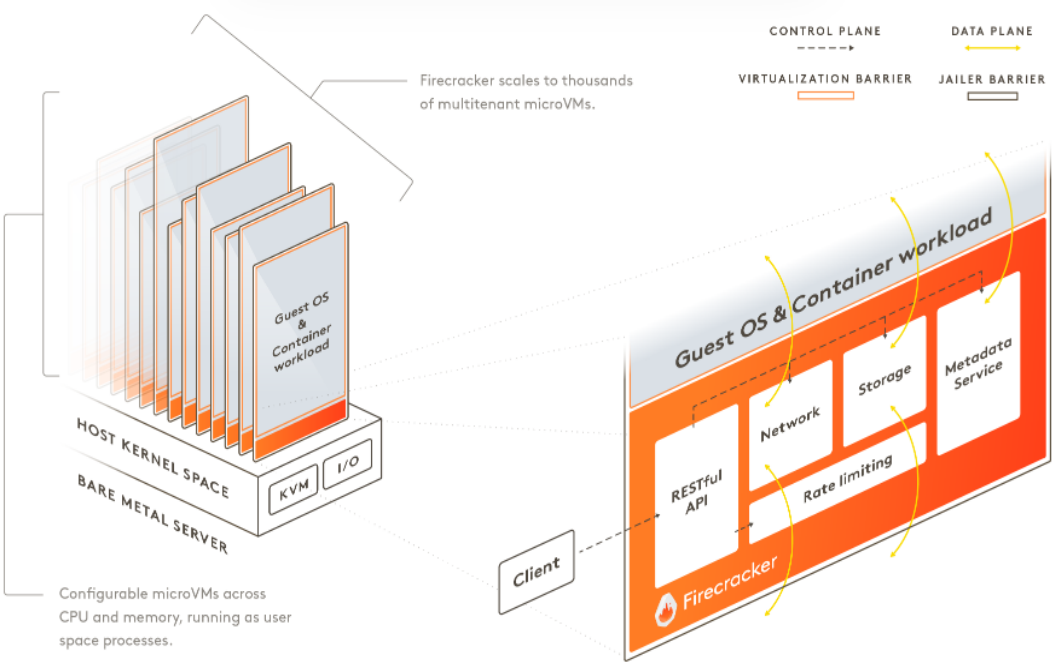
1.2.1. Firecracker.

Este producto nace con varios objetivos.

Lograr un nivel mayor de aislamiento del que ya es conseguido por parte de la virtualización ligera a través de contenedores.

Alto rendimiento y velocidad, además de un bajo consumo de elementos de hardware en comparación con las máquinas virtuales convencionales.

Permitir un alto grado de personalización por parte del usuario de las MMV desplegadas, así como, la capacidad de alterar sus configuraciones "en caliente", lo cual es una característica esencial en entornos de hiperdisponibilidad, como Data Centers o entornos Cloud.



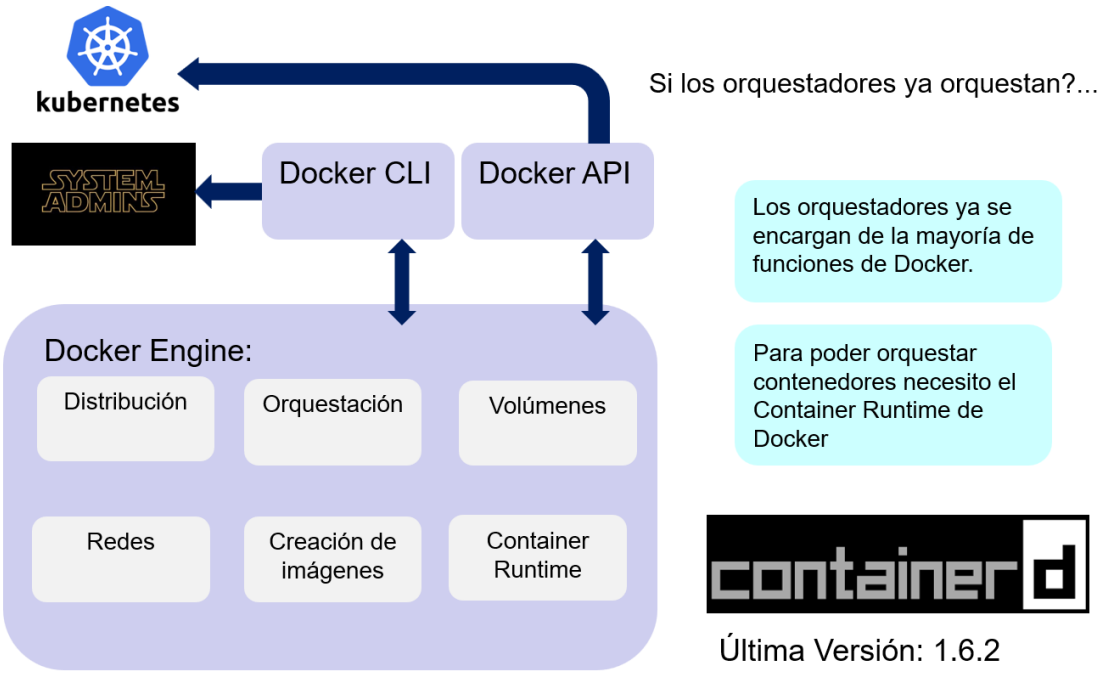
1.2.2. Containerd.

Al igual que el proyecto del que parte (**Docker[2]**), da soporte a la gestión y monitorización de contenedores basados en las especificaciones de la **OCI[1]**, término que se ha estandarizado como "**OCI compliant**".

Aunque es una pequeña porción de Docker en términos de funcionalidad. También es capaz de ofrecer herramientas interesantes de cara al uso directo por parte del usuario.

Sin embargo, su uso está enfocado a la gestión de contenedores por parte de orquestadores terceros como Kubernetes[4].

Tanto es así, que Containerd[11] nace de las exigencias de éste último al proyecto Docker para que liberase su container runtime (dockerd), a fin de simplificar la orquestación a través de la API de Kubernetes, haciendo que no fuera necesario utilizar al completo todo el sistema de Docker para este caso de uso.



1.3. Documentación bibliográfica

A lo largo de esta memoria, se van a citar numerosos productos, organizaciones y conceptos que resultan relevantes para el estudio de mi trabajo, por lo que se dejan referencias a los sitios Web donde poder encontrar información sobre los mismos.

[1] OCI : https://opencontainers.org/

[2] Docker: https://www.docker.com/

[3] Amazon Web Services: https://aws.amazon.com/es/?nc2=h_lg

[4] Kubernetes: https://kubernetes.io/es/

[5] Qemu: https://www.qemu.org/

[6] Terraform: https://www.terraform.io/

[7] Vagrant: https://www.vagrantup.com/

[8] AWS Fargate: https://aws.amazon.com/es/fargate/?nc=sn&loc=0

[9] Ansible: https://www.ansible.com/

A su vez, Se citan los principales sitios de documentación disponibles de las tecnologías estudiadas aquí.

[10] Firecracker: https://firecracker-microvm.github.io/

[11] Containerd: https://containerd.io/

2. Base Teórica

2.1. Firecracker:

Esta tecnología nos proporciona un motor de vitualización a nivel de kernel, lo cual, nos otorga una gran capacidad de personalización a la hora de poder configurar las Micro máquinas Virtuales, a partir de ahora, MMV.

Esto nos permite cargar imágenes de sistemas operativos muy simplificadas, reduciendo el coste computacional de arrancarlas y administrarlas.

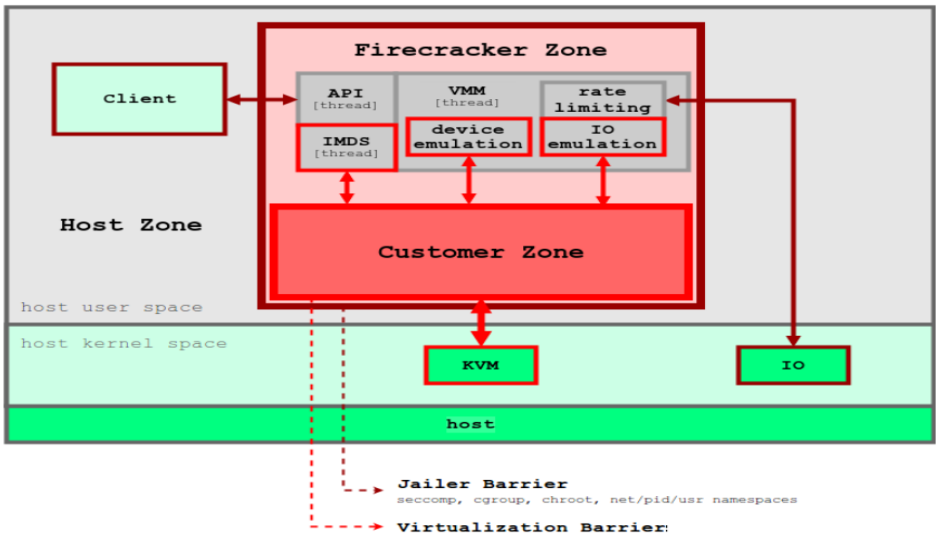
Firecracker nos permite utilizar una compilación del kernel de Linux propia, la cual podemos obtener de múltiples formas, siendo las más extendidas las siguientes:

- Compilación manual Had-Hoc de nuestro Kernel basado en Linux.
- Mediante una herramienta implementada en Docker y distribuida por Firecracker.
- Además podemos realizar este paso de una forma sencilla a través de QEMU[5].

Por otro lado, Amazon Web Services, desde ahora AWS[3], tiene repositorios públicos de donde podremos obtener binarios de algunos kernels preconfigurados.

Firecracker permite el aprovisionamiento, administración, arranque y parada de máquinas virtuales a través de ficheros de configuración JSON y a través de una API REST, además del intérprete firectl.

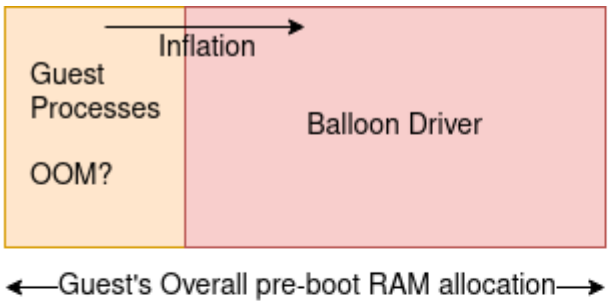
La arquitectura de un equipo que utilice este sistema sería la siguiente.



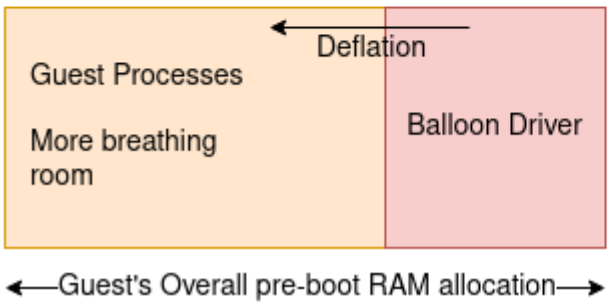
Adentrándonos más en profundidad en cada uno de sus componentes, tenemos para empezar el "VirtIO Ballooning", que se encarga de la gestión de memoria.

Esto es llevado a cabo a través del mecanismo conocido como "ballooning", donde cada uno de los huéspedes publica y solicita la cantidad de memoria que requiere, permitiendo que el resto pueda amoldarse.

Además, el sistema anfitrión puede cambiar en tiempo de ejecución la memoria reservada por cada una de estas máquinas virtuales.



Host may be giving away memory from the balloon to other guests

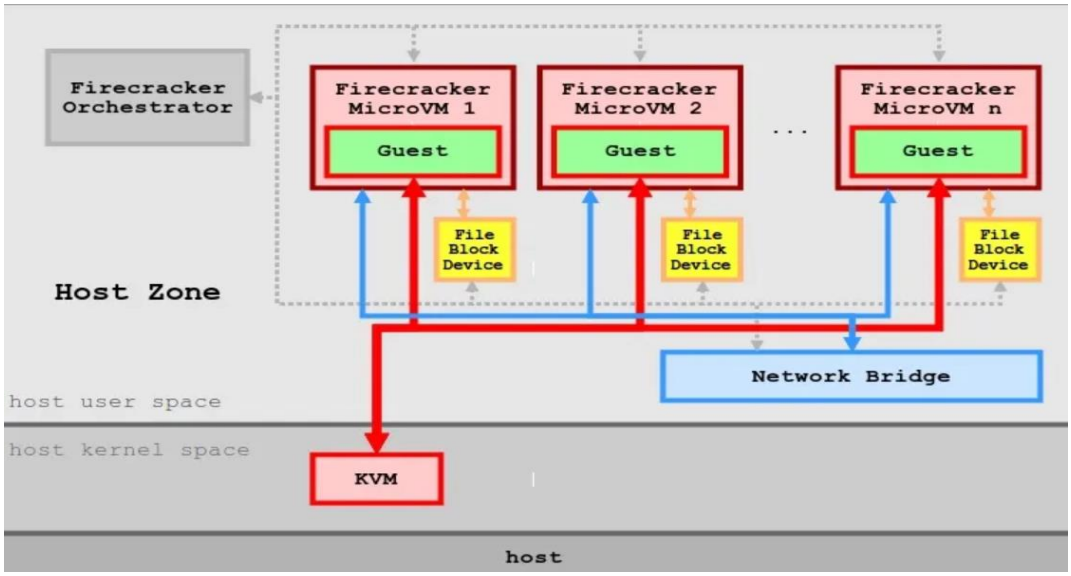


Host may be giving back memory to this guest by shrinking its target balloon size

En el ámbito del almacenamiento, hay que recordar que Firecracker no presenta una solución "**virtio-fs**" en la cual se emulan sistemas de archivos para cada máquina virtual o contenedor, como si ocurre en las especificaciones de la OCI.

Firecracker implementa una solución basada en "**VirtIo block devices**" , donde en la máquina anfitriona sí coexisten sistemas de archivos completos, no emulados, y separados por tanto para cada instancia o máquina virtual.

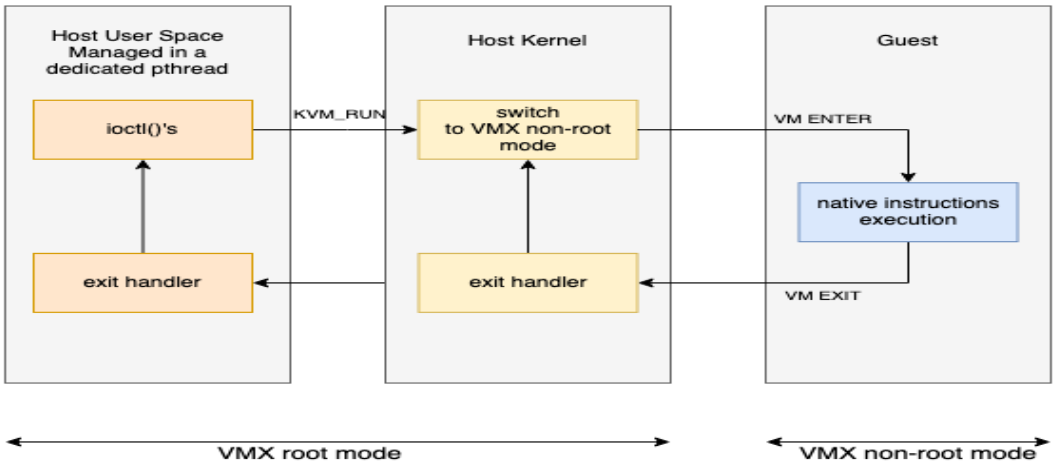
Por ello, en este aspecto, Firecracker[10] se posiciona como un fuerte competidor contra las tecnologías basadas en contenedores, ofreciendo un mayor grado de aislamiento y seguridad.



Firecracker también puede limitar el ancho de banda de cada uno de los sistemas huéspedes a través del componente IO Throttling, el cual se basa en el mecanismo de "bucket tokens".

Además, disponemos de emulación de microcontroladores, procesadores antiguos y otros dispositivos, además de drivers de periféricos como el i8042.

Por último, y siendo uno de sus aspectos más potentes, dentro de Firecracker, un vCPU no representa un CPU físico en la máquina anfitriona, estos vCPU son gestionados a través de hilos de POSIX.



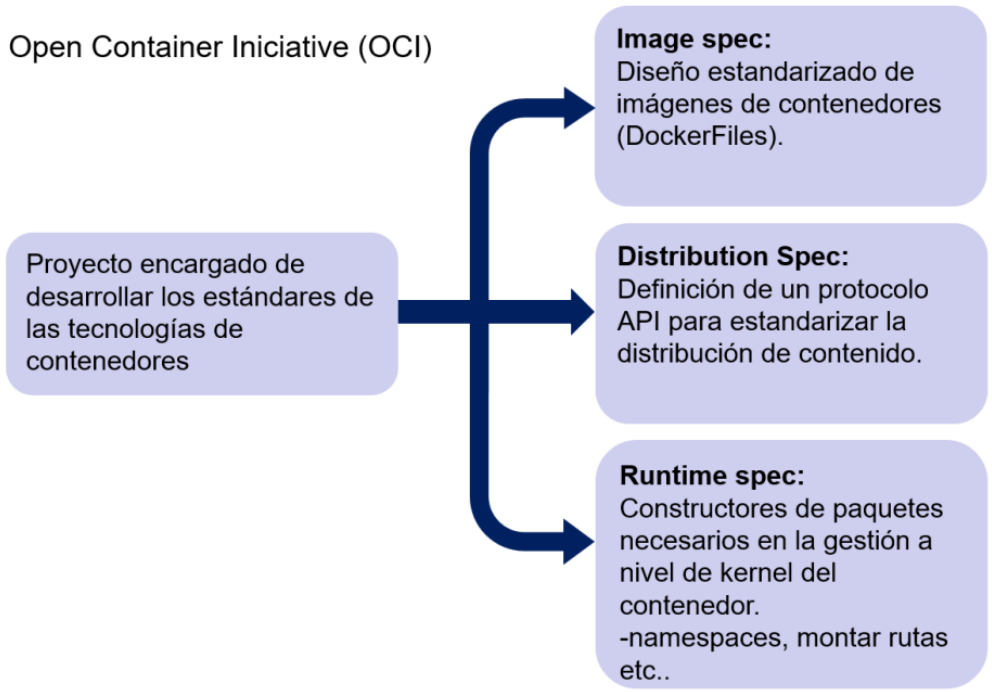
2.2. Containerd:

Siendo inicialmente parte del motor de contenedores propio del proyecto Docker[2] (dockerd), fue liberado y publicado, haciendo que no fuera necesario usar toda la infraestructura del motor Docker a la hora de realizar orquestación de contenedores basados en las especificaciones de la OCI[1] por parte de terceros como Kubernetes[4].

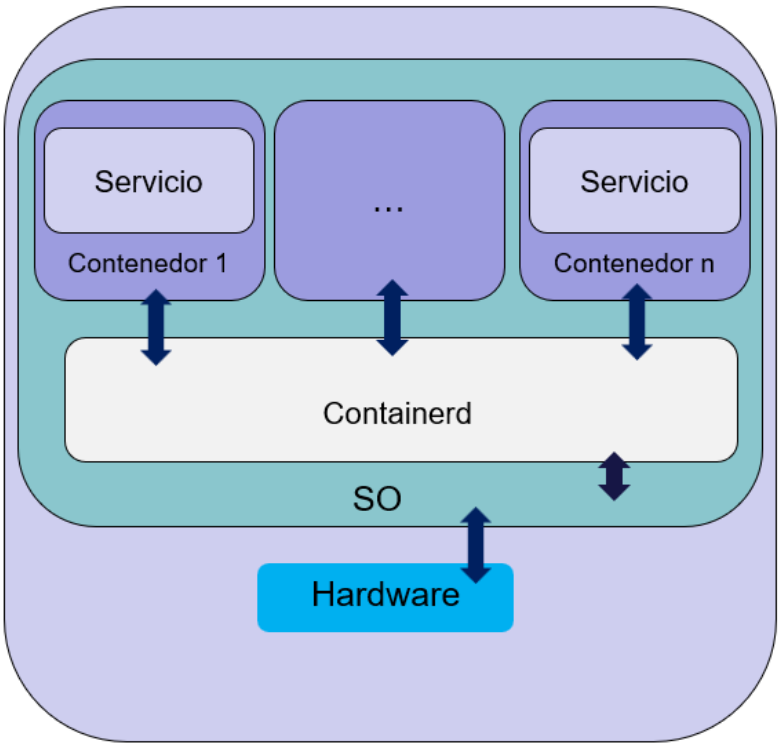
Mantiene parte de las funcionalidades de Docker, como la conectividad a nivel de red, sin embargo, no permite la construcción de imágenes a partir de ficheros tales como Dockerfile de forma nativa. Por otro lado, veremos que esto sí será posible mediante plugins instalados junto con uno de sus intérpretes, Nerdctl.

En el ámbito de los contenedores existe la Open Container Iniciative, la cual es una organización y proyecto de la Linux Foundation que se encarga de publicar especificaciones con el fin de estandarizar la creación, distribución y administración de contenedores.

Las especificaciones más importantes son las siguientes.



La arquitectura que encontraríamos dentro de un equipo que utilizase este motor de contenedores sería la siguiente:



3. Evaluación de las implementaciones estudiadas.

3.1.1. Firecracker, características.

Nombre Proyecto	**Firecracker**
Funcionalidades	➢ Provee de soporte para KVM. ➢ Gestión de MMV a través de API, ficheros de configuración JSON, así como el interpreté firectl. ➢ Compatibilidad con Kubernetes y otros orquestadores. ➢ Compatibilidad con motores de contenedores como Containerd para encapsulación. ➢ Gran capacidad de configuración de las MMV. ➢ Tiempos de ejecución muy por debajo de los de la competencia. ➢ Gestión de los recursos hardware de los equipos anfitriones altamente eficiente y personalizable.
Referencias	[10] Firecracker, [4] Kubernetes
Autores	Proyecto Firecracker, desarrollado por AWS. Amazon.com Inc.
Versión empleada	Firecracker: 1.1.0
Última versión existente	Firecracker: 1.2.0

3.1.2. Firecracker, equipamiento y dependencias de software.

Hardware necesario	➢ Disponible para CPU Intel ó AMD con arquitectura x64 ó x86 con soporte para virtualización. Se está testando en las siguientes arquitecturas: • Intel Skylake. • Intel Cascade Lake. • AMD Zen2. • ARM64 Neoverse N1. ➢ Memoria y disco suficiente para almacenar las MMV.
Infraestructura de red requerida	➢ Switches virtuales de Linux.
Dependencias software	➢ Kernel de Linux, 4.14 ó 5.10. ➢ Soporte y acceso a la virtualización a nivel de BIOS Y Sistema Operativo (Sin hipervisor) . ➢ Navegador en consola de comandos (Curl o Wget). ➢ Imagen del Kernel de Linux para ser virtualizada. ➢ Sistema de archivos de Linux para ligarlo a su respectiva imagen. ➢ Git, Nos servirá para el mantenimiento del servicio, así como, gestionar actualizaciones. Por otro lado, si queremos compilar nuestro propio kernel, una práctica común es hacerlo a partir de alguna rama del repositorio en Github de L.Torvald. ➢ Binario del proyecto Firecracker.
S.O empleado	➢ Ubuntu 20.04, disponible en laboratorio.

3.1.3. Containerd, Características.

Nombre Proyecto	Containerd
Funcionalidades	➢ Provee de soporte para la gestión de contenedores OCI ➢ Interfaz de usuario a través de diferentes intérpretes de comandos, por defecto ctr, el resto se estudiarán posteriormente. ➢ Interfaz con orquestadores de contenedores a través de API. ➢ Compatibilidad con imágenes de contenedores "OCI Compliant". ➢ Capacidad de encapsulación de contenedores a través de namespaces y chroot.
Referencias	[11] Containerd
Autores	Docker iniciative, actualmente de código abierto, proyecto Containerd.
Versión empleada	Containerd: 1.6.12
Última versión existente	Containerd: 1.6.12

3.1.4. Containerd, equipamiento y dependencias de software.

Hardware necesario	➢ PC con arquitectura x64 ó x86. ➢ Memoria y disco suficiente para almacenar las imágenes de contenedores
Infraestructura de red requerida	El sistema de red virtual lo proporciona Containerd, en este caso.
Dependencias software	Para su compilación a partir del código fuente: ➢ Compilador Protoc 3.x ➢ Golang 1.18 (en adelante). Para su uso dentro del proyecto: ➢ Navegador en consola de comandos (Curl o Wget). ➢ Git ➢ Motor de Docker(Docker CE). ➢ Makefile. ➢ ProtoC ➢ Brctl ➢ Rutime RunC ➢ Intérprete Nerdctl (no necesario, pero se usará en este trabajo)
S.O empleado	• Ubuntu 20.04, disponible en el laboratorio.

3.2 Comparativa de clientes existentes en el mercado.

3.2.1. Comparativa Qemu y Firecracker.

Tecnologías de virtualización como QEMU[5] rivalizan con Firecracker en el ámbito de la gestión de máquinas virtuales a través de interfaces cada vez más sencillas y configurables.

a su vez, comienzan a surgir tecnologías como Vagrant[7] que facilitan el aprovisionamiento y la administración de MV(en entornos de pruebas preferiblemente) apoyándose en terceros motores de virtualización como VirtualBox.

Por su similaridad con Firecracker, voy a optar por extenderme en las características de QEMU.

Producto	**Firecracker**	**Qemu**
Fecha última versión del servicio	07/12/2022 Versión 1.2.0	09/09/2022 Versión 7.1.94
Principales diferencias funcionales	➢ Herramientas para la personalización de Kernel. ➢ Solo virtualización de Kernel de Linux. ➢ No dispone de interfaz gráfica.	➢ Permite gestión de MV a través de Interfaz gráfica y por línea de comandos. ➢ Permite virtualización de sistemas operativos que no se basen en Linux.
Sistemas operativos soportados	➢ Linux	➢ Linux, Windows, MAC
Equipo desarrollador	Inicialmente Amazon.Inc. Actualmente de código abierto	QEMU Project Developers.
Tipo de licencia	Apache 2.0	GNU General Public License. V.2.
Lenguajes de programación empleados	➢ C ➢ Rust	➢ C
Nivel de uso en la actualidad	Muy extendido en entornos cloud, sobre todo en aquellos donde trabaja AWS	Más extendido entre usuarios finales, enfocado a la emulación de sistemas completos y con interfaz gráfica.

3.2.2. Comparativa entre Containerd y Docker.

Dados los orígenes de Containerd como parte de Docker, creo que sería interesante realizar una comparación entre estos dos productos para discernir claramente qué los diferencia en la actualidad.

Producto	**Containerd**	**Docker**
Fecha última versión del servicio	08/12/2022 Versión 1.6.12	09/09/2022 Versión 7.1.94
Principales diferencias funcionales	➢ No permite de forma nativa la construcción de imágenes de contenedores a partir de ficheros (Dockerfiles). ➢ Omite labores de orquestación, tales como las de mantener el control del rendimiento de un cluster de contenedores, levantando más de estos si hacen falta.	➢ Construcción de contenedores a través de ficheros Dockerfile. ➢ Realiza operaciones de orquestación de clústeres de contenedores
Sistemas operativos soportados	➢ Linux, de forma nativa. ➢ MAC, disponible en la APP Store. ➢ Windows, a través de WSL o empleando una máquina virtual.	➢ Linux, de forma nativa. ➢ MAC, disponible en la APP Store. ➢ Windows, a través de WSL o empleando una máquina virtual.
Equipo desarrollador	Inicialmente Amazon.Inc. Actualmente de código abierto.	Docker Iniciative.
Tipo de licencia	Apache 2.0.	Apache 2.0.
Lenguajes de programación empleados	➢ Go.	➢ Go.
Nivel de uso en la actualidad	Cada vez más extendido entre sistemas que requieren orquestación, dada la integración con Kubernetes.	Mucho más popular, sigue siendo más potente de forma nativa para los usuarios finales.

3.3. Herramientas de acceso al servicio.

Tanto Firecracker como Containerd son productos destinados a servir como soporte para otros sistemas/aplicaciones con funcionalidades concretas. Por ello, prefiero tratar a los diferentes mecanismos de acceso a estas tecnologías como herramientas, más que como clientes, dicho lo cual, voy a pasar a extenderme en aquellas existentes.

3.3.1.1. Api para Firecracker, Open API.

Nos permite comunicarnos con el gestor de MMV a través de un socket en el que se encuentra escuchando el servicio Firecracker.

Su principal fortaleza es que nos permite realizar cambios "En caliente" sobre la configuración de las máquinas virtuales.

Sobre todo, por este último apartado, voy a usar esta herramienta para comunicarme con el sistema Firecracker.

3.3.1.2. Comparativa con firectl.

Herramienta	**OpenAPI**	**firectl**
Fecha última versión	➢ v1.2.0	➢ v0.2.0
Principales diferencias funcionales	➢ Uso de un socket http para la comunicación con el sistema Firecracker ➢ Es la interfaz por defecto, presenta órdenes menos legibles y más extensas, por lo que en este trabajo, la mayoría se han sintetizado en scripts para mayor comodidad.	➢ Usa internamente tanto ficheros JSON de configuración como llamadas a la API. ➢ Mas sencilla de usar, pero no viene instalada por defecto y es más difícil de mantener. ➢ Es un intérprete de comandos.
Sistemas operativos soportados	Linux	Linux
Tipo de licencia	Apache 2.0	Apache 2.0
Lenguajes de programación empleados	➢ C	➢ C
Url	https://github.com/firecracker-microvm/firecracker/blob/main/docs/api_requests/actions.md	https://github.com/firecracker-microvm/firectl

3.3.2.1. Intérpretes de comandos de Containerd.

El motor Containerd nos ofrece varios intérpretes de comandos, así como acceso al mismo a través de su API.

En cuanto a los principales intérpretes tenemos:

Ctr: Intérprete por defecto, compatible con el proyecto Firecracker-Containerd. Aporta pocas funcionalidades.

Cabe destacar que no posee ningún comando que nos permita construir imágenes de contenedores a partir de un fichero de imagen.

Nerdctl: Similar al Docker-CLI en cuanto a los comandos disponibles y su sintaxis. Es más potente que ctr y nos permite construir imágenes de contenedores a partir de ficheros de imágenes, será el empleado en este trabajo.

En cuanto a su API:

CRI CTL: Soporte a la API (CRI-O), ésta, está enfocado hacia el uso del motor por parte de orquestadores.

En este trabajo no se contempla el uso de orquestadores de contenedores, por ello no vamos a hacer uso de la API de Containerd, de modo que, para tener acceso a funcionalidades que permitan realizar una comparativa más interesante con Firecracker se va a hacer uso de Nerdctl.

De esta manera, vamos a tener acceso de una forma más cómoda a los repositorios de imágenes (sin necesidad de indicar los registros) además de tener la posibilidad de construir nuestras propias imágenes de contenedores.

Dado que ctr y nerdctl son intérpretes, y, además, éste último presenta muchísimas características comunes con Docker, voy a optar por comparar estas dos soluciones.

3.3.2.2. Comparativa Nerdctl con Ctr.

Comparativa de clientes existentes en el mercado		
Cliente	**Ctr**	**Nerdctl**
Fecha última versión	08/12/2022 Versión 1.6.12	21/09/2022 Versión 1.0.0
Principales diferencias funcionales	➢ No apto para orquestación. ➢ No apto para construcción de imágenes.	➢ Apto para la construcción de imágenes a partir de ficheros de imágenes OCI Compliant. ➢ Capacidad de "traer" imágenes de contenedores de repositorios públicos sin tener que indicar el identificador de registro, a través de etiquetas. ➢ Labores de orquestación de clusters tales como "compose".
Sistemas operativos soportados	Linux	Linux
Tipo de licencia	Apache 2.0	Apache 2.0
Lenguajes de programación empleados	En su mayoría programado en Go, de igual manera que el resto del proyecto Containerd.	En su mayoría programado en Go, de igual manera que el resto del proyecto Containerd.
Url	https://github.com/projectatomic/containerd/blob/master/docs/cli.md	https://github.com/containerd/nerdctl

4. Proceso de instalación/administración del servidor

4.1. Obtención e Instalación de Firecracker.

4.1.1. Obtención e instalación de dependencias.

Si queremos realizar una instalación desde el código fuente necesitamos tener Docker instalado y corriendo como un demonio.

Descargamos e instalamos Docker.

```
sudo apt-get update
sudo apt-get install docker.io
```

Arrancamos Docker como un servicio de Systemd y añadimos nuestro usuario al grupo docker.

```
sudo systemctl start docker
sudo usermod -aG docker $USER
```

4.1.2. Obtención del binario ya compilado de Firecracker.

Descargamos el binario comprimido acorde a nuestra arquitectura de CPU del proyecto Firecracker

```
release_url=https://github.com/firecracker-microvm/firecracker/releases

latest=$(basename $(curl -fsSLI -o /dev/null -w %{url_effective} ${release_url}/latest))
arch=`uname -m`
curl -L ${release_url}/download/${latest}/firecracker-${latest}-${arch}.tgz | tar -xz
```

4.1.3. Obtención del código fuente de Firecracker para su compilación.

Clonamos el repositorio. Este equipo desarrolla sobre la rama main, por lo que no es necesario hacer checkout sobre ninguna otra, a no ser que estemos insteresados en alguna versión que no sea la última.

```
git clone https://github.com/firecracker-microvm/firecracker
```

4.1.4. Instalación de Firecracker.

4.1.4.1. Instalación de Firecracker desde el código fuente.

Entramos en el repositorio previamente descargado.

```
cd firecracker
```

Construimos a través de musl.

```
tools/devtool build
```

4.1.4.2. Instalación desde paquetes/repositorios.

Renombramos el binario obtenido en el punto ***4.1.2.*** y lo situamos dentro del directorio binaries de nuestro trabajo.

```
mv release-${latest}-$(uname -m)/firecracker-${latest}-$(uname -m) ../binaries/firecracker
```

Borramos el directorio intermedio descargado.

```
rm -rf release-${latest}-$(uname -m)
```

4.1.5. Actualización de Firecracker

4.1.5.1. Actualización de Firecracker desde el código fuente.

Para ello habría que repetir los pasos de los apartados ***4.1.3*** y ***4.1.4.1.***

4.1.5.2. Actualización de Firecracker desde el binario precompilado.

Para actualizar Firecracker desde el binario haría falta volver a realizar los apartados ***4.1.2*** y ***4.1.4.2.***

4.1.6. Desinstalación de Firecracker.

Tanto si se han seguido los pasos para instalar desde el código fuente como desde el repositorio los ficheros del servicio se encuentran en el mismo directorio, por tanto bastaría con borrar el mismo.

Suponiendo que nos encontramos dentro de proyecto_sta_mandommon/firecracker/

```
sudo rm -rf binaries/firecracker
```

4.2. Obtención e Instalación de Containerd.

4.2.1. Obtención e instalación de dependencias.

Descargamos los CNI Plugins para poder utilizar las funcionalidades de nerdctl que será instalado posteriormente.

```
wget
https://github.com/containernetworking/plugins/releases/download/v1.1.1/cni-plugins-linux-amd64-v1.1.1.tgz
```

En el caso de que queramos realizar una instalación a través del código fuente tendremos además que descargar e instalar las siguientes dependencias.

Containerd necesita de un containerd runtime, que en este caso será RunC.

Lo descargamos, instalamos y eliminamos los ficheros intermedios.

```
wget
https://github.com/opencontainers/runc/releases/download/v1.1.4/runc.amd64

sudo install -m 755 runc.amd64 /usr/local/sbin/runc
rm runc.amd64
```

Instalamos Go 1.18.

```
Sudo apt-get update
Sudo apt-get install go
```

Descargamos e instalamos ProtoC.

```
wget -c
https://github.com/protocolbuffers/protobuf/releases/download/v3.11.4/protoc-3.11.4-linux-x86_64.zip

sudo unzip protoc-3.11.4-linux-x86_64.zip -d /usr/local
```

Instalamos btrfs.

```
apt-get install btrfs-progs libbtrfs-dev
```

4.2.2. Obtención del binario Containerd.

Descargamos el binario de Containerd desde el repositorio de GitHub del proyecto Containerd.

```
wget
https://github.com/containerd/containerd/releases/download/v1.6.12/containerd-1.6.12-linux-amd64.tar.gz
```

4.2.3. Obtención de los ficheros fuente de Containerd.

Clonamos el repositorio

```
git clone https://github.com/containerd/containerd
```

4.2.4. Instalación de Containerd.

4.2.4.1. Instalación desde código fuente.

Una vez descargadas las dependencias, entramos en el repositorio clonado anteriormente e instalamos.

```
cd containerd
sudo make install
```

La opción install moverá el ejecutable al directorio /usr/local.

4.2.4.2. Instalación desde paquetes/repositorios.

Simplemente lo obtenemos desde los repositorios del gestor de paquetes apt, en el caso de que estemos usando Ubuntu.

```
Sudo apt-get install containerd
```

4.2.4.3. Instalación de Nerdctl.

Una vez instalado Containerd, pasamos a descargar e instalar el intérprete nerdctl.

En primer lugar, instalamos los CNI Plugins, obtenidos en el apartado

```
tar -C ~/.local -xzf /tmp/nerdctl.tar.gz libexec
echo 'export CNI_PATH=~/.local/libexec/cni' >> ~/.bashrc
source ~/.bashrc
```

Donde hemos descomprimido el fichero obtenido y situado en el directorio ~/.local, posteriormente actualizamos la variable PATH para poder llamar al comando sin ./.

Luego, descargamos el binario de Nerdcrl que coincida con la arquitectura del procesador de nuestro equipo desde el repositorio de GitHub del proyecto Containerd.

```
archType="amd64"
if test "$(uname -m)" = "aarch64"
then
    archType="arm64"
fi
wget -q
"https://github.com/containerd/nerdctl/releases/download/v${NERDCTL_VERSIO
N}/nerdctl-full-${NERDCTL_VERSION}-linux-${archType}.tar.gz" -O
/tmp/nerdctl.tar.gz
```

Creamos el directorio ~/.local/bin, y descomprimimos el archivo descargado.

```
mkdir -p ~/.local/bin
tar -C ~/.local/bin/ -xzf /tmp/nerdctl.tar.gz --strip-components 1
bin/nerdctl
```

Una vez con el ejecutable en el directorio ~/.local/bin añadimos la ruta al PATH para poder usar el comando sin necesidad de "./" y desde cualquier directorio del usuario con las siguientes órdenes.

```
echo -e '\nexport PATH="${PATH}:~/.local/bin"' >> ~/.bashrc
source ~/.bashrc
```

Por último, cambiamos el propietario del fichero ejecutable nerdctl a root.

```
sudo chown root "$(which nerdctl)"
sudo chmod +s "$(which nerdctl)"
```

A partir de este momento ya disponemos del intérprete nerdctl para acceder a las funcionalidades de Containerd.

4.2.4.4. Actualización de Containerd

4.2.4.5. Actualización de Containerd desde el código fuente.

Repetir los pasos ***4.2.3*** y ***4.2.4.1.***

En el caso de obtengamos algún problema con las dependencias, lo más conveniente es volver a realizar el paso 4.2.1 previamente.

4.2.4.6. Actualización de Containerd desde paquetes/repositorios.

Para realizar este apartado únicamente es necesario volver a ejecutar el script mencionado en el punto ***4.2.4.2.***

4.2.4.7. Desinstalación de Containerd.

Tanto si se han seguido los pasos para instalar desde el código fuente como si se han realizado los pasos para instalar a través del gestor de paquetes apt las acciones a realizar son las mismas.

Borramos el directorio /usr/local/bin/containerd.

```
Sudo rm –rf /usr/local/bin/containerd.
```

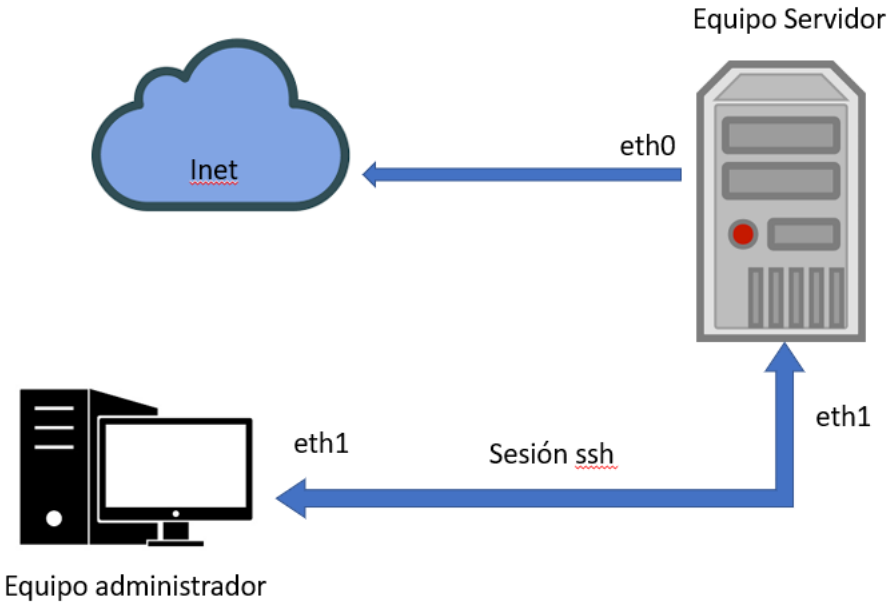
Si nos resulta mas cómodo, podemos hacerlo a través del comando purge, si hemos realizado previamente la instalación a través de paquetes, en cualquier caso, el resultado es el mismo, se borra el mismo directorio.

```
sudo purge containerd
```

5. Diseño de los escenarios de prueba

Como he explicado anteriormente, se realizarán los dos escenarios de manera concurrente para poder comprobar las diferencias de rendimiento de las dos tecnologías estudiadas.

El escenario físico presentaría la siguiente estructura.



Donde en el equipo servidor, correrán de forma concurrente los dos escenarios lógicos, uno en el que levantaremos máquinas virtuales con el kernel de Alpine Linux a través de Firecracker, y otro en el que levantaremos contenedores con imágenes de esta misma distribución de Linux.

Por otro lado, por comodidad y escalabilidad, se va a gestionar el equipo servidor desde OpenSSH a través de otro equipo.

Ambos estarán conectados por la interfaz eth1 la cual sabemos que previamente no está activada en los equipos del laboratorio.

Se procederá así para dejar libre la interfaz eth0 del equipo servidor para comprobar que las máquinas huéspedes tienen acceso a internet una vez levantadas.

5.1. Escenario 1: Máquina Virtual Alpine Linux a través de Firecracker.

En este apartado se detallarán todos los pasos a seguir hasta tener una máquina virtual Alpine Linux con funciones de red, 2 núcleos de procesador y 1GB de memoria RAM además de un sistema de archivos propio. Todo ello a través del gestor de máquinas virtuales Firecracker.

5.1.1. Escenario 1: Configuración del servidor.

(*1) En primer lugar, vamos a configurar algunas opciones de red dentro del equipo del laboratorio con el fin de lograr conectividad a nivel con la máquina virtual creada.

Primero Salvamos las reglas del cortafuegos Netfilter.

```
sudo iptables-save > iptables.rules.old
```

Configuramos un tap device virtual para dar conectividad a la máquina huésped con el equipo del laboratorio.

```
sudo ip tuntap add dev tap0 mode tap
sudo sysctl -w net.ipv4.conf.tap0.proxy_arp=1
sudo sysctl -w net.ipv6.conf.tap0.disable_ipv6=1
sudo ip addr add 172.17.0.1/16 dev tap0
sudo ip link set tap0 up
```

Añadimos las reglas necesarias al cortafuegos para permitir función de reenvío hacía internet por la interfaz eth0 del equipo del laboratorio.

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
sudo iptables -A FORWARD -i tap0 -o eth0 -j ACCEPT
```

A continuación, configuramos las características propias de la máquina virtual que queremos levantar con Firecracker.

Para ello, es necesario abrir un socket para poder comunicarnos con la API de Firecracker de la siguiente manera.

```
./../binaries/firecracker --api-sock /tmp/firecracker.socket
```

Cada vez que queramos levantar una máquina virtual debemos crear otro socket o eliminar el anterior con el siguiente comando.

```
sudo rm -f /tmp/firecracker.socket
```

Ahora sí, a través de la API.

Configuramos la Imagen del Kernel de Linux utilizada, en este caso, Alpine.

```
arch=`uname -m`

kernel_path="/home/dit/proyecto_sta_mandommon/firecracker/virtual_machines
/ubuntu-hello/hello-vmlinux.bin"

if [ ${arch} = "x86_64" ]; then
    curl --unix-socket /tmp/firecracker.socket -i \
      -X PUT 'http://localhost/boot-source'   \
      -H 'Accept: application/json'           \
      -H 'Content-Type: application/json'     \
      -d "{
            \"kernel_image_path\": \"${kernel_path}\",
            \"boot_args\": \"console=ttyS0 reboot=k panic=1 pci=off\"
       }"

elif [ ${arch} = "aarch64" ]; then
    curl --unix-socket /tmp/firecracker.socket -i \
      -X PUT 'http://localhost/boot-source'   \
      -H 'Accept: application/json'           \
      -H 'Content-Type: application/json'     \
      -d "{
            \"kernel_image_path\": \"${kernel_path}\",
            \"boot_args\": \"keep_bootcon console=ttyS0 reboot=k panic=1
pci=off\"
       }
```

En el paso anterior se ha utilizado una imagen del kernel de Linux previamente descargada y existente en el directorio virtual_machines/ubuntu-hello.

Configuramos el sistema de archivos empleado, que ha sido previamente creado.

```
rootfs_path="/home/dit/proyecto_sta_mandommon/firecracker/virtual_machines
/ubuntu-hello/hello-rootfs.ext4"
curl --unix-socket /tmp/firecracker.socket -i \
  -X PUT 'http://localhost/drives/rootfs' \
  -H 'Accept: application/json'           \
  -H 'Content-Type: application/json'     \
  -d "{
        \"drive_id\": \"rootfs\",
        \"path_on_host\": \"${rootfs_path}\",
        \"is_root_device\": true,
        \"is_read_only\": false
   }"
```

Configuramos el resto de elementos de Hardware.

```
curl --unix-socket /tmp/firecracker.socket -i  \
  -X PUT 'http://localhost/machine-config' \
  -H 'Accept: application/json'            \
  -H 'Content-Type: application/json'      \
  -d '{
      "vcpu_count": 2,
      "mem_size_mib": 1024
  }'
```

Ahora pasamos a configurar la red de la máquina virtual.

para realizar este paso es necesario haber seguido (*1) y haber obtenido el tap device con el nombre empleado en este paso.

```
curl --unix-socket /tmp/firecracker.socket \
    -X PUT 'http://localhost/network-interfaces/eth0' \
    -H 'accept:application/json' \
    -H 'Content-Type:application/json' \
    -d '{
      "iface_id": "eth0",
      "guest_mac": "AA:FC:00:00:00:01",
      "host_dev_name": "tap0"
   }
```

Donde hemos levantado la interfaz eth0 de la máquina virtual y la hemos enlazado al tap device previamente creado en la máquina del laboratorio.

Ahora sí, podemos iniciar la máquina virtual de la siguiente forma

```
curl --unix-socket /tmp/firecracker.socket -i \
  -X PUT 'http://localhost/actions'       \
  -H  'Accept: application/json'          \
  -H  'Content-Type: application/json'    \
  -d '{
      "action_type": "InstanceStart"
   }'
```

A partir de aquí disponemos de una Shell con la que podemos autenticarnos con usuario y contraseña root.

5.2. Escenario 2: Contenedor con imagen de Alpine corriendo sobre Containerd.

En este apartado se detallarán los pasos a seguir hasta obtener un contenedor con una imagen de Alpine Linux haciendo uso del intérprete nerdctl para acceder al motor Containerd.

5.2.1. Escenario 2: Configuración del servidor

Una vez instalado nerdctl obtenemos la imagen oficial de Alpine del respositorio Docker-io con el siguiente comando.

```
Nerdctl pull alpine
```

Arrancamos el contenedor.

```
nerdctl run -it --name alpine alpine
```

De forma automática ya tendríamos conectividad hacia internet a través de la interfaz eth0 del equipo del laboratorio.

Cabe mencionar que esto se lleva a cabo de igual manera que con el switch virtual que proporciona por defecto Docker.

Al instalar nerdctl nos hemos ahorrado la necesidad de realizar una configuración de red previa.

6. Puesta en funcionamiento del servicio

Dentro del entregable, y omitiendo este mismo documento, tenemos la siguiente estructura de directorios.

```
Defensa-Dominguez_Montero_Manuel.tar.gz
├── containerd
│   ├── images
│   └── scripts
│       ├── containerd_installation.sh
│       ├── erase_alpine.sh
│       ├── nerdctl_installation.sh
│       ├── start_alpine.sh
│       ├── start_containerd.sh
│       └── uninstall_containerd.sh
├── firecracker
│   ├── binaries
│   ├── scripts
│   │   ├── docker_installation.sh
│   │   ├── erase_socket.sh
│   │   ├── firecracker_installation.sh
│   │   ├── firecracker_source_installation.sh
│   │   ├── get_example_vm.sh
│   │   ├── init_firecracker_socket.sh
│   │   ├── iptables.rules.old
│   │   ├── networking_installation.sh
│   │   ├── uninstall_firecracker.sh
│   │   └── uninstall_networking.sh
│   └── virtual_machines
│       ├── ubuntu18
│       │   ├── config_files
│       │   │   └── mv_config.json
│       │   ├── hello-rootfs.ext4
│       │   └── hello-vmlinux.bin
│       └── ubuntu-hello
│           ├── config_files
│           │   ├── net_config.sh
│           │   ├── start_vm.sh
│           │   ├── vm_config.json
│           │   └── vm_config.sh
│           ├── hello-rootfs.ext4
│           └── hello-vmlinux.bin
├── README.txt
└── tests
    ├── init_test.sh
    ├── net_test.sh
    └── pi_test.sh
```

ACLARACIÓN

Dentro del fichero tarball comprimido para la entrega de este documento, he sintetizado la siguiente estructura de directorios a fin de facilitar la administración de los servicios estudiados.

A lo largo de esta memoria me referiré a los diferentes scripts y ficheros de configuración presentes aquí.

Queda este árbol de directorios aquí para consultar en caso de dudas.

6.1. Administración del servicio Firecracker.

6.1.1. Arranque de Firecracker como servicio al inicio del sistema.

En en caso, no voy a instalar Firecracker como servicio para que arranque al iniciar el sistema, ya que a diferencia de Containerd, Firecracker si me permite configurar en profundidad las máquinas virtuales que voy a levantar, por ello, prefiero controlar el arranque de las mismas de forma manual.

6.1.2. Administración de Firecracker.

Como se puede ver en el árbol de directorios, dentro de la carpeta Firecracker podemos encontrar el directorio scripts, en el cual se encuentran los ficheros Shell script que he elaborado para sintetizar y automatizar las tareas propias de la administración de Firecracker.

- Inicio del socket de la API: init_firecracker_socket.sh
- Eliminación del socket: erase_socket.sh
- Configuración de red del equipo servidor: networking_installation.sh
- Descarga del la imagen del kernel de Linux y sistema de ficheros empleados: get_example_vm.sh
- Desmantelamiento del sistema de red virtual del equipo servidor: uninstall_networking.sh

Así como dentro del directorio virtual_machines, he situado las carpetas correspondientes a las diferentes máquinas virtuales que se van a emplear.

Dentro de cada una de esas carpetas, se encuentra el directorio config_files, donde he situado los Shell scripts y ficheros JSON de configuración que he elaborado con el fin de automatizar la configuración las máquinas virtuales.

- Configuración de imagen del kernel, sistema de archivos, así como los aspectos principales del hardware: vm_config.sh
- Configuración a nivel de red de la máquina virtual: net_config.sh
- Arranque de la máquina virtual: start_vm.sh

6.1.3. Gestión de registros de Firecracker.

En primer lugar creamos el fichero de registro.

```
touch logs.file
```

Utilizamos la API para configurar el Logger de Firecracker, a través de la siguiente orden.

```
curl --unix-socket /tmp/firecracker.socket -i \
    -X PUT "http://localhost/logger" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -d "{
            "log_path": "logs.fifo",
            "level": "Warning",
            "show_level": false,
            "show_log_origin": false
    }"
```

Ahora tan solo es necesario consultar el fichero previamente creado para monitorizar los registros de Firecracker.

6.1.4. Aspectos de seguridad de Firecracker

- Se recomienda una buena configuración del cortafuegos a la hora de permitir el direccionamiento IP a través de la máquina anfitriona hacía otras redes.
- Hay que tener en cuenta que si vulneran una de las MMV podrían acceder a la máquina anfitriona a través del tap device.
- Es aconsejable usar el Jailer ofrecido por Firecracker si se va a usar esta tecnología en producción. Este mecanismo hace uso de chroot y namespaces para encapsular la máquina dentro del sistema de archivos de la máquina anfitriona.

6.2. Administración del servicio Containerd.

6.2.1. Arranque de Containerd como servicio al inicio del sistema.

Por la cantidad de documentación existente y por su facilidad de configuración, voy a utilizar systemd para gestionar el servicio Containerd.

En primer lugar, descargamos el fichero de servicio.

```
Wget -P /usr/local/lib/systemd/system/containerd.service
https://raw.githubusercontent.com/containerd/containerd/main/containerd.service
```

Recargamos systemd y habilitamos el servicio.

```
Systemctl daemon-reload
systemctl enable -now containerd
```

6.2.2. Administración de Containerd.

De igual manera que con Firecracker, a fin de automatizar las tareas de administración el día de la defensa, he elaborado una serie de scripts que abarcan los aspectos más importantes de la administración de este servicio.

Dentro del directorio Containerd podremos encontrar la carpeta scripts, donde tenemos:

- Arranque de Containerd: start_containerd.sh
- Instalación de Containerd: containerd_installation.sh
- Desinstalación de Containerd: uninstall_containerd.sh
- Obtención de una imagen de Alpine Linux y arranque de un contenedor con la misma: start_alpine.sh
- Borrado del contenedor Alpine: erase_alpine.sh
- Instalación del intérprete nerdctl: nerdctl_installation.sh

Todos los scripts mencionados en este apartado (6.2.) Se han explicado detalladamente paso a paso en los procesos de instalación de los servicios, así como en el despliegue de los escenarios 1 y 2. Por lo que no voy a extenderme en el contenido de los mismos.

6.2.3. Gestión de Registros de Containerd.

A través de systemd, si es que hemos escogido instalar Containerd como servicio, podemos hacer uso de la herramienta Journalctl y así obtener los elementos de registro.

```
Journalctl –u containerd
```

6.2.4. Aspectos de seguridad de Containerd.

- Aunque permite encapsulación, hay que tener en cuenta que, por defecto, las imágenes de contenedores no presentan un sistema de archivos propio, sino que guardan el contenido de las mismas dentro de un directorio presente en el sistema de archivos de la máquina anfitriona.

- Se recomienda hacer uso de namespaces y chroot para encapsular los procesos asociados a los contenedores en ambientes de producción, ya que este motor de contenedores ofrece esta característica.

- Al igual que con Firecracker, Containerd ofrece dispositivos virtuales de red, por lo que, llevada a cabo una mala configuración, si un contenedor es vulnerado, el sistema puede verse atacado a través de la red local virtual.

7. Tests y Sandboxing.

7.1. Comparativas de Rendimiento

Desde ahora, por comodidad, cuando se hable de instancias nos estaremos refiriendo tanto a MMV como a contenedores indiferentemente.

El objetivo de este apartado es ver si hay una diferencia notable entre el rendimiento de uno y otro de los escenarios.

Para ello he elaborado una serie de scripts para llevar a cabo pruebas de eficiencia en el uso de los recursos hardware, así como pruebas de capacidad de procesamiento.

7.1.1. Arranque de varias instancias

En este test, programado en el fichero init_test.sh, he llevado a cabo pruebas de arranque de múltiples instancias de forma concurrente para comprobar la velocidad de inicio y el uso de los recursos del equipo anfitrión.

7.1.1.1. Tabla Firecracker.

Nº de instancias	Tiempo de arranque	Recursos servidor
1	100ms	RAM: 40% Uso de CPU: 20%
5	400ms	RAM: 40% Uso de CPU: 25%
20	1,5s	RAM: 60% Uso de CPU: 60%

7.1.1.2. Tabla Containerd.

Nº de instancias	Tiempo de arranque	Recursos servidor
1	1s	RAM: 40% Uso de CPU: 25%
5	10s	RAM: 50% Uso de CPU: 35%
20	15s	RAM: 80% Uso de CPU: 80%

El porcentaje de recursos empleados lo he obtenido a partir del comando Top y los tiempos de arranque a través de el comando “time”.

Para la realización de los test, se ha empleado un equipo anfitrión con 12GB de RAM y CPU Intel de 4 núcleos.

7.1.2. Test de capacidad de procesamiento.

En este test, he programado un sencillo Shell script (pi_test.sh) que calcula los n primeros decimales del número π.

Para ello he utilizado el comando “time” para saber el tiempo empleado en realizar la orden por parte del intérprete de comandos.

El algoritmo que he empleado es el de la serie de Leibniz-Gregory, el cual responde a la siguiente expresión.

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

Tabla comparativa de tiempos de ejecución		
Numero de decimales	***Tiempo Firecracker***	***Tiempo Containerd***
150	***0s***	***0s***
500	***1s***	***2s***
500	***6s***	***6s***
700	***10s***	***12s***
1000	***16s***	***16s***
10000	***2min***	***1,45min***

Como podemos ver los tiempos de ejecución son muy parecidos, la máquina virtual se ha configurado con apenas un 1GB de RAM y 2 vCPU’s.

El contenedor de Alpine no se ha configurado a nivel de recursos de hardware requeridos, por lo que parte con la configuración por defecto.

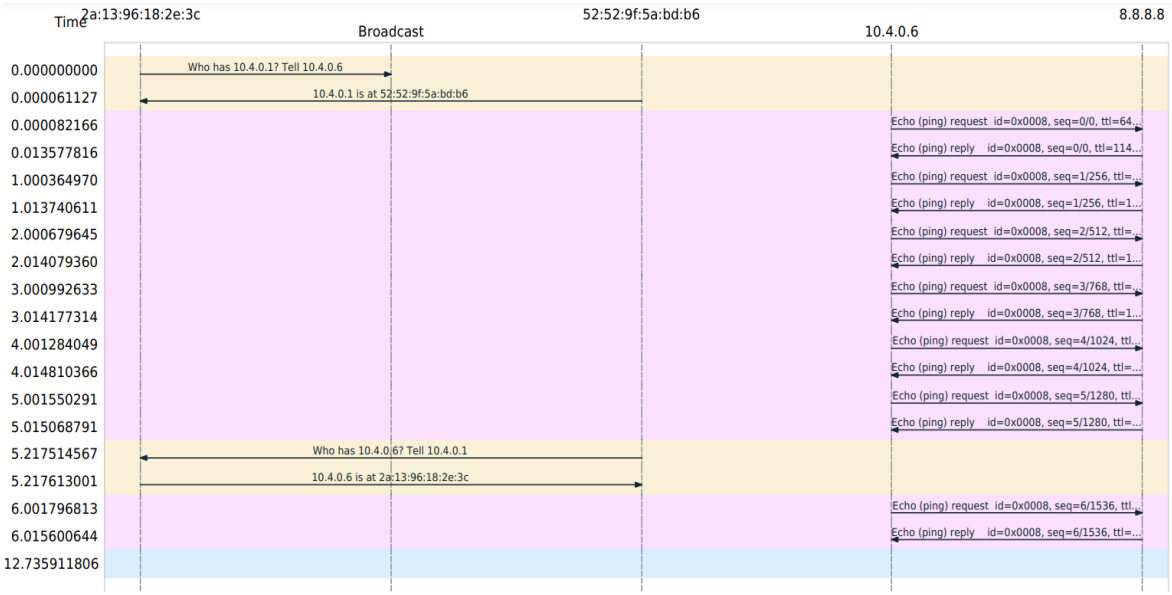
7.1.3. Comparativa del paso de mensajes por la red

7.1.3.1. Mensajes de red Firecracker.

Este es el diagrama de paso de mensajes observado tras capturar la interfaz virtual tap0 a través de Wireshark realizando un ping desde la máquina huésped a la anfitriona.

7.1.3.2. Mensajes de red Containerd.

Este es el diagrama de paso de mensajes observado tras capturar la interfaz virtual nerdctl0 mencionada en el paso de instalación de Containerd.



En este último diagrama, se ve como realizamos ping al servidor DNS 8.8.8.8 de Google, recibiendo respuesta.

8. Deficiencias del servicio

Dada la naturaleza de las tecnologías estudiadas, las cuales sirven de soporte para el despliegue de otras aplicaciones, así como, las dificultades de configuración sobre todo de Firecracker, no me ha sido posible por tiempo el realizar un escenario que presente una funcionalidad llamativa.

Por otro lado, de cara al ámbito de Firecracker, me gustaría destacar la falta de un repositorio o registro público de donde descargar y distribuir kernels simplificados adaptados para Firecracker.

Podemos descargar imágenes de ejemplo del respositorio oficial de AWS pero son escasas en número y funcionalidad.

Siendo la capacidad de distribución de imágenes uno de los fuertes de Docker, ésta podría servir de ejemplo para tecnologías como Firecracker

9. Ampliaciones/mejoras del servicio

Me hubiera gustado probar Kubernetes sobre los escenarios desplegados y levantar algún aplicativo de alta disponibilidad como un servidor Web distribuido en un clúster.

Sería de interés realizar un estudio de cuál de las tecnologías es más eficiente en entornos de alta concurrencia.

Sin embargo, Kubernetes fue materia de un trabajo en el pasado año y no he podido usarlo en este.

10. Incidencias y principales problemas detectados

Respecto a Firecracker:

- Pese a la potencia que ofrece Firecracker en términos de rendimiento y modularidad, es difícil encontrar documentación fiable y de calidad.

Respecto a Containerd:

- Docker sigue siendo tan popular que resulta muy difícil encontrar documentación fiable y de calidad sobre el uso de Containerd, sobre todo del uso adecuado de su inérprete de comandos por defecto ctr.

- Dado que Containerd nace de la exigencia de Kubernetes de lograr una mejor integración entre si mismo y el motor de contenedores Docker a través de la CRI-O API, Containerd no está enfocado a usarse a través de intérprete de comandos, por lo que sus funcionalidades a través de los mismos es limitada.

11. Resumen y Conclusiones

Durante gran parte del tiempo empleado en este trabajo he intentado desplegar el servicio ofrecido por el equipo Containerd-Firecracker, cuyo repositorio en GitHub es el siguiente.

https://github.com/firecracker-microvm/firecracker-containerd

Dicho proyecto, consiste en dotar al motor de contenedores Containerd de la capacidad de levantar sus contenedores sobre MMV de forma transparente al usuario, cambiando el container runtime por defecto (RunC), por uno derivado del mismo que tuviese conexión a través de una API con Firecracker.

Pese a proponer una solución a mi parecer bastante llamativa presentaba los siguientes inconvenientes:

- Muy poca documentación debido a que ese tipo de arquitecturas las han adoptado otros proyectos mucho más extendidos como Kata Containers.
- Una difícil gestión de dependencias debido a la complejidad del sistema propuesto.
- Imposibilidad de realizar el trabajo en ninguna de las máquinas virtuales ofrecidas en la asignatura debido a los problemas que presenta la virtualización anidada. Por ello he tenido que copiar el sistema de archivos desde una de las máquinas virtuales y desplegarlo en un equipo real para poder trabajar desde casa.

No ha sido hasta dos días antes de la entrega de este documento, cuando decidí estudiar por separado Firecracker y Containerd abandonando la idea inicial, Con el fin de abarcar todos los conceptos que en el título del trabajo se me requerían.

He de decir que esta decisión ha conseguido que profundice mucho más de lo que el proyecto anterior me hubiese permitido en ambas tecnologías.

Conocía con anterioridad tecnologías como Qemu y Vagrant que facilitan el uso de máquinas virtuales y su personalización, pero no tenía ni idea del potencial de Firecracker.