Building Debian system images with vmdb2

Lars Wirzenius

vmdb2-0.8-156-g65adab0

Contents

Chapter 1

Introduction

vmdb2 builds disk images with Debian installed. The images can be used for virtual machines, or can be written to USB flash memory devices, and hardware computers can be booted off them. It is a successor of the vmdebootstrap program, written by the same author, to fix a number of architectural problems with the old program. The new program is not compatible with the old one; that would've required keeping the problems, as well.

This manual is published as HTML at https://vmdb2-manual.liw.fi/ and as a PDF at https://vmdb2-manual.liw.fi/vmdb2.pdf.

1.1 Why vmdb2 given vmdebootstrap already existed

`vmdebootstrap` was the first attempt by Lars Wirzenius to write a tool to build system images. It turned out to not be well designed. Specifically, it was not easily extensible to be as flexible as a tool of this sort should be.

1.2 Why vmdb2 given other tools already exist

Lars likes to write tools for himself and had some free time. He sometimes prefers to write his own tools rather than spend time and energy evaluating and improving existing tools. He admits this is a character flaw.

Also, he felt ashamed of how messy **`vmdebootstrap`** turned out to be.

If nobody else likes **`vmdb2`**, that just means Lars had some fun on his own.

Chapter 2

Installation

You can get vmdb2 by getting the source code from git:

```
git clone git://git.liw.fi/vmdb2
```

You can then run it from the source tree:

```
sudo /path/to/vmdb2/vmdb2 ...
```

In Debian 10 (“buster”) and its derivatives, you can also install the vmdb2 package:

```
apt install vmdb2
```

For any other systems, we have no instructions. If you figure it out, please tell us how.

Chapter 3

Getting started

vmdb2 works by reading specification file with instructions for how an image should be built, using YAML syntax, and following those instructions. A minimal specification file example:

```
steps:
  - mkimg: "{{ output }}"
    size: 4G

  - mklabel: gpt
    device: "{{ output }}"

  - mkpart: primary
    device: "{{ output }}"
    start: 0%
    end: 100%
    tag: root

  - mkfs: ext4
    partition: root

  - mount: root

  - debootstrap: stretch
    mirror: http://deb.debian.org/debian
    target: root

  - apt: install
    packages:
      - linux-image-amd64
    tag: root-fs

  - grub: bios
    tag: root
```

The above creates a four gigabyte file, creates a GPT partition table, a single partition, with a filesystem, and installs Debian release stretch onto it. It also installs a kernel, and a boot loader.

To use this, save the specification into **test.vmdb**, and run the following command:

```
sudo vmdb2 test.vmdb --output test.img --verbose
```

This will take a long time, mostly at the **debootstrap** step.

3.1 Tags

Instead of device filenames, vmdb2 steps refer to block devices inside the image, and their mount points, by symbolic names called tags. Tags are any names that the user likes, and vmdb2 does not assign meaning to them. They're just strings.

3.2 Jinja2 expansion

To refer to the filename specified with the **--output** or **--image** command line options, you can use Jinja2 templating. The variables **output** and **image** can be used.

```
- mkimg: "{{ output }}"

- mklabel: "{{ image }}"
```

The difference is that **--output** creates a new file, or truncates an existing file, whereas **--images** requires the file to already exist. The former is better for image file, the latter for real block devices.

3.3 Speed up image creasing by caching the root filesystem

Building an image can take several minutes, and that's with fast access to a Debian mirror and an SSD. The slowest part is typically running debootstrap, and that always results in the same output, for a given Debian release. This means its easy to cache.

vmdb2 has the two actions **cache-roots** and **unpack-rootfs** and the command line option **--rootfs-tarball** to allow user to cache. Thhe user uses the option to name a file. **cache-rootfs** takes the root filesystem and stores it into the file as a compress tar archive ("tarball"). **unpack-rootfs** unpacks the tarball. This allows vmdb2 to skip running debootstrap needlessly.

The specify which steps should be skipped, the **unless** field can be used: **unpack-rootfs** sets the **rootfs-unpacked** flag if it actually unpacks a tarball, and **unless** allows checking for that flag. If the tarball doesn't exist, the flag is not set.

```
- unpack-rootfs: root

- debootstrap: stretch
  target: root
  unless: rootfs-unpacked

- cache-rootfs: root
  unless: rootfs-unpacked
```

If the tarball exists, it's unpacked, and the **`debootstrap`** and **`cache-rootfs`** steps are skipped.

It's possible to have any number of steps between the unpack and the cache steps. However, note that if you change the steps, you need to delete the tarball to run them.

TODO: unless, caching, tags, jinja2

Chapter 4

Step reference manual

4.1 Step: `ansible`

Run Ansible using a provided playbook, to configure the image. vmdb2 sets up Ansible so that it treats the image as the host being configured (via the **`chroot`** connecion). The image MUST have Python installed (version 2 or 3 depending on Ansible version).

Step keys:

- **`ansible`** — REQUIRED; value is the tag of the root filesystem.
- **`playbook`** — REQUIRED; value is the filename of the Ansible playbook, relative to the .vmdb file.

Example (in the .vmdb file):

```
- apt: install
  tag: root
  packages: [python]

- ansible: root
  playbook: foo.yml
```

Example (**`foo.yml`**):

```
- hosts: image
  tasks:

    - name: "set /etc/hostname"
      shell: |
        echo "{{ hostname }}" > /etc/hostname

    - name: "unset root password"
      shell: |
        sed -i '/^root:[^:]*:/s//root::/' /etc/passwd

    - name: "configure networking"
      copy:
        content: |
          auto eth0
```

```
          iface eth0 inet dhcp
          iface eth0 inet6 auto
        dest: /etc/network/interfaces.d/wired

  vars:
    hostname: discworld
```

4.2 Step: `apt`

Install packages using apt, which needs to already have been installed.

Step keys:

- **`apt`** — REQUIRED; value MUST be **`install`**.
- **`tag`** — REQUIRED; value is the tag for the root filesystem.
- **`packages`** — REQUIRED; value is a list of packages to install.

Example (in the .vmdb file):

```
- apt: install
  tag: root
  packages:
  - python
  - linux-image-amd64
```

4.3 Step: `chroot`

Run a shell snippet in a chroot inside the image.

Step keys:

- **`chroot`** — REQUIRED; value is the tag for the root filesystem.
- **`shell`** — REQUIRED; the shell snippet to run

Example (in the .vmdb file):

```
- chroot: root
  shell: |
      echo I am in chroot
```

4.4 Step: `shell`

Run a shell snippet on the host. This is not run in a chroot, and can access the host system.

Step keys:

- **`root-fs`** — REQUIRED; value is the tag for the root filesystem.

- **shell** — REQUIRED; the shell snippet to run

Example (in the .vmdb file):

```
- root-fs: root
  shell: |
      echo I am in NOT in chroot.
```

4.5 Step: debootstrap

Install packages using apt, which needs to already have been installed.

Step keys:

- **debootstrap** — REQUIRED; value is the codename of the Debian release to install: **stretch**, **buster**, etc.
- **target** — REQUIRED; value is the tag for the root filesystem.
- **mirror** — OPTIONAL; which Debian mirror to use

Example (in the .vmdb file):

```
- debootstrap: stretch
  target: root
  mirror: http://mirror.example.com/debian
```

4.6 Step: grub

Install the GRUB bootloader to the image. Works on a PC, for traditional BIOS booting or modern UEFI booting. Does not (yet?) support Secure Boot.

Warning: This is the least robust part of vmdb2.

Step keys:

- **grub** — REQUIRED; value MUST be one of **uefi** and **bios**, for a UEFI or a BIOS boot, respectively. (FIXME: these are valid for a PC; not sure what other archs require, if grub even works there.)
- **tag** — REQUIRED; value is the tag for the root filesystem.
- **efi** — REQUIRED for UEFI; value is the tag for the EFI filesystem.
- **console** — OPTIONAL; set to **serial** to configure the image to use a serial console.
- **device** — OPTIONAL; which device to install GRUB onto; this is needed when installing to a real hard drive, instead of an image.

Example (in the .vmdb file):

```
- grub: bios
  tag: root
```

Same, but for UEFI:

```
- grub: uefi
  tag: root
  efi: efi
  console: serial
```

Install to a real hard disk (named with the **--image** option):

```
- grub: uefi
  tag: root
  efi: efi
  image-dev: "{{ image }}"
```

4.7 Step: luks

Set up disk encryption using LUKS with the **cryptsetup** utility. The encryption passphrase is read from a file or from the output of a command. The encrypted disk gets opened and can be mounted using a separate tag for the cleartext view.

Step keys:

- **cryptsetup** — REQUIRED; value is the tag for the encrypted block device. This is not directly useable by users, or mountable.
- **tag** — REQUIRED; the tag for the de-crypted block device. This is what gets mounted and visible to users.
- **key-file** — OPTIONAL; file from where passphrase is read.
- **key-cmd** — OPTIONAL; command to run, passphrase is the first line of its standard output.

Example (in the .vmdb file):

```
- cryptsetup: root
  tag: root_crypt
  key-file: disk.pass
```

Same, except run a command to get passphrase (in this case pass):

```
- cryptsetup: root
  tag: root_crypt
  key-cmd: pass show disk-encryption
```

4.8 Step: vgcreate

Create an LVM2 volume group (VG), and also initialise the physical volumes for it.

Step keys:

- **vgcreate** — REQUIRED; value is the tag for the volume group. This gets initialised with **vgcreate**.
- **physical** — REQUIRED; list of tags for block devices (partitions) to use as physical volumes. These get initialised with **pvcreate**.

Example (in the .vmdb file):

```
- vgcreate: rootvg
  physical:
  - my_partition
  - other_partition
```

4.9 Step: `lvcreate`

Create an LVM2 logical volume (LV) in an existing volume group.

Step keys:

- **`lvcreate`** — REQUIRED; value is the tag for the volume group.
- **`name`** — REQUIRED; tag for the new LV block device.
- **`size`** — REQUIRED; size of the new LV.

Example (in the .vmdb file):

```
- lvcreate: rootvg
  name: rootfs
  size: 1G
```

4.10 Step: `mkfs`

Create a filesystem.

Step keys:

- **`mkfs`** — REQUIRED; filesystem type, such as **`mkfs`** or **`vfat`**.
- **`partition`** — REQUIRED; tag for the block device to use.

Example (in the .vmdb file):

```
- mkfs: ext4
  partition: root
```

4.11 Step: `mkimg`

Create a new image file of a desired size.

Step keys:

- **`mkimage`** — REQUIRED; name of file to create.
- **`size`** — REQUIRED; size of the image.

Example (in the .vmdb file):

```
- mkimg: "{{ output }}"
  size: 4G
```

4.12 Step: `mount`

Mount a filesystem.

Step keys:

- **`mount`** — REQUIRED; tag of filesystem to mount.
- **`dirname`** — OPTIONAL; the mount point.
- **`mount-on`** — OPTIONAL; tag of already mounted filesystem in image. (FIXME: this may be wrong?)

Example (in the .vmdb file):

```
- mount: root
```

4.13 Step: `mklabel`

Create a partition table on a block device.

Step keys:

- **`mklabel`** — REQUIRED; type of partition table, MUST be one of **`msdos`** and **`gpt`**.
- **`device`** — REQUIRED; tag for the block device.

Example (in the .vmdb file):

```
- mklabel: "{{ output }}"
  size: 4G
```

4.14 Step: `mkpart`

Create a partition.

Step keys:

- **`mkpart`** — REQUIRED; type of partition to create: use **`primary`** (but any value acceped by **`parted`** is OK).
- **`device`** — REQUIRED; filename of block device where to create partition.
- **`start`** — REQUIRED; where does the partition start?
- **`end`** — REQUIRED; where does the partition end?
- **`tag`** — REQUIRED; tag for the new partition.

Example (in the .vmdb file):

```
- mkpart: primary
  device: "{{ output }}"
  start: 0%
  end: 100%
  tag: root
```

4.15 Step: `kpartx`

Create loop devices for partitions in an image file. Not needed when installing to a real block device, instead of an image file.

Step keys:

- **`kpartx`** — REQUIRED; filename of block device with partitions.

Example (in the .vmdb file):

```
- kpartx: "{{ output }}"
```

4.16 Step: `qemu-debootstrap`

Install packages using apt, which needs to already have been installed, for a different architecture than the host where vmdb2 is being run. For example, for building an image for a Raspberry Pi on an Intel PC.

Step keys:

- **`qemu-debootstrap`** — REQUIRED; value is the codename of the Debian release to install: **`stretch`**, **`buster`**, etc.
- **`target`** — REQUIRED; value is the tag for the root filesystem.
- **`mirror`** — OPTIONAL; which Debian mirror to use.
- **`arch`** — REQUIRED; the foreign architecture touse.
- **`variant`** — OPTIONAL; the variant for debootstrap.

Example (in the .vmdb file):

```
- qemu-debootstrap: stretch
  target: root
  mirror: http://mirror.example.com/debian
  arch: arm64
  variant: buildd
```

4.17 Step: `cache-rootfs`

Create a tarball of the root filesystem in the image.

Step keys:

- **`cache-rootfs`** — REQUIRED; tag of root filesystem on image.

Example (in the .vmdb file):

```
- cache-rootfs: root
  unless: rootfs_unpacked
```

4.18 Step: `unpack-rootfs`

Unpack a tarball of the root filesystem to the image, and set the **`rootfs_unpacked`** condition to true. If the tarball doesn't exist, do nothing and leave the **`rootfs_unpacked`** condition to false.

Step keys:

- **`unpack-rootfs`** — REQUIRED; tag for the root filesystem.

Example (in the .vmdb file):

```
- unpack-rootfs: root
```