NAME

virt−builder − Build virtual machine images quickly

SYNOPSIS

```
virt−builder os−version
   [−o|−−output DISKIMAGE] [−−size SIZE] [−−format raw|qcow2]
   [−−attach ISOFILE]
   [−−root−password SELECTOR]
   [−−hostname HOSTNAME]
   [−−install PKG,[PKG...]]
   [−−upload FILE:DEST]
   [−−edit FILE:EXPR]
   [−−delete FILE] [−−scrub FILE]
   [−−run SCRIPT] [−−run−command 'CMD ARGS ...']
   [−−firstboot SCRIPT] [−−firstboot−command 'CMD ARGS ...']
   [−−firstboot−install PKG,[PKG...]]

virt−builder −l|−−list [−−long]

virt−builder −−notes os−version

virt−builder −−print−cache

virt−builder −−cache−all−templates

virt−builder −−delete−cache

virt−builder −−get−kernel DISKIMAGE
   [−−format raw|qcow2] [−−output OUTPUTDIR]
```

DESCRIPTION

Virt-builder is a tool for quickly building new virtual machines. You can build a variety of VMs for local or cloud use, usually within a few minutes or less. Virt-builder also has many ways to customize these VMs. Everything is run from the command line and nothing requires root privileges, so automation and scripting is simple.

Note that virt-builder does not install guests from scratch. It takes cleanly prepared, digitally signed OS templates and customizes them. This approach is used because it is much faster, but if you need to do fresh installs you may want to look at *virt−install* (1) and *oz−install* (1).

The easiest way to get started is by looking at the examples in the next section.

EXAMPLES

List the virtual machines available

```
virt−builder −−list
```

will list out the operating systems available to install. A selection of freely redistributable OSes is available as standard. You can add your own too (see below).

After choosing a guest from the list, you may want to see if there are any installation notes:

```
virt−builder −−notes fedora−20
```

Build a virtual machine

```
virt−builder fedora−20
```

will build a Fedora 20 image. This will have all default configuration (minimal size, no user accounts, random root password, only the bare minimum installed software, etc.).

You *do not* need to run this command as root.

The first time this runs it has to download the template over the network, but this gets cached (see

‘‘CACHING’’).

The name of the output file is derived from the template name, so above it will be `fedora-20.img`. You can change the output filename using the *-o* option:

```
virt-builder fedora-20 -o mydisk.img
```

You can also use the *-o* option to write to existing devices or logical volumes.

```
virt-builder fedora-20 --format qcow2
```

As above, but write the output in qcow2 format to `fedora-20.qcow2`.

```
virt-builder fedora-20 --size 20G
```

As above, but the output size will be 20 GB. The guest OS is resized as it is copied to the output (automatically, using *virt-resize* (1)).

Setting the root password

```
virt-builder fedora-20 --root-password file:/tmp/rootpw
```

Create a Fedora 20 image. The root password is taken from the file `/tmp/rootpw`.

Note if you *don't* set *--root-password* then the guest is given a *random* root password.

You can also create user accounts. See ‘‘USERS AND PASSWORDS’’ below.

Set the hostname

```
virt-builder fedora-20 --hostname virt.example.com
```

Set the hostname to `virt.example.com`.

Installing software

To install packages from the ordinary (guest) software repository (eg. yum or apt):

```
virt-builder fedora-20 --install "inkscape,@Xfce Desktop"
```

(In Fedora, `@` is used to install groups of packages. On Debian you would install a meta-package instead.)

Customizing the installation

There are many options that let you customize the installation. These include: *--run*/*--run-command*, which run a shell script or command while the disk image is being generated and lets you add or edit files that go into the disk image. *--firstboot*/*--firstboot-command*, which let you add scripts/commands that are run the first time the guest boots. *--edit* to edit files. *--upload* to upload files.

For example:

```
cat <<'EOF' > /tmp/yum-update.sh
yum -y update
EOF

virt-builder fedora-20 --firstboot /tmp/yum-update.sh
```

or simply:

```
virt-builder fedora-20 --firstboot-command 'yum -y update'
```

which makes the *yum* (8) `update` command run once the first time the guest boots.

Or:

```
virt-builder fedora-20 --edit '/etc/yum.conf: s/gpgcheck=1/gpgcheck=0/'
```

which edits `/etc/yum.conf` inside the disk image (during disk image creation, long before boot).

You can combine these options, and have multiple options of all types.

OPTIONS

--help

Display help.

--attach ISOFILE
: During the customization phase, the given disk is attached to the libguestfs appliance. This is used to provide extra software repositories or other data for customization.

  You probably want to ensure the volume(s) or filesystems in the attached disks are labelled (or use an ISO volume name) so that you can mount them by label in your run-scripts:

```
mkdir /tmp/mount
mount LABEL=EXTRA /tmp/mount
```

  You can have multiple *--attach* options, and the format can be any disk format (not just an ISO).

  See also: *--run*, ‘‘Installing packages at build time from a side repository’’, *genisoimage*(1), *virt-make-fs*(1).

--attach-format FORMAT
: Specify the disk format for the next *--attach* option. The `FORMAT` is usually `raw` or `qcow2`. Use `raw` for ISOs.

--cache DIR
--no-cache
: *--cache* DIR sets the directory to use/check for cached template files. If not set, defaults to either `$XDG_CACHE_HOME/virt-builder/` or `$HOME/.cache/virt-builder/`.

  --no-cache disables template caching.

--cache-all-templates
: Download all templates to the cache and then exit. See ‘‘CACHING’’.

  Note this doesn’t cache everything. More templates might be uploaded. Also this doesn’t cache packages (the *--install* option).

--check-signature
--no-check-signature
: Check/don’t check the digital signature of the OS template. The default is to check the signature and exit if it is not correct. Using *--no-check-signature* bypasses this check.

  See also *--fingerprint*.

--curl CURL
: Specify an alternate *curl*(1) binary. You can also use this to add curl parameters, for example to disable https certificate checks:

```
virt-builder --curl "curl --insecure" [...]
```

--delete FILE
--delete DIR
: Delete a file from the guest. Or delete a directory (and all its contents, recursively).

  See also: *--upload*, *--scrub*.

--delete-cache
: Delete the template cache. See ‘‘CACHING’’.

--edit FILE:EXPR
: Edit `FILE` using the Perl expression `EXPR`.

  Be careful to properly quote the expression to prevent it from being altered by the shell.

  Note that this option is only available when Perl 5 is installed.

  See ‘‘NON-INTERACTIVE EDITING’’ in *virt-edit*(1).

--fingerprint ’AAAA BBBB ...’
: Check that the index and templates are signed by the key with the given fingerprint. (The fingerprint is a long string, usually written as 10 groups of 4 hexadecimal digits).

If signature checking is enabled and the *--fingerprint* option is not given, then this checks the download was signed by F7774FB1AD074A7E8C8767EA91738F73E1B768A0 (which is RichardW.M.Jones's key).

You can also set the `VIRT_BUILDER_FINGERPRINT` environment variable.

--firstboot SCRIPT

--firstboot-command 'CMD ARGS ...'

Install `SCRIPT` inside the guest, so that when the guest first boots up, the script runs (as root, late in the boot process).

The script is automatically chmod +x after installation in the guest.

The alternative version *--firstboot-command* is the same, but it conveniently wraps the command up in a single line script for you.

You can have multiple *--firstboot* and *--firstboot-command* options. They run in the same order that they appear on the command line.

See also *--run*.

--firstboot-install PKG[,PKG,...]

Install the named packages (a comma-separated list). These are installed when the guest first boots using the guest's package manager (eg. apt, yum, etc.) and the guest's network connection.

For an overview on the different ways to install packages, see ''INSTALLING PACKAGES''.

--format qcow2

--format raw

For ordinary builds, this selects the output format. The default is *raw*.

With *--get-kernel* this specifies the input format.

--get-kernel IMAGE

This option extracts the kernel and initramfs from a previously built disk image called `IMAGE` (in fact it works for any VM disk image, not just ones built using virt-builder).

The kernel and initramfs are written to the current directory, unless you also specify the *--output* `outputdir` **directory** name.

The format of the disk image is automatically detected unless you specify it by using the *--format* option.

In the case where the guest contains multiple kernels, the one with the highest version number is chosen. To extract arbitrary kernels from the disk image, see *guestfish*(1). To extract the entire `/boot` directory of a guest, see *virt-copy-out*(1).

--gpg GPG

Specify an alternate *gpg*(1) (GNU Privacy Guard) binary. You can also use this to add gpg parameters, for example to specify an alternate home directory:

```
virt-builder --gpg "gpg --homedir /tmp" [...]
```

--hostname HOSTNAME

Set the hostname of the guest to `HOSTNAME`. You can use a dotted hostname.domainname (FQDN) if you want.

--install PKG[,PKG,...]

Install the named packages (a comma-separated list). These are installed during the image build using the guest's package manager (eg. apt, yum, etc.) and the host's network connection.

For an overview on the different ways to install packages, see ''INSTALLING PACKAGES''.

-l

--list
--list --long
List available templates.

The alternative *--list --long* form shows lots more details about each operating system option.

See also: *--source*, *--notes*, ''CREATING YOUR OWN TEMPLATES''.

--no-logfile
Scrub `builder.log` (log file from build commands) from the image after building is complete. If you don't want to reveal precisely how the image was built, use this option.

See also: ''LOG FILE''.

--network
--no-network
Enable or disable network access from the guest during the installation.

Enabled is the default. Use *--no-network* to disable access.

The network only allows outgoing connections and has other minor limitations. See ''NETWORK'' in *virt-rescue* (1).

If you use *--no-network* then certain other options such as *--install* will not work.

This does not affect whether the guest can access the network once it has been booted, because that is controlled by your hypervisor or cloud environment and has nothing to do with virt-builder.

Generally speaking you should *not* use *--no-network*. But here are some reasons why you might want to:

1. Because the libguestfs backend that you are using doesn't support the network. (See: ''BACKEND'' in *guestfs* (3)).
2. Any software you need to install comes from an attached ISO, so you don't need the network.
3. You don't want untrusted guest code trying to access your host network when running virt-builder. This is particularly an issue when you don't trust the source of the operating system templates. (See ''SECURITY'' below).
4. You don't have a host network (eg. in secure/restricted environments).

--notes os-version
List any notes associated with this guest, then exit (this does not do the install).

-o filename
--output filename
Write the output to `filename`. If you don't specify this option, then the output filename is generated by taking the `os-version` string and adding `.img` (for raw format) or `.qcow2` (for qcow2 format).

Note that the output filename could be a device, partition or logical volume.

When used with *--get-kernel*, this option specifies the output directory.

--password-crypto password-crypto
Set the password encryption to `md5`, `sha256` or `sha512`.

`sha256` and `sha512` require glibc ≥ 2.7 (check *crypt* (3) inside the guest).

`md5` will work with relatively old Linux guests (eg. RHEL 3), but is not secure against modern attacks.

The default is `sha512` unless libguestfs detects an old guest that didn't have support for SHA-512, in which case it will use `md5`. You can override libguestfs by specifying this option.

--print-cache
Print information about the template cache. See ''CACHING''.

--quiet
Don't print ordinary progress messages.

--root-password SELECTOR
Set the root password.

See ''USERS AND PASSWORDS'' below for the format of the `SELECTOR` field, and also how to set up user accounts.

Note if you *don't* set *--root-password* then the guest is given a *random* root password.

--run SCRIPT
--run-command 'CMD ARGS ...'
Run the shell script (or any program) called `SCRIPT` on the disk image. The script runs virtualized inside a small appliance, chrooted into the guest filesystem.

The script is automatically chmod +x.

If libguestfs supports it then a limited network connection is available but it only allows outgoing network connections. You can also attach data disks (eg. ISO files) as another way to provide data (eg. software packages) to the script without needing a network connection (*--attach*). You can also upload data files (*--upload*).

The alternative version *--run-command* is the same, but it conveniently wraps the command up in a single line script for you.

You can have multiple *--run* and *--run-command* options. They run in the same order that they appear on the command line.

See also: *--firstboot*, *--attach*, *--upload*.

--scrub FILE
Scrub a file from the guest. This is like *--delete* except that:

- It scrubs the data so a guest could not recover it.
- It cannot delete directories, only regular files.

--size SIZE
Select the size of the output disk, where the size can be specified using common names such as `32G` (32 gigabytes) etc.

Virt-builder will resize filesystems inside the disk image automatically.

If the size is not specified, then one of two things happens. If the output is a file, then the size is the same as the template. If the output is a device, partition, etc then the size of that device is used.

To specify size in bytes, the number must be followed by the lowercase letter *b*, eg: `--size10737418240b`.

--source URL
Set the source URL to look for templates. If not specified it defaults to http://libguestfs.org/download/builder/index.asc

See also ''CREATING YOUR OWN TEMPLATES'' below.

You can also set the `VIRT_BUILDER_SOURCE` environment variable.

Note that you should not point *--source* to sources that you don't trust (unless the source is signed by someone you do trust). See also the *--no-network* option.

--upload FILE:DEST
Upload local file `FILE` to destination `DEST` in the disk image. File owner and permissions from the original are preserved, so you should set them to what you want them to be in the disk image.

`DEST` could be the final filename. This can be used to rename the file on upload.

If DEST is a directory name (which must already exist in the guest) then the file is uploaded into that directory, and it keeps the same name as on the local filesystem.

See also: *--mkdir*, *--delete*, *--scrub*.

-v
--verbose
Enable debug messages and/or produce verbose output.

When reporting bugs, use this option and attach the complete output to your bug report.

-V
--version
Display version number and exit.

REFERENCE

INSTALLING PACKAGES

There are several approaches to installing packages or applications in the guest which have different trade-offs.

Installing packages at build time

If the guest OS you are installing is similar to the host OS (eg. both are Linux), and if libguestfs supports network connections, then you can use *--install* to install packages like this:

```
virt-builder fedora-20 --install inkscape
```

This uses the guest's package manager and the host's network connection.

Installing packages at first boot

Another option is to install the packages when the guest first boots:

```
virt-builder fedora-20 --firstboot-install inkscape
```

This uses the guest's package manager and the guest's network connection.

The downsides are that it will take the guest a lot longer to boot first time, and there's nothing much you can do if package installation fails (eg. if a network problem means the guest can't reach the package repositories).

Installing packages at build time from a side repository

If the software you want to install is not available in the main package repository of the guest, then you can add a side repository. Usually this is presented as an ISO (CD disk image) file containing extra packages.

You can create the disk image using either *genisoimage*(1) or *virt-make-fs*(1). For genisoimage, use a command like this:

```
genisoimage -o extra-packages.iso -R -J -V EXTRA cdcontents/
```

Create a script that mounts the ISO and sets up the repository. For yum, create /tmp/install.sh containing:

```
mkdir /tmp/mount
mount LABEL=EXTRA /tmp/mount

cat <<'EOF' > /etc/yum.repos.d/extra.repo
[extra]
name=extra
baseurl=file:///tmp/mount
enabled=1
EOF

yum -y install famousdatabase
```

For apt, create /tmp/install.sh containing:

```
mkdir /tmp/mount
mount LABEL=EXTRA /tmp/mount

apt-cdrom -d=/tmp/mount add
apt-get -y install famousdatabase
```

Use the *--attach* option to attach the CD / disk image and the *--run* option to run the script:

```
virt-builder fedora-20 \
  --attach extra-packages.iso \
  --run /tmp/install.sh
```

USERS AND PASSWORDS

The *--root-password* option is used to change the root password (otherwise a random password is used). This option takes a password `SELECTOR` in one of the following formats:

--root-password file:FILENAME

Read the root password from `FILENAME`. The whole first line of this file is the replacement password. Any other lines are ignored. You should create the file with mode 0600 to ensure no one else can read it.

--root-password password:PASSWORD

Set the root password to the literal string `PASSWORD`.

Note: this is not secure since any user on the same machine can see the cleartext password using *ps* (1).

Creating user accounts

To create user accounts, use the *useradd* (8) command with --firstboot-command like this:

```
virt-builder --firstboot-command \
   'useradd -m -p "" rjones ; chage -d 0 rjones'
```

The above command will create an `rjones` account with no password, and force the user to set a password when they first log in. There are other ways to manage passwords, see *useradd* (8) for details.

LOG FILE

Scripts and package installation that runs at build time (*--run*, *--run-command*, *--install*, but *not* firstboot) is logged in one of the following locations:

`/tmp/builder.log`

On Linux, BSD and other guests.

`C:\Temp\builder.log`

On Windows, DOS guests.

`/builder.log`

If `/tmp` or `C:\Temp` is missing.

If you don't want the log file to appear in the final image, then use the *--no-logfile* command line option.

INSTALLATION PROCESS

When you invoke virt-builder, installation proceeds as follows:

- The template image is downloaded.

  If the template image is present in the cache, the cached version is used instead. (See ''CACHING'').
- The template signature is checked.
- The template is uncompressed to a tmp file.
- The template image is resized into the destination, using *virt-resize* (1).
- Extra disks are attached (*--attach*).

- A new random seed is generated for the guest.
- The hostname is set (*--hostname*).
- The root password is changed (*--root-password*).
- Packages are installed (*--install*).
- Files are uploaded (*--upload*).
- Files are edited (*--edit*).
- Files are deleted (*--delete*, *--scrub*).
- Firstboot scripts are installed (*--firstboot*, *--firstboot-command*, *--firstboot-install*).

  Note that although firstboot scripts are installed at this step, they do not run until the guest is booted first time. Firstboot scripts will run in the order they appear on the command line.
- Scripts are run (*--run*, *--run-command*).

  Scripts run in the order they appear on the command line.

IMPORTING THE DISK IMAGE

Importing into libvirt

Import the disk image into libvirt using *virt-install* (1) *--import* option.

```
virt-install --import \
  --name guest --ram 2048 --disk path=disk.img,format=raw
```

Notes:

1. You *must* specify the correct format. The format is `raw` unless you used virt-builder's *--format* option.
2. You can run virt-install as root or non-root. Each works slightly differently because libvirt manages a different set of virtual machines for each user. In particular virt-manager normally shows the root-owned VMs, whereas Boxes shows the user-owned VMs, and other tools probably work differently as well.

Importing into OpenStack

Import the image into Glance (the OpenStack image store) by doing:

```
glance image-create --name fedora-20-image --file fedora-20.img \
  --disk-format raw --container-format bare \
  --is-public True
```

The *--file* parameter is the virt-builder-generated disk image. It should match virt-builder's *--output* option. The *--disk-format* parameter should match virt-builder's *--format* option (or `raw` if you didn't use that option). The *--container-format* should always be `bare` since virt-builder doesn't put images into containers.

You can use the `glanceimage-show`*`fedora-20-image`* command to display the properties of the image.

To boot up an instance of your image on a Nova compute node, do:

```
nova boot fedora-20-server --image fedora-20-image \
  --flavor m1.medium
```

Use `novaflavor-list` to list possible machine flavors. Use `novalist` to list running instances.

DEBUGGING BUILDS

If virt-builder fails with an error, then enable debugging (*-v*) and report a bug (see ''BUGS'' below).

If virt-builder is successful but the image doesn't work, here are some things to try:

Use virt-rescue

Run *virt−rescue* (1) on the disk image:

```
virt−rescue −a disk.img
```

This gives you a rescue shell. You can mount the filesystems from the disk image on `/sysroot` and examine them using ordinary Linux commands. You can also chroot into the guest to reinstall the bootloader. The virt-rescue man page has a lot more information and examples.

Use guestfish

Run *guestfish* (1) on the disk image:

```
guestfish −a disk.img −i
```

Use guestfish commands like `ll /directory` and `cat /file` to examine directories and files.

Use guestmount

Mount the disk image safely on the host using FUSE and *guestmount* (1):

```
mkdir /tmp/mp
guestmount −a disk.img −i /tmp/mp
cd /tmp/mp
```

To unmount the disk image do:

```
fusermount −u /tmp/mp
```

Add a serial console

If the guest hangs during boot, it can be helpful to add a serial console to the guest, and direct kernel messages to the serial console. Adding the serial console will involve looking at the documentation for your hypervisor. To direct kernel messages to the serial console, add the following on the kernel command line:

```
console=tty0 console=ttyS0,115200
```

CREATING YOUR OWN TEMPLATES

For serious virt-builder use, you may want to create your own repository of templates.

Libguestfs.org repository

Out of the box, virt-builder downloads the file http://libguestfs.org/download/builder/index.asc which is an index of available templates plus some information about each one, wrapped up in a digital signature. The command `virt−builder −−list` lists out the information in this index file.

The templates hosted on libguestfs.org were created using shell scripts, kickstart files and preseed files which can be found in the libguestfs source tree, in `builder/website`.

Setting up the repository

You can set up your own site containing an index file and some templates, and then point virt-builder at the site by using the *−−source* option:

```
virt−builder −−source https://example.com/builder/index.asc \
   −−fingerprint 'AAAA BBBB ...' \
   −−list
```

(Note setting the environment variables `VIRT_BUILDER_SOURCE` and `VIRT_BUILDER_FINGERPRINT` may be easier to type!)

You can host this on any web or FTP server, or a local or network filesystem.

Setting up a GPG key

If you don't have a GnuPG key, you will need to set one up. (Strictly speaking this is optional, but if your index and template files are not signed then virt-builder users will have to use the *−−no−check−signature* flag every time they use virt-builder.)

To create a key, see the GPG manual http://www.gnupg.org/gph/en/manual.html.

Export your GPG public key and add it to the keyring of all virt-builder users:

```
gpg --export -a "you@example.com" > pubkey

# For each virt-builder user:
gpg --import pubkey
```

Also find the fingerprint of your key:

```
gpg --list-keys --fingerprint
```

Create the templates

There are many ways to create the templates. For example you could clone existing guests (see *virt-sysprep*(1)), or you could install a guest by hand (*virt-install*(1)). To see how the templates were created for virt-builder, look at the scripts in `builder/website`

For best results when compressing the templates, use the following xz options (see *nbdkit-xz-plugin*(1) for further explanation):

```
xz --best --block-size=16777216 disk
```

Creating and signing the index file

The index file has a simple text format (shown here without the digital signature):

```
[fedora-18]
name=Fedora® 18
osinfo=fedora18
file=fedora-18.xz
checksum[sha512]=...
format=raw
size=6442450944
compressed_size=148947524
expand=/dev/sda3

[fedora-19]
name=Fedora® 19
osinfo=fedora19
file=fedora-19.xz
checksum[sha512]=...
revision=3
format=raw
size=4294967296
compressed_size=172190964
expand=/dev/sda3
```

The part in square brackets is the `os-version`, which is the same string that is used on the virt-builder command line to build that OS.

After preparing the `index` file in the correct format, clearsign it using the following command:

```
gpg --clearsign --armor index
```

This will create the final file called `index.asc` which can be uploaded to the server (and is the *--source* URL). As noted above, signing the index file is optional, but recommended.

The following fields can appear:

`name=NAME`

The user-friendly name of this template. This is displayed in the *--list* output but is otherwise not significant.

`osinfo=ID`
: This optional field maps the operating system to the associated libosinfo ID. Virt-builder does not use it (yet).

`file=PATH`
: The path (relative to the index) of the xz-compressed template.

  Note that absolute paths or URIs are **not** permitted here. This is because virt-builder has a ‘‘same origin’’ policy for templates so they cannot come from other servers.

`sig=PATH`
: **This option is deprecated**. Use the checksum field instead.

  The path (relative to the index) of the GPG detached signature of the xz file.

  Note that absolute paths or URIs are **not** permitted here. This is because virt-builder has a ‘‘same origin’’ policy for templates so they cannot come from other servers.

  The file can be created as follows:

```
gpg --detach-sign --armor -o disk.xz.sig disk.xz
```

`checksum[sha512]=7b882fe9b82eb0fef...`
: The SHA-512 checksum of the **compressed** file is checked after it is downloaded. To work out the signature, do:

```
sha512sum disk.xz
```

  Note if you use this, you don’t need to sign the file, ie. don’t use `sig`. This option overrides `sig`.

`checksum=7b882fe9b82eb0fef...`
: `checksum` is an alias for `checksum[sha512]`.

  If you need to interoperate with virt-builder = 1.24.0 then you have to use `checksum` because that version would give a parse error with square brackets and numbers in the key of a field. This is fixed in virt-builder ≥ 1.24.1.

`revision=N`
: The revision is an integer which is used to control the template cache. Increasing the revision number causes clients to download the template again even if they have a copy in the cache.

  The revision number is optional. If omitted it defaults to `1`.

`format=raw`
`format=qcow2`
: Specify the format of the disk image (before it was compressed). If not given, the format is autodetected, but generally it is better to be explicit about the intended format.

  Note this is the source format, which is different from the *--format* option (requested output format). Virt-builder does on-the-fly conversion from the source format to the requested output format.

`size=NNN`
: The virtual size of the image in bytes. This is the size of the image when uncompressed. If using a non-raw format such as qcow2 then it means the virtual disk size, not the size of the qcow2 file.

  This field is required.

  Virt-builder also uses this as the minimum size that users can request via the *--size* option, or as the default size if there is no *--size* option.

`compressed_size=NNN`
: The compressed size of the disk image in bytes. This is just used for information (when using *--list --long*).

`expand=/dev/sdaX`
When expanding the image to its final size, instruct *virt−resize* (1) to expand the named partition in the guest image to fill up all available space. This works like the virt-resize `−−`*expand* option.

You should usually put the device name of the guest's root filesystem here.

It's a good idea to use this, but not required. If the field is omitted then virt-resize will create an extra partition at the end of the disk to cover the free space, which is much less user-friendly.

`lvexpand=/dev/VolGroup/LogVol`
When expanding the image to its final size, instruct *virt−resize* (1) to expand the named logical volume in the guest image to fill up all available space. This works like the virt-resize `−−`*lv−expand* option.

If the guest uses LVM2 you should usually put the LV of the guest's root filesystem here. If the guest does not use LVM2 or its root filesystem is not on an LV, don't use this option.

`notes=NOTES`
Any notes that go with this image, especially notes describing what packages are in the image, how the image was prepared, and licensing information.

This information is shown in the `−−`*notes* and `−−`*list* `−−`*long* modes.

You can use multi-line notes here by indenting each new line with at least one character of whitespace (even on blank lines):

```
notes=This image was prepared using
 the following kickstart script:
                                <-- one space at beginning of line
 part /boot --fstype ext3
 ...
```

`hidden=true`
Using the hidden flag prevents the template from being listed by the `−−`*list* option (but it is still installable). This is used for test images.

Running virt-builder against the alternate repository

Ensure each virt-builder user has imported your public key into their gpg keyring (see above).

Each virt-builder user should export these environment variables:

- `VIRT_BUILDER_SOURCE` to point to the URL of the `index.asc` file.
- `VIRT_BUILDER_FINGERPRINT` to contain the fingerprint (long hex string) of the user who signed the index file and the templates.

Now run virt-builder commands as normal, eg:

```
virt-builder --list --long

virt-builder os-version
```

To debug problems, add the `-v` option to these commands.

Licensing of templates

You should be aware of the licensing of images that you distribute. For open source guests, provide a link to the source code in the `notes` field and comply with other requirements (eg. around trademarks).

CACHING

Since the templates are usually very large, downloaded templates are cached in the user's home directory.

The location of the cache is `$XDG_CACHE_HOME/virt-builder/` or `$HOME/.cache/virt-builder`.

You can print out information about the cache directory, including which guests are currently cached, by doing:

```
virt-builder --print-cache
```

The cache can be deleted if you want to save space by doing:

```
virt-builder --delete-cache
```

You can download all (current) templates to the local cache by doing:

```
virt-builder --cache-all-templates
```

To disable the template cache, use *--no-cache*.

Only templates are cached. The index and detached digital signatures are not cached.

Virt-builder uses *curl*(1) to download files and it also uses the current `http_proxy` (etc) settings when installing packages (*--install*). You may therefore want to set those environment variables in order to maximize the amount of local caching that happens. See ''ENVIRONMENT VARIABLES'' and *curl*(1).

DIGITAL SIGNATURES

Virt-builder uses GNU Privacy Guard (GnuPG or gpg) to verify that the index and templates have not been tampered with.

The source points to an index file, which is optionally signed.

Virt-builder downloads the index and checks that the signature is valid and the signer's fingerprint matches the specified fingerprint (ie. *--fingerprint*, `VIRT_BUILDER_FINGERPRINT`, or a built-in fingerprint, in that order).

For checking against the built-in public key/fingerprint, this requires importing the public key into the user's local gpg keyring (that's just the way that gpg works).

When a template is downloaded, its signature is checked in the same way.

Although the signatures are optional, if you don't have them then virt-builder users will have to use *--no-check-signature* on the command line. This prevents an attacker from replacing the signed index file with an unsigned index file and having virt-builder silently work without checking the signature. In any case it is highly recommended that you always create signed index and templates.

ARCHITECTURE

Virt-builder can build a guest for any architecture no matter what the host architecture is. For example an x86-64 guest on an ARM host.

However certain options may not work correctly, specifically options that require running commands in the guest during the build process: *--install*, *--run*, *--run-command*. You may need to replace these with their firstboot-equivalents.

An x86-64 host building 32 bit i686 guests should work without any special steps.

SECURITY

Virt-builder does not need to run as root (in fact, should not be run as root), and doesn't use setuid, `sudo` or any similar mechanism.

--install, *--run* and *--run-command* are implemented using an appliance (a small virtual machine) so these commands do not run on the host. If you are using the libguestfs libvirt backend and have SELinux enabled then the virtual machine is additionally encapsulated in an SELinux container (sVirt).

However these options will have access to the host's network and since the template may contain untrusted code, the code might try to access host network resources which it should not. You can use *--no-network* to prevent this.

Firstboot commands run in the context of the guest when it is booted, and so the security of your hypervisor / cloud should be considered.

Virt-builder injects a random seed into every guest which it builds. This helps to ensure that TCP sequence numbers, UUIDs, ssh host keys etc are truly random when the guest boots.

You should check digital signatures and not ignore any signing errors.

USER MODE LINUX

You can use virt-builder with the User-Mode Linux (UML) backend. This may be faster when running virt-builder inside a virtual machine (eg. in the cloud).

To enable the UML backend, read the instructions in ‘‘USER-MODE LINUX BACKEND’’ in *guestfs* (3).

Currently you have to use the *−−no−network* option. This should be fixed in a future version.

The qcow2 output format is not supported by UML. You can only create raw-format guests.

SELINUX

Guests which use SELinux (such as Fedora and Red Hat Enterprise Linux) require that each file has a correct SELinux label.

Since virt-builder does not know how to give new files a correct label, it touches `/.autorelabel` in the guest and relies on the guest to relabel itself at first boot.

This usually means that these guests will reboot themselves once the first time you use them. This is normal, and harmless.

ENVIRONMENT VARIABLES

For other environment variables which affect all libguestfs programs, see ‘‘ENVIRONMENT VARIABLES’’ in *guestfs* (3).

`http_proxy`
`https_proxy`
`no_proxy`

Set the proxy for downloads. These environment variables (and more) are actually interpreted by *curl* (1), not virt-builder.

`HOME`

Used to determine the location of the template cache. See ‘‘CACHING’’.

`LIBGUESTFS_MEMSIZE`

The size (in megabytes) of the appliance. The default can be found using the command `guestfishget−memsize`. Increase this if you find that *−−run* scripts are running out of memory.

`VIRT_BUILDER_FINGERPRINT`

Set the default value for the GPG signature fingerprint (see *−−fingerprint* option).

`VIRT_BUILDER_SOURCE`

Set the default value for the source URL for the template repository (see *−−source* option).

`XDG_CACHE_HOME`

Used to determine the location of the template cache. See ‘‘CACHING’’.

EXIT STATUS

This program returns 0 if successful, or non-zero if there was an error.

SEE ALSO

guestfs (3), *guestfish* (1), *guestmount* (1), *virt−copy−out* (1), *virt−install* (1), *virt−rescue* (1), *virt−resize* (1), *virt−sysprep* (1), *oz−install* (1), *gpg* (1), *curl* (1), *virt−make−fs* (1), *genisoimage* (1), http://libguestfs.org/.

AUTHOR

Richard W.M. Jones http://people.redhat.com/˜rjones/

COPYRIGHT

LICENSE

General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110−1301 USA.

BUGS

To get a list of bugs against libguestfs, use this link: https://bugzilla.redhat.com/buglist.cgi?component=libguestfs&product=Virtualization+Tools

To report a new bug against libguestfs, use this link: https://bugzilla.redhat.com/enter_bug.cgi?component=libguestfs&product=Virtualization+Tools

When reporting a bug, please supply:

- The version of libguestfs.
- Where you got libguestfs (eg. which Linux distro, compiled from source, etc)
- Describe the bug accurately and give a way to reproduce it.
- Run *libguestfs−test−tool*(1) and paste the **complete, unedited** output into the bug report.