NAME

virt−builder − Build virtual machine images quickly

SYNOPSIS

```
 virt-builder os-version
    [-o|--output DISKIMAGE] [--size SIZE] [--format raw|qcow2]
    [--arch ARCHITECTURE] [--attach ISOFILE]
    [--append-line FILE:LINE] [--chmod PERMISSIONS:FILE]
    [--commands-from-file FILENAME] [--copy SOURCE:DEST]
    [--copy-in LOCALPATH:REMOTEDIR] [--delete PATH] [--edit FILE:EXPR]
    [--firstboot SCRIPT] [--firstboot-command 'CMD+ARGS']
    [--firstboot-install PKG,PKG..] [--hostname HOSTNAME]
    [--install PKG,PKG..] [--link TARGET:LINK[:LINK..]] [--mkdir DIR]
    [--move SOURCE:DEST] [--password USER:SELECTOR]
    [--root-password SELECTOR] [--run SCRIPT]
    [--run-command 'CMD+ARGS'] [--scrub FILE] [--sm-attach SELECTOR]
    [--sm-register] [--sm-remove] [--sm-unregister]
    [--ssh-inject USER[:SELECTOR]] [--truncate FILE]
    [--truncate-recursive PATH] [--timezone TIMEZONE] [--touch FILE]
    [--uninstall PKG,PKG..] [--update] [--upload FILE:DEST]
    [--write FILE:CONTENT] [--no-logfile]
    [--password-crypto md5|sha256|sha512] [--selinux-relabel]
    [--sm-credentials SELECTOR]


 virt-builder -l|--list [--long] [--list-format short|long|json] [os-version]

 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-27
```

Build a virtual machine

```
virt-builder fedora-27
```

will build a Fedora 25 image for the same architecture as virt-builder (so running it from an i686 installation will try to build an i686 image, if available). 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−27.img*. You can change the output filename using the *−o* option:

```
virt-builder fedora-27 -o mydisk.img
```

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

```
virt-builder fedora-27 --format qcow2
```

As above, but write the output in qcow2 format to *fedora−27.qcow2*.

```
virt-builder fedora-27 --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)).

```
virt-builder fedora-27 --arch i686
```

As above, but using an i686 template, if available.

Setting the root password

```
virt-builder fedora-27 --root-password file:/tmp/rootpw
```

Create a Fedora 25 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 which is printed on stdout.

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

Set the hostname

```
virt-builder fedora-27 --hostname virt.example.com
```

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

Installing software

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

```
virt-builder fedora-27 --install "inkscape,@Xfce Desktop"
```

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

To update the installed packages to the latest version:

```
virt-builder debian-7 --update
```

For guests which use SELinux, like Fedora and Red Hat Enterprise Linux, you may need to do SELinux relabelling after installing or updating packages (see "SELINUX" below):

```
virt-builder fedora-27 --update --selinux-relabel
```

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. *−−fir stboot*/*−−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/dnf-update.sh
dnf -y --best update
EOF

virt-builder fedora-27 --firstboot /tmp/dnf-update.sh
```

or simply:

```
virt-builder fedora-27 --firstboot-command 'dnf -y --best update'
```

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

Or:

```
virt-builder fedora-27 \
  --edit '/etc/dnf/dnf.conf:
          s/gpgcheck=1/gpgcheck=0/'
```

which edits */etc/dnf/dnf.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.

--arch ARCHITECTURE
Use the specified architecture for the output image. This means there must be sources providing the requested template for the requested architecture.

See also "ARCHITECTURE".

--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*, *--update* options).

−−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*.

−−colors
−−colours

Use ANSI colour sequences to colourize messages. This is the default when the output is a tty. If the output of the program is redirected to a file, ANSI colour sequences are disabled unless you use this option.

−−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−cache

Delete the template cache. See ''CACHING''.

−−no−delete−on−failure

Don't delete the output file on failure to build. You can use this to debug failures to run scripts. See ''DEBUGGING BUILDS'' for ways to debug images.

The default is to delete the output file if virt-builder fails (or, for example, some script that it runs fails).

−−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).

You can give this option multiple times. If you have multiple source URLs, then you can have either no fingerprint, one fingerprint or multiple fingerprints. If you have multiple, then each must correspond 1−1 with a source URL.

−−format qcow2
−−format raw

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

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

To create an old-style qcow2 file (for compatibility with RHEL 6 or very old qemu < 1.1), after running virt-builder, use this command:

```
 qemu-img amend -f qcow2 -o compat=0.10 output.qcow2
```

−−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).

Note this method is **deprecated**: there is a separate tool for this, **virt−get−kernel**(1), which has more options for the file extraction.

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. By default virt-builder looks for either `gpg2` or `gpg` in the `$PATH`.

You can also use this to add gpg parameters, for example to specify an alternate home directory:

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

−l [os−version]
−−list [os−version]
−−list −−list−format format [os−version]
−−list −−long [os−version]

List all the available templates if no guest is specified, or only for the specified one.

It is possible to choose with *−−list−format* the output format for the list templates:

short

The default format, prints only the template identifier and, next to it, its short description.

long

Prints a textual list with the details of the available sources, followed by the details of the available templates.

json

Prints a JSON object with the details of the available sources and the details of the available templates.

The `version` key in the main object represents the ‘‘compatibility version’’, and it is bumped every time the resulting JSON output is incompatible with the previous versions (for example the structure has changed, or non-optional keys are no more present).

−−long is a shorthand for the `long` format.

See also: *−−source*, *−−notes*, ‘‘SOURCES OF TEMPLATES’’.

−−machine−readable
−−machine−readable=format

This option is used to make the output more machine friendly when being parsed by other programs. See ‘‘MACHINE READABLE OUTPUT’’ below.

−m MB
−−memsize MB

Change the amount of memory allocated to *−−run* scripts. Increase this if you find that *−−run* scripts or the *−−install* option are running out of memory.

The default can be found with this command:

```
 guestfish get-memsize
```

−−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).

−−no−sync
Do not sync the output file on exit.

Virt-builder `fsync`s the output file or disk image when it exits.

The reason is that qemu/KVM's default caching mode is `none` or `directsync`, both of which bypass the host page cache. Therefore these would not work correctly if you immediately started the guest after running virt-builder − they would not see the complete output file. (Note that you should not use these caching modes − they are fundamentally broken for this and other reasons.)

If you are not using these broken caching modes, you can use *−−no−sync* to avoid this unnecessary sync and gain considerable extra performance.

−−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.

−−print−cache
Print information about the template cache. See "CACHING".

−q
−−quiet
Don't print ordinary progress messages.

−−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`.

−−smp N
Enable N ≥ 2 virtual CPUs for *−−run* scripts to use.

−−source URL
Set the source URL to look for indexes.

You can give this option multiple times to specify multiple sources.

See also "SOURCES OF TEMPLATES" below.

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.

−−no−warn−if−partition
Do not emit a warning if the output device is a partition. This warning avoids a common user error when writing to a USB key or external drive, when you should normally write to the whole device (−−*output/dev/sdX*), not to a partition on the device (−−*output/dev/sdX1*). Use this option to*suppr ess* this warning.

−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.

−x Enable tracing of libguestfs API calls.

Customization options

−−append−line FILE:LINE
Append a single line of text to the `FILE`. If the file does not already end with a newline, then one is added before the appended line. Also a newline is added to the end of the `LINE` string automatically.

For example (assuming ordinary shell quoting) this command:

```
 --append-line '/etc/hosts:10.0.0.1 foo'
```

will add either `10.0.0.1 foo` or `10.0.0.1 foo` to the file, the latter only if the existing file does not already end with a newline.

represents a newline character, which is guessed by looking at the existing content of the file, so this command does the right thing for files using Unix or Windows line endings. It also works for empty or non-existent files.

To insert several lines, use the same option several times:

```
 --append-line '/etc/hosts:10.0.0.1 foo'
 --append-line '/etc/hosts:10.0.0.2 bar'
```

To insert a blank line before the appended line, do:

```
 --append-line '/etc/hosts:'
 --append-line '/etc/hosts:10.0.0.1 foo'
```

−−chmod PERMISSIONS:FILE
Change the permissions of `FILE` to `PERMISSIONS`.

Note: `PERMISSIONS` by default would be decimal, unless you prefix it with `0` to get octal, ie. use `0700` not `700`.

−−commands−from−file FILENAME
Read the customize commands from a file, one (and its arguments) each line.

Each line contains a single customization command and its arguments, for example:

```
 delete /some/file
 install some-package
 password some-user:password:its-new-password
```

Empty lines are ignored, and lines starting with # are comments and are ignored as well. Furthermore, arguments can be spread across multiple lines, by adding a `\` (continuation character) at the of a line, for example

```
 edit /some/file:\
   s/^OPT=.*/OPT=ok/
```

The commands are handled in the same order as they are in the file, as if they were specified as *−−delete /some/file* on the command line.

−−copy SOURCE:DEST
Copy files or directories recursively inside the guest.

Wildcards cannot be used.

−−copy−in LOCALPATH:REMOTEDIR
Copy local files or directories recursively into the disk image, placing them in the directory `REMOTEDIR` (which must exist).

Wildcards cannot be used.

−−delete PATH
Delete a file from the guest. Or delete a directory (and all its contents, recursively).

You can use shell glob characters in the specified path. Be careful to escape glob characters from the host shell, if that is required. For example:

```
 virt-customize --delete '/var/log/*.log'.
```

See also: *−−upload*, *−−scrub*.

−−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).

−−firstboot SCRIPT
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* options. They run in the same order that they appear on the command line.

Please take a look at "FIRST BOOT SCRIPTS" for more information and caveats about the first boot scripts.

See also *−−run*.

−−firstboot−command 'CMD+ARGS'
Run command (and arguments) inside the guest when the guest first boots up (as root, late in the boot process).

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

Please take a look at "FIRST BOOT SCRIPTS" for more information and caveats about the first boot scripts.

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''.

−−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''.

See also *−−update*, *−−uninstall*.

−−link TARGET:LINK[:LINK..]
Create symbolic link(s) in the guest, starting at `LINK` and pointing at `TARGET`.

−−mkdir DIR
Create a directory in the guest.

This uses `mkdir−p` so any intermediate directories are created, and it also works if the directory already exists.

−−move SOURCE:DEST
Move files or directories inside the guest.

Wildcards cannot be used.

−−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''.

−−password USER:SELECTOR
Set the password for `USER`. (Note this option does*not* create the user account).

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

−−password−crypto md5|sha256|sha512
When the virt tools change or set a password in the guest, this option sets the password encryption of that password 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.

Note this does not change the default password encryption used by the guest when you create new user accounts inside the guest. If you want to do that, then you should use the *−−edit* option to modify `/etc/sysconfig/authconfig` (Fedora, RHEL) or `/etc/pam.d/common−password` (Debian, Ubuntu).

−−root−password SELECTOR
Set the root password.

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

Note: In virt-builder, if you *don't* set *−−root−password* then the guest is given a *random* root password.

−−run SCRIPT
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*).

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

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

−−run−command 'CMD+ARGS'
Run the command and arguments on the disk image. The command runs virtualized inside a small appliance, chrooted into the guest filesystem.

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*).

You can have multiple *−−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.

−−selinux−relabel
Relabel files in the guest so that they have the correct SELinux label.

This will attempt to relabel files immediately, but if the operation fails this will instead touch */.autorelabel* on the image to schedule a relabel operation for the next time the image boots.

You should only use this option for guests which support SELinux.

−−sm−attach SELECTOR
Attach to a pool using `subscription−manager`.

See "SUBSCRIPTION-MANAGER" for the format of the `SELECTOR` field.

−−sm−credentials SELECTOR
Set the credentials for `subscription−manager`.

See "SUBSCRIPTION-MANAGER" for the format of the `SELECTOR` field.

−−sm−register
Register the guest using `subscription−manager`.

This requires credentials being set using *−−sm−credentials*.

−−sm−remove
Remove all the subscriptions from the guest using `subscription−manager`.

−−sm−unregister
Unregister the guest using `subscription−manager`.

−−ssh−inject USER[:SELECTOR]
Inject an ssh key so the given `USER` will be able to log in over ssh without supplying a password. The `USER` must exist already in the guest.

See "SSH KEYS" for the format of the `SELECTOR` field.

You can have multiple *−−ssh−inject* options, for different users and also for more keys for each user.

−−timezone TIMEZONE
Set the default timezone of the guest to `TIMEZONE`. Use a location string like `Europe/London`

−−touch FILE
This command performs a **touch** (1)−like operation on `FILE`.

−−truncate FILE
This command truncates `FILE` to a zero-length file. The file must exist already.

−−truncate−recursive PATH
This command recursively truncates all files under `PATH` to zero-length.

−−uninstall PKG,PKG..
Uninstall the named packages (a comma-separated list). These are removed during the image build using the guest's package manager (eg. apt, yum, etc.). Dependent packages may also need to be uninstalled to satisfy the request.

See also *−−install*, *−−update*.

−−update
Do the equivalent of `yum update`, `apt−get upgrade`, or whatever command is required to update the packages already installed in the template to their latest versions.

See also *−−install*, *−−uninstall*.

−−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*.

−−write FILE:CONTENT
Write `CONTENT` to `FILE`.

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−27 −−install inkscape
```

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

Updating packages at build time

To update the installed packages in the template at build time:

```
virt-builder fedora-27 --update
```

Most of the templates that ship with virt-builder come with a very minimal selection of packages (known as a "JEOS" or "Just Enough Operating System"), which are up to date at the time the template is created, but could be out of date by the time you come to install an OS from the template. This option updates those template packages.

Installing packages at first boot

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

```
virt-builder fedora-27 --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 dnf, 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

dnf -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-27 \
  --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).

−−root−password random
Choose a random password, which is printed on stdout. The password has approximately 120 bits of randomness.

This is the default.

−−root−password disabled
The root account password is disabled. This is like putting `*` in the password field.

−−root−password locked:file:FILENAME
−−root−password locked:password:PASSWORD
−−root−password locked:random
The root account is locked, but a password is placed on the account. If first unlocked (using `passwd -u`) then logins will use the given password.

−−root−password locked
−−root−password locked:disabled
The root account is locked *and* password is disabled.

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.

KEYBOARD LAYOUT

Because there are so many different ways to set the keyboard layout in Linux distributions, virt-builder does not yet attempt to have a simple command line option. This section describes how to set the keyboard for some common Linux distributions.

Keyboard layout with systemd

For distros that use systemd `localectl`, use a command like this:

```
virt-builder fedora-27 \
  --firstboot-command 'localectl set-keymap uk'
```

See **localectl**(1) and https://www.happyassassin.net/2013/11/23/keyboard−layouts−in−fedora−20−and−previously/ for more details.

Keyboard layout using /etc/sysconfig/keyboard

For RHEL ≤ 6, Fedora ≤ 18 and similar, upload or modify the keyboard configuration file using the −−*upload*, −−*write* or −−*edit* options. For example:

```
virt-builder centos-6 \
  --edit '/etc/sysconfig/keyboard: s/^KEYTABLE=.*/KEYTABLE="uk"/'
```

The format of this file can be found documented in many places online.

Keyboard layout with Debian-derived distros

For Debian-derived distros using */etc/default/keyboard*, upload or modify the keyboard file using the −−*upload*, −−*write* or −−*edit* options. For example:

```
virt-builder debian-8 \
  --edit '/etc/default/keyboard: s/^XKBLAYOUT=.*/XKBLAYOUT="gb"/'
```

See https://wiki.debian.org/Keyboard.

LANGUAGE

Most Linux distributions support multiple locale settings so that you can have guest messages printed in another language such as Russian.

However there is no single setting which controls this, since extra packages may need to be installed to support console and X fonts, and keyboard input methods. The packages required, and their configuration is highly distro-specific, and it is outside the scope of virt-builder to do this.

This section contains examples for some common Linux distributions.

Setting Japanese in Fedora 25

```
virt-builder fedora-27 \
  --size 20G \
  --update \
  --install @japanese-support \
  --install @xfce \
  --install xorg-x11-server-Xorg,xorg-x11-drivers,rsyslog \
  --link /usr/lib/systemd/system/graphical.target:/etc/systemd/system/default.ta
  --firstboot-command 'localectl set-locale LANG=ja_JP.utf8' \
  --firstboot-command 'localectl set-keymap jp' \
  --firstboot-command 'systemctl isolate graphical.target'
```

Setting Japanese in Debian 8 (Jessie)

Note that although this enables Japanese in the text console too, it is unlikely that you will see properly rendered Japanese there. However Japanese is properly rendered in X applications and terminals.

```
pkgs=locales,xfce4,\
ibus,ibus-anthy,\
fonts-ipafont-gothic,fonts-ipafont-mincho,\
fonts-takao-mincho,\
xfonts-intl-japanese,xfonts-intl-japanese-big,\
iceweasel-l10n-ja,manpages-ja

virt-builder debian-8 \
  --size 20G \
  --install $pkgs \
  --edit '/etc/locale.gen: s,^#\s*ja,ja,' \
  --write '/etc/default/locale:LANG="ja_JP.UTF-8"' \
  --run-command "locale-gen"
```

LOG FILE

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

/tmp/builder.log
: On Linux, BSD, and other non-Windows 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.

SSH KEYS

The *−−ssh−inject* option is used to inject ssh keys for users in the guest, so they can login without supplying a password.

The `SELECTOR` part of the option value is optional; in this case, *−−ssh−inject* `USER` means that we look in the *current* user's *~/.ssh* directory to find the default public ID file. That key is uploaded. "default public ID" is the *default_ID_file* file described in **ssh−copy−id** (1).

If specified, the `SELECTOR` can be in one of the following formats:

−−ssh−inject USER:file:FILENAME
: Read the ssh key from *FILENAME*. *FILENAME* is usually a *.pub* file.

−−ssh−inject USER:string:KEY_STRING
: Use the specified `KEY_STRING`. `KEY_STRING` is usually a public string like *ssh-rsa AAAA.... user@localhost*.

In any case, the *~USER/.ssh* directory and the *~USER/.ssh/authorized_keys* file will be created if not existing already.

FIRST BOOT SCRIPTS

The *−−firstboot* and *−−firstboot−command* options allow you to execute commands at the first boot of the guest. To do so, an init script for the guest init system is installed, which takes care of running all the added scripts and commands.

Supported operating systems are:

Linux
: Init systems supported are: systemd, System-V init (known also as sysvinit), and Upstart (using the System-V scripts).

  Note that usually init scripts run as root, but with a more limited environment than what could be available from a normal shell: for example, `$HOME` may be unset or empty.

  The output of the first boot scripts is available in the guest as *~root/virt−sysprep−firstboot.log*.

Windows
: *rhsrvany.exe*, available from sources at https://github.com/rwmjones/rhsrvany, or *pvvxsvc.exe*, available with SUSE VMDP is installed to run the first boot scripts. It is required, and the setup of first boot scripts will fail if it is not present.

  rhsrvany.exe or *pvvxsvc.exe* is copied from the location pointed to by the `VIRT_TOOLS_DATA_DIR` environment variable; if not set, a compiled-in default will be used (something like */usr/share/virt−tools*).

  The output of the first boot scripts is available in the guest as *C:\Program Files\Guestfs\Firstboot\log.txt*.

SUBSCRIPTION-MANAGER

It is possible to automate the registration and attaching of the system using `subscription−manager`. This is typical on Red Hat Enterprise Linux guests. There are few options which ease this process, avoid executing commands manually and exposing passwords on command line.

−−sm−register starts the registration process, and requires *−−sm−credentials* to be specified; the format of the `SELECTOR` of *−−sm−credentials* is one of the following formats:

−−sm−credentials USER:file:FILENAME
: Read the password for the specified `USER` from *FILENAME*.

−−sm−credentials USER:password:PASSWORD
: Use the literal string `PASSWORD` for the specified `USER`.

−−sm−attach attaches the system to subscriptions; the format of its `SELECTOR` is one of the following:

−−sm−attach auto
: `subscription−manager` attaches to the best-fitting subscriptions for the system.

−−sm−attach file:FILENAME
: Read the pool ID from *FILENAME*.

−−sm−attach pool:POOL
: Use the literal string `POOL` as pool ID.

−−sm−remove removes all the subscriptions from the guest, while *−−sm−unregister* completely unregister the system.

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.
- Guest customization is performed, in the order specified on the command line.
- SELinux relabelling is done (*−−selinux−relabel*).

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 −−os−variant fedora27
```

Notes:

1. You *must* specify the correct format. The format is `raw` unless you used virt−builder's *−−format* option.
2. *−−os−variant* is highly recommended, because it will present optimum devices to enable the guest to run most efficiently. To get a list of all variants, do:

   ```
    osinfo−query os
   ```

   The above tool is provided by libosinfo package.
3. 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−27−image −−file fedora−27.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−b uilder'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−27−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−27−server −−image fedora−27−image \
   −−flavor m1.medium
```

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

Booting directly using qemu or KVM

The qemu command line is not very stable or easy to use, hence libvirt should be used if possible. However a command line similar to the following could be used to boot the virtual machine:

```
 qemu−system−x86_64 \
   −machine accel=kvm:tcg \
   −cpu host \
   −m 2048 \
   −drive file=disk.img,format=raw,if=virtio
```

As with libvirt, it is very important that the correct format is chosen. It will be `raw` unless the *−−format* option was used.

CONFIGURATION MANAGEMENT

Puppet

To enable the Puppet agent in a guest, install the package, point the configuration at your Puppetmaster, and ensure the agent runs at boot.

A typical virt-builder command would be:

```
 virt−builder fedora−27 \
   −−hostname client.example.com \
   −−update \
   −−install puppet \
   −−append−line '/etc/puppet/puppet.conf:[agent]' \
   −−append−line '/etc/puppet/puppet.conf:server = puppetmaster.example.com/' \
   −−run−command 'systemctl enable puppet' \
   −−selinux−relabel
```

The precise instructions vary according to the Linux distro. For further information see: https://docs.puppet.com/puppet/latest/install_pre.html

DEBUGGING BUILDS

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

If virt-builder fails because some script or package it is installing fails, try using *−−no−delete−on−failure* to preserve the output file, and continue reading this section.

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
```

SOURCES OF TEMPLATES

virt-builder reads the available sources from configuration files, with the *.conf* extension and located in the following paths:

- `$XDG_CONFIG_HOME`/virt−builder/repos.d/ (`$XDG_CONFIG_HOME` is *$HOME/.config* if not set).
- `$VIRT_BUILDER_DIRS`/virt−builder/repos.d/ (where `$VIRT_BUILDER_DIRS` means any of the directories in that environment variable, or just */etc* if not set).

Each *.conf* file in those paths has a simple text format like the following:

```
[libguestfs.org]
uri=http://libguestfs.org/download/builder/index.asc
gpgkey=file:///etc/xdg/virt-builder/repos.d/libguestfs.gpg
```

The part in square brackets is the repository identifier, which is used as unique identifier.

The following fields can appear:

`uri=URI`

The URI of the index file which this repository refers to.

This field is required.

`gpgkey=URI`

This optional field represents the URI (although only *file://* URIs are accepted) of the key used to sign the index file. If not present, the index file referred by *uri*=.. is not signed.

`proxy=MODE`

This optional field specifies the proxy mode, to be used when downloading the index file of this repository. The possible values are:

no, **off**

No proxy is being used at all, even overriding the system configuration.

system

The proxy used is the system one.

anything else

Specifies the actual proxy configuration to be used, overriding the system configuration.

If not present, the assumed value is to respect the proxy settings of the system (i.e. as if **system** would be specified).

`format=FORMAT`
This optional field specifies the format of the repository. The possible values are:

native
The native format of the `virt-builder` repository. See also "Creating and signing the index file" below.

simplestreams
The URI represents the root of a Simple Streams v1.0 tree of metadata.

For more information about Simple Streams, see also https://launchpad.net/simplestreams.

If not present, the assumed value is `native`.

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/templates`.

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 creating a *.conf* file pointing to it.

Note that if your index is signed, you will need to properly fill *gpgkey=..* in your *.conf* file, making sure to deploy also the GPG key file.

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

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:

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

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/templates`

Virt-builder supports any image format (e.g. raw, qcow2, etc) as template, both as-is, and compressed as XZ. This way, existing images (e.g. cleaned using **virt-sysprep**(1)) can be used as templates.

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
arch=x86_64
file=fedora−18.xz
checksum[sha512]=...
format=raw
size=6442450944
compressed_size=148947524
expand=/dev/sda3

[fedora−19]
name=Fedora® 19
osinfo=fedora19
arch=x86_64
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.

The index file creation and signature can be eased with the **virt−builder−repository** (1) tool.

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 *uri*=.. 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).

`arch=ARCH`
: The architecture of the operating system installed within the template. This field is required.

`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 file specified in *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; in case it is compressed, that is the format before the compression. 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 actual size of the disk image in bytes, i.e. what was specified in *file*=... This is just used for information (when using `long`, and `json` formats of −−*list*).

`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.

`aliases=ALIAS1 ALIAS2 ...`
This optional field specifies a list of aliases, separated by spaces, for the image. For example, an alias could be used to always point to the latest version of a certain image, leaving the old versions available in the index instead of updating the same image (see the `revision` field).

Running virt-builder against multiple sources

It is possible to use multiple sources with virt-builder. The recommended way is to deploy *.conf* files pointing to the index files. Another way is to specify the sources using multiple −−*source* and/or −−*fingerprint* options:

```
virt-builder \
  --source http://example.com/s1/index.asc \
  --source http://example.com/s2/index.asc
```

You can provide N or 1 fingerprints. In the case where you provide N fingerprints, N = number of sources and there is a 1−1 correspondence between each source and each fingerprint:

```
virt-builder \
  --source http://example.com/s1/index.asc --fingerprint '0123 ...' \
  --source http://example.com/s2/index.asc --fingerprint '9876 ...'
```

In the case where you provide 1 fingerprint, the same fingerprint is used for all sources.

You `must` provide at least 1 fingerprint.

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).

Formal specification of the index file

The index file format has a formal specification defined by the flex scanner and bison parser used to parse the file. This can be found in the following files in the libguestfs source tree:

```
builder/index-scan.l
builder/index-parse.y
```

A tool called **virt−index−validate** (1) is available to validate the index file to ensure it is correct.

Note that the parser and tool can work on either the signed or unsigned index file (ie. *index* or *index.asc*).

The index is always encoded in UTF−8.

CACHING

Caching templates

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.

Caching packages

Virt-builder uses **curl**(1) to download files and it also uses the current `http_proxy` (etc) settings when installing packages (*--install*, *--update*).

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).

Local mirrors

To increase both speed and reliability of installing packages, you can set up a local mirror of the target distribution, and point the guest package manager at that.

Using a local mirror with Fedora

To install a Fedora guest using a local mirror:

```
virt-builder fedora-27 \
  --edit '/etc/yum.repos.d/fedora.repo:
      s{.*baseurl=.*}{baseurl=http://example.com/mirror/};
      s{.*metalink=.*}{};
  ' \
  --edit '/etc/yum.repos.d/fedora-updates.repo:
      s{.*baseurl=.*}{baseurl=http://example.com/mirror-updates/};
      s{.*metalink=.*}{};
  ' \
  --run-command 'dnf -y update' \
  --install 'pkg1,pkg2,...'
```

Using a local mirror with Debian

Assuming that you are using `apt-proxy` to mirror the repository, you should create a new *sources.list* file to point to your proxy (see https://help.ubuntu.com/community/AptProxy) and then do:

```
virt-builder debian-8 \
  --upload sources.list:/etc/apt/sources.list \
  --run-command 'apt-get -y update' \
  --install 'pkg1,pkg2,...'
```

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. the one specified in *gpgkey=..* in the *.conf*, or with *--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, specifically options that require running commands in the guest during the build process: *−−install*, *−−update*, *−−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, *−−update*, *−−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.

CLONES

If you wish to create many new guests of the same type, it is tempting to run virt-builder once and then copy the output file. You should **not** do this. You should run virt-builder once for each new guest you need.

The reason is that each clone needs to have (at least) a separate random seed, and possibly other unique features (such as filesystem UUIDs) in future versions of virt-builder.

Another thing you should *not* do is to boot the guest, then clone the booted disk image. The reason is that some guests create unique machine IDs, SSH host keys and so on at first boot, and you would not want clones to have duplicate identities.

See also: **virt−sysprep** (1).

PERFORMANCE

The most important aspect of getting good performance is caching. Templates gets downloaded into the cache the first time they are used, or if you use the *−−cache−all−templates* option. See "CACHING" above for further information.

Packages required for the *−−install* and *−−update* options are downloaded using the host network connection. Setting the`http_proxy`, `https_proxy` and `ftp_proxy` en vironment variables to point to a local web cache may ensure they only need to be downloaded once. You can also try using a local package repository, although this can be complex to set up and varies according to which Linux distro you are trying to install.

Using −−no−sync

Use *−−no−sync*. However read the caveats in the "OPTIONS" section above, since this can cause disk corruption if not used correctly.

Skipping virt-resize

Virt-builder can skip the virt-resize step under certain conditions. This makes virt-builder much faster. The conditions are:

- the output must be a regular file (not a block device), **and**
- the user did **not** use the *−−size* option, **and**
- the output format is the same as the template format (usually raw).

pxzcat

Virt-builder uses an internal implementation of pxzcat (parallel xzcat) if liblzma was found at build time. If liblzma was not found at build time, regular `xzcat` is used which is single-threaded.

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.

Virt-builder does not know how to give new files a label, so there are two possible strategies it can use to ensure correct labelling:

Using *−−selinux−relabel*

This runs **setfiles** (8) just before finalizing the guest, which sets SELinux labels correctly in the disk image.

This is the recommended method.

−−touch /.autorelabel

Guest templates may already contain a file called */.autorelabel* or you may touch it.

For guests that use SELinux, this causes **restorecon** (8) to run at first boot. Guests will reboot themselves once the first time you use them, which is normal and harmless.

Please note that if your guest uses SELinux, and you are doing operations on it which might create new files or change existing ones, you are recommended to use *−−selinux−relabel*. This will help in making sure that files have the right SELinux labels.

MACHINE READABLE OUTPUT

The *−−machine−readable* option can be used to make the output more machine friendly, which is useful when calling virt-builder from other programs, GUIs etc.

Use the option on its own to query the capabilities of the virt-builder binary. Typical output looks like this:

```
$ virt-builder --machine-readable
virt-builder
arch
config-file
customize
json-list
pxzcat
```

A list of features is printed, one per line, and the program exits with status 0.

It is possible to specify a format string for controlling the output; see ''ADVANCED MACHINE READABLE OUTPUT'' in **guestfs** (3).

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, and the location of the user' sources. See "CACHING" and "SOURCES OF TEMPLATES".

`VIRT_TOOLS_DATA_DIR`
: This can point to the directory containing data files used for Windows firstboot installation.

Normally you do not need to set this. If not set, a compiled-in default will be used (something like */usr/share/virt−tools*).

This directory may contain the following files:

rhsrvany.exe
: This is the RHSrvAny Windows binary, used to install a "firstboot" script in Windows guests. It is required if you intend to use the *−−firstboot* or *−−firstboot−command* options with Windows guests.

See also: `https://github.com/rwmjones/rhsrvany`

pvvxsvc.exe
: This is a Windows binary shipped with SUSE VMDP, used to install a "firstboot" script in Windows guests. It is required if you intend to use the *−−firstboot* or *−−firstboot−command* options with Windows guests.

`XDG_CACHE_HOME`
: Used to determine the location of the template cache. See "CACHING".

`XDG_CONFIG_HOME`
: Used to determine the location of the user' sources. See "SOURCES OF TEMPLATES".

`VIRT_BUILDER_DIRS`
: Used to determine the location of the system sources. See "SOURCES OF TEMPLATES".

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−builder−repository** (1), **virt−copy−out** (1), **virt−customize** (1), **virt−get−kernel** (1), **virt−install** (1), **virt−rescue** (1), **virt−resize** (1), **virt−sysprep** (1), **oz−install** (1), **gpg** (1), **gpg2** (1), **curl** (1), **virt−make−fs** (1), **genisoimage** (1), http://libguestfs.org/.

AUTHOR

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

COPYRIGHT

LICENSE

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.