Build Containers with unshare

AU: 2024 - 2025

ISIMM-Institut Supérieur d'Informatique et de Mathématiques de Monastir

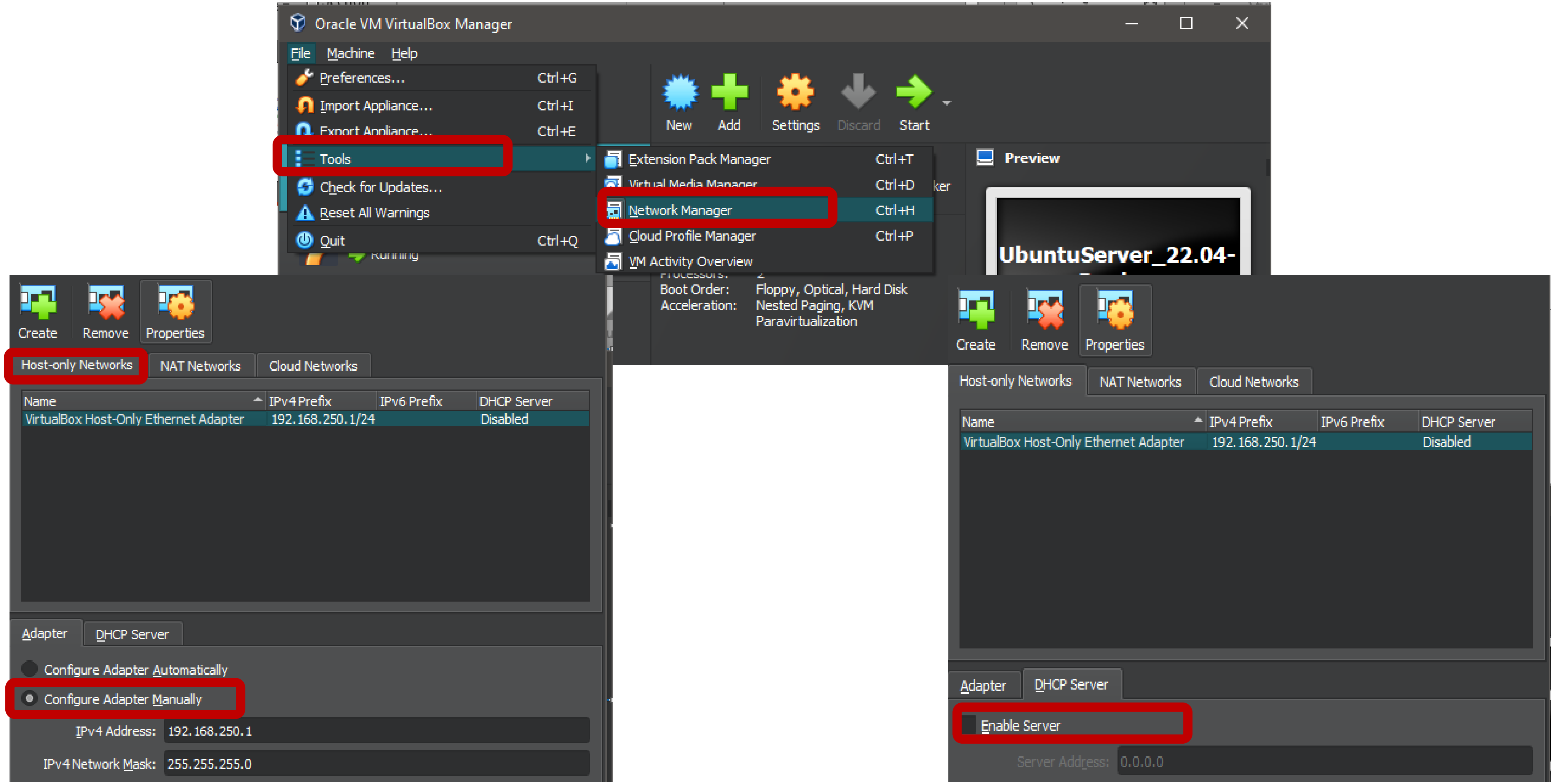
Prerequisites

- Install Linux Ubuntu Server 22.04 or download VM Images:

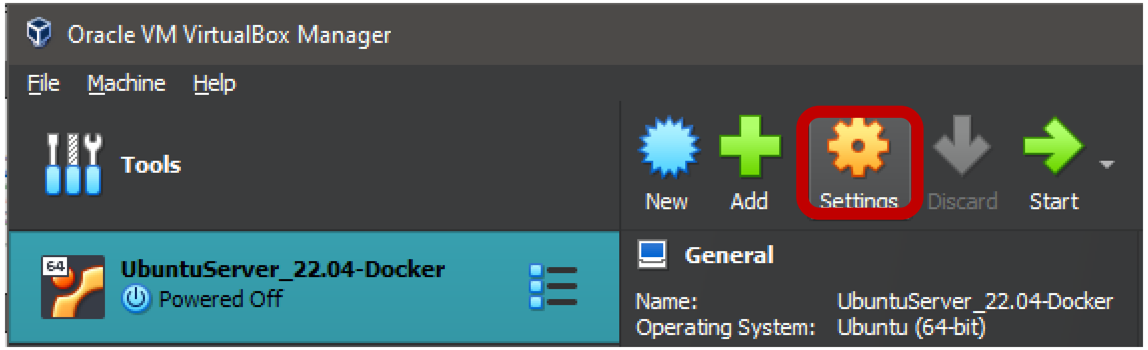
  https://www.linuxvmimages.com/images/ubuntuserver-2204/

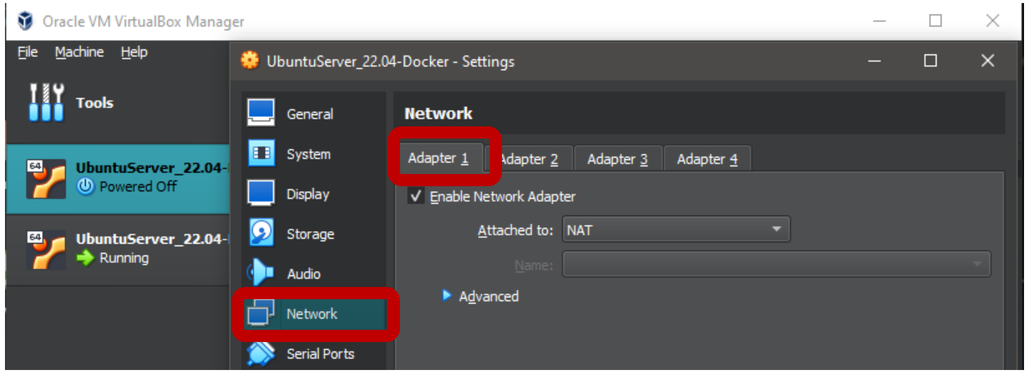
  - User/Password: ubuntu/ubuntu
- Network connections of the Linux VM:
  - Network Adapter 1 (enp0s3): NAT (Internet connection)
  - Network Adapter 2 (enp0s8): Host Only Adapter (connection with physical Machine)
  - Network Adapter 3 (enp0s8): Internal Network(connection with VMs)
- Putty installed on physical machine.

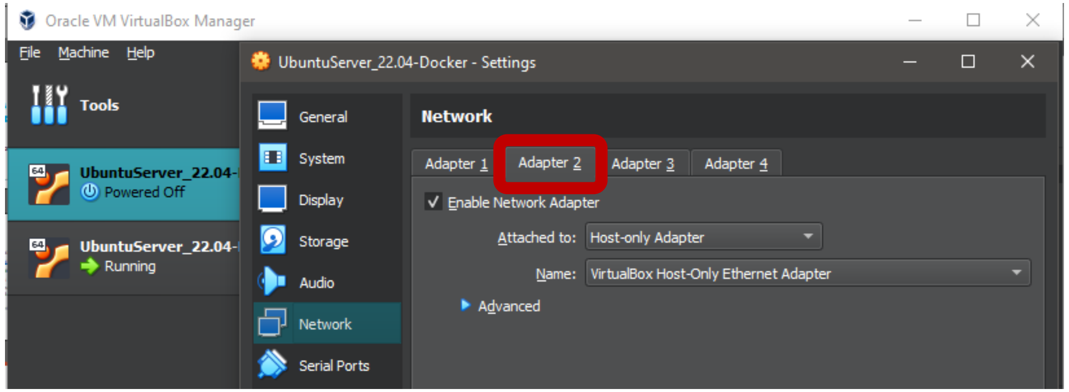
Vbox Configuration:
Add Host Only Network without DHCP

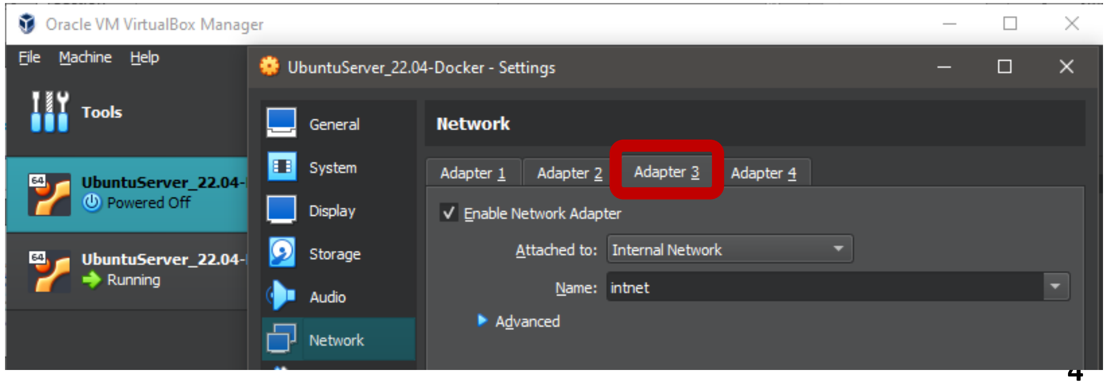


VM Network Configuration









VM IP Configuration

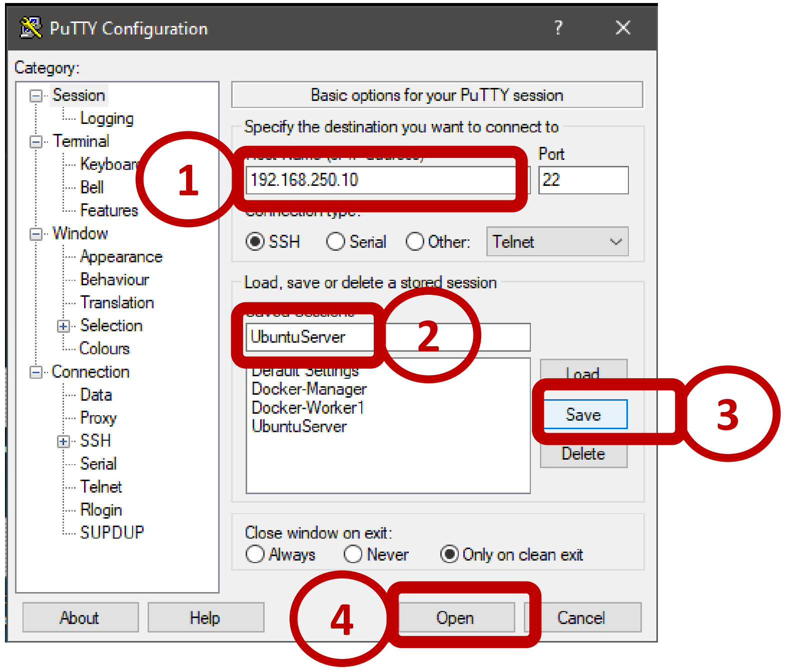
- Verify the network interfaces (enp0s3,enp0s8 and enp0s9) with: **$ ip a**
- Edit netplan configuration file: **$sudo nano /etc/netplan/00-installer-config.yaml**

```yaml
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false
      addresses:
       - 192.168.250.10/24
    enp0s9:
      dhcp4: false
      addresses:
       - 172.31.0.1/24
  version: 2
```

Connectivity Host - VM

- Apply the new network config: **$sudo netplan apply**
- Verify the VM IP configuration: **$ ip a**
- On physical host:

- Test ping with the VM, run putty and open an ssh connection:



Creating namespaces with unshare

- Namespaces: Restricted views of systems like the process tree, network interfaces, mounts, …
- Usage:

unshare [options] [<program> [<argument>...]]

- Options:

-f, --fork: To run the program in the modified namespaces. unshare will:

1) create namespaces,
2) fork(),
3) Then, exec()

Without -f: Don’t fork(): create namespaces and exec() the program directly.

The UTS Namespace

- **Unix Time Sharing (UTS) Namespace**: store the system hostname.

$ sudo unshare -u bash

- ❖The –u (--uts) flag creates a new UTS Namespace

- Use **pstree** to verify the parent Process (compare with **$sudo unshare -uf bash**)

#hostname sandbox

- ❖To change the hostname in the new UTS name space.

#hostname

- ❖Verify the new hostname

- Switch in the native namespaces and check:

#hostname

The UTS Namespace

- **Unix Time Sharing (UTS) Namespace**: store the system hostname.

$ sudo unshare -u bash

❖The –u (--uts) flag creates a new UTS Namespace

- Use **pstree** to verify the parent Process (compare with **$sudo unshare -uf bash**)

#hostname sandbox

❖To change the hostname in the new UTS name space.

#hostname

❖Verify the new hostname

- Switch in the native namespaces and check:

#hostname

Without sudo ?

User Namespace

$unshare -u bash

unshare:unshare failed: Operation not permitted

→New User namespace: the USER namespace of the created UTS namespace.

$unshare -Uu bash

- The flag **-U**, **--user**: Create a new USER namespace.
- The flag **-u**, **--uts**: Create a new UTS namespace.

User Namespace

$unshare -Uu bash   # New user not specified

$id

nobody

- Display the UID map:

$cat /proc/self/uid_map #Empty file: without UID map → nobody in all namespaces (all resources)

$exit

$unshare -Uu --map-user=root bash

Or: $unshare -Uu -r bash: The -r flag and --map-user=root are equivalent.

- The **--map-user=root** or **-r** option: Map current user to root in created name spaces.

$unshare -Uur   bash

$whoami

root

#cat /proc/self/uid_map

0   1000   #User with uid 0 is maped to the user with the UID 1000 (ubuntu)

Verify Namespaces

- The lsns command provides information about system namespaces(in native namespaces):

  #lsns

- With -p PID option: Display only the namespaces held by the process with this PID:

  #lsns -p $$

\- Verify that a new **UTS** namespaces is created (compare with the native namespace).

- Reads directly the /proc directory:

  #ls -l /proc/Process_ID/ns

Process_ID can also be: **$$** or **self**

Entering namespaces: nsenter

- Processes may choose to share/join an exited namespaces with **nsenter.**
- In the first terminal:

  $unshare -uUr bash
  #hostname test
  #hostname
  #echo $$ # PID of container Process

- In second terminal, run a bash Sharing the new UTS namespace:

  $hostname
  $sudo nsenter --uts=/proc/ContainerProcess_ID/ns/uts bash
  #Synthax: nsenter --uts=MountNamespaceName Command
  # or   : nsenter -u -t ContainerProcess_ID Command
  $hostname
  $exit
  sudo nsenter --uts=/proc/ContainerProcess_ID/ns/uts hostname test2

- On the first terminal (container): **hostname # To check the hostname modification**

The Mount Namespace: An example

- Simulate disk partition:
  - In the native namespace, create a virtual disk partition:

    $sudo dd if=/dev/zero of=dev1 bs=1024 count=1024
  - Create the file system (ext2):

    $sudo mkfs -t ext2 dev1
  - Mount partition, change root owner, create a test file then unmount the partition:

    $sudo mkdir MountPoint

    $sudo mount dev1 MountPoint

    $sudo chown ubuntu:ubuntu MountPoint

    $sudo touch MountPoint/file1

    $sudo umount MountPoint

The Mount Namespace: An example

In terminal 1	In terminal 2 (Native namespaces)
$unshare -mUf bash # to create a new Mount Namespace for the bash process. ##Synthax: unshare –m (or –mount) command #mount #echo $$ # PID of the process # #[Before Running the mount command of terminal 2] #df -kh # list of mounted partitions # ls /home/ubuntu/MountPoint # [After Running the mount command of terminal 2] #df -kh # list of mounted partitions # ls MountPoint #mkdir MountPoint/rep1 #exit	df -kh # list of mounted partitions ls MountPoint sudo nsenter -m -t ContainerPID \ mount /home/ubuntu/dev1 \ /home/ubuntu/MountPoint df -kh # list of mounted partitions ls MountPoint

The PID Namespace

$unshare -Umfr -p   bash

- The flag –p (or –pid): Create a new PID namespace.
- The - m flag: creates a new mount namespace (**why? (1)**)
- The - f to fork after creating the new namespaces and starting bash (**why?(2)**)

 # echo $$

1

#ps aux

 # pstree

- **Why ps report that systemd have the PID 1?**
- **And the pstree command is referencing the native PID namespace?**

The PID Namespace

- The Linux Kernel uses the /proc pseudo filesystem to report running process status.
  - → /proc is used by commands like pstree, ps … to report information about processes

With (unshare -Umfrp bash) /proc is not modified (**The why (1)**).

→ Since we created a new PID namespace, we need to mount a new /proc that matches the new namespace.

The PID Namespace: remount /proc

```
$ unshare -Umfpr  bash
# mount -t proc none /proc
#ps aux
# pstree
#exit
```

- Or directly on command line with **--mount-proc** option:

```
$ unshare -Umfpr --mount-proc  bash
# ps aux
```

The PID Namespace: Why -f option?

Run unshare with (-f)	Run unshare without (-f)
$unshare -umUr -p -f sh #mount -t proc none /proc #ls #echo $$ 1 #exit	$unshare -umUr -p sh #mount -t proc none /proc #ls /bin/sh: 2: Cannot fork #echo $$ xxx #exit

The PID Namespace: Why -f option?

Run unshare with (-f)	Run unshare without (-f)
#share -umUr -p -f sh #lsns -p $$ The sh process is attached to: • New PID namespace • New uts namespace • New mount namespace	#share -umUr -p sh #lsns -p $$ The sh process is attached to a: ▪ PID namespace of the parent process (where is the new PID namespace?) ▪ New uts namespace ▪ New mount namespace

Entering in PID namespaces: nsenter

- In a first terminal:

```
$unshare -fUuprm --mount-proc  bash
# tty
        <Terminal name>
```

- In native namespaces: (Identify the PID of bash process)

```
$ ps aux |  grep <result of tty command (without /dev)>
        root    xxx  0.0  0.0  20272  3064 pts/5    S+   17:25   0:00 bash
$pid=xxx
$sudo nsenter -p -t $pid -m -t $pid unshare -Ufur  bash
#ps aux
#exit
```

Stop/Re-Start a container

- Stop a running container:

kill -SIGSTOP $pid

- Re-start stopped container:

kill -SIGCONT $pid

- Run an interactive shell in a container:

sudo nsenter -a -t $pid sh #by default the command is bash.

Changing the Filesystem Example 1: Bash Container

- Create the Directory tree:

```
$TARGET=rootfs
$mkdir -p ${TARGET}/{dev,proc,bin,lib,lib64}
```

- Determine libraries needed by the bash shell:

```
$ldd /bin/bash
    linux-vdso.so.1 (0x00007ffc3af34000)
    libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007fbea7d89000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fbea7b60000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fbea7f23000)
```

○ **linux-vdso.so.1** : A virtual shared object that doesn't have any physical file on the disk.
→ it's a part of the linux kernel.

Changing the Filesystem
Example 1: Bash Container

- Copy files shared object lib

```
$cp -afL /lib/x86_64-linux-gnu/libtinfo.so.6 ${TARGET}/lib/
$ cp -afL /lib/x86_64-linux-gnu/libc.so.6 ${TARGET}/lib/
$ cp -afL /lib64/ld-linux-x86-64.so.2  ${TARGET}/lib64/
```

- Copy the binary file

```
$ cp -aL /bin/bash ${TARGET}/bin/
```

- Run container with chroot

```
$ unshare -mipunUrf  chroot ${TARGET} /bin/bash
        bash-5.1# echo message
        bash-5.1# pwd
        bash-5.1# ls
        /bin/sh: 1: ls: not found
        bash-5.1# exit
```

Changing the Filesystem Example 1: Bash Container

- Determine libraries needed by the ls command:

```
$ldd /bin/ls
    linux-vdso.so.1 (0x00007fffccf9c000)
    libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fdbe9896000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fdbe966d000)
    libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007fdbe95d6000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fdbe98ed000)
```

- Copy files shared object lib used by ls:

```
$cp -afL /lib/x86_64-linux-gnu/libselinux.so.1  ${TARGET}/lib/
$cp -afL /lib/x86_64-linux-gnu/libpcre2-8.so.0  ${TARGET}/lib/
```

- Copy ls binary file:

```
$cp -aL /bin/ls ${TARGET}/bin/
```

- Re-Run the container:

```
$unshare -mipunUrf  chroot ${TARGET} /bin/bash
        bash-5.1# pwd
        bash-5.1# ls  -l
        bash-5.1# exit
```

Changing the Filesystem
Example 2: Debian Container

- Create the root file system:
  - Install debootstrap : A tool that install a Debian-based system into a subdirectory of already-installed system

    $sudo apt install debootstrap
  - create a new ~/chroot-debian the root directory

    $TARGET="chroot-Debian"

    $mkdir -p ${TARGET}
  - With debootstrap, install the debian distribution in TARGET:

    $sudo debootstrap stable ${TARGET} https://deb.debian.org/debian
  - Verify installed files:

    $ls ${TARGET}

Changing the Filesystem Example 2: Debian Container

- Modify the owner/group of the file system:

```
$uid=$(id -u)
$gid=$(id -g)
$sudo chown -R $uid:$gid ${TARGET}
```

- Create a Debian Container:

```
$ unshare -mipunUrf chroot ${TARGET} /bin/bash
    #echo $$
    #cat /etc/os-release
    #exit
```

Containers with conventional Linux file-systems

- To run a second bash container:
  - Copy rootfs directory in rootfs2 : cp rootfs rootfs2
  - Start the container: chroot rootfs2 /bin/bash

☹ **inefficient disk space optimization**: common files branches (exp:/bin) can't be shared by containers.

☹With (n) instances of a container and a container file system size (s)G, (n*s) G of physical memory would be reserved by running containers.

Another file-system for the container ?

UnionFS : A File System of Containers

UFS: Union File Systems

chroot

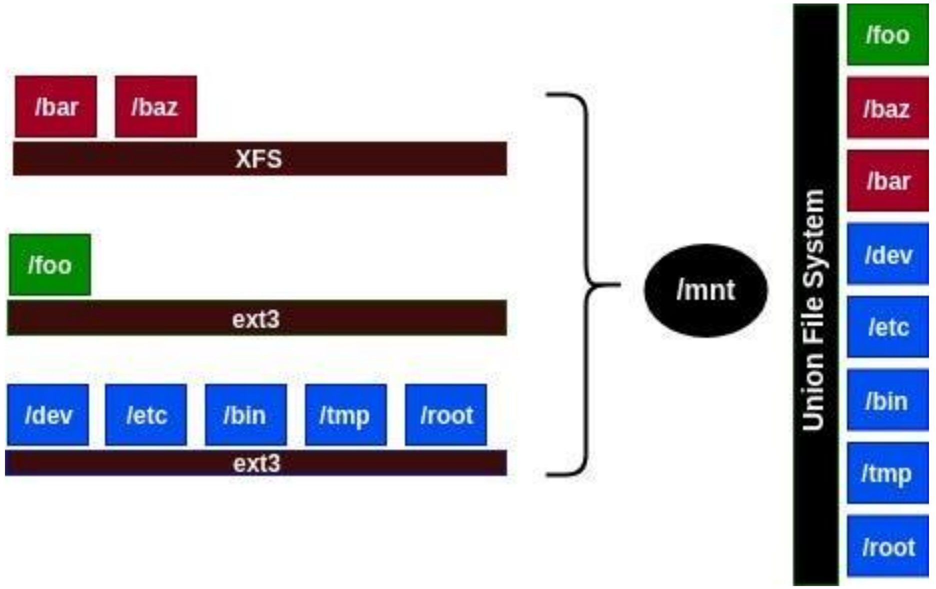
Namespace

Union File System

- AUFS (advanced multi-layered unification filesystem)
- Overlay FS
- Overlay2 FS
- etc…

Union File System

- ✓On top of other file-systems.
- ✓ A single and unified view to files and directories of **<u>separate file-system</u>**.
- ✓ it mounts multiple directories to a single mount point.
- ✓It is more of a mounting mechanism than a file system.

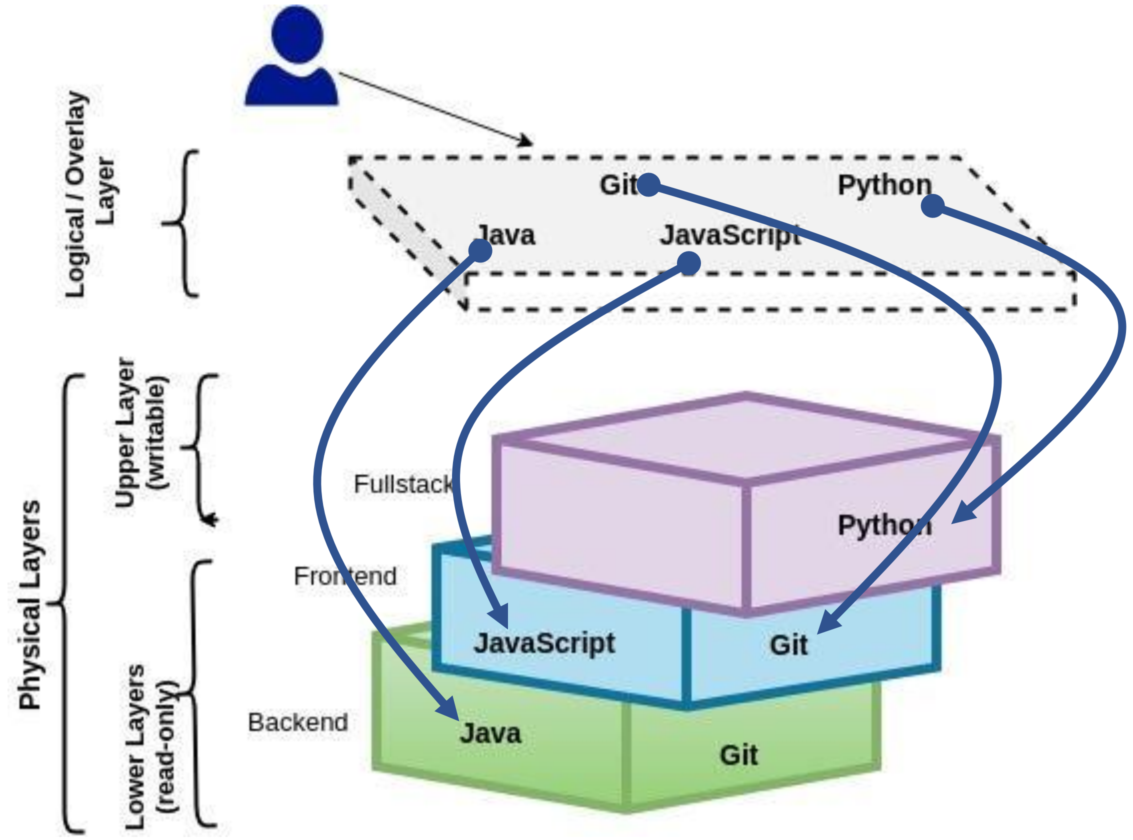


Union File System

- **Logical merge** of multiple layers (Layer = branch).
- **Multiple** Read-only **lower layers**, **One** writable **upper layer**.
- Start reading from the **upper layer** than defaults to **lower layers**.
- Copy on Write (CoW) to the upper layer:

  A modified file is copied from "lower" to "upper" layer. All the modification will take place in upper layer: the only layer with write access.
- Simulate removal from lower directory through **whiteout** file.

Union File System: A Use Case



Debian Container with UFS

- Create container bundle directory:

$TARGET=C1_Debian_Bundle

$mkdir -p ${TARGET}/{UPPER,WORK,ROOTFS} #UPPER et WORK must be in the same filesystem

- Create an OverlayFS (lower layer: rootfs, upperlayer:UPPER):

```
$sudo mount -t overlay \
        -o lowerdir=chroot-Debian,upperdir=${TARGET}/UPPER,workdir=${TARGET}/WORK \
        none  ${TARGET}/ROOTFS
```

- Verify the UFS File system

$ls ${TARGET}/ROOTFS

- Run the Container:

```
$unshare -mipunUrf  chroot ${TARGET}/ROOTFS /bin/bash
        # echo $$
```

Debian Container with UFS

- Create a second Debian container: Same manip. with (**TARGET=C2_Debian_Bundle**).
- With C1_Debian:

  # echo Conatiner1 > fileC1

  # rm /etc/os-release
- With C2_Debian:

  # echo Conatiner2 > fileC2

  # cat /etc/os-release
- List the Upper layer of C1 container:

$ls -l C1_Debian_Bundle/UPPER/

drwxr-xr-x 2 ubuntu ubuntu 4096 Sep 14 14:03 etc

-rw-rw-r-- 1 ubuntu ubuntu 11 Sep 14 14:02 fileC1

$ls -l C1_Debian_Bundle/UPPER/etc

c--------- 2 root root 0, 0 Sep 14 14:03 os-release

→ A whiteout file in upper layer (/etc/os-release): Block the visibility of the file.

Network Namespace

- Run two Debian Containers (C1 & C2) on 2 terminals:
  - C1 Debian container:

  TARGET=C1_Debian_Bundle; unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash
  - C2 Debian container:

  TARGET=C2_Debian_Bundle; unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash
  - With C1 (or C2):

  # ip a

  1: lo:………. (Only loopback interface)
- Unshare whith **-n** option: create an anonymous network namespace.

  → Can be referred by the PID of one process in that namespace.

  → Named network name space are under : /var/run/netns/

Network Namespace

- Switch terminal in the native namespaces:
  - The PID(s) of C1 unshare processes (var:pidC1):

TARGET=C1_Debian_Bundle; pidC1=$(pgrep -f "unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash")

  - Find the PID of C2 unshare process(var:pidC2 var) :

TARGET=C2_Debian_Bundle; pidC2=$(pgrep -f "unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash")

  - Verify pidC1 and pidC2:

echo -e " - C1 PID: $pidC1 \n - C2 PID: $pidC2"

Network Namespace

In the native namespaces:

- Create a virtual switch to connect containers (C1 and C2):

sudo ip link add br_1 type bridge

sudo ip link set br_1 up

sudo ip addr add 10.10.10.1/24 dev br_1

- Create pair of virtual ethernet interfaces to connect C1 to switch:

sudo ip link add veth_01 netns $pidC1 type veth peer veth_11

sudo ip link set veth_11 master br_1

sudo ip link set veth_11 up

- Create pair of virtual ethernet interfaces to connect C2 to switch:

sudo ip link add veth_02 netns $pidC2 type veth peer veth_12

sudo ip link set veth_12 master br_1

sudo ip link set veth_12 up

Network Namespace

- **C1 container:**

  ip address add 10.10.10.2/24 dev veth_01

  ip link set veth_01 up

  - To verify the IP configuration: **ip a**

- **C2 container:**

  ip address add 10.10.10.3/24 dev veth_02

  ip link set veth_02 up

  - To verify the IP configuration: **ip a**

Container Networking

- Test connectivity:
- **C1 container (respectively C2):**

    ping 10.10.10.1

    ping 10.10.10.3 (respectively **10.10.10.2**)

- **In the native namespaces**, verify that ACCEPT is the default policy in iptables:

    sudo iptables -L

    sudo iptables -P FORWARD ACCEPT

Expose a network port: iptables

- Run/simulate web service with netcat (nc):

nc [<options>] <host> <port>

- The nc command: manipulate (read/write) a TCP socket (by default)
- Netcat has two working modes:
  - Listen mode (-l option): Server. If <host> is omitted, nc listens on all addresses
  - Connect mode (the default mode): Client.
- The -k option: When a connection is completed, listen for another one.

Expose a network port: iptables

- Install nc command on container:
  - Exit containers(C1 and C2).
  - Unmount the UFS filesystem (layer0 will be modified):

TARGET=C1_Debian_Bundle; sudo umount ${TARGET}/ROOTFS
TARGET=C2_Debian_Bundle; sudo umount ${TARGET}/ROOTFS

  - Copy nc libraries and binary from VM File system to Layer 0 Container FS :

```
cd
RootFS=chroot-Debian
cp -afL /lib/x86_64-linux-gnu/libbsd.so.0 ${RootFS}/lib/
cp -afL /lib/x86_64-linux-gnu/libresolv.so.2 ${RootFS}/lib/
cp -afL /lib/x86_64-linux-gnu/libc.so.6 ${RootFS}/lib/
cp -afL /lib/x86_64-linux-gnu/libmd.so.0 ${RootFS}/lib/
cp -aL /usr/bin/nc ${RootFS}/usr/bin/
```

Expose a network port: iptables

- **Remount File Systems/Run container:**
- **Terminal 1:**

```
TARGET=C1_Debian_Bundle
sudo umount ${TARGET}/ROOTFS
sudo mount -t overlay -o lowerdir=chroot-Debian,upperdir=${TARGET}/UPPER,workdir=${TARGET}/WORK \
        none  ${TARGET}/ROOTFS

unshare -mipunUrf  chroot ${TARGET}/ROOTFS /bin/bash
```

- **Terminal2:**

```
TARGET=C2_Debian_Bundle
sudo umount ${TARGET}/ROOTFS
sudo mount -t overlay  -o lowerdir=chroot-Debian,upperdir=${TARGET}/UPPER,workdir=${TARGET}/WORK \
        none  ${TARGET}/ROOTFS

unshare -mipunUrf  chroot ${TARGET}/ROOTFS /bin/bash
```

Expose a network port: iptables

In the native namespaces:

> **TARGET=C1_Debian_Bundle; pidC1=$(pgrep -f "unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash")**
>
> **TARGET=C2_Debian_Bundle; pidC2=$(pgrep -f "unshare -mipunUrf chroot ${TARGET}/ROOTFS /bin/bash")**

- Connect C1 to br_1 switch:

```
sudo ip link add veth_01 netns $pidC1 type veth peer veth_11
sudo ip link set veth_11 master br_1
sudo ip link set veth_11 up
```

- Connect C2 to br_1 switch:

```
sudo ip link add veth_02 netns $pidC2 type veth peer veth_12
sudo ip link set veth_12 master br_1
sudo ip link set veth_12 up
```

Expose a network port: iptables

- **Containers IP configuration:**

❖**C1 container:**

ip address add 10.10.10.2/24 dev veth_01
ip link set veth_01 up
ip route add default via 10.10.10.1

- Verify the IP configuration: **ip a**
- Verify the routing table: **ip route show**

❖**C2 container:**

ip address add 10.10.10.3/24 dev veth_02
ip link set veth_02 up
ip route add default via 10.10.10.1

- Verify the IP configuration: **ip a**
- Verify the routing table: **ip route show**

Expose a network port: iptables

- **Start web service with nc:**

❖ **C1 container:**

```
while true;
  do echo -e "HTTP/1.1 200 OK\n\n$(echo 'Debian Container 1' )" \
  | nc -l -k -p 8080 -q 1;
done
```

- **Ctrl+Z** to stop process

❖ **C2 container:**

```
while true;
  do echo -e "HTTP/1.1 200 OK\n\n$(echo 'Debian Container 2' )" \
  | nc -l -k -p 8090 -q 1;
done
```

Expose a network port: iptables

- **In native namespace(VM Machine):** Test web server

  $curl http://10.10.10.2:8080
  $curl http://10.10.10.3:8090

- **Expose TCP port:**
  - C1 container: 8080 – 80
  - C2 container: 8090 – 8080

- **Configure iptables:**

```
sudo iptables -F
sudo iptables -F -t nat
sudo iptables -t nat -A PREROUTING -p tcp -d  192.168.250.10 --dport 80 -j DNAT --to-destination 10.10.10.2:8080
sudo iptables -t nat -A PREROUTING -p tcp -d  192.168.250.10 --dport 8080 -j DNAT --to-destination 10.10.10.3:8090
```

Expose a network port: iptables

- On physical machine: