The first Algerian Linux distribution

FennecOS

AZIL Hamza, HADJAZI Mohammed Hisham

University of SAIDA

November 22, 2024

1. Introduction
2. The Setup Environment
3. Building
4. Productions VMs
5. Testing our Machines with Jenkins & sshpass

1 Introduction
AZIL Hamza
HADJAZI Mohammed Hisham

2 The Setup Environment

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

1 Introduction
AZIL Hamza
HADJAZI Mohammed Hisham

2 The Setup Environment

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

Who we are !

- AZIL Hamza
- Embedded System/Linux Engeneer
- C/C++ developper
- Chess, cooking and comedy
- https://github.com/hamzastar7
- feel free to email me on `azilhamza@gmail.com`

1 Introduction
AZIL Hamza
HADJAZI Mohammed Hisham

2 The Setup Environment

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

Who we are !

- HADJAZI Mohammed Hisham
- Master's in CS, specialized in Networks and Security
- Network/System Admin at ESI SBA
- Addicted to Containers and Kubernetes
- https://github.com/the6wheels

1 Introduction

2 The Setup Environment
- Work Station Hamza
- Work Station Hisham

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

1 Introduction

2 The Setup Environment
Work Station Hamza
Work Station Hisham

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

- Debian12 (kernel6.1) in DualBoot with ubuntu20.04.6 LTS(kernel5.15)
- build-essentiel, asciinema, screen
- QEMU (x86_64 & arch)

1 Introduction

2 The Setup Environment
- Work Station Hamza
- Work Station Hisham

3 Building

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

- Ubuntu 20.04.6 LTS.
- Virtual Box 6.1.
  - VM with live disto of Ubuntu 20.04.6 LTS (8 cores 24GB RAM) with openssh-server.
- Jenkins for tests automation.
- sshpass for ssh automation.
- Ansible and Ansible modules for shell script automation.

1 Introduction

2 The Setup Environment

3 Building
- HelloWorld!
- Linux From Scratch

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

1 Introduction

2 The Setup Environment

3 Building
- HelloWorld!
- Linux From Scratch

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

HelloWorld

When I wanted to create my OS from scratch for the first time, my question was **"where all of this started?"** How to make every piece of the OS and how understand what it does. I wanted to launch a HelloWorld instead of the kernel, and to make this possible, I went through a simple HelloWorld example.

```
hamza@FennecHOST: ~/hello 105x24
hamza@FennecHOST:~/hello$ cat hello.c
#include<stdio.h>
int main(int , char **){

printf("\nSALAM ALYKOM\n");

sleep(9999999);
}
hamza@FennecHOST:~/hello$ gcc -o init hello.c --static -w
hamza@FennecHOST:~/hello$ file init
file: Compiled magic version [545] does not match with shared library magic version

init: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked,
ec31c613cab0c6eeb5c49e26c8d6ea27, for GNU/Linux 3.2.0, not stripped
hamza@FennecHOST:~/hello$
```

HelloWorld!

we compile the hello to init (the kernel recognize as a starting point)

```
hamza@FennecHOST:~/hello$ find . | cpio -o -H newc | gzip > hello_rmfs.cpio.gz
12550 blocks
```

we will emulate the result with QEMU

```
[    1.852398] netconsole: network logging started
[    1.853351] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[    1.859804] kworker/u4:0 (48) used greatest stack depth: 15032 bytes left
[    1.868045] Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    1.868703] Loaded X.509 cert 'wens: 61c038651aabdcf94bd0ac7ff06c7248db18c600'
[    1.869541] ALSA device list:
[    1.870696]   No soundcards found.
[    1.871740] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[    1.872101] cfg80211: failed to load regulatory.db
[    1.937314] Freeing unused kernel image (initmem) memory: 2764K
[    1.937746] Write protecting the kernel read-only data: 26624k
[    1.939301] Freeing unused kernel image (rodata/data gap) memory: 1184K
[    2.099485] x86/mm: Checked W+X mappings: passed, no W+X pages found.
[    2.100221] tsc: Refined TSC clocksource calibration: 2903.983 MHz
[    2.100613] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x29dbf5a3d38, max_idle_ns: 44079525155
[    2.100883] clocksource: Switched to clocksource tsc
[    2.101237] Run /init as init process

SALAM ALYKOM
[    2.311025] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3

tsl                                                    hamza@FennecHOST: ~/hello 100x1
hamza@FennecHOST:~/hello$ sudo kill $(pgrep qemu)
```

minimal

from this point we wanted to create a very basic rootfs that could rebuild it self from source we found that for installing a basic linux system that can build it self from the code source we need about 7 packages: linux(kernel), busybox, uClibc, gcc, binutils, make & bash

Building Linux Kernel

we get the latest code source from github at linus trovad repo:
then we clone it

```
hamza@FennecHOST:~$ git clone https://github.com/torvalds/linux.git
fatal: destination path 'linux' already exists and is not an empty directory.
hamza@FennecHOST:~$ cd linux
hamza@FennecHOST:~/linux$ make menuconfig
```

```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
```

```
.config - Linux/x86 6.12.0-rc1 Kernel Configuration
  Debugging kernel
                                General setup
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing
  excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <
  module capable

        [ ] Compile also drivers which will not load
        [ ] Compile the kernel with warnings as errors
        () Local version - append to kernel release
        [*] Automatically append version information to the version string
        () Build ID Salt
            Kernel compression mode (Gzip)  --->
        () Default init path
        ((none)) Default hostname
        [*] System V IPC
        [*] POSIX Message Queues
        [ ] General notification queue
        [*] Enable process_vm_readv/writev syscalls
        [ ] uselib syscall (for libc5 and earlier)
        [*] Auditing support
            IRQ subsystem  --->
            Timers subsystem  --->
            BPF subsystem  --->
            Preemption Model (Voluntary Kernel Preemption (Desktop))  --->
```

Configuring the Kernel

```
[*] CPU isolation
    RCU Subsystem  --->
< > Kernel .config support
<M> Enable kernel headers through /sys/kernel/kheaders.tar.xz
(18) Kernel log buffer size (16 => 64KB, 17 => 128KB)
(12) CPU kernel log buffer size contribution (13 => 8 KB, 17 => 128KB)
[ ] Printk indexing debugfs interface
    Scheduler features  --->
[ ] Memory placement aware NUMA scheduler
v(+)
```

Configuring the Kernel

```
< >    I2C/SMBus Test Stub
[*]    I2C slave support
<M>      I2C eeprom slave driver
<M>      I2C testunit driver
```

Configuring the Kernel

```
hamza@FennecHOST:~/linux$ make menuconfig
configuration written to .config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

hamza@FennecHOST:~/linux$ make
```

Configuring the Kernel

```
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#3)
hamza@FennecHOST:~/linux$
```

Configuring the Kernel

```
hamza@FennecHOST:~/linux$ sudo make install
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/dracut 6.12.0-rc7-00212-g4a5df3796467 /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467
depmod: ERROR: could not open directory /lib/modules/6.12.0-rc7-00212-g4a5df3796467: No such file or directory
depmod: FATAL: could not search modules: No such file or directory
dracut: Generating /boot/initrd.img-6.12.0-rc7-00212-g4a5df3796467
dracut: Cannot find module directory /lib/modules/6.12.0-rc7-00212-g4a5df3796467/
dracut: and --no-kernel was not specified
run-parts: /etc/kernel/postinst.d/dracut exited with return code 1
make[1]: *** [arch/x86/Makefile:320: install] Error 1
make: *** [Makefile:224: __sub-make] Error 2
```

Configuring the Kernel

```
hamza@FennecHOST:~/linux$ sudo make modules_install
  SYMLINK /lib/modules/6.12.0-rc7-00212-g4a5df3796467/build
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/modules.order
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/modules.builtin
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/modules.builtin.modinfo
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/fs/efivarfs/efivarfs.ko
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/drivers/thermal/intel/x86_pkg_temp_thermal.ko
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/net/netfilter/nf_log_syslog.ko
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/net/netfilter/xt_mark.ko
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/net/netfilter/xt_nat.ko
  INSTALL /lib/modules/6.12.0-rc7-00212-g4a5df3796467/kernel/net/netfilter/xt_LOG.ko
```

testing in the current machine

```
hamza@FennecHOST:~/linux$ sudo make install
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/dracut 6.12.0-rc7-00212-g4a5df3796467 /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467
dracut: Generating /boot/initrd.img-6.12.0-rc7-00212-g4a5df3796467
dracut: Disabling early microcode, because kernel does not support it. CONFIG_MICROCODE_[AMD|INTEL]!=y
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.12.0-rc7-00212-g4a5df3796467 /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.12.0-rc7-00212-g4a5df3796467 /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467
Generating grub configuration file ...
Found background image: /usr/share/images/desktop-base/desktop-grub.png
Found linux image: /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467
Found initrd image: /boot/initrd.img-6.12.0-rc7-00212-g4a5df3796467
Found linux image: /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467.old
Found initrd image: /boot/initrd.img-6.12.0-rc7-00212-g4a5df3796467
Found linux image: /boot/vmlinuz-6.1.0-10-amd64
Found initrd image: /boot/initrd.img-6.1.0-10-amd64
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
```

testing in the current machine

```
hamza@FennecHOST:~/linux$ uname -ra
Linux FennecHOST 6.1.0-10-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.38-1 (2023-07-14) x86_64 GNU/Linux
```

testing in the current machine

```
hamza@FennecHOST:~$ uname -ar
Linux FennecHOST 6.12.0-rc7-00212-g4a5df3796467 #4 SMP PREEMPT_DYNAMIC Mon Nov 18 17:19:21 CET 2024
x86_64 GNU/Linux
hamza@FennecHOST:~$
```

testing in the current machine

```
hamza@FennecHOST: ~ 136x29
T:~$ sudo grep 'vmlinuz' /boot/grub/grub.cfg
/boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467 root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro  quiet
linux   /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467 root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro  quiet
linux   /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467 root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro single
linux   /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467.old root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro  quiet
linux   /boot/vmlinuz-6.12.0-rc7-00212-g4a5df3796467.old root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro single
linux   /boot/vmlinuz-6.1.0-10-amd64 root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro  quiet
linux   /boot/vmlinuz-6.1.0-10-amd64 root=UUID=2a4202d0-feab-4a48-b96d-a97532070320 ro single
T:~$
```

minimal linux live

- getting and building linux, glibc, busybox
- Creating iso
- Generating Image
- testing it on QEMU

minimal linux live

```
#-----------minimal-------------
mkdir -p build/
tar xvf linux-5.18.3.tar.xz
cd linux-5.18.3
make defconfig
make --perfix=../build
cd ..

wget https://ftp.gnu.org/gnu/glibc/glibc-2.35.tar.bz2
tar xvf glibc-2.35.tar.bz2
cd xvf glibc-2.35
./configure --prefix=  --with-headers=../build/include \
    --without-gd --without-selinux\
    --disable-werror CFLAGS="$CFLAGS"
make -j16 && make install DESTDIR=../build/glibc
cd ..

mkdir -p ./{sysroot,rootfs}
cp -r ../build/glibc/* sysroot/
cp -r ../build/include sysroot/
SYSROOT=$(realpath sysroot/)
mkdir -p $SYSROOT/usr
```

minimal linux live

```
mkdir ../rootfs
ROOTFS=$(realpath ../rootfs)
cp -r busybox ../rootfs
cp -r $ROOTFS/bin/busybox
cp $SYSROOT/lib/ld-linux* $ROOTFS/lib64
cp $SYSROOT/lib/libm.so.6 $ROOTFS/lib
cp $SYSROOT/lib/libc.so.6 $ROOTFS/lib
cp $SYSROOT/lib/libresolv.so.2 $ROOTFS/lib
set +e
strip -g \
  $ROOTFS/bin/* \
  $ROOTFS/sbin/* \
  $ROOTFS/lib/* \
  2>/dev/null
set -e
find . | cpio -R root:root -H newc -o | xz -9 --check=crc32 > ../rootfs.cpio.xz

#------creating iso-------
xorriso -as mkisofs \
    -isohybrid-mbr ../syslinux/syslinux-*/bios/mbr/isohdpfx.bin \
    -c boot/syslinux/boot.cat \
    -b boot/syslinux/isolinux.bin \
```

minimal linux live

```
    #----------GENERATE IMAGE---------

cd $ROOTFS
mkdir image
cp -r $ROOTFS/*
cd image && tar -zcf image.tgz *

#-----runing the live distro------

qemu-system-x86_64 -m 128M -cdrom live.iso -boot d -vga std
```

minimal linux live

```
/hamza/KINGSTON$ qemu-system-x86_64 -m 128M -cdrom live.iso -boot d -vga std
```

QEMU - Press Ctrl+Alt+G to release grab

Machine View

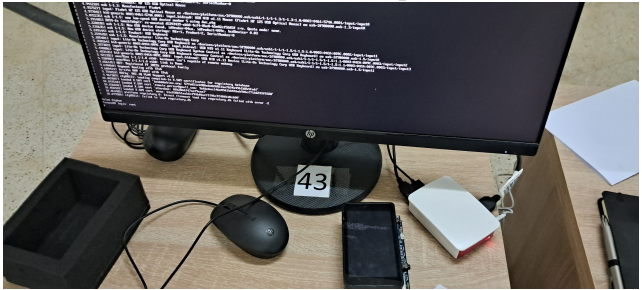
```
[    2.281974] PM:   Magic number: 0:630:997
[    2.282483] printk: console [netcon0] enabled
[    2.282537] netconsole: network logging started
[    2.284309] cfg80211: Loading compiled-in X.509 certificates for regulatory d
atabase
[    2.346629] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    2.347634] platform regulatory.0: Direct firmware load for regulatory.db fai
led with error -2
[    2.347804] cfg80211: failed to load regulatory.db
[    2.348457] ALSA device list:
[    2.348529]   No soundcards found.
[    2.436256] Freeing unused kernel image memory: 1316K
[    2.436519] Write protecting the kernel read-only data: 20480k
[    2.438140] Freeing unused kernel image memory: 2004K
[    2.438736] Freeing unused kernel image memory: 900K
```

minimal linux live

```
/ # ls
README  dev     lib     proc    sbin    tmp     var
bin     etc     lib64   root    sys     usr
/ # pzd
/bin/sh: pzd: not found
/ # pwd
/
/ # whoami
whoami: unknown uid 0
/ # _
```

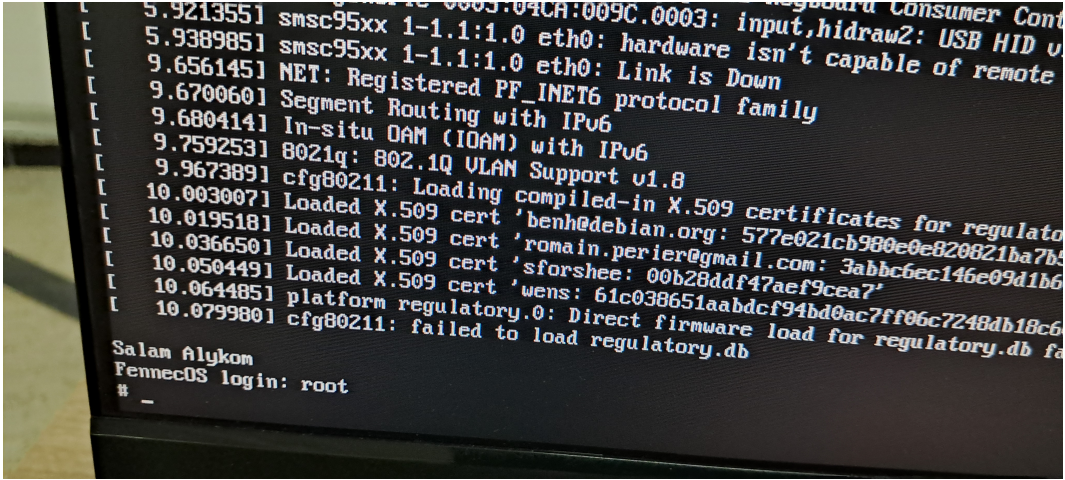
ARM Version on Pi3

Due to cross-compilation complications we choose to call for the help of other utilities that can provide cross-compile toolchain.



We have created an ARM OS using **Buildroot** (pre-built make files for cross-comp) and successfully booted it to a Pi3 device after converting the image with dd command to the SD card.

ARM Version on Pi3



NEW USB Driver

we choose to implement a very small driver that could be usufull in the future a driver that:

- identify a new usb
- identify devices containing LEDs (keybord as exemple)
- toogling thoos LEDs

NEW USB Driver

a basic exemple that interigate with the leds of the keybord

```
// Define the notifier block to register with the USB subsystem
static struct notifier_block usb_nb = {
    .notifier_call = usb_notify,
};

// Module initialization
static int __init usb_led_init(void)
{
    printk(KERN_INFO "USB LED Driver module loaded\n");

    // Register with the USB subsystem to receive device add/remove notifications
    usb_register_notify(&usb_nb);

    return 0;
}

// Module exit
```

NEW testing the driver

after building the driver we should insert the driver using insmod

```
sudo rmmod test.ko
sudo insmod test.ko
```

after inserting the module we could see diffrent prints on dmesg

```
[ 7520.573248] USB LED Driver module unloaded
[ 7523.605160] USB LED Driver module loaded
```

1 Introduction

2 The Setup Environment

3 Building
- HelloWorld!
- Linux From Scratch

4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

LFS

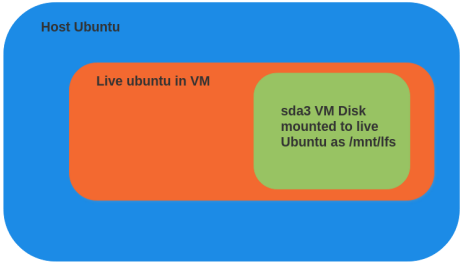
Linux From Scratch

Version 12.2-systemd

Published September 1st, 2024

We have used LFS book to make a our minimal linux distobution FennecOS which was used later on the rest of our VM's as a base system.

Method of building



- We boot via a live OS inside our VM.
- we create 3 partitons in our VM disk (boot,rootfs,swap)(ex2,ex4,swap-file).
- we install a list of required tools for building and compiling our LFS, these tools are installed on the live RAM disk and belong to the live distro.

Final minimal OS

```
-bash-5.2# name -r
-bash: name: command not found
-bash-5.2# cat /proc/version
Linux version 6.10.5 (root@ubuntu) (gcc (GCC) 14.2.0, GNU ld (GNU Binutils) 2.43
.1) #1 SMP PREEMPT_DYNAMIC Sun Nov 17 04:51:17 CET 2024
-bash-5.2#
```

After rebooting the vm and removing the ISO of our live Ubuntu, we are introduced to our final minimal OS using Linux from scratch.

Why use virtualization route

- Create multiple copies of our OS.
- Snapshots for recovery of failures.
- Network options like bridge interfaces and creation of separate network ranges.
- Using VM tools to export images.

1 Introduction

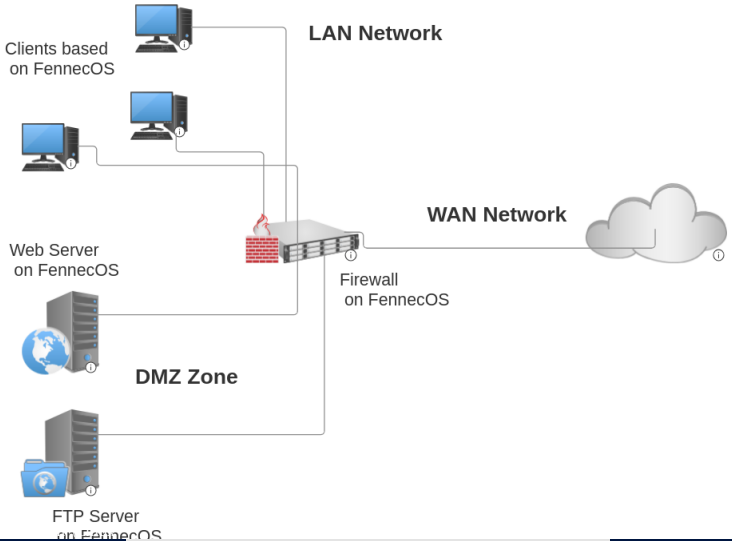
2 The Setup Environment

3 Building

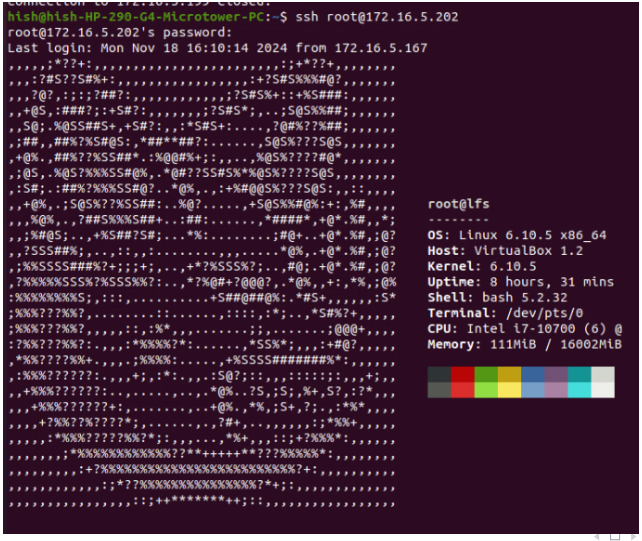
4 Productions VMs

5 Testing our Machines with Jenkins & sshpass

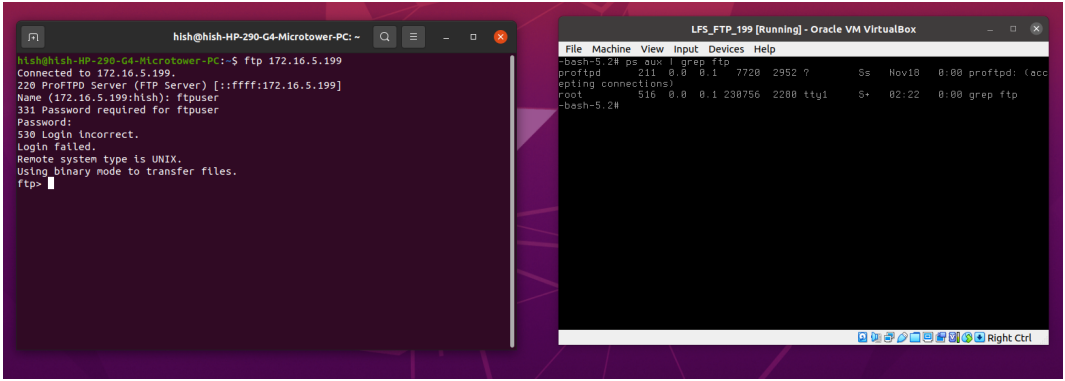
Network Topology



FennecOS clients

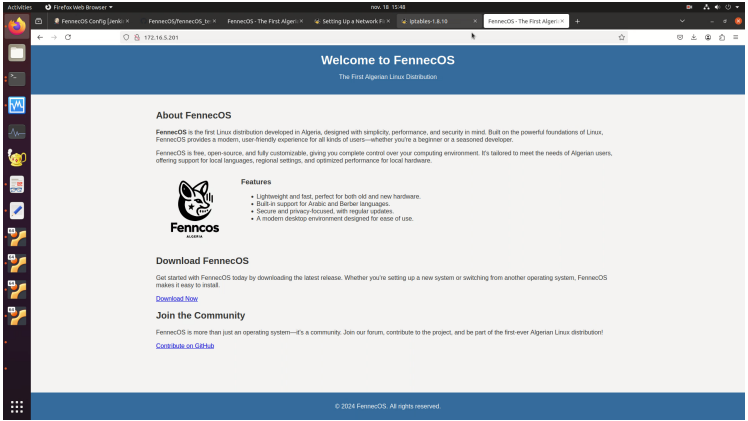


FTP Server



FTP Server using portftpd.

Web Server



A simple static landing page using Apache web server.

Firewall

- We used IPTables.
- Required Kernel modification to add NETFILTER.
- DMZ zone for FTP server and Web server.
- WAN network with bridged interface.
- LAN network on third interface.

IPtables

- LAN_IF="eth0"
- WAN_IF="enp0s"
- DMZ_IF="eth1"
- iptables -P INPUT DROP
- iptables -P FORWARD DROP
- iptables -P OUTPUT ACCEPT
- iptables -A INPUT -i lo -j ACCEPT
- iptables -A OUTPUT -o lo -j ACCEPT
- iptables -A INPUT -p tcp –dport 21 -j ACCEPT
- iptables -A OUTPUT -p tcp –sport 21 -j ACCEPT
- iptables -A INPUT -p tcp –dport 80 -j ACCEPT
- iptables -A OUTPUT -p tcp –sport 80 -j ACCEPT
- iptables -A INPUT -p tcp –dport 443 -j ACCEPT
- iptables -A OUTPUT -p tcp –sport 443 -j ACCEPT
- iptables -A INPUT -p tcp –dport 22 -j ACCEPT
- iptables -A OUTPUT -p tcp –sport 22 -j ACCEPT
- iptables -t nat -A POSTROUTING -o $WAN_IF -j MASQUERADE

1 Introduction

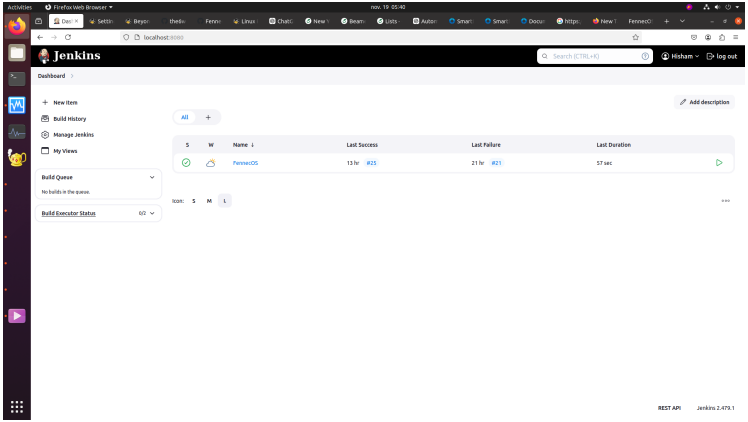
2 The Setup Environment

3 Building

4 Productions VMs

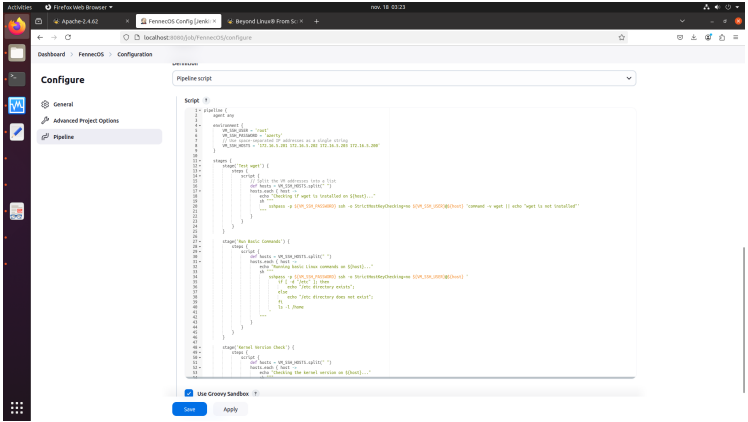
5 Testing our Machines with Jenkins & sshpass

Jenkins new pipeline



We used Jenkins with sshpass and virualbox to automate the testing process of our machines.

Jenkins pipeline script



here we defined a list of tests that Jenkins will perform to test our machines.

Pipeline Features

- The pipeline loops over the **VM_SSH_HOSTS** list to perform tests on each specified host.
- The use of **sshpass** ensures password-based SSH authentication.
- Each stage checks a specific aspect of the system and reports the results.

Test wget

```
command -v wget || echo "wget is not installed"
```

Run Basic Commands

```
if [ -d "/etc" ]; then echo "/etc directory exists"; else
    echo "/etc directory does not exist"; fi
ls -l /home
```

Kernel Version Check

```
uname -r
```

Kernel Logs Check

```
dmesg | grep -i "error\|warn" || echo "No kernel errors
    or warnings"
```

CPU Info Check

```
cat /proc/cpuinfo
```

Memory Info Check

```
cat /proc/meminfo
```

Network Connectivity Test

```
ping -c 4 8.8.8.8 || echo "Network connectivity test
    failed"
```

Tests Finished successfully in Jenkins

localhost:8080/job/FennecOS/27/console

Dashboard > FennecOS > #27

```
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=36.123 ms
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 35.145/35.593/36.123/0.368 ms
[Pipeline] echo (hide)
Testing network connectivity on 172.16.5.199...
[Pipeline] sh
+ sshpass -p azerty ssh -o StrictHostKeyChecking=no root@172.16.5.199 ping -c 4 8.8.8.8 || echo "Network connectivity test failed"
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=113 time=35.756 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=35.718 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=35.940 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=35.629 ms
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 35.629/35.761/35.940/0.113 ms
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline finished
[Pipeline] echo
All tests passed!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

What are our perspectives ?

- scalable architecture with the due to the ease of containerizing and deploying our OS into Docker or kubernetes.
- very simple and helpful for educational purposes.
- Widely customizable and deployable on different machines with different architectures X86 and ARM.