# Docker: LAB 4 image de base Debian/Ubuntu

| | |
|---|---|
| **Objet** | Création d'une image de base Debian/ubuntu |
| **Niveau requis** | débutant, avisé |
| **Débutant, à savoir** | |
| **Suivi** | 🔧 Fix Me! |

Création d'une image `Debian/Ubuntu` avec `debootstrap`.

### moby/contrib/mkimage/debootstrap

```
#!/usr/bin/env bash
set -e

mkimgdeb="$(basename "$0")"
mkimg="$(dirname "$0").sh"

usage() {
    echo >&2 "usage: $mkimgdeb rootfsDir suite [debootstrap-args]"
    echo >&2 " note: $mkimgdeb meant to be used from $mkimg"
    exit 1
}

rootfsDir="$1"
if [ -z "$rootfsDir" ]; then
    echo >&2 "error: rootfsDir is missing"
    echo >&2
    usage
fi
shift

# we have to do a little fancy footwork to make sure "rootfsDir" becomes the
second non-option argument to debootstrap

before=()
while [ $# -gt 0 ] && [[ "$1" == -* ]]; do
    before+=( "$1" )
    shift
done

suite="$1"
if [ -z "$suite" ]; then
    echo >&2 "error: suite is missing"
    echo >&2
    usage
fi
shift
```

```
# get path to "chroot" in our current PATH
chrootPath="$(type -P chroot || :)"
if [ -z "$chrootPath" ]; then
    echo >&2 "error: chroot not found. Are you root?"
    echo >&2
    usage
fi

rootfs_chroot() {
    # "chroot" doesn't set PATH, so we need to set it explicitly to
something our new debootstrap chroot can use appropriately!

    # set PATH and chroot away!
    PATH='/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin' \
        "$chrootPath" "$rootfsDir" "$@"
}

# allow for DEBOOTSTRAP=qemu-debootstrap ./mkimage.sh ...
: ${DEBOOTSTRAP:=debootstrap}

(
    set -x
    $DEBOOTSTRAP "${before[@]}" "$suite" "$rootfsDir" "$@"
)

# now for some Docker-specific tweaks

# prevent init scripts from running during install/update
echo >&2 "+ echo exit 101 > '$rootfsDir/usr/sbin/policy-rc.d'"
cat > "$rootfsDir/usr/sbin/policy-rc.d" <<-'EOF'
    #!/bin/sh
    # For most Docker users, "apt-get install" only happens during "docker
build",
    # where starting services doesn't work and often fails in humorous ways.
This
    # prevents those failures by stopping the services from attempting to
start.
    exit 101
EOF
chmod +x "$rootfsDir/usr/sbin/policy-rc.d"

# prevent upstart scripts from running during install/update
(
    set -x
    rootfs_chroot dpkg-divert --local --rename --add /sbin/initctl
    cp -a "$rootfsDir/usr/sbin/policy-rc.d" "$rootfsDir/sbin/initctl"
    sed -i 's/^exit.*/exit 0/' "$rootfsDir/sbin/initctl"
)

# shrink a little, since apt makes us cache-fat (wheezy: ~157.5MB vs ~120MB)
```

```
( set -x; rootfs_chroot apt-get clean )

# this file is one APT creates to make sure we don't "autoremove" our
currently
# in-use kernel, which doesn't really apply to debootstraps/Docker images
that
# don't even have kernels installed
rm -f "$rootfsDir/etc/apt/apt.conf.d/01autoremove-kernels"

# Ubuntu 10.04 sucks... :)
if strings "$rootfsDir/usr/bin/dpkg" | grep -q unsafe-io; then
    # force dpkg not to call sync() after package extraction (speeding up
installs)
    echo >&2 "+ echo force-unsafe-io >
'$rootfsDir/etc/dpkg/dpkg.cfg.d/docker-apt-speedup'"
    cat > "$rootfsDir/etc/dpkg/dpkg.cfg.d/docker-apt-speedup" <<-'EOF'
        # For most Docker users, package installs happen during "docker
build", which
        # doesn't survive power loss and gets restarted clean afterwards
anyhow, so
        # this minor tweak gives us a nice speedup (much nicer on spinning
disks,
        # obviously).
        force-unsafe-io
    EOF
fi

if [ -d "$rootfsDir/etc/apt/apt.conf.d" ]; then
    # _keep_ us lean by effectively running "apt-get clean" after every
install
    aptGetClean='"rm -f /var/cache/apt/archives/*.deb
/var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin || true";'
    echo >&2 "+ cat > '$rootfsDir/etc/apt/apt.conf.d/docker-clean'"
    cat > "$rootfsDir/etc/apt/apt.conf.d/docker-clean" <<-EOF
        # Since for most Docker users, package installs happen in "docker
build" steps,
        # they essentially become individual layers due to the way Docker
handles
        # layering, especially using CoW filesystems.  What this means for
us is that
        # the caches that APT keeps end up just wasting space in those
layers, making
        # our layers unnecessarily large (especially since we'll normally
never use
        # these caches again and will instead just "docker build" again and
make a brand
        # new image).
        # Ideally, these would just be invoking "apt-get clean", but in our
testing,
        # that ended up being cyclic and we got stuck on APT's lock, so we
get this fun
```

```
        # creation that's essentially just "apt-get clean".
        DPkg::Post-Invoke { ${aptGetClean} };
        APT::Update::Post-Invoke { ${aptGetClean} };
        Dir::Cache::pkgcache "";
        Dir::Cache::srcpkgcache "";
        # Note that we do realize this isn't the ideal way to do this, and
are always
        # open to better suggestions
(https://github.com/docker/docker/issues).
    EOF

    # remove apt-cache translations for fast "apt-get update"
    echo >&2 "+ echo Acquire::Languages 'none' >
'$rootfsDir/etc/apt/apt.conf.d/docker-no-languages'"
    cat > "$rootfsDir/etc/apt/apt.conf.d/docker-no-languages" <<-'EOF'
        # In Docker, we don't often need the "Translations" files, so we're
just wasting
        # time and space by downloading them, and this inhibits that.  For
users that do
        # need them, it's a simple matter to delete this file and "apt-get
update". :)
        Acquire::Languages "none";
    EOF

    echo >&2 "+ echo Acquire::GzipIndexes 'true' >
'$rootfsDir/etc/apt/apt.conf.d/docker-gzip-indexes'"
    cat > "$rootfsDir/etc/apt/apt.conf.d/docker-gzip-indexes" <<-'EOF'
        # Since Docker users using "RUN apt-get update && apt-get install -y
..." in
        # their Dockerfiles don't go delete the lists files afterwards, we
want them to
        # be as small as possible on-disk, so we explicitly request "gz"
versions and
        # tell Apt to keep them gzipped on-disk.
        # For comparison, an "apt-get update" layer without this on a
pristine
        # "debian:wheezy" base image was "29.88 MB", where with this it was
only
        # "8.273 MB".
        Acquire::GzipIndexes "true";
        Acquire::CompressionTypes::Order:: "gz";
    EOF

    # update "autoremove" configuration to be aggressive about removing
suggests deps that weren't manually installed
    echo >&2 "+ echo Apt::AutoRemove::SuggestsImportant 'false' >
'$rootfsDir/etc/apt/apt.conf.d/docker-autoremove-suggests'"
    cat > "$rootfsDir/etc/apt/apt.conf.d/docker-autoremove-suggests" <<-
'EOF'
        # Since Docker users are looking for the smallest possible final
```

```
images, the
        # following emerges as a very common pattern:
        #   RUN apt-get update \
        #       && apt-get install -y <packages> \
        #       && <do some compilation work> \
        #       && apt-get purge -y --auto-remove <packages>
        # By default, APT will actually _keep_ packages installed via
Recommends or
        # Depends if another package Suggests them, even and including if
the package
        # that originally caused them to be installed is removed.  Setting
this to
        # "false" ensures that APT is appropriately aggressive about
removing the
        # packages it added.
        #
https://aptitude.alioth.debian.org/doc/en/ch02s05s05.html#configApt-AutoRemo
ve-SuggestsImportant
        Apt::AutoRemove::SuggestsImportant "false";
    EOF
fi

if [ -z "$DONT_TOUCH_SOURCES_LIST" ]; then
    # tweak sources.list, where appropriate
    lsbDist=
    if [ -z "$lsbDist" -a -r "$rootfsDir/etc/os-release" ]; then
        lsbDist="$(. "$rootfsDir/etc/os-release" && echo "$ID")"
    fi
    if [ -z "$lsbDist" -a -r "$rootfsDir/etc/lsb-release" ]; then
        lsbDist="$(. "$rootfsDir/etc/lsb-release" && echo "$DISTRIB_ID")"
    fi
    if [ -z "$lsbDist" -a -r "$rootfsDir/etc/debian_version" ]; then
        lsbDist='Debian'
    fi
    # normalize to lowercase for easier matching
    lsbDist="$(echo "$lsbDist" | tr '[:upper:]' '[:lower:]')"
    case "$lsbDist" in
        debian)
            # updates and security!
            if curl -o /dev/null -s --head --location --fail
"http://security.debian.org/dists/$suite/updates/main/binary-$(rootfs_chroot
dpkg --print-architecture)/Packages.gz"; then
                (
                    set -x
                    sed -i "
                        p;
                        s/ $suite / ${suite}-updates /
                    " "$rootfsDir/etc/apt/sources.list"
                    echo "deb http://security.debian.org $suite/updates
main" >> "$rootfsDir/etc/apt/sources.list"
                )
```

```
            fi
            ;;
        ubuntu)
            # add the updates and security repositories
            (
                set -x
                sed -i "
                    p;
                    s/ $suite / ${suite}-updates /; p;
                    s/ $suite-updates / ${suite}-security /
                " "$rootfsDir/etc/apt/sources.list"
            )
            ;;
        tanglu)
            # add the updates repository
            if [ "$suite" != 'devel' ]; then
                (
                    set -x
                    sed -i "
                        p;
                        s/ $suite / ${suite}-updates /
                    " "$rootfsDir/etc/apt/sources.list"
                )
            fi
            ;;
        steamos)
            # add contrib and non-free if "main" is the only component
            (
                set -x
                sed -i "s/ $suite main$/ $suite main contrib non-free/"
"$rootfsDir/etc/apt/sources.list"
            )
            ;;
    esac
fi

(
    set -x

    # make sure we're fully up-to-date
    rootfs_chroot sh -xc 'apt-get update && apt-get dist-upgrade -y'

    # delete all the apt list files since they're big and get stale quickly
    rm -rf "$rootfsDir/var/lib/apt/lists"/*
    # this forces "apt-get update" in dependent images, which is also good

    mkdir "$rootfsDir/var/lib/apt/lists/partial" # Lucid... "E: Lists
directory /var/lib/apt/lists/partial is missing."
)
```

```
## Archlinux

### moby/contrib/mkimage-arch.sh


#!/usr/bin/env bash
# Generate a minimal filesystem for archlinux and load it into the local
# docker as "archlinux"
# requires root
set -e

# reset umask to default
umask 022

hash pacstrap &>/dev/null || {
    echo "Could not find pacstrap. Run pacman -S arch-install-scripts"
    exit 1
}

hash expect &>/dev/null || {
    echo "Could not find expect. Run pacman -S expect"
    exit 1
}

export LANG="C.UTF-8"

ROOTFS=$(mktemp -d ${TMPDIR:-/var/tmp}/rootfs-archlinux-XXXXXXXXXX)
chmod 755 $ROOTFS

# required packages
PKGREQUIRED=(
    bash
    haveged
    pacman
    pacman-mirrorlist
)

# packages to ignore for space savings
PKGIGNORE=(
    dhcpcd
    diffutils
    file
    inetutils
    iproute2
    iputils
    jfsutils
    licenses
    linux
    linux-firmware
    lvm2
    man-db
```

```
    man-pages
    mdadm
    nano
    netctl
    openresolv
    pciutils
    pcmciautils
    psmisc
    reiserfsprogs
    s-nail
    sysfsutils
    systemd-sysvcompat
    usbutils
    vi
    which
    xfsprogs
)

PKGREMOVE=(
    gawk
    haveged
    less
    linux-libre
    linux-libre-firmware
)

PKGREQUIRED="${PKGREQUIRED[*]}"
IFS=','
PKGIGNORE="${PKGIGNORE[*]}"
unset IFS
PKGREMOVE="${PKGREMOVE[*]}"

arch="$(uname -m)"
case "$arch" in
    armv*)
        if pacman -Q archlinuxarm-keyring >/dev/null 2>&1; then
            pacman-key --init
            pacman-key --populate archlinuxarm
        else
            echo "Could not find archlinuxarm-keyring. Please, install it
and run pacman-key --populate archlinuxarm"
            exit 1
        fi
        PACMAN_CONF=$(mktemp ${TMPDIR:-/var/tmp}/pacman-conf-archlinux-
XXXXXXXXX)
        version="$(echo $arch | cut -c 5)"
        sed "s/Architecture = armv/Architecture = armv${version}h/g"
'./mkimage-archarm-pacman.conf' > "${PACMAN_CONF}"
        PACMAN_MIRRORLIST='Server =
http://mirror.archlinuxarm.org/$arch/$repo'
```

```
        PACMAN_EXTRA_PKGS='archlinuxarm-keyring'
        EXPECT_TIMEOUT=1800 # Most armv* based devices can be very slow
(e.g. RPiv1)
        ARCH_KEYRING=archlinuxarm
        DOCKER_IMAGE_NAME="armv${version}h/archlinux"
        ;;
    *)
        PACMAN_CONF='./mkimage-arch-pacman.conf'
        PACMAN_MIRRORLIST='Server =
https://mirrors.kernel.org/archlinux/$repo/os/$arch'
        PACMAN_EXTRA_PKGS=''
        EXPECT_TIMEOUT=60
        ARCH_KEYRING=archlinux
        DOCKER_IMAGE_NAME=archlinux
        ;;
esac

export PACMAN_MIRRORLIST

expect <<EOF
    set send_slow {1 .1}
    proc send {ignore arg} {
        sleep .1
        exp_send -s -- \$arg
    }
    set timeout $EXPECT_TIMEOUT
    spawn pacstrap -C $PACMAN_CONF -c -d -G -i $ROOTFS base $PKGREQUIRED
$PACMAN_EXTRA_PKGS --ignore $PKGIGNORE
    expect {
        -exact "anyway? \[Y/n\] " { send -- "n\r"; exp_continue }
        -exact "(default=all): " { send -- "\r"; exp_continue }
        -exact "installation? \[Y/n\]" { send -- "y\r"; exp_continue }
        -exact "delete it? \[Y/n\]" { send -- "y\r"; exp_continue }
    }
EOF

arch-chroot $ROOTFS /bin/sh -c 'rm -r /usr/share/man/*'
arch-chroot $ROOTFS /bin/sh -c "haveged -w 1024; pacman-key --init; pkill
haveged; pacman-key --populate $ARCH_KEYRING"
arch-chroot $ROOTFS /bin/sh -c "ln -sf /usr/share/zoneinfo/UTC
/etc/localtime"
arch-chroot $ROOTFS /bin/sh -c "for pkg in $PKGREMOVE; do if pacman -Qi
\$pkg > /dev/null 2>&1; then pacman -Rs --noconfirm \$pkg; fi; done"
echo 'en_US.UTF-8 UTF-8' > $ROOTFS/etc/locale.gen
arch-chroot $ROOTFS locale-gen

# udev doesn't work in containers, rebuild /dev
DEV=$ROOTFS/dev
rm -rf $DEV
mkdir -p $DEV
mknod -m 666 $DEV/null c 1 3
```

```
mknod -m 666 $DEV/zero c 1 5
mknod -m 666 $DEV/random c 1 8
mknod -m 666 $DEV/urandom c 1 9
mkdir -m 755 $DEV/pts
mkdir -m 1777 $DEV/shm
mknod -m 666 $DEV/tty c 5 0
mknod -m 600 $DEV/console c 5 1
mknod -m 666 $DEV/tty0 c 4 0
mknod -m 666 $DEV/full c 1 7
mknod -m 600 $DEV/initctl p
mknod -m 666 $DEV/ptmx c 5 2
ln -sf /proc/self/fd $DEV/fd

tar --numeric-owner --xattrs --acls -C $ROOTFS -c . | docker import -
$DOCKER_IMAGE_NAME
docker run --rm -t $DOCKER_IMAGE_NAME echo Success.
rm -rf $ROOTFS

### moby/contrib/mkimage-arch-pacman.conf

#
# /etc/pacman.conf
#
# See the pacman.conf(5) manpage for option and repository directives

#
# GENERAL OPTIONS
#
[options]
# The following paths are commented out with their default values listed.
# If you wish to use different paths, uncomment and update the paths.
#RootDir     = /
#DBPath      = /var/lib/pacman/
#CacheDir    = /var/cache/pacman/pkg/
#LogFile     = /var/log/pacman.log
#GPGDir      = /etc/pacman.d/gnupg/
HoldPkg     = pacman glibc
#XferCommand = /usr/bin/curl -C - -f %u > %o
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
#CleanMethod = KeepInstalled
#UseDelta    = 0.7
Architecture = auto

# Pacman won't upgrade packages listed in IgnorePkg and members of
IgnoreGroup
#IgnorePkg   =
#IgnoreGroup =

#NoUpgrade   =
#NoExtract   =
```

```
# Misc options
#UseSyslog
#Color
#TotalDownload
# We cannot check disk space from within a chroot environment
#CheckSpace
#VerbosePkgLists

# By default, pacman accepts packages signed by keys that its local keyring
# trusts (see pacman-key and its man page), as well as unsigned packages.
SigLevel    = Required DatabaseOptional
LocalFileSigLevel = Optional
#RemoteFileSigLevel = Required

# NOTE: You must run `pacman-key --init` before first using pacman; the
local
# keyring can then be populated with the keys of all official Arch Linux
# packagers with `pacman-key --populate archlinux`.

#
# REPOSITORIES
#   - can be defined here or included from another file
#   - pacman will search repositories in the order defined here
#   - local/custom mirrors can be added here or in separate files
#   - repositories listed first will take precedence when packages
#     have identical names, regardless of version number
#   - URLs will have $repo replaced by the name of the current repo
#   - URLs will have $arch replaced by the name of the architecture
#
# Repository entries are of the format:
#       [repo-name]
#       Server = ServerName
#       Include = IncludePath
#
# The header [repo-name] is crucial - it must be present and
# uncommented to enable the repo.
#

# The testing repositories are disabled by default. To enable, uncomment the
# repo name header and Include lines. You can add preferred servers
immediately
# after the header, and they will be used before the default mirrors.

#[testing]
#Include = /etc/pacman.d/mirrorlist

[core]
Include = /etc/pacman.d/mirrorlist

[extra]
Include = /etc/pacman.d/mirrorlist
```

```
#[community-testing]
#Include = /etc/pacman.d/mirrorlist

[community]
Include = /etc/pacman.d/mirrorlist

# An example of a custom package repository.  See the pacman manpage for
# tips on creating your own repositories.
#[custom]
#SigLevel = Optional TrustAll
#Server = file:///home/custompkgs


## Alpine

### moby/contrib/mkimage-alpine.sh

#!/bin/sh

set -e

[ $(id -u) -eq 0 ] || {
    printf >&2 '%s requires root\n' "$0"
    exit 1
}

usage() {
    printf >&2 '%s: [-r release] [-m mirror] [-s] [-c additional repository]
[-a arch]\n' "$0"
    exit 1
}

tmp() {
    TMP=$(mktemp -d ${TMPDIR:-/var/tmp}/alpine-docker-XXXXXXXXXX)
    ROOTFS=$(mktemp -d ${TMPDIR:-/var/tmp}/alpine-docker-rootfs-XXXXXXXXXX)
    trap "rm -rf $TMP $ROOTFS" EXIT TERM INT
}

apkv() {
    curl -sSL $MAINREPO/$ARCH/APKINDEX.tar.gz | tar -Oxz |
        grep --text '^P:apk-tools-static$' -A1 | tail -n1 | cut -d: -f2
}

getapk() {
    curl -sSL $MAINREPO/$ARCH/apk-tools-static-$(apkv).apk |
        tar -xz -C $TMP sbin/apk.static
}

mkbase() {
    $TMP/sbin/apk.static --repository $MAINREPO --no-cache --allow-untrusted
```

```
\
        --root $ROOTFS --initdb add alpine-base
}

conf() {
    printf '%s\n' $MAINREPO > $ROOTFS/etc/apk/repositories
    printf '%s\n' $ADDITIONALREPO >> $ROOTFS/etc/apk/repositories
}

pack() {
    local id
    id=$(tar --numeric-owner -C $ROOTFS -c . | docker import - alpine:$REL)

    docker tag $id alpine:latest
    docker run --rm alpine printf 'alpine:%s with id=%s created!\n' $REL $id
}

save() {
    [ $SAVE -eq 1 ] || return 0

    tar --numeric-owner -C $ROOTFS -c . | xz > rootfs.tar.xz
}

while getopts "hr:m:sc:a:" opt; do
    case $opt in
        r)
            REL=$OPTARG
            ;;
        m)
            MIRROR=$OPTARG
            ;;
        s)
            SAVE=1
            ;;
        c)
            ADDITIONALREPO=$OPTARG
            ;;
        a)
            ARCH=$OPTARG
            ;;
        *)
            usage
            ;;
    esac
done

REL=${REL:-edge}
MIRROR=${MIRROR:-http://nl.alpinelinux.org/alpine}
SAVE=${SAVE:-0}
MAINREPO=$MIRROR/$REL/main
ADDITIONALREPO=$MIRROR/$REL/${ADDITIONALREPO:-community}
```

```
ARCH=${ARCH:-$(uname -m)}

tmp
getapk
mkbase
conf
pack
save
```