Docker: LAB DI1 Base Image BusyBox

Objet	Création d'une image de base BusyBox
Niveau requis	débutant, avisé
Débutant, à savoir	
Suivi	:DRAFT:

BusyBox regroupe des versions minuscules de nombreux utilitaires UNIX courants en un seul petit exécutable, qui remplace la plupart des utilitaires qu'on trouve habituellement dans GNU `fileutils`, `shellutils`, etc. Cependant, les options incluses fournissent les fonctionnalités attendues et se comportent de manière très similaire à leurs homologues GNU: **BusyBox** fournit un environnement assez complet pour tout système de petite taille ou intégré.

BusyBox a été conçu dans un souci d'optimisation de la taille et de ressources limitées, et est extrêmement modulaire, permettant d'inclure ou d'exclure facilement des commandes (ou fonctionnalités) lors de la compilation, ce qui facilite la personnalisation des systèmes intégrés.

Ce document décrit la construction de la base de SliTaz, pour cela on utilise: un noyau Linux, **BusyBox**, et Syslinux pour booter le système. SliTaz utilise une archive initramfs décompressée en RAM par le kernel au démarrage du système. On va créer la box, pour qu'elle tienne dans un système de fichier racine de 3 à 4 Mb, en utilisant 'strip' sur les bibliothèques et les binaires, pour gagner de la place.

Construction d'une Image de base BusyBox

Parfois, on a juste besoin d’expérimenter une configuration avec un environnement minimal sans reconstruire de petites images Linux.

On peut utiliser une installation existante dans une prison chroot. Une prison chroot est un environnement isolé fonctionnant au-dessus d'un autre. Le nom lui-même dit en réalité ce qu'il est, il exécute des processus sous une racine modifiée (un sous-répertoire de la racine d'origine).

Cet article déceit la construction dans une prison chroot d'un environnement minimal ne contenant qu'un shell `BusyBox` et accessible via le serveur SSH de `Dropbear`.

Création du jail chroot

Description de la prison

Afin de ne pas toucher au système, les étapes de l'installation s’exécuteront dans la prison chroot. Le système de fichiers racine sera `/chroot/jail`. Lorsque cet environnement est en place, on peut facilement ajouter ou supprimer des bibliothèques pour vérifier les dépendances ou la compatibilité. Ces bibliothèques ne doivent pas nécessairement correspondre à celles de /lib.

Création de la la structure du système de fichiers racine

La première chose à faire est de créer la structure du système de fichiers racine dont on a besoin pour un système minimal sous le répertoire spécifique, dans cet exemple: /chroot/busybox.

```
# mkdir -pv /chroot/busybox
mkdir: created directory ‘/chroot’
mkdir: created directory ‘/chroot/busybox’
# cd /chroot/busybox/
# mkdir -pv dev/pts proc etc lib usr/lib var/run var/log
mkdir: created directory ‘dev’
mkdir: created directory ‘dev/pts’
mkdir: created directory ‘proc’
mkdir: created directory ‘etc’
mkdir: created directory ‘lib’
mkdir: created directory ‘usr’
mkdir: created directory ‘usr/lib’
mkdir: created directory ‘var’
mkdir: created directory ‘var/run’
mkdir: created directory ‘var/log’
# ln -s lib/ lib64
```

Ajout de /dev/pts et /dev

Outre la structure de répertoires, on a également besoin de périphériques et de liens vers les répertoires `/dev/pts` et `/proc` d'origine:

```
# mknod dev/urandom c 1 9
# chmod 0666 dev/urandom
# mknod dev/ptmx c 5 2
# chmod 0666 dev/ptmx
# mknod dev/tty c 5 0
# chmod 0666 dev/tty
# mount -o bind /dev/pts dev/pts/
# mount -o bind /proc proc/
```

Puisque nous on veut se connecter au chroot, il faut aussi d’autres fichiers:

```
# cp /etc/localtime etc/
# cp /etc/nsswitch.conf etc/
# cp /etc/resolv.conf etc/
# cp /etc/host.conf etc/
# cp /etc/hosts etc/
# cp /etc/shells etc/
# touch var/log/lastlog
# touch var/run/utmp
```

```
# touch var/log/wtmp
```

Démarrer la prison chroot

Démarrer la prison.

```
|  | root@cen7 ~ |# chroot /chroot/busybox/
```

Construction du noyau

Parmi les nombreuses raisons, la construction d'un noyau Linux minimal et son démarrage sur un émulateur permettent aux développeurs de créer rapidement des fonctionnalités supplémentaires du noyau Linux.

Pré requis

```
sudo apt-get install curl
sudo apt-get install libncurses5-dev
sudo apt install qemu-system-x86
```

Télécharger et extraire le noyau Linux et BusyBox

```
curl https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.10.6.tar.xz | tar
xJf -
curl https://busybox.net/downloads/busybox-1.26.2.tar.bz2 | tar xjf -
```

Créer un espace utilisateur minimal

```
cd busybox-1.26.2
mkdir -pv $TOP/obj/busybox-x86
make O=$TOP/obj/busybox-x86 defconfig
```

- **-pv**: p signifie ne pas générer d'erreur et quitter si le répertoire existe et v signifie imprimer un message pour chaque répertoire créé.
- **0=** signifie mettre la sortie ici

Activer la liaison statique dans BusyBox

```
make O=$TOP/obj/busybox-x86 menuconfig
```

1. Appuyer sur enter dans `Busybox Settings --->`

2. Appuyer sur la flèche vers le bas et choisir `Build BusyBox as a static binary (no shared libs).`
3. Appuyer sur Y
4. Sélectionner Exit deux fois et appuyer sur Entrée pour enregistrer (le curseur doit être sur `yes`).

Construire BusyBox

```
cd $TOP/obj/busybox-x86
make -j2
make install
```

Make -j2 a pris environ 2 min, le temps d'installation était négligeable.

Construisre la structure de répertoire de initramfs

```
mkdir -pv $TOP/initramfs/x86-busybox
cd $TOP/initramfs/x86-busybox
mkdir -pv
{bin,dev,sbin,etc,proc,sys/kernel/debug,usr/{bin,sbin},lib,lib64,mnt/root,ro
ot}
cp -av $TOP/obj/busybox-x86/\_install/* $TOP/initramfs/x86-busybox
sudo cp -av /dev/{null,console,tty,sda1} $TOP/initramfs/x86-busybox/dev/
```

- **-av**: signifie `--dR --preserve=all`
  - **d**: `--no-dereference --preserve=links`
  - **R**: copier les répertoires de manière récursive

Créer init et le rendre exécutable

- Modifier init

```
vi $TOP/initramfs/x86-busybox/init
```

```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
mount -t debugfs none /sys/kernel/debug

echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"

exec /bin/sh
```

Rendre le fichier exécutable

```

```

```
chmod +x $TOP/initramfs/x86-busybox/init
```

Créer les initramfs

```
cd $TOP/initramfs/x86-busybox
find . | cpio -H newc -o > ../initramfs.cpio
cd ..
cat initramfs.cpio | gzip > $TOP/obj/initramfs.igz
```

- **find** - recherche des fichiers dans une hiérarchie de répertoires
  - **$TOP**: commence la recherche de fichiers danss $TOP, aucun filtre n'étant spécifié, tous les fichiers seront retournés.
  - **-print0**: affiche le nom du fichier de remplissage suivi d'un NULL au lieu d'une nouvelle ligne
- **cpio** - copie des fichiers depuis et vers des archives
  - **-o**: crée l'archive (exécutée en mode copie)
  - **-v**: Liste textuelle des fichiers traités
  - **-H newc**: Utilise l'archive donnée FORMAT
    - **Newc**: nouveau format portable (SVR4), qui prend en charge les systèmes de fichiers comportant plus de 65 536 i-nœuds (4294967295 octets).
  - **Gzip** - compresse des fichiers

Pour vérifier l'initramfs, faire :

```
ls -hl $TOP/obj/initramfs-busybox-x86.cpio.gz
```

On doit voir quelque chose comme:

```
-rw-rw-r-- 1 pfefferz pfefferz 8.7M Mar  6 21:06 /home/pfefferz/tl/teeny-
linux/obj/initramfs-busybox-x86.cpio.gz
```

Pour vérifier le contenu, faire:

```
mkdir /checkarchive
cd /checkarchive
zcat $TOP/obj/initramfs.igz | cpio -idmv --no-absolute-filenames
```

Ensuite, pour recréer à partir de ce répertoire, faire:

```
cd /checkarchive
find . | cpio -H newc -o > ../checked-initramfs.cpio
cd ..
cat checked-initramfs.cpio | gzip > $TOP/obj/checked-initramfs.igz
```

Configurer le noyau

- Configurer le noyau avec la configuration minimale

```
cd /linux-4.10.6
make O=$TOP/obj/linux-x86-allnoconfig allnoconfig
```

- Activer les options de configuration

```
cd /linux-4.10.6
make O=$TOP/obj/linux-x86-allnoconfig nconfig
```

`Nconfig` est menu pseudo-graphique basé sur `ncurses`

```
*] 64-bit kernel

-> General setup
  -> Configure standard kernel features
[*] Enable support for printk

-> General setup
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support

-> Executable file formats / Emulations
[*] Kernel support for ELF binaries
[*] Kernel support for scripts starting with #!

-> Device Drivers
  -> Character devices
[*] Enable TTY

-> Device Drivers
  -> Character devices
    -> Serial drivers
[*] 8250/16550 and compatible serial support
[*]   Console on 8250/16550 and compatible serial port

-> File systems
  -> Pseudo filesystems
[*] /proc file system support
[*] sysfs file system support

-> Kernel hacking
    -> Compile-time checks and compiler options
[*] Debug filesystem
```

```
-> Kernel hacking
[*] Early printk
```

`Debug filesystem` n'est pas nécessaire mais l'inclure permet de disposer d'une interface de débogage ce qui est un bon moyen de commencer à travailler avec le noyau. `Early printk` n'est pas non plus nécessaires, mais cela aide à voir le noyau démarrer.

Choix des options de configuration

Les options décrites ci-dessous permettent de construire un NUTSHELL busybox pouvant être utilisé dans Docker.

Généralement nécéssaire:

clé	Commentaire
NAMESPACES {NET,PID,IPC,UTS}_NS	
CGROUPS CGROUP_CPUACCT CGROUP_DEVICE CGROUP_FREEZER CGROUP_SCHED CPUSETS MEMCG	Les installations par défaut de docker utilisent toujours cgroupfs et la plupart, pour modifier les valeurs par défaut utiliser `--exec-opt native.cgroupdriver=systemd`
KEYS	
VETH BRIDGE BRIDGE_NETFILTER	
NF_NAT_IPV4 IP_NF_FILTER IP_NF_TARGET_MASQUERADE	
NETFILTER_XT_MATCH_{ADDRTYPE,CONNTRACK,IPVS}	
IP_NF_NAT NF_NAT NF_NAT_NEEDED	
POSIX_MQUEUE	required for bind-mounting /dev/mqueue into containers
DEVPTS_MULTIPLE_INSTANCES	pour les kernel < 4.8
CONFIG_SECURITY_APPARMOR CONFIG_DEFAULT_SECURITY=“apparmor” CONFIG_SECURITY_APPARMOR_BOOTPARAM_VALUE=1	AppArmor et apparmor_parser doivent être installés

Optional Features:

clé	Commentaire
USER_NS bad	dans RHEL7/CentOS si User namespaces disabled 'user_namespace.enable=0 ; add 'user_namespace.enable=1' to boot command line
SECCOMP	
CGROUP_PIDS	
MEMCG_SWAP MEMCG_SWAP_ENABLED	/sys/fs/cgroup/memory/memory.memsw.limit_in_bytes ne doit pas être défini
LEGACY_VSYSCALL_NATIVE	bad dangereux, fournit une cible contournant ASLR avec des gadgets ROP

clé	Commentaire
LEGACY_VSYSCALL_EMULATE	good
LEGACY_VSYSCALL_NONE	bad Les ontainers utilisant eglibc ⇐ 2.13 ne fonctionneront pas. Cela désactivera ASLR pour le VDSO, ce qui peut aider à exploiter des vulnérabilités de sécurité.
MEMCG_KMEM	pour les kernel < 4.5
RESOURCE_COUNTERS	pour les kernel < 3.18
NETPRIO_CGROUP	
CGROUP_NET_PRIO	
BLK_CGROUP BLK_DEV_THROTTLING IOSCHED_CFQ CFQ_GROUP_IOSCHED	
CGROUP_PERF	
CGROUP_HUGETLB	
NET_CLS_CGROUP $netprio	
CFS_BANDWIDTH FAIR_GROUP_SCHED RT_GROUP_SCHED	
IP_NF_TARGET_REDIRECT	
IP_VS	
IP_VS_NFCT	
IP_VS_PROTO_TCP	
IP_VS_PROTO_UDP	
IP_VS_RR	
EXT4_USE_FOR_EXT2	
EXT3_FS EXT3_FS_XATTR EXT3_FS_POSIX_ACL EXT3_FS_SECURITY	pour utiliser EXT3 comme support activer
EXT4_FS EXT4_FS_POSIX_ACL EXT4_FS_SECURITY	pour utiliser EXT3 ou EXT4 comme support

Network Drivers

clé	Commentaire
VXLAN BRIDGE_VLAN_FILTERING	
CRYPTO CRYPTO_AEAD CRYPTO_GCM CRYPTO_SEQIV CRYPTO_GHASH XFRM XFRM_USER XFRM_ALGO INET_ESP INET_XFRM_MODE_TRANSPORT	Optionnel (for encrypted networks)
IPVLAN	ipvlan
MACVLAN DUMMY	Macvlan
NF_NAT_FTP NF_CONNTRACK_FTP NF_NAT_TFTP NF_CONNTRACK_TFTP	ftp,tftp client in container

Storage Drivers

clé	Commentaire
AUFS_FS	certains noyaux incluent des correctifs AUFS mais pas l'indicateur AUFS_FS
BTRFS_FS BTRFS_FS_POSIX_ACL	btrfs
BLK_DEV_DM DM_THIN_PROVISIONING	devicemapper
OVERLAY_FS	overlay

Pour `zfs` il faut s'assurer d' avoir `/dev/zfs` et les ommandes `zfs` et `zpool` opérationnelles.

Contruire le noyau Linux

```
cd /linux-4.10.6
make O=$TOP/obj/linux-x86-allnoconfig -j2
```

À la fin du journal, on doit voir:

```
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Setup is 15804 bytes (padded to 15872 bytes).
System is 550 kB
CRC 57d75ca8
Kernel: arch/x86/boot/bzImage is ready  (#1)
make[1]: Leaving directory '/home/pfefferz/tl/teeny-linux/obj/linux-x86-
allnoconfig'
```

Lancer le noyau