
curtin Documentation

Release 17.1

Scott Moser

Dec 15, 2017

Contents

1	Overview	3
1.1	Stages	3
2	Curtin Configuration	7
2.1	Configuration options	7
3	APT Source	17
3.1	Features	17
3.2	Configuration	18
3.3	Common snippets	19
3.4	Timing	20
3.5	Dependencies	20
4	Networking	21
4.1	Configuration Types	21
4.2	Multi-layered configurations	28
4.3	More Examples	29
5	Storage	31
5.1	Configuration Types	31
5.2	Additional Examples	40
6	Curthooks / New OS Support	47
6.1	Image provided curtin-hooks	47
6.2	Networking configuration	48
6.3	Storage configuration	48
6.4	System boot	48
6.5	finalize hook	48
7	Reporting	49
7.1	Events	49
7.2	Configuration	50
7.3	Webhook Reporter	50
7.4	Journald Reporter	50
7.5	Development / Debug Reporting	52
7.6	Legacy Reporter	52

8	Developing Curtin	53
8.1	Install dependencies	53
8.2	Download cloud images	53
8.3	Getting the source	54
8.4	Using curtin	54
8.5	Notes about ‘launch’	54
9	Integration Testing	55
9.1	Background	55
9.2	Debugging	56
9.3	Setup	56
9.4	Running	57
9.5	Environment Variables	57
9.6	Environment ‘boolean’ values	59
9.7	Test Class Variables	60
10	Indices and tables	61

This is ‘curtin’, the curt installer. It is blunt, brief, snappish, snippety and unceremonious. Its goal is to install an operating system as quick as possible.

Contents:

Curtin is intended to be a bare bones “installer”. Its goal is to take data from a source, and get it onto disk as quick as possible and then boot it. The key difference from traditional package based installers is that curtin assumes the thing its installing is intelligent and will do the right thing.

1.1 Stages

A usage of curtin will go through the following stages:

- Install Environment boot
- Early Commands
- Partitioning
- Network Discovery and Setup
- Extraction of sources
- Hook for installed OS to customize itself
- Final Commands

1.1.1 Install Environment boot

At the moment, curtin doesn’t address how the system that it is running on is booted. It could be booted from a live-cd or from a pxe boot environment. It could even be booted off a disk in the system (although installation to that disk would probably break things).

Curtin’s assumption is that a fairly rich Linux (Ubuntu) environment is booted.

1.1.2 Command Environment

Stages and commands invoked by curtin always have the following environment variables defined.

- `WORKING_DIR`: This is for inter-command state. It will be the same directory for each command run and will only be deleted at the end of the install. Files referenced in other environment variables will be in this directory.
- `TARGET_MOUNT_POINT`: The path in the filesystem where the target filesystem will be mounted.
- `OUTPUT_NETWORK_CONFIG`: After the network discovery stage, this file should contain networking config information that should then be written to the target.
- `OUTPUT_FSTAB`: After partitioning and filesystem creation, this file will contain fstab(5) style content representing mounts.
- `CONFIG`: This variable contains a path to a yaml formatted file with the fully rendered config.

1.1.3 Early Commands

Early commands are executed on the system, and non-zero exit status will terminate the installation process. These commands are intended to be used for things like

- module loading
- hardware setup
- environment setup for subsequent stages of curtin.

Config Example:

```
early_commands:
  05_load_loop: [modprobe, loop]
  99_update: apt-get update && apt-get dist-upgrade
```

1.1.4 Partitioning

Partitioning covers setting up filesystems on the system. A series of commands are run serially in order. At the end, a fstab formatted file must be populated in `OUTPUT_FSTAB` that contains mount information, and the filesystems are expected to be mounted at the `TARGET_MOUNT_POINT`.

Any commands can be used to create this filesystem, but curtin contains some tools to facilitate with this process.

Config Example:

```
partitioning_commands:
  10_wipe_filesystems: curtin wipe --quick --all-unused-disks
  50_setup_raid: curtin disk-setup --all-disks raid0 /
```

1.1.5 Network Discovery

Networking configuration is *discovered* in the ‘network’ stage. The default command run at this stage is `curtin net-meta auto`. After execution, it will write the discovered networking to the file specified in the environment variable `OUTPUT_NETWORK_CONFIG`. The format of this file is as described in [Networking](#).

If curtin’s config has a network section, the net-meta will simply parrot the data to the output file. If there is no network section, then its default behavior is to copy existing config from the running environment.

Note, that as with fstab, this file is not copied verbatim to the target filesystem, but rather made available to the OS customization stage. That stage may just copy the file verbatim, but may also parse it, and apply the settings.

1.1.6 Extraction of sources

Sources are the things to install. Curtin prefers to install root filesystem tar files.

Config Example:

```
sources:  
  05_primary: http://cloud-images.ubuntu.com/releases/precise/release/ubuntu-12.04-  
    ↪server-cloudimg-amd64-root.tar.gz
```

Given the source above, curtin will essentially do a:

```
wget $URL | tar -Sxvzf
```

1.1.7 Final Commands

Config Example:

```
final_commands:  
  05_callhome_finished: wget http://example.com/i-am-done
```


Curtin exposes a number of configuration options for controlling Curtin behavior during installation.

2.1 Configuration options

Curtin's top level config keys are as follows:

- `apt_mirrors` (`apt_mirrors`)
- `apt_proxy` (`apt_proxy`)
- `block-meta` (`block`)
- `debconf_selections` (`debconf_selections`)
- `disable_overlayroot` (`disable_overlayroot`)
- `grub` (`grub`)
- `http_proxy` (`http_proxy`)
- `install` (`install`)
- `kernel` (`kernel`)
- `kexec` (`kexec`)
- `multipath` (`multipath`)
- `network` (`network`)
- `power_state` (`power_state`)
- `proxy` (`proxy`)
- `reporting` (`reporting`)
- `restore_dist_interfaces` (`restore_dist_interfaces`)
- `sources` (`sources`)

- `stages` (`stages`)
- `storage` (`storage`)
- `swap` (`swap`)
- `system_upgrade` (`system_upgrade`)
- `write_files` (`write_files`)

2.1.1 apt_mirrors

Configure APT mirrors for `ubuntu_archive` and `ubuntu_security`

ubuntu_archive: `<http://local.archive/ubuntu>`

ubuntu_security: `<http://local.archive/ubuntu>`

If the target OS includes `/etc/apt/sources.list`, Curtin will replace the default values for each key set with the supplied mirror URL.

Example:

```
apt_mirrors:
  ubuntu_archive: http://local.archive/ubuntu
  ubuntu_security: http://local.archive/ubuntu
```

2.1.2 apt_proxy

Curtin will configure an APT HTTP proxy in the target OS

apt_proxy: `<URL to APT proxy>`

Example:

```
apt_proxy: http://squid.mirror:3267/
```

2.1.3 block-meta

Configure how Curtin selects and configures disks on the target system without providing a custom configuration (`mode=simple`).

devices: `<List of block devices for use>`

The `devices` parameter is a list of block device paths that Curtin may select from with choosing where to install the OS.

boot-partition: `<dictionary of configuration>`

The `boot-partition` parameter controls how to configure the boot partition with the following parameters:

enabled: `<boolean>`

Enabled will forcibly setup a partition on the target device for booting.

format: `<['uefi', 'gpt', 'prep', 'mbr']>`

Specify the partition format. Some formats, like `uefi` and `prep` are restricted by platform characteristics.

fstype: `<filesystem type: one of ['ext3', 'ext4'], defaults to 'ext4'>`

Specify the filesystem format on the boot partition.

label: <filesystem label: defaults to 'boot'>

Specify the filesystem label on the boot partition.

Example:

```
block-meta:
  devices:
    - /dev/sda
    - /dev/sdb
  boot-partition:
    - enabled: True
      format: gpt
      fstype: ext4
      label: my-boot-partition
```

2.1.4 debconf_selections

Curtin will update the target with debconf set-selection values. Users will need to be familiar with the package debconf options. Users can probe a packages' debconf settings by using `debconf-get-selections`.

selection_name: <debconf-set-selections input>

`debconf-set-selections` is in the form:

```
<packagename> <packagename/option-name> <type> <value>
```

Example:

```
debconf_selections:
  set1: |
    cloud-init cloud-init/datasources multiselect MAAS
    lxd lxd/bridge-name string lxdbr0
  set2: lxd lxd/setup-bridge boolean true
```

2.1.5 disable_overlayroot

Curtin disables overlayroot in the target by default.

disable_overlayroot: <boolean: default True>

Example:

```
disable_overlayroot: False
```

2.1.6 grub

Curtin configures grub as the target machine's boot loader. Users can control a few options to tailor how the system will boot after installation.

install_devices: <list of block device names to install grub>

Specify a list of devices onto which grub will attempt to install.

replace_linux_default: <boolean: default True>

Controls whether grub-install will update the Linux Default target value during installation.

update_nvram: *<boolean: default False>*

Certain platforms, like `uefi` and `prep` systems utilize NVRAM to hold boot configuration settings which control the order in which devices are booted. Curtin by default will not attempt to update the NVRAM settings to preserve the system configuration. Users may want to force NVRAM to be updated such that the next boot of the system will boot from the installed device.

Example:

```
grub:
  install_devices:
    - /dev/sda1
  replace_linux_default: False
  update_nvram: True
```

2.1.7 http_proxy

Curtin will export `http_proxy` value into the installer environment. **Deprecated:** This setting is deprecated in favor of `proxy` below.

http_proxy: *<HTTP Proxy URL>*

Example:

```
http_proxy: http://squid.proxy:3728/
```

2.1.8 install

Configure Curtin's install options.

log_file: *<path to write Curtin's install.log data>*

Curtin logs install progress by default to `/var/log/curtin/install.log`

post_files: *<List of files to read from host to include in reporting data>*

Curtin by default will post the `log_file` value to any configured reporter.

save_install_config: *<Path to save merged curtin configuration file>*

Curtin will save the merged configuration data into the target OS at the path of `save_install_config`. This defaults to `/root/curtin-install-cfg.yaml`

save_install_logs: *<Path to save curtin install log>*

Curtin will copy the install log to a specific path in the target filesystem. This defaults to `/root/install.log`

target: *<path to mount install target>*

Control where curtin mounts the target device for installing the OS. If this value is unset, curtin picks a suitable path under a temporary directory. If a value is set, then curtin will utilize the `target` value instead.

unmount: *disabled*

If this key is set to the string 'disabled' then curtin will not unmount the target filesystem when install is complete. This skips unmounting in all cases of install success or failure.

Example:

```
install:
  log_file: /tmp/install.log
  post_files:
    - /tmp/install.log
    - /var/log/syslog
  save_install_config: /root/myconf.yaml
  save_install_log: /var/log/curtin-install.log
  target: /my_mount_point
  unmount: disabled
```

2.1.9 kernel

Configure how Curtin selects which kernel to install into the target image. If `kernel` is not configured, Curtin will use the default mapping below and determine which package value by looking up the current release and current kernel version running.

fallback-package: *<kernel package-name to be used as fallback>*

Specify a kernel package name to be used if the default package is not available.

mapping: *<Dictionary mapping Ubuntu release to HWE kernel names>*

Default mapping for Releases to package names is as follows:

```
precise:
  3.2.0:
  3.5.0: -lts-quantal
  3.8.0: -lts-raring
  3.11.0: -lts-saucy
  3.13.0: -lts-trusty
trusty:
  3.13.0:
  3.16.0: -lts-utopic
  3.19.0: -lts-vivid
  4.2.0: -lts-wily
  4.4.0: -lts-xenial
xenial:
  4.3.0:
  4.4.0:
```

package: *<Linux kernel package name>*

Specify the exact package to install in the target OS.

Example:

```
kernel:
  fallback-package: linux-image-generic
  package: linux-image-generic-lts-xenial
  mapping:
    - xenial:
      - 4.4.0: -my-custom-kernel
```

2.1.10 kexec

Curtin can use kexec to “reboot” into the target OS.

mode: *<on>*

Enable rebooting with kexec.

Example:

```
kexec: on
```

2.1.11 multipath

Curtin will detect and autoconfigure multipath by default to enable boot for systems with multipath. Curtin does not apply any advanced configuration or tuning, rather it uses distro defaults and provides enough configuration to enable booting.

mode: *<['auto', ['disabled']]>*

Defaults to auto which will configure enough to enable booting on multipath devices. Disabled will prevent curtin from installing or configuring multipath.

overwrite_bindings: *<boolean>*

If `overwrite_bindings` is `True` then Curtin will generate new bindings file for multipath, overriding any existing binding in the target image.

Example:

```
multipath:
  mode: auto
  overwrite_bindings: True
```

2.1.12 network

Configure networking (see Networking section for details).

network_option_1: *<option value>*

Example:

```
network:
  version: 1
  config:
    - type: physical
      name: eth0
      mac_address: "c0:d6:9f:2c:e8:80"
      subnets:
        - type: dhcp4
```

2.1.13 power_state

Curtin can configure the target machine into a specific power state after completing an installation. Default is to do nothing.

delay: *<Integer seconds to delay change in state>*

Curtin will wait `delay` seconds before changing the power state.

mode: *<New power state is one of: [halt, poweroff, reboot]>*

Curtin will transition the node into one of the new states listed.

`halt` will stop a machine, but may not cut the power to the system. `poweroff` will stop a machine and request it shut off the power. `reboot` will perform a platform reset.

message: *<message string>*

The message string will be broadcast to system consoles prior to power state change.

Example:

```
power_state:
  mode: poweroff
  delay: 5
  message: Bye Bye
```

2.1.14 proxy

Curtin will put `http_proxy`, `https_proxy` and `no_proxy` into its install environment. This is in affect for curtin's process and subprocesses.

proxy: A dictionary containing `http_proxy`, `https_proxy`, and `no_proxy`.

Example:

```
proxy:
  http_proxy: http://squid.proxy:3728/
  https_proxy: http://squid.proxy:3728/
  no_proxy: localhost,127.0.0.1,10.0.2.1
```

2.1.15 reporting

Configure installation reporting (see Reporting section for details).

Example:

```
reporting:
  maas:
    level: DEBUG
    type: webhook
    endpoint: http://localhost:8000/
```

2.1.16 restore_dist_interfaces

Curtin can restore a copy of `/etc/network/interfaces` built in to cloud images.

restore_dist_interfaces: *<boolean>*

If True, then Curtin will restore the interfaces file into the target.

Example:

```
restore_dist_interfaces: True
```

2.1.17 sources

Specify the root image to install on to the target system. The URI also configures the method used to copy the image to the target system.

sources: <List of source URIs>

source URI may be one of:

- **dd-:** Use `dd` command to write image to target.
- **cp://:** Use `rsync` command to copy source directory to target.
- **file://:** Use `tar` command to extract source to target.
- **http[s]://:** Use `wget` | `tar` commands to extract source to target.

Example Cloud-image:

```
sources:
- https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-root.
↪tar.gz
```

Example Custom DD image:

```
sources:
- dd-img: https://localhost/raw_images/centos-6-3.img
```

Example Copy from booted environment:

```
sources:
- cp:///
```

Example Copy from local tarball:

```
sources:
- file:///tmp/root.tar.gz
```

2.1.18 stages

Curtin installation executes in stages. At each stage, Curtin will look for a list of commands to run at each stage by reading in from the Curtin config <stage_name>_commands which is a dictionary and each key contains a list of commands to run. Users may override the stages value to control what curtin stages execute. During each stage, the commands are executed in C Locale sort order. Users should name keys in a NN-XXX format where NN is a two-digit number to exercise control over execution order.

The following stages are defined in Curtin and run by default.

- **early:** *Preparing for Installation*

This stage runs before any actions are taken for installation. By default this stage does nothing.

- **partitioning:** *Select and partition disks for installation*

This stage runs `curtin block-meta simple` by default.

- **network:** *Probe and configure networking*

This stage runs `curtin net-meta auto` by default.

- **extract:** *Writing install sources to disk*

This stage runs `curtin extract` by default.

- **extract:** *Writing install sources to disk*

This stage runs `curtin extract` by default.

- **curthooks:** *Configuring installed system*

This stage runs `curtin curthooks` by default.

- **hooks:** *Finalizing installation*

This stage runs `curtin hook` by default.

- **late:** *Executing late commands*

This stage runs after Curtin has completed the installation. By default this stage does nothing.

Example Custom Stages:

```
# Skip the whole install and just run `mystage`
stages: ['early', 'late', 'mystage']
mystage_commands:
  00-cmd: ['/usr/bin/foo']
```

Example Early and Late commands:

```
early_commands:
  99-cmd: ['echo', 'I ran last']
  00-cmd: ['echo', 'I ran first']
late_commands:
  50-cmd: ['curtin', 'in-target' '--', 'touch', '/etc/disable_overlayroot']
```

2.1.19 swap

Curtin can configure a swapfile on the filesystem in the target system. Size settings can be integer or string values with suffix. Curtin supports the following suffixes which multiply the value.

- **B:** 1
- **K[B]:** 1 << 10
- **M[B]:** 1 << 20
- **G[B]:** 1 << 30
- **T[B]:** 1 << 40

Curtin will use a heuristic to configure the swapfile size if the `size` parameter is not set to a specific value. The `maxsize` sets the upper bound of the heuristic calculation.

filename: <path to swap file>

Configure the filename of the swap file. Defaults to `/swap.img`

maxsize: <Size string>

Configure the max size of the swapfile, defaults to 8GB

size: <Size string>

Configure the exact size of the swapfile. Setting `size` to 0 will disable swap.

Example:

```
swap:
  filename: swap.img
  size: None
  maxsize: 4GB
```

2.1.20 system_upgrade

Control if Curtin runs *dist-upgrade* in target after install. Defaults to False.

enabled: *<boolean>*

Example:

```
system_upgrade:
  enabled: False
```

2.1.21 write_files

Curtin supports writing out arbitrary data to a file. `write_files` accepts a dictionary of entries formatted as follows:

path: *<path and filename to save content>*

Specify the name and location of where to write the content.

permissions: *<Unix permission string>*

Specify the permissions mode as an integer or string of numbers.

content: *<data>*

Specify the content.

Example:

```
write_files:
  f1:
    path: /file1
    content: !!binary |
      f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAwAAAAAAAAABAAAAAAAAAAJAVAAAAAA
  f2: {path: /file2, content: "foobar", permissions: '0666'}
```

This part of curtin is meant to allow influencing the apt behaviour and configuration.

By default - if no apt config is provided - it does nothing. That keeps behavior compatible on upgrades.

The feature has an optional target argument which - by default - is used to modify the environment that curtin currently installs (@TARGET_MOUNT_POINT).

3.1 Features

- Add PGP keys to the APT trusted keyring
 - add via short keyid
 - add via long key fingerprint
 - specify a custom keyserver to pull from
 - add raw keys (which makes you independent of keyservers)
- Influence global apt configuration
 - adding ppa's
 - replacing mirror, security mirror and release in sources.list
 - able to provide a fully custom template for sources.list
 - add arbitrary apt.conf settings
 - provide debconf configurations
 - disabling suites (=pockets)
 - per architecture mirror definition

3.2 Configuration

The general configuration of the apt feature is under an element called `apt`.

This can have various “global” subelements as listed in the examples below. The file `apt-source.yaml` holds more examples.

These global configurations are valid throughout all of the apt feature. So for example a global specification of a primary mirror will apply to all rendered sources entries.

Then there is a section `sources` which can hold any number of source subelements itself. The key is the filename and will be prepended by `/etc/apt/sources.list.d/` if it doesn't start with a `/`. There are certain cases - where no content is written into a `source.list` file where the filename will be ignored - yet it can still be used as index for merging.

The values inside the entries consist of the following optional entries

- `source`: a `sources.list` entry (some variable replacements apply)
- `keyid`: providing a key to import via shortid or fingerprint
- `key`: providing a raw PGP key
- `keyserver`: specify an alternate keyserver to pull keys from that were specified by `keyid`

The section “sources” is is a dictionary (unlike most block/net configs which are lists). This format allows merging between multiple input files than a list like

```
sources:
  s1: {'key': 'key1', 'source': 'source1'}

sources:
  s2: {'key': 'key2'}
  s1: {'keyserver': 'foo'}

This would be merged into
  s1: {'key': 'key1', 'source': 'source1', 'keyserver': 'foo'}
  s2: {'key': 'key2'}
```

Here is just one of the most common examples for this feature: install with curtin in an isolated environment (derived repository):

For that we need to: * insert the PGP key of the local repository to be trusted

- since you are locked down you can't pull from `keyserver.ubuntu.com`
- if you have an internal keyserver you could pull from there, but let us assume you don't even have that; so you have to provide the raw key
- in the example I'll use the key of the “Ubuntu CD Image Automatic Signing Key” which makes no sense as it is in the trusted keyring anyway, but it is a good example. (Also the key is shortened to stay readable)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1
mQGIBEFEnz8RBAC7LstGskD7McXZgd58oN68KquARLBl6rjA2vdhw177KkPPOr3O
RwIbDAAKCRBA126vQ30FtdxYAJsfjU+xbex7gevyGQ2/mhqidES4MwCggqQyo+w1
Twx6DKLF+3rF5nflF3Q=
=PBAe
-----END PGP PUBLIC KEY BLOCK-----
```

- replace the mirrors used to some mirrors available inside the isolated environment for apt to pull repository data from.

- lets consider we have a local mirror at `mymirror.local` but otherwise following the usual paths
- make an example with a partial mirror that doesn't mirror the backports suite, so backports have to be disabled

That would be specified as

```
apt:
  primary:
    - arches [default]
      uri: http://mymirror.local/ubuntu/
  disable_suites: [backports]
  sources:
    localrepokey:
      key: | # full key as block
            -----BEGIN PGP PUBLIC KEY BLOCK-----
            Version: GnuPG v1

            mQGiBEFEnz8RBAC7LstGsKD7McXZgd58oN68KquARLB16rjA2vdhw177KkPPOr3O
            RwIbDAAKCRBA126vQ30FtdxYAJsfjU+xbex7gevyGQ2/mhqidES4MwCggqQyo+w1
            Twx6DKLF+3rF5nf1F3Q=
            =PB Ae
            -----END PGP PUBLIC KEY BLOCK-----
```

The file `examples/apt-source.yaml` holds various further examples that can be configured with this feature.

3.3 Common snippets

This is a collection of additional ideas people can use the feature for customizing their to-be-installed system.

- enable proposed on installing

```
apt:
  sources:
    proposed.list:
      source: |
        deb $MIRROR $RELEASE-proposed main restricted universe multiverse
```

- Make debug symbols available

```
apt:
  sources:
    ddebs.list:
      source: |
        deb http://ddebs.ubuntu.com $RELEASE main restricted universe multiverse
        deb http://ddebs.ubuntu.com $RELEASE-updates main restricted universe_
↪multiverse
        deb http://ddebs.ubuntu.com $RELEASE-security main restricted universe_
↪multiverse
        deb http://ddebs.ubuntu.com $RELEASE-proposed main restricted universe_
↪multiverse
```

3.4 Timing

The feature is implemented at the stage of `curthooks_commands`, which runs just after `curtin` has extracted the image to the target. Additionally it can be ran as standalone command “`curtin -v --config <yourconfigfile> apt-config`”.

This will pick up the target from the environment variable that is set by `curtin`, if you want to use it to a different target or outside of usual `curtin` handling you can add `--target <path>` to it to overwrite the target path. This target should have at least a minimal system with `apt`, `apt-add-repository` and `dpkg` being installed for the functionality to work.

3.5 Dependencies

Cloud-init might need to resolve dependencies and install packages in the ephemeral environment to run `curtin`. Therefore it is recommended to not only provide an `apt` configuration to `curtin` for the target, but also one to the install environment via `cloud-init`.

Curtin supports a user-configurable networking configuration format. This format lets users (including via MAAS) to customize their machines' networking interfaces by assigning subnet configuration, virtual device creation (bonds, bridges, vlans) routes and DNS configuration.

Curtin accepts a YAML input under the top-level `network` key to indicate that a user would like to specify a custom networking configuration. Required elements of a network configuration are `config` and `version`. Currently curtin only supports network config version=1.

```
network:
  version: 1
  config: []
```

4.1 Configuration Types

Within the network `config` portion, users include a list of configuration types. The current list of support `type` values are as follows:

- Physical (`physical`)
- Bond (`bond`)
- Bridge (`bridge`)
- VLAN (`vlan`)
- Nameserver (`nameserver`)
- Route (`route`)

Physical, Bond, Bridge and VLAN types may also include IP configuration under the key `subnets`.

- Subnet/IP (`subnets`)

4.1.1 Physical

The `physical` type configuration represents a “physical” network device, typically Ethernet-based. At least one of these entries is required for external network connectivity. Type `physical` requires only one key: `name`. A `physical` device may contain some or all of the following keys:

name: *<desired device name>*

A device's name must be less than 15 characters. Names exceeding the maximum will be truncated. This is a limitation of the Linux kernel network-device structure.

mac_address: *<MAC Address>*

The MAC Address is a device unique identifier that most Ethernet-based network devices possess. Specifying a MAC Address is optional.

Note: Curtin will emit a udev rule to provide a persistent mapping between a device's `name` and the `mac_address`.

mtu: *<MTU SizeBytes>*

The MTU key represents a device's Maximum Transmission Unit, the largest size packet or frame, specified in octets (eight-bit bytes), that can be sent in a packet- or frame-based network. Specifying `mtu` is optional.

Note: The possible supported values of a device's MTU is not available at configuration time. It's possible to specify a value too large or too small for a device and may be ignored by the device.

Physical Example:

```
network:
  version: 1
  config:
    # Simple network adapter
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
    # Second nic with Jumbo frames
    - type: physical
      name: jumbo0
      mac_address: aa:11:22:33:44:55
      mtu: 9000
    # 10G pair
    - type: physical
      name: gbe0
      mac_address: cd:11:22:33:44:00
    - type: physical
      name: gbe1
      mac_address: cd:11:22:33:44:02
```

4.1.2 Bond

A `bond` type will configure a Linux software Bond with one or more network devices. A `bond` type requires the following keys:

name: *<desired device name>*

A device's name must be less than 15 characters. Names exceeding the maximum will be truncated. This is a limitation of the Linux kernel network-device structure.

mac_address: *<MAC Address>*

When specifying MAC Address on a bond this value will be assigned to the bond device and may be different than the MAC address of any of the underlying bond interfaces. Specifying a MAC Address is optional. If `mac_address` is not present, then the bond will use one of the MAC Address values from one of the bond interfaces.

bond_interfaces: *<List of network device names>*

The `bond_interfaces` key accepts a list of network device name values from the configuration. This list may be empty.

params: *<Dictionary of key: value bonding parameter pairs>*

The `params` key in a bond holds a dictionary of bonding parameters. This dictionary may be empty. For more details on what the various bonding parameters mean please read the `Linux Kernel Bonding.txt`.

Valid `params` keys are:

- `active_slave`: Set bond attribute
- `ad_actor_key`: Set bond attribute
- `ad_actor_sys_prio`: Set bond attribute
- `ad_actor_system`: Set bond attribute
- `ad_aggregator`: Set bond attribute
- `ad_num_ports`: Set bond attribute
- `ad_partner_key`: Set bond attribute
- `ad_partner_mac`: Set bond attribute
- `ad_select`: Set bond attribute
- `ad_user_port_key`: Set bond attribute
- `all_slaves_active`: Set bond attribute
- `arp_all_targets`: Set bond attribute
- `arp_interval`: Set bond attribute
- `arp_ip_target`: Set bond attribute
- `arp_validate`: Set bond attribute
- `downdelay`: Set bond attribute
- `fail_over_mac`: Set bond attribute
- `lacp_rate`: Set bond attribute
- `lp_interval`: Set bond attribute
- `miimon`: Set bond attribute
- `mii_status`: Set bond attribute
- `min_links`: Set bond attribute
- `mode`: Set bond attribute
- `num_grat_arp`: Set bond attribute
- `num_unsol_na`: Set bond attribute

- `packets_per_slave`: Set bond attribute
- `primary`: Set bond attribute
- `primary_reselect`: Set bond attribute
- `queue_id`: Set bond attribute
- `resend_igmp`: Set bond attribute
- `slaves`: Set bond attribute
- `tlb_dynamic_lb`: Set bond attribute
- `updelay`: Set bond attribute
- `use_carrier`: Set bond attribute
- `xmit_hash_policy`: Set bond attribute

Bond Example:

```
network:
  version: 1
  config:
    # Simple network adapter
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
    # 10G pair
    - type: physical
      name: gbe0
      mac_address: cd:11:22:33:44:00
    - type: physical
      name: gbe1
      mac_address: cd:11:22:33:44:02
    - type: bond
      name: bond0
      bond_interfaces:
        - gbe0
        - gbe1
      params:
        bond-mode: active-backup
```

4.1.3 Bridge

Type bridge requires the following keys:

- `name`: Set the name of the bridge.
- `bridge_interfaces`: Specify the ports of a bridge via their name. This list may be empty.
- `params`: A list of bridge params. For more details, please read the `bridge-utils-interfaces` manpage.

Valid keys are:

- `bridge_ageing`: Set the bridge's ageing value.
- `bridge_bridgeprio`: Set the bridge device network priority.
- `bridge_fd`: Set the bridge's forward delay.
- `bridge_hello`: Set the bridge's hello value.

- `bridge_hw`: Set the bridge's MAC address.
- `bridge_maxage`: Set the bridge's maxage value.
- `bridge_maxwait`: Set how long network scripts should wait for the bridge to be up.
- `bridge_pathcost`: Set the cost of a specific port on the bridge.
- `bridge_portprio`: Set the priority of a specific port on the bridge.
- `bridge_ports`: List of devices that are part of the bridge.
- `bridge_stp`: Set spanning tree protocol on or off.
- `bridge_waitport`: Set amount of time in seconds to wait on specific ports to become available.

Bridge Example:

```
network:
  version: 1
  config:
    # Simple network adapter
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
    # Second nic with Jumbo frames
    - type: physical
      name: jumbo0
      mac_address: aa:11:22:33:44:55
      mtu: 9000
    - type: bridge
      name: br0
      bridge_interfaces:
        - jumbo0
      params:
        bridge_ageing: 250
        bridge_bridgeprio: 22
        bridge_fd: 1
        bridge_hello: 1
        bridge_maxage: 10
        bridge_maxwait: 0
        bridge_pathcost:
          - jumbo0 75
        bridge_pathprio:
          - jumbo0 28
        bridge_stp: 'off'
        bridge_maxwait:
          - jumbo0 0
```

4.1.4 VLAN

Type `vlan` requires the following keys:

- `name`: Set the name of the VLAN
- `vlan_link`: Specify the underlying link via its name.
- `vlan_id`: Specify the VLAN numeric id.

VLAN Example:

```
network:
  version: 1
  config:
    # Physical interfaces.
    - type: physical
      name: eth0
      mac_address: "c0:d6:9f:2c:e8:80"
    # VLAN interface.
    - type: vlan
      name: eth0.101
      vlan_link: eth0
      vlan_id: 101
      mtu: 1500
```

4.1.5 Nameserver

Users can specify a nameserver type. Nameserver dictionaries include the following keys:

- `address`: List of IPv4 or IPv6 address of nameservers.
- `search`: List of hostnames to include in the `resolv.conf` search path.

Nameserver Example:

```
network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
      subnets:
        - type: static
          address: 192.168.23.14/27
          gateway: 192.168.23.1
    - type: nameserver:
      address:
        - 192.168.23.2
        - 8.8.8.8
      search:
        - exemplary
```

4.1.6 Route

Users can include static routing information as well. A route dictionary has the following keys:

- `destination`: IPv4 network address with CIDR netmask notation.
- `gateway`: IPv4 gateway address with CIDR netmask notation.
- `metric`: Integer which sets the network metric value for this route.

Route Example:

```
network:
  version: 1
  config:
    - type: physical
```

```

name: interface0
mac_address: 00:11:22:33:44:55
subnets:
  - type: static
    address: 192.168.23.14/24
    gateway: 192.168.23.1
  - type: route
    destination: 192.168.24.0/24
    gateway: 192.168.24.1
    metric: 3

```

4.1.7 Subnet/IP

For any network device (one of the Config Types) users can define a list of `subnets` which contain ip configuration dictionaries. Multiple subnet entries will create interface alias allowing a single interface to use different ip configurations.

Valid keys for `subnets` include the following:

- `type`: Specify the subnet type.
- `control`: Specify manual, auto or hotplug. Indicates how the interface will be handled during boot.
- `address`: IPv4 or IPv6 address. It may include CIDR netmask notation.
- `netmask`: IPv4 subnet mask in dotted format or CIDR notation.
- `gateway`: IPv4 address of the default gateway for this subnet.
- `dns_nameserver`: Specify a list of IPv4 dns server IPs to end up in `resolv.conf`.
- `dns_search`: Specify a list of search paths to be included in `resolv.conf`.

Subnet types are one of the following:

- `dhcp4`: Configure this interface with IPv4 dhcp.
- `dhcp`: Alias for `dhcp4`
- `dhcp6`: Configure this interface with IPv6 dhcp.
- `static`: Configure this interface with a static IPv4.
- `static6`: Configure this interface with a static IPv6 .

When making use of `dhcp` types, no additional configuration is needed in the subnet dictionary.

Subnet DHCP Example:

```

network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
      subnets:
        - type: dhcp

```

Subnet Static Example:

```
network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
      subnets:
        - type: static
          address: 192.168.23.14/27
          gateway: 192.168.23.1
          dns_nameservers:
            - 192.168.23.2
            - 8.8.8.8
          dns_search:
            - exemplary.maas
```

The following will result in an interface0 using DHCP and interface0 : 1 using the static subnet configuration.

Multiple subnet Example:

```
network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
      subnets:
        - type: dhcp
        - type: static
          address: 192.168.23.14/27
          gateway: 192.168.23.1
          dns_nameservers:
            - 192.168.23.2
            - 8.8.8.8
          dns_search:
            - exemplary
```

4.2 Multi-layered configurations

Complex networking sometimes uses layers of configuration. The syntax allows users to build those layers one at a time. All of the virtual network devices supported allow specifying an underlying device by their `name` value.

Bonded VLAN Example:

```
network:
  version: 1
  config:
    # 10G pair
    - type: physical
      name: gbe0
      mac_address: cd:11:22:33:44:00
    - type: physical
      name: gbe1
      mac_address: cd:11:22:33:44:02
    # Bond.
    - type: bond
```



```

name: bond0
bond_interfaces:
  - gbe0
  - gbe1
params:
  bond-mode: 802.3ad
  bond-lacp-rate: fast
# A Bond VLAN.
- type: vlan
  name: bond0.200
  vlan_link: bond0
  vlan_id: 200
  subnets:
    - type: dhcp4

```

4.3 More Examples

Some more examples to explore the various options available.

Multiple VLAN example:

```

network:
  version: 1
  config:
    - id: eth0
      mac_address: d4:be:d9:a8:49:13
      mtu: 1500
      name: eth0
      subnets:
        - address: 10.245.168.16/21
          dns_nameservers:
            - 10.245.168.2
          gateway: 10.245.168.1
          type: static
      type: physical
    - id: eth1
      mac_address: d4:be:d9:a8:49:15
      mtu: 1500
      name: eth1
      subnets:
        - address: 10.245.188.2/24
          dns_nameservers: []
          type: static
      type: physical
    - id: eth1.2667
      mtu: 1500
      name: eth1.2667
      subnets:
        - address: 10.245.184.2/24
          dns_nameservers: []
          type: static
      type: vlan
      vlan_id: 2667
      vlan_link: eth1
    - id: eth1.2668
      mtu: 1500

```

```
name: eth1.2668
subnets:
- address: 10.245.185.1/24
  dns_nameservers: []
  type: static
type: vlan
vlan_id: 2668
vlan_link: eth1
- id: eth1.2669
  mtu: 1500
  name: eth1.2669
  subnets:
  - address: 10.245.186.1/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2669
  vlan_link: eth1
- id: eth1.2670
  mtu: 1500
  name: eth1.2670
  subnets:
  - address: 10.245.187.2/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2670
  vlan_link: eth1
- address: 10.245.168.2
  search:
  - dellstack
  type: nameserver
```

Curtin supports a user-configurable storage layout. This format lets users (including via MAAS) to customize their machines' storage configuration by creating partitions, RAID's, LVM's, formatting with file systems and setting mount points.

Custom storage configuration is handled by the `block-meta custom` command in curtin. Partitioning layout is read as a list of in-order modifications to make to achieve the desired configuration. The top level configuration key containing this configuration is `storage`. This key should contain a dictionary with at least a version number and the configuration list. The current config specification is `version: 1`.

Config Example:

```
storage:
  version: 1
  config:
    - id: sda
      type: disk
      ptable: gpt
      serial: QM00002
      model: QEMU_HARDDISK
```

5.1 Configuration Types

Each entry in the config list is a dictionary with several keys which vary between commands. The two dictionary keys that every entry in the list needs to have are `id: <id>` and `type: <type>`.

An entry's `id` allows other entries in the config to refer to a specific entry. It can be any string other than one with a special meaning in yaml, such as `true` or `none`.

An entry's `type` tells curtin how to handle a particular entry. Available commands include:

- Disk Command (`disk`)
- Partition Command (`partition`)

- Format Command (`format`)
- Mount Command (`mount`)
- LVM_VolGroup Command (`lvm_volgroup`)
- LVM_Partition Command (`lvm_partition`)
- DM_Crypt Command (`dm_crypt`)
- RAID Command (`raid`)
- Bcache Command (`bcache`)

5.1.1 Disk Command

The disk command sets up disks for use by curtin. It can wipe the disks, create partition tables, or just verify that the disks exist with an existing partition table. A disk command may contain all or some of the following keys:

ptable: *msdos, gpt*

If the `ptable` key is present and a valid type of partition table, curtin will create an empty partition table of that type on the disk. At the moment, `msdos` and `gpt` partition tables are supported.

serial: *<serial number>*

In order to uniquely identify a disk on the system its serial number should be specified. This ensures that even if additional storage devices are added to the system during installation, or `udev` rules cause the path to a disk to change curtin will still be able to correctly identify the disk it should be operating on using `/dev/disk/by-id`.

This is the preferred way to identify a disk and should be used in all production environments as it is less likely to point to an incorrect device.

path: *<path to device with leading /dev>*

The `path` key can be used to identify the disk. If both `serial` and `path` are specified, curtin will use the serial number and ignore the path that was specified.

iSCSI disks are supported via a special path prefix of `'iscsi:'`. If this prefix is found in the path specification for a disk, it is assumed to be an iSCSI disk specification and must be in a [RFC4173](#) compliant format, with extensions from Debian for supporting authentication:

`iscsi:[user:password[:iuser:ipassword]@]host:proto:port:lun:targetname`

- `user`: User to authenticate with, if needed, for iSCSI initiator authentication. Only CHAP authentication is supported at this time.
- `password`: Password to authenticate with, if needed, for iSCSI initiator authentication. Only CHAP authentication is supported at this time.
- `iuser`: User to authenticate with, if needed, for iSCSI target authentication. Only CHAP authentication is supported at this time.
- `ipassword`: Password to authenticate with, if needed, for iSCSI target authentication. Only CHAP authentication is supported at this time.

Note: Curtin will treat it as an error if the user and password are not both specified for initiator and target authentication.

- `host`: iSCSI server hosting the specified target. It can be a hostname, IPv4 or IPv6 address. If specified as an IPv6 address, it must be specified as `[address]`.

- `proto`: Specifies the protocol used for iSCSI. Currently only `6`, or `TCP`, is supported and any other value is ignored. If not specified, `6` is assumed.
- `port`: Specifies the port the iSCSI server is listening on. If not specified, `3260` is assumed.
- `lun`: Specifies the LUN of the iSCSI target to connect to. If not specified, `0` is assumed.
- `targetname`: Specifies the iSCSI target to connect to, by its name on the iSCSI server.

Note: Curtin will treat it as an error if the `host` and `targetname` are not specified.

Any iSCSI disks specified will be configured to login at boot in the target.

model: *<disk model>*

This can specify the manufacturer or model of the disk. It is not currently used by curtin, but can be useful for a human reading a config file. Future versions of curtin may make use of this information.

wipe: *superblock, superblock-recursive, zero, random*

If `wipe` is specified, **the disk contents will be destroyed**. In the case that a disk is a part of virtual block device, like `bcache`, `RAID` array, or `LVM`, then curtin will attempt to tear down the virtual device to allow access to the disk for resetting the disk.

The most common option for clearing a disk is `wipe: superblock`. In some cases use of `wipe: superblock-recursive` is useful to ensure that embedded superblocks on a disk aren't rediscovered during probing. For example, `LVM`, `bcache` and `RAID` on a partition would have metadata outside of the range of a superblock wipe of the start and end sections of the disk.

The `wipe: zero` option will write zeros to each sector of the disk. Depending on the size and speed of the disk; it may take a long time to complete.

The `wipe: random` option will write pseudo-random data from `/dev/urandom` Depending on the size and speed of the disk; it may take a long time to complete.

preserve: *true, false*

When the `preserve` key is present and set to `true` curtin will attempt to use the disk without damaging data present on it. If `preserve` is set and `ptable` is also set, then curtin will validate that the partition table specified by `ptable` exists on the disk and will raise an error if it does not. If `preserve` is set and `ptable` is not, then curtin will be able to use the disk in later commands, but will not check if the disk has a valid partition table, and will only verify that the disk exists.

It can be dangerous to try to move or re-size filesystems and partitions containing data that needs to be preserved. Therefore curtin does not support preserving a disk without also preserving the partitions on it. If a disk is set to be preserved and curtin is told to move a partition on that disk, installation will stop. It is still possible to reformat partitions that do not need to be preserved.

name: *<name>*

If the `name` key is present, curtin will create a `udev` rule that makes a symbolic link to the disk with the given name value. This makes it easy to find disks on an installed system. The links are created in `/dev/disk/by-dname/<name>`. A link to each partition on the disk will also be created at `/dev/disk/by-dname/<name>-part<number>`, so if `name: maindisk` is set, the disk will be at `/dev/disk/by-dname/maindisk` and the first partition on it will be at `/dev/disk/by-dname/maindisk-part1`.

grub_device: *true, false*

If the `grub_device` key is present and set to `true`, then when post installation hooks are run `grub` will be installed onto this disk. In most situations it is not necessary to specify this value as curtin will detect and determine which

device to use as a boot disk. In cases where the boot device is on a special volume, such as a RAID array or a LVM Logical Volume, it may be necessary to specify the device that will hold the grub bootloader.

Config Example:

```
- id: disk0
  type: disk
  ptable: gpt
  serial: QM00002
  model: QEMU_HARDDISK
  name: maindisk
  wipe: superblock
```

5.1.2 Partition Command

The partition command creates a single partition on a disk. Curtin only needs to be told which disk to use and the size of the partition. Additional options are available.

number: *<number>*

The partition number can be specified using `number`. However, numbers must be in order and some situations, such as extended/logical partitions on msdos partition tables will require special numbering, so it maybe better to omit the partition number. If the `number` key is not present, curtin will attempt determine the right number to use.

size: *<size>*

The partition size can be specified with the `size` key. Sizes must be given with an appropriate SI unit, such as *B*, *kB*, *MB*, *GB*, *TB*, or using just the appropriate SI prefix, i.e. *B*, *k*, *M*, *G*, *T*...

Note: Curtin does not adjust size values. If you specific a size that exceeds the capacity of a device then installation will fail.

device: *<device id>*

The `device` key refers to the `id` of a disk in the storage configuration. The disk entry must already be defined in the list of commands to ensure that it has already been processed.

wipe: *superblock, pvremove, zero, random*

After the partition is added to the disk's partition table, curtin can run a wipe command on the partition. The wipe command values are the same as for disks.

flag: *logical, extended, boot, bios_grub, swap, lvm, raid, home, prep*

If the `flag` key is present, curtin will set the specified flag on the partition. Note that some flags only apply to msdos partition tables, and some only apply to gpt partition tables.

The *logical/extended* partition flags can be used to create logical partitions on a msdos table. An extended partition should be created containing all of the empty space on the drive, and logical partitions can be created within it. A extended partition must already be present to create logical partitions. If the `number` flag is set for an extended partition it must be set to 4, and each logical partition should be numbered starting from 5.

On msdos partition tables, the *boot* flag sets the boot parameter to that partition. On gpt partition tables, the boot flag sets the esp flag on the partition.

If the host system for curtin has been booted using UEFI then curtin will install grub to the esp partition. If the system installation media has been booted using an MBR, grub will be installed onto the disk's MBR. However, on a disk with a gpt partition table, there is not enough space after the MBR for grub to store its second stage core.img, so a

small un-formatted partition with the *bios_grub* flag is needed. This partition should be placed at the beginning of the disk and should be 1MB in size. It should not contain a filesystem or be mounted anywhere on the system.

preserve: *true, false*

If the preserve flag is set to true, curtin will verify that the partition exists and will not modify the partition.

Config Example:

```
- id: disk0-part1
  type: partition
  number: 1
  size: 8GB
  device: disk0
  flag: boot
```

5.1.3 Format Command

The format command makes filesystems on a volume. The filesystem type and target volume can be specified, as well as a few other options.

fstype: *ext4, ext3, fat32, fat16, swap, xfs*

The *fstype* key specifies what type of filesystem format curtin should use for this volume. Curtin knows about common Linux filesystems such as ext4/3 and fat filesystems and makes use of additional parameters and flags to optimize the filesystem. If the *fstype* value is not known to curtin, that is not fatal. Curtin will check if *mkfs.<fstype>* exists and if so, will use that tool to format the target volume.

For fat filesystems, the size of the fat table can be specified by entering *fat64*, *fat32*, *fat16*, or *fat12* instead of just entering *fat*. If *fat* is used, then *mkfs.fat* will automatically determine the best size fat table to use, probably *fat32*.

If *fstype: swap* is set, curtin will create a swap partition on the target volume.

volume: *<volume id>*

The *volume* key refers to the *id* of the target volume in the storage config. The target volume must already exist and be accessible. Any type of target volume can be used as long as it has a block device that curtin can locate.

label: *<volume name>*

The *label* key tells curtin to create a filesystem LABEL when formatting a volume. Note that not all filesystem types support names and that there are length limits for names. For fat filesystems, names are limited to 11 characters. For ext4/3 filesystems, names are limited to 16 characters.

If curtin does not know about the filesystem type it is using, then the *label* key will be ignored, because curtin will not know the correct flags to set the label value in the filesystem metadata.

uuid: *<uuid>*

If the *uuid* key is set and *fstype* is set to *ext4* or *ext3*, then curtin will set the uuid of the new filesystem to the specified value.

preserve: *true, false*

If the *preserve* key is set to true, curtin will not format the partition.

Config Example:

```
- id: disk0-part1-fs1
  type: format
  fstype: ext4
```

```
label: cloud-image
volume: disk0-part1
```

5.1.4 Mount Command

The mount command mounts the target filesystem and creates an entry for it in the newly installed system's `/etc/fstab`. The path to the target mountpoint must be specified as well as the target filesystem.

path: *<path>*

The path key tells curtin where the filesystem should be mounted on the target system. An entry in the target system's `/etc/fstab` will be created for the target device which will mount it in the correct place once the installed system boots.

If the device specified is formatted as swap space, then an entry will be added to the target system's `/etc/fstab` to make use of this swap space.

When entries are created in `/etc/fstab`, curtin will use the most reliable method available to identify each device. For regular partitions, curtin will use the UUID of the filesystem present on the partition. For special devices, such as RAID arrays, or LVM logical volumes, curtin will use their normal path in `/dev`.

device: *<device id>*

The device key refers to the `id` of the target device in the storage config. The target device must already contain a valid filesystem and be accessible.

Note: If the specified device refers to an iSCSI device, the corresponding `fstab` entry will contain `_netdev` to indicate networking is required to mount this filesystem.

options: *<mount(8) comma-separated options string>*

The options key will replace the default options value of defaults.

Warning: The kernel and user-space utilities may differ between the install environment and the runtime environment. Not all kernels and user-space combinations will support all options. Providing options for a mount point will have both of the following effects:

- curtin will mount the filesystems with the provided options during the installation.
- curtin will ensure the target OS uses the provided mount options by updating the target OS (`/etc/fstab`).

If either of the environments (install or target) do not have support for the provided options, the behavior is undefined.

Config Example:

```
- id: disk0-part1-fs1-mount0
  type: mount
  path: /home
  device: disk0-part1-fs1
  options: 'noatime,errors=remount-ro'
```


5.1.5 Lvm Volgroup Command

The `lvm_volgroup` command creates LVM Physical Volumes (PV) and connects them in a LVM Volume Group (vg). The command requires a name for the volgroup and a list of the devices that should be used as physical volumes.

name: *<name>*

The `name` key specifies the name of the volume group. It anything can be used except words with special meanings in YAML, such as *true*, or *none*.

devices: *[]*

The `devices` key gives a list of devices to use as physical volumes. Each device is specified using the `id` of existing devices in the storage config. Almost anything can be used as a device such as partitions, whole disks, RAID.

Config Example:

```
- id: volgroup1
  type: lvm_volgroup
  name: vg1
  devices:
    - disk0-part2
    - disk1
```

5.1.6 Lvm Partition Command

The `lvm_partition` command creates a lvm logical volume on the specified volgroup with the specified size. It also assigns it the specified name.

name: *<name>*

The `name` key specifies the name of the Logical Volume (LV) to be created.

Curtin creates udev rules for Logical Volumes to give them consistently named symbolic links in the target system under `/dev/disk/by-dname/`. The naming scheme for Logical Volumes follows the pattern `<volgroup name>-<logical volume name>`. For example a `lvm_partition` with name *lv1* on a `lvm_volgroup` named *vg1* would have the path `/dev/disk/by-dname/vg1-lv1`.

volgroup: *<volgroup id>*

The `volgroup` key specifies the `id` of the Volume Group in which to create the logical volume. The volgroup must already have been created and must have enough free space on it to create the logical volume. The volgroup should be specified using the `id` key of the volgroup in the storage config, not the name of the volgroup.

size: *<size>*

The `size` key tells curtin what size to make the logical volume. The size can be entered in any format that can be processed by the lvm2 tools, so a number followed by a SI unit should work, i.e. *B*, *kB*, *MB*, *GB*, *TB*.

If the `size` key is omitted then all remaining space on the volgroup will be used for the logical volume.

Note: Curtin does not adjust size values. If you specific a size that exceeds the capacity of a device then installation will fail.

Config Example:

```
- id: lvm_partition_1
  type: lvm_partition
  name: lv1
```

```
volgroup: volgroup1
size: 10G
```

Combined Example:

```
- id: volgroup1
  type: lvm_volgroup
  name: vg1
  devices:
    - disk0-part2
    - disk1
- id: lvm_partition_1
  type: lvm_partition
  name: lv1
  volgroup: volgroup1
  size: 10G
```

5.1.7 Dm-Crypt Command

The `dm_crypt` command creates encrypted volumes using `cryptsetup`. It requires a name for the encrypted volume, the volume to be encrypted and a key. Note that this should not be used for systems where security is a requirement. The key is stored in plain-text in the storage configuration and it could be possible for the storage configuration to be intercepted between the utility that generates it and curtin.

volume: <volume id>

The `volume` key gives the volume that is to be encrypted.

dm_name: <name>

The `name` key specifies the name of the encrypted volume.

key: <key>

The `key` key specifies the password of the encryption key. The target system will prompt for this password in order to mount the disk.

Note: Encrypted disks and partitions are tracked in `/etc/crypttab` and will be mounted at boot time.

Config Example:

```
- id: lvm_partition_1
  type: dm_crypt
  dm_name: crypto
  volume: sdb1
  key: testkey
```

5.1.8 RAID Command

The `RAID` command configures Linux Software RAID using `mdadm`. It needs to be given a name for the md device, a list of volumes for to compose the md device, an optional list of devices to be used as spare volumes, and RAID level.

name: <name>

The `name` key specifies the name of the md device.

Note: Curtin creates a udev rule to create a link to the md device in `/dev/disk/by-dname/<name>` using the specified name.

raidlevel: *0, 1, 5, 6, 10*

The `raidlevel` key specifies the raid level of the array.

devices: *[]*

The `devices` key specifies a list of the devices that will be used for the raid array. Each device must be referenced by `id` and the device must be previously defined in the storage configuration. Must not be empty.

Devices can either be full disks or partition.

spare_devices: *[]*

The `spare_devices` key specifies a list of the devices that will be used for spares in the raid array. Each device must be referenced by `id` and the device must be previously defined in the storage configuration. May be empty.

Config Example:

```
- id: raid_array
  type: raid
  name: md0
  raidlevel: 1
  devices:
    - sdb
    - sdc
  spare_devices:
    - sdd
```

5.1.9 Bcache Command

The `bcache` command will configure a block-cache device using the Linux kernel `bcache` module. Bcache allows users to use a typically small, but fast SSD or NVME device as a cache for larger, slower spinning disks.

The `bcache` command needs to be told which device to use hold the data and which device to use as its cache device. A cache device may be reused with multiple backing devices.

backing_device: *<device id>*

The `backing_device` key specifies the item in storage configuration to use as the backing device. This can be any device that would normally be used with a filesystem on it, such as a partition or a raid array.

cache_device: *<device id>*

The `cache_device` key specifies the item in the storage configuration to use as the cache device. This can be a partition or a whole disk. It should be on a `ssd` in most cases, as `bcache` is designed around the performance characteristics of a `ssd`.

cache_mode: *writethrough, writeback, writearound, none*

The `cache_mode` key specifies the mode in which `bcache` operates. The default mode is `writethrough` which ensures data hits the backing device before completing the operation. `writeback` mode will have higher performance but exposes dataloss if the cache device fails. `writearound` will avoid using the cache for large sequential writes; useful for not evicting smaller reads/writes from the cache. `None` effectively disables `bcache`.

name: *<name>*

If the `name` key is present, `curtin` will create a link to the device at `/dev/disk/by-dname/<name>`.

Config Example:

```
- id: bcache0
  type: bcache
  name: cached_raid
  backing_device: raid_array
  cache_device: sdb
```

5.2 Additional Examples

Learn by examples.

- Basic
- LVM
- Bcache
- RAID Boot
- RAID5 + Bcache

5.2.1 Basic Layout

```
storage:
  version: 1
  config:
    - id: disk0
      type: disk
      ptable: msdos
      model: QEMU HARDDISK
      path: /dev/vdb
      name: main_disk
      wipe: superblock
      grub_device: true
    - id: disk0-part1
      type: partition
      number: 1
      size: 3GB
      device: disk0
      flag: boot
    - id: disk0-part2
      type: partition
      number: 2
      size: 1GB
      device: disk0
    - id: disk0-part1-format-root
      type: format
      fstype: ext4
      volume: disk0-part1
    - id: disk0-part2-format-home
      type: format
      fstype: ext4
      volume: disk0-part2
    - id: disk0-part1-mount-root
      type: mount
```

```

    path: /
    device: disk0-part1-format-root
-   id: disk0-part2-mount-home
    type: mount
    path: /home
    device: disk0-part2-format-home

```

5.2.2 LVM

```

storage:
  version: 1
  config:
    - id: sda
      type: disk
      ptable: msdos
      model: QEMU HARDDISK
      path: /dev/vdb
      name: main_disk
    - id: sda1
      type: partition
      size: 3GB
      device: sda
      flag: boot
    - id: sda_extended
      type: partition
      size: 5G
      flag: extended
      device: sda
    - id: sda2
      type: partition
      size: 2G
      flag: logical
      device: sda
    - id: sda3
      type: partition
      size: 3G
      flag: logical
      device: sda
    - id: volgroup1
      name: vg1
      type: lvm_volgroup
      devices:
        - sda2
        - sda3
    - id: lvmpart1
      name: lv1
      size: 1G
      type: lvm_partition
      volgroup: volgroup1
    - id: lvmpart2
      name: lv2
      type: lvm_partition
      volgroup: volgroup1
    - id: sda1_root
      type: format
      fstype: ext4

```

```
volume: sdal
- id: lv1_fs
  name: storage
  type: format
  fstype: fat32
  volume: lvmpart1
- id: lv2_fs
  name: storage
  type: format
  fstype: ext3
  volume: lvmpart2
- id: sdal_mount
  type: mount
  path: /
  device: sdal_root
- id: lv1_mount
  type: mount
  path: /srv/data
  device: lv1_fs
- id: lv2_mount
  type: mount
  path: /srv/backup
  device: lv2_fs
```

5.2.3 Bcache

```
storage:
  version: 1
  config:
    - id: id_rotary0
      type: disk
      name: rotary0
      path: /dev/vdb
      ptable: msdos
      wipe: superblock
      grub_device: true
    - id: id_ssd0
      type: disk
      name: ssd0
      path: /dev/vdc
      wipe: superblock
    - id: id_rotary0_part1
      type: partition
      name: rotary0-part1
      device: id_rotary0
      number: 1
      size: 999M
      wipe: superblock
    - id: id_rotary0_part2
      type: partition
      name: rotary0-part2
      device: id_rotary0
      number: 2
      size: 9G
      wipe: superblock
    - id: id_bcache0
```

```

    type: bcache
    name: bcache0
    backing_device: id_rotary0_part2
    cache_device: id_ssd0
    cache_mode: writeback
  - id: bootfs
    type: format
    label: boot-fs
    volume: id_rotary0_part1
    fstype: ext4
  - id: rootfs
    type: format
    label: root-fs
    volume: id_bcache0
    fstype: ext4
  - id: rootfs_mount
    type: mount
    path: /
    device: rootfs
  - id: bootfs_mount
    type: mount
    path: /boot
    device: bootfs

```

5.2.4 RAID Boot

```

storage:
  version: 1
  config:
    - id: sda
      type: disk
      ptable: gpt
      model: QEMU HARDDISK
      path: /dev/vdb
      name: main_disk
      grub_device: 1
    - id: bios_boot_partition
      type: partition
      size: 1MB
      device: sda
      flag: bios_grub
    - id: sdal
      type: partition
      size: 3GB
      device: sda
    - id: sdb
      type: disk
      ptable: gpt
      model: QEMU HARDDISK
      path: /dev/vdc
      name: second_disk
    - id: sdb1
      type: partition
      size: 3GB
      device: sdb
    - id: sdc

```

```
    type: disk
    ptable: gpt
    model: QEMU HARDDISK
    path: /dev/vdd
    name: third_disk
  - id: sdc1
    type: partition
    size: 3GB
    device: sdc
  - id: mddevice
    name: md0
    type: raid
    raidlevel: 5
    devices:
      - sda1
      - sdb1
      - sdc1
  - id: md_root
    type: format
    fstype: ext4
    volume: mddevice
  - id: md_mount
    type: mount
    path: /
    device: md_root
```

5.2.5 RAID5 + Bcache

```
storage:
  config:
  - grub_device: true
    id: sda
    model: QEMU HARDDISK
    name: sda
    ptable: msdos
    path: /dev/vdb
    type: disk
    wipe: superblock
  - id: sdb
    model: QEMU HARDDISK
    name: sdb
    path: /dev/vdc
    type: disk
    wipe: superblock
  - id: sdc
    model: QEMU HARDDISK
    name: sdc
    path: /dev/vdd
    type: disk
    wipe: superblock
  - id: sdd
    model: QEMU HARDDISK
    name: sdd
    path: /dev/vde
    type: disk
    wipe: superblock
```



```

- id: sde
  model: QEMU HARDDISK
  name: sde
  path: /dev/vdf
  type: disk
  wipe: superblock
- devices:
  - sdc
  - sdd
  - sde
  id: md0
  name: md0
  raidlevel: 5
  spare_devices: []
  type: raid
- device: sda
  id: sda-part1
  name: sda-part1
  number: 1
  size: 1000001536B
  type: partition
  uuid: 3a38820c-d675-4069-b060-509a3d9d13cc
  wipe: superblock
- device: sda
  id: sda-part2
  name: sda-part2
  number: 2
  size: 7586787328B
  type: partition
  uuid: 17747faa-4b9e-4411-97e5-12fd3d199fb8
  wipe: superblock
- backing_device: sda-part2
  cache_device: sdb
  cache_mode: writeback
  id: bcache0
  name: bcache0
  type: bcache
- fstype: ext4
  id: sda-part1_format
  label: ''
  type: format
  uuid: 71blef6f-5cab-4a77-b4c8-5a209ec11d7c
  volume: sda-part1
- fstype: ext4
  id: md0_format
  label: ''
  type: format
  uuid: b031f0a0-adb3-43be-bb43-ce0fc8a224a4
  volume: md0
- fstype: ext4
  id: bcache0_format
  label: ''
  type: format
  uuid: ce45bbaf-5a44-4487-b89e-035c2dd40657
  volume: bcache0
- device: bcache0_format
  id: bcache0_mount
  path: /

```

```
type: mount
- device: sda-part1_format
  id: sda-part1_mount
  path: /boot
  type: mount
- device: md0_format
  id: md0_mount
  path: /srv/data
  type: mount
version: 1
```

Curthooks / New OS Support

Curtin has built-in support for installation of Ubuntu. Other operating systems are supported through a mechanism called ‘curthooks’ or ‘curtin-hooks’.

A curtin install runs through different stages. See the *Stages* documentation for function of each stage. The stages communicate with each other via data in a working directory and environment variables as described in *Command Environment*.

Curtin handles partitioning, filesystem creation and target filesystem population for all operating systems. Curthooks are the mechanism provided so that the operating system can customize itself before reboot. This customization typically would need to include:

- ensuring that appropriate device drivers are loaded on first boot
- consuming the network interfaces file and applying its declarations.
- ensuring that necessary packages are installed to utilize storage configuration or networking configuration.
- making the system boot (running grub-install or equivalent).

6.1 Image provided curtin-hooks

An image provides curtin hooks support by containing a file `/curtin/curtin-hooks`.

If an Ubuntu image contains this path it will override the builtin curtin support.

The `curtin-hooks` program should be executable in the filesystem and will be executed without any arguments. It will be executed in the install environment, *not* the target environment. A change of root to the target environment can be done with `curtin in-target`.

The hook is provided with some environment variables that can be used to find more information. See the *Command Environment* doc for details. Specifically interesting to this stage are:

- `OUTPUT_NETWORK_CONFIG`: This is a path to the file created during network discovery stage.
- `OUTPUT_FSTAB`: This is a path to the file created during partitioning stage.

- `CONFIG`: This is a path to the curtin config file. It is provided so that additional configuration could be provided through to the OS customization.

6.2 Networking configuration

Access to the network configuration that is desired is inside the config and is in the format described in [Networking](#).

The `curtin-hooks` program must read the configuration from the path contained in `OUTPUT_NETWORK_CONFIG` and then set up the installed system to use it.

If the installed system has `cloud-init` at version 17.1 or higher, it may be possible to simply copy this section into the target in `/etc/cloud/cloud.cfg.d/` and let `cloud-init` render the correct networking on first boot.

6.3 Storage configuration

Access to the storage configuration that was set up is inside the config and is in the format described in [Storage](#).

To apply this storage configuration, the curthooks may need to:

- update `/etc/fstab` to add the expected mounts entries. The environment variable `OUTPUT_FSTAB` contains a path to a file that may be suitable for use.
- install any packages that are not already installed that are required to boot with the provided storage config. For example, if the storage layout includes raid you may need to install the `mdadm` package.
- update or create an `initramfs`.

6.4 System boot

In Ubuntu, curtin will run ‘`grub-setup`’ and to install `grub`. This covers putting the bootloader onto the disk(s) that are marked as `grub_device`. The provided hook will need to do the equivalent operation.

6.5 finalize hook

There is one other hook that curtin will invoke in an install, called `finalize`. This program is invoked in the same environment as `curtin-hooks` above. It is intended to give the OS a final opportunity make updates before reboot. It is called before `late_commands`.

Curtin is capable of reporting its progress via the reporting framework. This enables the user to obtain status information from curtin.

7.1 Events

Reporting consists of notification of a series of ‘events. Each event has:

- **event_type:** ‘start’ or ‘finish’
- **description:** human readable text
- **level:** the log level of the event, DEBUG/INFO/WARN etc.
- **name:** and id for this event
- **result:** only present when event_type is ‘finish’, its value is one of “SUCCESS”, “WARN”, or “FAIL”. A result of WARN indicates something is likely wrong, but a non-fatal error. A result of “FAIL” is fatal.
- **origin:** literal value ‘curtin’
- **timestamp:** the unix timestamp at which this event occurred

names are unique and hierarchical. For example, a series of names might look like:

- cmd-install (start)
- cmd-install/stage-early (start)
- cmd-install/stage-early (finish)
- cmd-install (finish)

You are guaranteed to always get a finish for each sub-item before finish of the parent item, and guaranteed to get finish for all events. A FAIL result of a sub-item will bubble up to its parent item.

7.2 Configuration

Reporting configuration is done through the `reporting` item in config. An example config:

```
reporting:
  keyname1:
    type: webhook
    endpoint: "http://127.0.1.1:8000/"
  keyname2:
    type: print

install:
  log_file: /tmp/install.log
  post_files: [/tmp/install.log, /var/log/syslog]
```

Each entry in the `reporting` dictionary must be a dictionary. The key is only used for reference and to aid in config merging.

Each entry must have a ‘type’. The currently supported values are:

- **log:** logs via python logger
- **print:** prints messages to stdout (for debugging)
- **webhook:** posts json formatted data to a remote url. Supports Oauth.

Additionally, the webhook reporter will post files on finish of curtin. The user can declare which files should be posted in the `install` item via `post_files` as shown above. If `post_files` is not present, it will default to the value of `log_file`.

7.3 Webhook Reporter

The webhook reporter posts the event in json format to an endpoint. To enable, provide curtin with config like:

```
reporting:
  mylistener:
    type: webhook
    endpoint: http://example.com/endpoint/path
    consumer_key: "ck_foo"
    consumer_secret: "cs_foo"
    token_key: "tk_foo"
    token_secret: "tk_secret"
    level: INFO
```

The `endpoint` key is required. Oauth information (`consumer_key`, `consumer_secret`, `token_key`, `token_secret`) is not required, but if provided then oauth will be used to authenticate to the endpoint on each post. If `level` is specified then all messages with a lower priority than specified will be ignored. Default is INFO.

7.4 Journald Reporter

The journald reporter sends the events to systemd’s `journald`. To enable, provide curtin with config like:

```
reporting:
  mylistener:
    type: journald
```

```

identifier: "my_identifier"
level: DEBUG

```

The event's fields are mapped to fields of the resulting journal entry as follows:

- **description** maps to **CURTIN_MESSAGE**
- **level** maps to **PRIORITY**
- **name** maps to **CURTIN_NAME**
- **event_type** maps to **CURTIN_EVENT_TYPE**
- **result**, if present, maps to **CURTIN_RESULT**

The configured *identifier*, which defaults to “curtin_event”, becomes the entry's **SYSLOG_IDENTIFIER**.

The python-systemd package must be installed to use this handler.

7.4.1 Example Events

The following is an example event that would be posted:

```

{
  "origin": "curtin",
  "timestamp": 1440688425.6038516,
  "event_type": "start",
  "name": "cmd-install",
  "description": "curtin command install",
  "level": "INFO"
}

```

The post files will look like this:

```

{
  "origin": "curtin",
  "files": [
    {
      "content": "fCBzZmRpc2s...gLS1ubylyZX",
      "path": "/var/log/curtin/install.log",
      "encoding": "base64"
    },
    {
      "content": "fCBzZmRpc2s...gLS1ubylyZX",
      "path": "/var/log/syslog",
      "encoding": "base64"
    }
  ],
  "description": "curtin command install",
  "timestamp": 1440688425.6038516,
  "name": "cmd-install",
  "result": "SUCCESS",
  "event_type": "finish"
}

```

7.4.2 Example Http Request

The following is an example http request from curtin:

```
Accept-Encoding: identity
Host: localhost:8000
Content-Type: application/json
Connection: close
User-Agent: Curtin/0.1
Content-Length: 156

{
  "origin": "curtin",
  "timestamp": 1440688425.6038516,
  "event_type": "start",
  "name": "cmd-install/stage-early",
  "description": "preparing for installation"
}
```

7.5 Development / Debug Reporting

For debugging and development a simple web server is provided in *tools/report-webhook-logger*.

Run the web service like:

```
./tools/report-webhook-logger 8000
```

And then run your install with appropriate config, like:

```
sudo ./bin/curtin -vvv install \
  --set install/logfile=/tmp/foo \
  --set reporting/mypost/type=webhook \
  --set reporting/mypost/endpoint=http://localhost:8000/
file://$root_tgz
```

7.6 Legacy Reporter

The legacy ‘reporter’ config entry is still supported. This was utilized by MAAS for start/end and posting of the install log at the end of installation.

Its configuration looks like this:

Legacy Reporter Config Example:

```
reporter:
  url: http://example.com/your/path/to/post
  consumer_key: "ck_foo"
  consumer_secret: "cs_foo"
  token_key: "tk_foo"
  token_secret: "tk_secret"
```


Curtin developers make use of cloud-images and kvm to help develop and test new curtin features.

8.1 Install dependencies

Install some virtualization and cloud packages to get started.:

```
sudo apt-get -qy install kvm libvirt-bin cloud-utils bzip2
```

8.2 Download cloud images

Curtin will use two cloud images (-disk1.img) is for booting, (-root.tar.gz) is used for installing.:

```
mkdir -p ~/download
DLDIR=$( cd ~/download && pwd )
rel="trusty"
arch=amd64
burl="http://cloud-images.ubuntu.com/$rel/current/"
for f in $rel-server-cloudimg-${arch}-root.tar.gz $rel-server-cloudimg-${arch}-disk1.
↪img; do
    wget "$burl/$f" -O $DLDIR/$f; done
( cd $DLDIR && qemu-img convert -O qcow $rel-server-cloudimg-${arch}-disk1.img $rel-
↪server-cloudimg-${arch}-disk1.qcow2)

export BOOTIMG="$DLDIR/$rel-server-cloudimg-${arch}-disk1.qcow2"
export ROOTTGZ="$DLDIR/$rel-server-cloudimg-${arch}-root.tar.gz"
```

8.3 Getting the source

Download curtin source from launchpad via *bzr* command.:

```
mkdir -p ~/src
bzr init-repo ~/src/curtin
( cd ~/src/curtin && bzr branch lp:curtin trunk.dist )
( cd ~/src/curtin && bzr branch trunk.dist trunk )
```

8.4 Using curtin

Use *launch* to launch a kvm instance with user data to pack up local curtin and run it inside an instance.:

```
cd ~/src/curtin/trunk
./tools/launch $BOOTIMG --publish $ROOTTGZ -- curtin install "PUBURL/${ROOTTGZ##*/}"
```

8.5 Notes about ‘launch’

- *launch* has *--help* so you can see that for some info.
- *--publish* adds a web server at `${HTTP_PORT:-$RANDOM_PORT}` and puts the files you want available there. You can reference this url in config or cmdline with ‘PUBURL’. For example *--publish foo.img* will put *foo.img* at `PUBURL/foo.img`.
- *launch* sets ‘ubuntu’ user password to ‘passw0rd’
- *launch* runs ‘kvm -curses’ kvm -curses keyboard info: ‘alt-2’ to go to qemu console
- *launch* puts serial console to ‘serial.log’ (look there for stuff)
- when logged in - you can look at `/var/log/cloud-init-output.log` - archive should be extracted in `/curtin` - shell archive should be in `/var/lib/cloud/instance/scripts/part-002`

Curtin includes an in-tree integration suite that runs hundreds of tests validating various forms of custom storage and network configurations across all of the supported Ubuntu LTS releases as well as some of the currently supported interim releases.

9.1 Background

Curtin includes a mechanism called ‘vmtest’ that allows it to actually do installs and validate a number of configurations.

The general flow of the vmtests is:

1. each test has an associated yaml config file for curtin in examples/tests
2. uses curtin-pack to create the user-data for cloud-init to trigger install
3. create and install a system using ‘tools/launch’.
 - (a) The install environment is booted from a maas ephemeral image.
 - (b) kernel & initrd used are from maas images (not part of the image)
 - (c) network by default is handled via user networking
 - (d) It creates all empty disks required
 - (e) cloud-init datasource is provided by launch
 - i. like: ds=nocloud-net;seedfrom=http://10.7.0.41:41518/ provided by python webserver start_http
 - ii. via -drive file=/tmp/launch.8VOiOn/seed.img,if=virtio,media=cdrom as a seed disk (if booted without external kernel)
 - (f) dependencies and other preparations are installed at the beginning by curtin inside the ephemeral image prior to configuring the target
4. power off the system.

5. configure a ‘NoCloud’ datasource seed image that provides scripts that will run on first boot.
 - (a) this will contain all our code to gather health data on the install
 - (b) by cloud-init design this runs only once per instance, if you start the system again this won’t be called again
6. boot the installed system with ‘tools/xkvm’.
 - (a) reuses the disks that were installed/configured in the former steps
 - (b) also adds an output disk
 - (c) additionally the seed image for the data gathering is added
 - (d) On this boot it will run the provided scripts, write their output to a “data” disk and then shut itself down.
7. extract the data from the output disk
8. vmtest python code now verifies if the output is as expected.

9.2 Debugging

At 3.1 one can pull data out of the maas image the command `mount-image-callback`. For example:

```
sudo mount-image-callback your.img -- sh -c 'cp $MOUNTPOINT/boot/* .'
```

At step 3.6 through 4 `tools/launch` can be called in a way to give you console access. To do so just call `tools/launch` but drop the `-serial=x` parameter. One might want to change “power_state”: {‘mode’: ‘poweroff’}” to avoid the auto reboot before getting control. Replace the directory usually seen in the launch calls with a clean fresh directory.

In `/curtin` `curtin` and its config can be found. If the system gets that far cloud-init will create a user `creds:ubuntu/passw0rd`, otherwise one can use a cloud-image from <https://cloud-images.ubuntu.com/> and add a backdoor user via:

```
bzr branch lp:~maas-maintainers/maas/backdoor-image backdoor-image
sudo ./backdoor-image -v --user=<USER> --password-auth --password=<PW> IMG
```

At step 6 -> 7 you might want to keep all the temporary images around. To do so you can set `CURTIN_VMTEST_KEEP_DATA_PASS=all` in your environment.

```
export CURTIN_VMTEST_KEEP_DATA_PASS=all CURTIN_VMTEST_KEEP_DATA_FAIL=all
```

That will keep the `/tmp/tmpXXXXXX` directories and all files in there for further execution.

At step 7 you might want to take a look at the output disk yourself. It is a normal qcow image, so one can use `mount-image-callback` as described above.

To invoke `xkvm` on your own take the command you see in the output and remove the “-serial ...” but add `-nographic` instead For graphical console one can add `--vnc 127.0.0.1:1`

9.3 Setup

In order to run `vmtest` you’ll need some dependencies. To get them, you can run:

```
make vmtest-deps
make sync-images # Uses the IMAGE_DIR environment variable mentioned below
```

9.4 Running

Running tests is done most simply by:

```
make vmtest
```

Note: By default, the vmtests for iSCSI will be skipped (see Environment Variable section for details).

If you wish to all tests in test_network.py, do so with:

```
nosetests3 tests/vmtests/test_network.py
```

Or run a single test with:

```
nosetests3 tests/vmtests/test_network.py::XenialTestNetworkBasic
```

9.5 Environment Variables

Some environment variables affect the running of vmtest

- `apt_proxy`:
test will set apt: { proxy } in the guests to the value of `apt_proxy` environment variable. If that is not set it will look at the host's apt config and read `Acquire::HTTP::Proxy`
- `CURTIN_VMTEST_CURTIN_EXE`: Defaults to ''
This is the path to the curtin executable that should be used for testing. It will need to set any environment that is needed and correctly pack itself. The default 'curtin' command in bin/ or installed by apt-get install curtin should work.
If the value is unset or empty, then curtin from <topdir>/bin is used.
So to run vmtest on an installed version of curtin with, you can simply set this variable to 'curtin' or '/usr/bin/curtin'
- `CURTIN_VMTEST_KEEP_DATA_PASS`: Defaults to none.
- `CURTIN_VMTEST_KEEP_DATA_FAIL`: Defaults to all.

These 2 variables determine what portions of the temporary test data are kept.

The variables contain a comma ',' delimited list of directories that should be kept in the case of pass or fail. Additionally, the values 'all' and 'none' are accepted.

Each vmtest that runs has its own sub-directory under the top level `CURTIN_VMTEST_TOPDIR`. In that directory are directories:

- `boot`: inputs to the system boot (after install)
- `install`: install phase related files
- `disks`: the disks used for installation and boot
- `logs`: install and boot logs
- `collect`: data collected by the boot phase

- CURTIN_VMTEST_TAR_DISKS: default 0

Vmtest writes out disk image files sparsely into a disks directory. If this flag is set to a non-zero number, vmtest will tar all disks in the directory into a single disks.tar and remove the sparse disk files.

- CURTIN_VMTEST_TOPDIR: default \$TMPDIR/vmtest-<timestamp>

Vmtest puts all test data under this value. By default, it creates a directory in TMPDIR (/tmp) named with as vmtest-<timestamp>

If you set this value, you must ensure that the directory is either non-existent or clean.

- CURTIN_VMTEST_REUSE_TOPDIR: default 0

If this variable is set to 1, then vmtest will detect if the test specified already exists in the configured CURTIN_VMTEST_TOPDIR location. If present, vmtest will skip executing the install and boot phase of vmtest, and install just execute the unittests specified. This allows developers to re-run unittests on existing data that's already been collected.

- CURTIN_VMTEST_LOG: default \$TMPDIR/vmtest-<timestamp>.log

Vmtest writes extended log information to this file. The default puts the log along side the TOPDIR.

- CURTIN_VMTEST_IMAGE_SYNC: default false (boolean)

If set to true, each run will attempt a sync of images. If you want to make sure images are always up to date, then set to true.

- CURTIN_VMTEST_BRIDGE: user

The network devices will be attached to this bridge. The default is user, which means to use qemu user mode networking. Set it to virbr0 or lxdbr0 to use those bridges and then be able to ssh in directly.

- CURTIN_VMTEST_BOOT_TIMEOUT: default 300

timeout before giving up on the boot of the installed system.

- CURTIN_VMTEST_INSTALL_TIMEOUT: default 3000

timeout before giving up on installation.

- CURTIN_VMTEST_PARALLEL: default ''

only supported through ./tools/jenkins-runner .

- -1: then run one per core.
- 0 or ' ': run with no parallel
- >0: run with N processes

This modifies the invocation of nosetests to add '-processes' and other necessary nose arguments (-process-timeout)

- CURTIN_VMTEST_NR_CPUS: default ''

Allow environment to override the number of virtual cpus to allocate in the target virtual machines.

- IMAGE_DIR: default /srv/images

Vmtest keeps a mirror of maas ephemeral images in this directory.

- IMAGES_TO_KEEP: default 1

Controls the number of images of each release retained in the IMAGE_DIR.

- CURTIN_VMTEST_EXTRA_CONFIG: default “

This can be set to a valid path to a config yaml. That can be used to change behaviour of the tests however a current debugging session needs it. The following example shows how it can be used for tests against a ppa, but this can also be used to test proposed or actually any modification to ephemeral or target as needed:

```
# example ppa to test into install environment
early_commands:
  10_add_ppa: ['sh', '-xc', 'DEBIAN_FRONTEND=noninteractive add-apt-repository --
↪yes <yourppa>']
  # update & upgrade what is there already
  97_update: ['apt-get', 'update']
  98_upgrade: ['sh', '-xc', 'DEBIAN_FRONTEND=noninteractive apt-get upgrade --yes
↪']
# example ppa into target environment via apt feature
apt:
  sources:
    ignored1:
      source: "<yourppa>"
# example of any other modification
early_commands:
  01_something: ['sh', '-xc', '<yourcommand>']
# in target
late_commands:
  02_something: ['sh', '-xc', 'curtin in-target -- <yourcommand>']
```

- CURTIN_VMTEST_ISCSI_PORTAL: default “

By default, iSCSI tests are skipped when running *make vmtest*, as iSCSI server configuration is necessary. *tools/jenkins-runner* will configure a *tgt* server if possible and set the necessary environment variables.

If an accessible iSCSI server is available, it can be specified in this environment variable as *HOST:PORT*. *HOST* can be a hostname, IPv4 address or IPv6 address. If an IPv6 address is used, it must be enclosed in `[]`.

Additionally, if a *tgt* server is running locally as the iSCSI server and is configured to listen on a non-default socket, it is necessary to specify *TGT_IPC_SOCKET* to indicate the path to the socket in use.

As iSCSI server configuration by-hand can be difficult, there is a script in *tools/find-tgt* which can be used to run a local *tgt* server. It will find an available port and use the default route-able IPv4 address on the system. The script takes a directory as parameter, and will emit a *info* file in that directory which can be sourced as a shell script to set the relevant environment variables needed to run the iSCSI vmtests. For example:

```
mkdir output
./tools/find-tgt output
. output/info
nosetests3 tests/vmtests/test_iscsi.py
```

Or, using *jenkins-runner*:

```
./tools/jenkins-runner tests/vmtests/test_iscsi.py
```

9.6 Environment ‘boolean’ values

For boolean environment variables the value is considered True if it is any value other than case insensitive ‘false’, ‘’ or ‘0’.

9.7 Test Class Variables

The base VMBaseClass defines several variables that help creating a new test easily. Among those the common ones are:

Generic:

- `arch_skip`:
If a test is not supported on an architecture it can list the arch in this variable to auto-skip the test if executed on that arch.
- `conf_file`:
The configuration that will be processed by this vmtest.
- `extra_kern_args`:
Extra arguments to the guest kernel on boot.

Data Collection:

- `collect_scripts`:
The commands run when booting into the installed environment to collect the data for the test to verify a proper execution.
- `boot_cloudconf`:
Extra cloud-init config content for the install phase. This allows to gather content of the install phase if needed for test verification.

Disk Setup:

- `disk_block_size`:
Default block size 512 bytes.
- `disk_driver`:
Default block device driver is `virtio-blk`.

iSCSI Setup:

- `iscsi_disks`:

CHAPTER 10

Indices and tables

- `genindex`
- `modindex`
- `search`