
cloud-init

Release 22.1

Feb 15, 2022

1	Getting help	3
1.1	Availability	3
1.2	Boot Stages	5
1.3	First Boot Determination	7
1.4	CLI Interface	8
1.5	FAQ	13
1.6	Reporting Bugs	17
1.7	User-Data Formats	18
1.8	Cloud config examples	22
1.9	Events and Updates	55
1.10	Modules	57
1.11	Merging User-Data Sections	105
1.12	Instance Metadata	109
1.13	Datasources	120
1.14	Vendor Data	151
1.15	Network Configuration	152
1.16	Contributing to cloud-init	174
1.17	Code Review Process	183
1.18	Security Policy	186
1.19	Debugging cloud-init	187
1.20	Logging	191
1.21	Directory layout	193
1.22	Analyze	195
1.23	Docs	200
1.24	Testing	201
1.25	Integration Testing	204
	Python Module Index	209
	Index	211

Cloud-init is the *industry standard* multi-distribution method for cross-platform cloud instance initialization. It is supported across all major public cloud providers, provisioning systems for private cloud infrastructure, and bare-metal installations.

Cloud instances are initialized from a disk image and instance data:

- Cloud metadata
- User data (optional)
- Vendor data (optional)

Cloud-init will identify the cloud it is running on during boot, read any provided metadata from the cloud and initialize the system accordingly. This may involve setting up the network and storage devices to configuring SSH access key and many other aspects of a system. Later on the cloud-init will also parse and process any optional user or vendor data that was passed to the instance.

Having trouble? We would like to help!

- Try the [FAQ](#) – its got answers to some common questions
- Ask a question in the `#cloud-init` IRC channel on Libera
- Join and ask questions on the [cloud-init mailing list](#)
- Find a bug? [Report bugs on Launchpad](#)

1.1 Availability

Below outlines the current availability of cloud-init across distributions and clouds, both public and private.

Note: If a distribution or cloud does not show up in the list below contact them and ask for images to be generated using cloud-init!

1.1.1 Distributions

Cloud-init has support across all major Linux distributions, FreeBSD, NetBSD, OpenBSD and DragonFlyBSD:

- Alpine Linux
- ArchLinux
- Debian
- DragonFlyBSD
- Fedora
- FreeBSD
- Gentoo Linux

- NetBSD
- OpenBSD
- Photon OS
- RHEL/CentOS/AlmaLinux/Rocky Linux/EuroLinux
- SLES/openSUSE
- Ubuntu

1.1.2 Clouds

Cloud-init provides support across a wide ranging list of execution environments in the public cloud:

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform
- Oracle Cloud Infrastructure
- Softlayer
- Rackspace Public Cloud
- IBM Cloud
- DigitalOcean
- Bigstep
- Hetzner
- Joyent
- CloudSigma
- Alibaba Cloud
- OVH
- OpenNebula
- Exoscale
- Scaleway
- CloudStack
- AltCloud
- SmartOS
- UpCloud
- Vultr
- Zadara Edge Cloud Platform

Additionally, cloud-init is supported on these private clouds:

- Bare metal installs
- OpenStack
- LXD

- KVM
- Metal-as-a-Service (MAAS)
- VMware

1.2 Boot Stages

In order to be able to provide the functionality that it does, cloud-init must be integrated into the boot in fairly controlled way. There are five stages to boot:

1. Generator
2. Local
3. Network
4. Config
5. Final

1.2.1 Generator

When booting under systemd, a [generator](#) will run that determines if cloud-init.target should be included in the boot goals. By default, this generator will enable cloud-init. It will not enable cloud-init if either:

- The file `/etc/cloud/cloud-init.disabled` exists
- The kernel command line as found in `/proc/cmdline` contains `cloud-init=disabled`. When running in a container, the kernel command line is not honored, but cloud-init will read an environment variable named `KERNEL_CMDLINE` in its place.

Again, these mechanisms for disabling cloud-init at runtime currently only exist in systemd.

1.2.2 Local

systemd service	<code>cloud-init-local.service</code>
runs	as soon as possible with <code>/</code> mounted read-write
blocks	as much of boot as possible, <i>must</i> block network
modules	none

The purpose of the local stage is to:

- locate “local” data sources.
- apply networking configuration to the system (including “Fallback”)

In most cases, this stage does not do much more than that. It finds the datasource and determines the network configuration to be used. That network configuration can come from:

- **datasource**: cloud provided network configuration via metadata
- **fallback**: cloud-init’s fallback networking consists of rendering the equivalent to “dhcp on eth0”, which was historically the most popular mechanism for network configuration of a guest
- **none**: network configuration can be disabled by writing the file `/etc/cloud/cloud.cfg` with the content:

```
network: {config: disabled}
```

If this is an instance's first boot, then the selected network configuration is rendered. This includes clearing of all previous (stale) configuration including persistent device naming with old mac addresses.

This stage must block network bring-up or any stale configuration that might have already been applied. Otherwise, that could have negative effects such as DHCP hooks or broadcast of an old hostname. It would also put the system in an odd state to recover from, as it may then have to restart network devices.

Cloud-init then exits and expects for the continued boot of the operating system to bring network configuration up as configured.

Note: In the past, local data sources have been only those that were available without network (such as 'ConfigDrive'). However, as seen in the recent additions to the DigitalOcean datasource, even data sources that require a network can operate at this stage.

1.2.3 Network

systemd service	<code>cloud-init.service</code>
runs	after local stage and configured networking is up
blocks	as much of remaining boot as possible
modules	<code>cloud_init_modules</code> in <code>/etc/cloud/cloud.cfg</code>

This stage requires all configured networking to be online, as it will fully process any user-data that is found. Here processing means:

- retrieve any `#include` or `#include-once` (recursively) including `http`
- decompress any compressed content
- run any part-handler found.

This stage runs the `disk_setup` and `mounts` modules which may partition and format disks and configure mount points (such as in `/etc/fstab`). Those modules cannot run earlier as they may receive configuration input from sources only available via network. For example, a user may have provided user-data in a network resource that describes how local mounts should be done.

On some clouds, such as Azure, this stage will create filesystems to be mounted, including ones that have stale (previous instance) references in `/etc/fstab`. As such, entries `/etc/fstab` other than those necessary for cloud-init to run should not be done until after this stage.

A part-handler will run at this stage, as will boot-hooks including `cloud-config bootcmd`. The user of this functionality has to be aware that the system is in the process of booting when their code runs.

1.2.4 Config

systemd service	<code>cloud-config.service</code>
runs	after network
blocks	nothing
modules	<code>cloud_config_modules</code> in <code>/etc/cloud/cloud.cfg</code>

This stage runs config modules only. Modules that do not really have an effect on other stages of boot are run here, including `runcmd`.

1.2.5 Final

systemd service	<code>cloud-final.service</code>
runs	as final part of boot (traditional “rc.local”)
blocks	nothing
modules	<code>cloud_final_modules</code> in <code>/etc/cloud/cloud.cfg</code>

This stage runs as late in boot as possible. Any scripts that a user is accustomed to running after logging into a system should run correctly here. Things that run here include:

- package installations
- configuration management plugins (puppet, chef, salt-minion)
- user-defined scripts (i.e. shell scripts passed as user-data)

For scripts external to cloud-init looking to wait until cloud-init is finished, the `cloud-init status` subcommand can help block external scripts until cloud-init is done without having to write your own systemd units dependency chains. See [status](#) for more info.

1.3 First Boot Determination

cloud-init has to determine whether or not the current boot is the first boot of a new instance or not, so that it applies the appropriate configuration. On an instance’s first boot, it should run all “per-instance” configuration, whereas on a subsequent boot it should run only “per-boot” configuration. This section describes how cloud-init performs this determination, as well as why it is necessary.

When it runs, cloud-init stores a cache of its internal state for use across stages and boots.

If this cache is present, then cloud-init has run on this system before.¹ There are two cases where this could occur. Most commonly, the instance has been rebooted, and this is a second/subsequent boot. Alternatively, the filesystem has been attached to a *new* instance, and this is an instance’s first boot. The most obvious case where this happens is when an instance is launched from an image captured from a launched instance.

By default, cloud-init attempts to determine which case it is running in by checking the instance ID in the cache against the instance ID it determines at runtime. If they do not match, then this is an instance’s first boot; otherwise, it’s a subsequent boot. Internally, cloud-init refers to this behavior as `check`.

This behavior is required for images captured from launched instances to behave correctly, and so is the default which generic cloud images ship with. However, there are cases where it can cause problems.² For these cases, cloud-init has support for modifying its behavior to trust the instance ID that is present in the system unconditionally. This means that cloud-init will never detect a new instance when the cache is present, and it follows that the only way to cause cloud-init to detect a new instance (and therefore its first boot) is to manually remove cloud-init’s cache. Internally, this behavior is referred to as `trust`.

¹ It follows that if this cache is not present, cloud-init has not run on this system before, so this is unambiguously this instance’s first boot.

² A couple of ways in which this strict reliance on the presence of a datasource has been observed to cause problems:

- If a cloud’s metadata service is flaky and cloud-init cannot obtain the instance ID locally on that platform, cloud-init’s instance ID determination will sometimes fail to determine the current instance ID, which makes it impossible to determine if this is an instance’s first or subsequent boot (#1885527).
- If cloud-init is used to provision a physical appliance or device and an attacker can present a datasource to the device with a different instance ID, then cloud-init’s default behavior will detect this as an instance’s first boot and reset the device using the attacker’s configuration (this has been observed with the NoCloud datasource in #1879530).

To configure which of these behaviors to use, cloud-init exposes the `manual_cache_clean` configuration option. When `false` (the default), cloud-init will check and clean the cache if the instance IDs do not match (this is the default, as discussed above). When `true`, cloud-init will trust the existing cache (and therefore not clean it).

1.3.1 Manual Cache Cleaning

cloud-init ships a command for manually cleaning the cache: `cloud-init clean`. See [clean](#)'s documentation for further details.

1.3.2 Reverting `manual_cache_clean` Setting

Currently there is no support for switching an instance that is launched with `manual_cache_clean: true` from `trust` behavior to `check` behavior, other than manually cleaning the cache.

Warning: If you want to capture an instance that is currently in `trust` mode as an image for launching other instances, you **must** manually clean the cache. If you do not do so, then instances launched from the captured image will all detect their first boot as a subsequent boot of the captured instance, and will not apply any per-instance configuration.

This is a functional issue, but also a potential security one: cloud-init is responsible for rotating SSH host keys on first boot, and this will not happen on these instances.

1.4 CLI Interface

For the latest list of subcommands and arguments use cloud-init's `--help` option. This can be used against cloud-init itself or any of its subcommands.

```
$ cloud-init --help
usage: /usr/bin/cloud-init [-h] [--version] [--file FILES] [--debug] [--force]
                           {init,modules,single,query,dhclient-hook,features,analyze,
➤ devel,collect-logs,clean,status}
                           ...

optional arguments:
-h, --help            show this help message and exit
--version, -v         show program's version number and exit
--file FILES, -f FILES
                       additional yaml configuration files to use
--debug, -d          show additional pre-action logging (default: False)
--force              force running even if no datasource is found (use at
                       your own risk)

Subcommands:
{init,modules,single,query,dhclient-hook,features,analyze,devel,collect-logs,clean,
➤ status}
  init                initializes cloud-init and performs initial modules
  modules             activates modules using a given configuration key
  single              run a single module
  query               Query standardized instance metadata from the command
                       line.
  dhclient-hook       Run the dhclient hook to record network info.
```

(continues on next page)

(continued from previous page)

features	list defined features
analyze	Devel tool: Analyze cloud-init logs and data
devel	Run development tools
collect-logs	Collect and tar all cloud-init debug info
clean	Remove logs and artifacts so cloud-init can re-run.
status	Report cloud-init status or wait on completion.

The rest of this document will give an overview of each of the subcommands.

1.4.1 analyze

Get detailed reports of where cloud-init spends its time during the boot process. For more complete reference see [analyze](#).

Possible subcommands include:

- *blame*: report ordered by most costly operations
- *dump*: machine-readable JSON dump of all cloud-init tracked events
- *show*: show time-ordered report of the cost of operations during each boot stage
- *boot*: show timestamps from kernel initialization, kernel finish initialization, and cloud-init start

1.4.2 clean

Remove cloud-init artifacts from `/var/lib/cloud` to simulate a clean instance. On reboot, cloud-init will re-run all stages as it did on first boot.

- *-logs*: optionally remove all cloud-init log files in `/var/log/`
- *-reboot*: reboot the system after removing artifacts

1.4.3 collect-logs

Collect and tar cloud-init generated logs, data files, and system information for triage. This subcommand is integrated with `apport`.

Logs collected include:

- `/var/log/cloud-init.log`
- `/var/log/cloud-init-output.log`
- `/run/cloud-init`
- `/var/lib/cloud/instance/user-data.txt`
- cloud-init package version
- `dmesg` output
- `journalctl` output

Note: Ubuntu users can file bugs with `ubuntu-bug cloud-init` to automatically attach these logs to a bug report

1.4.4 devel

Collection of development tools under active development. These tools will likely be promoted to top-level subcommands when stable.

Do **NOT** rely on the output of these commands as they can and will change.

Current subcommands:

- `net-convert`: manually use cloud-init's network format conversion, useful for testing configuration or testing changes to the network conversion logic itself.
- `render`: use cloud-init's jinja template render to process **#cloud-config** or **custom-scripts**, injecting any variables from `/run/cloud-init/instance-data.json`. It accepts a user-data file containing the jinja template header `## template: jinja` and renders that content with any instance-data.json variables present.
- `schema`: a **#cloud-config** format and schema validator. It accepts a cloud-config YAML file and annotates potential schema errors locally without the need for deployment. Schema validation is work in progress and supports a subset of cloud-config modules.
- `hotplug-hook`: respond to newly added system devices by retrieving updated system metadata and bringing up/down the corresponding device. This command is intended to be called via a systemd service and is not considered user-accessible except for debugging purposes.

1.4.5 features

Print out each feature supported. If cloud-init does not have the features subcommand, it also does not support any features described in this document.

```
$ cloud-init features
NETWORK_CONFIG_V1
NETWORK_CONFIG_V2
```

1.4.6 init

Generally run by OS init systems to execute cloud-init's stages *init* and *init-local*. See [Boot Stages](#) for more info. Can be run on the commandline, but is generally gated to run only once due to semaphores in `/var/lib/cloud/instance/sem/` and `/var/lib/cloud/sem`.

- `-local`: run *init-local* stage instead of *init*

1.4.7 modules

Generally run by OS init systems to execute *modules:config* and *modules:final* boot stages. This executes cloud config [Modules](#) configured to run in the *init*, *config* and *final* stages. The modules are declared to run in various boot stages in the file `/etc/cloud/cloud.cfg` under keys:

- `cloud_init_modules`
- `cloud_config_modules`
- `cloud_final_modules`

Can be run on the command line, but each module is gated to run only once due to semaphores in `/var/lib/cloud/`.

- `--mode [init|config|final]`: run `modules:init`, `modules:config` or `modules:final` cloud-init stages. See [Boot Stages](#) for more info.

1.4.8 query

Query standardized cloud instance metadata crawled by cloud-init and stored in `/run/cloud-init/instance-data.json`. This is a convenience command-line interface to reference any cached configuration metadata that cloud-init crawls when booting the instance. See [Instance Metadata](#) for more info.

- `--all`: dump all available instance data as json which can be queried
- `--instance-data`: optional path to a different instance-data.json file to source for queries
- `--list-keys`: list available query keys from cached instance data
- `--format`: a string that will use jinja-template syntax to render a string replacing
- `<varname>`: a dot-delimited variable path into the instance-data.json object

Below demonstrates how to list all top-level query keys that are standardized aliases:

```
$ cloud-init query --list-keys
_beta_keys
availability_zone
base64_encoded_keys
cloud_name
ds
instance_id
local_hostname
platform
public_ssh_keys
region
sensitive_keys
subplatform
userdata
v1
vendordata
```

Below demonstrates how to query standardized metadata from clouds:

```
% cloud-init query v1.cloud_name
aws # or openstack, azure, gce etc.

# Any standardized instance-data under a <v#> key is aliased as a top-level key for_
↳convenience.
% cloud-init query cloud_name
aws # or openstack, azure, gce etc.

# Query datasource-specific metadata on EC2
% cloud-init query ds.meta_data.public_ipv4
```

Note: The standardized instance data keys under `v#` are guaranteed not to change behavior or format. If using top-level convenience aliases for any standardized instance data keys, the most value (highest `v#`) of that key name is what is reported as the top-level value. So these aliases act as a ‘latest’.

This data can then be formatted to generate custom strings or data:

```
# Generate a custom hostname fqdn based on instance-id, cloud and region
% cloud-init query --format 'custom-{{instance_id}}.{{region}}.{{v1.cloud_name}}.com'
custom-i-0e91f69987f37ec74.us-east-2.aws.com
```

1.4.9 single

Attempt to run a single named cloud config module.

- *--name*: the cloud-config module name to run
- *--frequency*: optionally override the declared module frequency with one of (always|once-per-instance|once)

The following example re-runs the `cc_set_hostname` module ignoring the module default frequency of once-per-instance:

```
$ cloud-init single --name set_hostname --frequency always
```

Note: Mileage may vary trying to re-run each cloud-config module, as some are not idempotent.

1.4.10 status

Report whether cloud-init is running, done, disabled or errored. Exits non-zero if an error is detected in cloud-init.

- *--long*: detailed status information
- *--wait*: block until cloud-init completes

Below are examples of output when cloud-init is running, showing status and the currently running modules, as well as when it is done.

```
$ cloud-init status
status: running

$ cloud-init status --long
status: running
time: Fri, 26 Jan 2018 21:39:43 +0000
detail:
Running in stage: init-local

$ cloud-init status
status: done

$ cloud-init status --long
status: done
time: Wed, 17 Jan 2018 20:41:59 +0000
detail:
DataSourceNoCloud [seed=/var/lib/cloud/seed/nocloud-net][dsmode=net]
```


1.5 FAQ

1.5.1 How do I get help?

Having trouble? We would like to help!

- First go through this page with answers to common questions
- Use the search bar at the upper left to search these docs
- Ask a question in the `#cloud-init` IRC channel on Libera
- Join and ask questions on the [cloud-init mailing list](#)
- Find a bug? Check out the [Reporting Bugs](#) topic for how to report one

1.5.2 Where are the logs?

Cloud-init uses two files to log to:

- `/var/log/cloud-init-output.log`: captures the output from each stage of cloud-init when it runs
- `/var/log/cloud-init.log`: very detailed log with debugging output, detailing each action taken
- `/run/cloud-init`: contains logs about how cloud-init decided to enable or disable itself, as well as what platforms/datasources were detected. These logs are most useful when trying to determine what cloud-init ran or did not run.

Be aware that each time a system boots, new logs are appended to the files in `/var/log`. Therefore, the files may have more than one boot worth of information present.

When reviewing these logs look for any errors or Python tracebacks to check for any errors.

1.5.3 Where are the configuration files?

Cloud-init config is provided in two places:

- `/etc/cloud/cloud.cfg`
- `/etc/cloud/cloud.cfg.d/*.cfg`

These files can define the modules that run during instance initialization, the datasources to evaluate on boot, and other settings.

1.5.4 Where are the data files?

Inside the `/var/lib/cloud/` directory there are two important subdirectories:

instance

The `/var/lib/cloud/instance` directory is a symbolic link that points to the most recently used instance-id directory. This folder contains the information cloud-init received from datasources, including vendor and user data. This can be helpful to review to ensure the correct data was passed.

It also contains the `datasource` file that contains the full information about what datasource was identified and used to setup the system.

Finally, the *boot-finished* file is the last thing that cloud-init does.

data

The `/var/lib/cloud/data` directory contains information related to the previous boot:

- *instance-id*: id of the instance as discovered by cloud-init. Changing this file has no effect.
- *result.json*: json file will show both the datasource used to setup the instance, and if any errors occurred
- *status.json*: json file shows the datasource used and a break down of all four modules if any errors occurred and the start and stop times.

1.5.5 What datasource am I using?

To correctly setup an instance, cloud-init must correctly identify the cloud that it is on. Therefore knowing what datasource is used on an instance launch can help aid in debugging.

To find what datasource is getting used run the *cloud-id* command:

```
$ cloud-id
nocloud
```

If the cloud-id is not what is expected, then running the *ds-identify* script in debug mode and providing that in a bug can help aid in resolving any issues:

```
$ sudo DEBUG_LEVEL=2 DI_LOG=stderr /usr/lib/cloud-init/ds-identify --force
```

The force parameter allows the command to be run again since the instance has already launched. The other options increase the verbosity of logging and put the logs to STDERR.

1.5.6 How can I re-run datasource detection and cloud-init?

If a user is developing a new datasource or working on debugging an issue it may be useful to re-run datasource detection and the initial setup of cloud-init.

To do this, force ds-identify to re-run, clean up any logs, and re-run cloud-init:

```
$ sudo DI_LOG=stderr /usr/lib/cloud-init/ds-identify --force
$ sudo cloud-init clean --logs
$ sudo cloud-init init --local
$ sudo cloud-init init
```

Warning: These commands will re-run cloud-init as if this were first boot of a system: this will, at the very least, cycle SSH host keys and may do substantially more. Do not run these commands on production systems.

1.5.7 How can I debug my user data?

Two of the most common issues with user data, that also happens to be cloud-config is:

1. Incorrectly formatted YAML
2. First line does not contain *#cloud-config*

To verify your YAML, we do have a short script called `validate-yaml.py` that can validate your user data offline.

Another option is to run the following on an instance to debug userdata provided to the system:

```
$ cloud-init devel schema --system --annotate
```

As launching instances in the cloud can cost money and take a bit longer, sometimes it is easier to launch instances locally using Multipass or LXD:

Multipass

`Multipass` is a cross-platform tool to launch Ubuntu VMs across Linux, Windows, and macOS.

When a user launches a Multipass VM, user data can be passed by adding the `-cloud-init` flag and the appropriate YAML file containing user data:

```
$ multipass launch bionic --name test-vm --cloud-init userdata.yaml
```

Multipass will validate the YAML syntax of the cloud-config file before attempting to start the VM! A nice addition to help save time when experimenting with launching instances with various cloud-configs.

Multipass only supports passing user-data and only as YAML cloud-config files. Passing a script, a MIME archive, or any of the other user-data formats cloud-init supports will result in an error from the YAML syntax validator.

LXD

`LXD` offers a streamlined user experience for using linux system containers. With LXD, a user can pass:

- user data
- vendor data
- metadata
- network configuration

The following initializes a container with user data:

```
$ lxc init ubuntu-daily:bionic test-container
$ lxc config set test-container user.user-data - < userdata.yaml
$ lxc start test-container
```

To avoid the extra commands this can also be done at launch:

```
$ lxc launch ubuntu-daily:bionic test-container --config=user.user-data="$(cat ↵
↵userdata.yaml)"
```

Finally, a profile can be setup with the specific data if a user needs to launch this multiple times:

```
$ lxc profile create dev-user-data
$ lxc profile set dev-user-data user.user-data - < cloud-init-config.yaml
$ lxc launch ubuntu-daily:bionic test-container -p default -p dev-user-data
```

The above examples all show how to pass user data. To pass other types of configuration data use the config option specified below:

Data	Config Option
user data	user.user-data
vendor data	user.vendor-data
metadata	user.meta-data
network config	user.network-config

See the [LXD Instance Configuration](#) docs for more info about configuration values or the [LXD Custom Network Configuration](#) document for more about custom network config.

cloud-localds

The *cloud-localds* command from the [cloud-utils](#) package generates a disk with user supplied data. The NoCloud datasource allows users to provide their own user data, metadata, or network configuration directly to an instance without running a network service. This is helpful for launching local cloud images with QEMU for example.

The following is an example of creating the local disk using the *cloud-localds* command:

```
$ cat >user-data <<EOF
#cloud-config
password: password
chpasswd:
  expire: False
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAAA...UlIsqdaO+w==
EOF
$ cloud-localds seed.img user-data
```

The resulting *seed.img* can then get passed along to a cloud image containing cloud-init. Below is an example of passing the *seed.img* with QEMU:

```
$ qemu-system-x86_64 -m 1024 -net nic -net user \
  -hda ubuntu-20.04-server-cloudimg-amd64.img \
  -hdb seed.img
```

The now booted image will allow for login using the password provided above.

For additional configuration, users can provide much more detailed configuration, including network configuration and metadata:

```
$ cloud-localds --network-config=network-config-v2.yaml \
  seed.img userdata.yaml metadata.yaml
```

See the [Networking Config Version 2](#) page for details on the format and config of network configuration. To learn more about the possible values for metadata, check out the [NoCloud](#) page.

1.5.8 Where can I learn more?

Below are some videos, blog posts, and white papers about cloud-init from a variety of sources.

- [cloud-init - The Good Parts](#)
- [cloud-init Summit 2019](#)
- [Utilising cloud-init on Microsoft Azure \(Whitepaper\)](#)

- [Cloud Instance Initialization with cloud-init \(Whitepaper\)](#)
- [cloud-init Summit 2018](#)
- [cloud-init - The cross-cloud Magic Sauce \(PDF\)](#)
- [cloud-init Summit 2017](#)
- [cloud-init - Building clouds one Linux box at a time \(Video\)](#)
- [cloud-init - Building clouds one Linux box at a time \(PDF\)](#)
- [Metadata and cloud-init](#)
- [The beauty of cloud-init](#)
- [Introduction to cloud-init](#)

1.6 Reporting Bugs

The following documents:

- 1) [How to collect information for reporting bugs](#)
- 2) [How to file bugs to the upstream cloud-init project or for distro specific packages](#)

1.6.1 Collect Logs

To aid in debugging, please collect the necessary logs. To do so, run the *collect-logs* subcommand to produce a tarfile that you can easily upload:

```
$ sudo cloud-init collect-logs
Wrote /home/ubuntu/cloud-init.tar.gz
```

If your version of cloud-init does not have the *collect-logs* subcommand, then please manually collect the base log files by doing the following:

```
$ sudo dmesg > dmesg.txt
$ sudo journalctl -o short-precise > journal.txt
$ sudo tar -cvf cloud-init.tar dmesg.txt journal.txt /run/cloud-init \
/var/log/cloud-init.log /var/log/cloud-init-output.log
```

1.6.2 Report Upstream Bug

Bugs for upstream cloud-init are tracked using Launchpad. To file a bug:

1. Collect the necessary debug logs as described above
2. [Create a Launchpad account](#) or login to your existing account
3. [Report an upstream cloud-init bug](#)

If debug logs are not provided, you will be asked for them before any further time is spent debugging. If you are unable to obtain the required logs please explain why in the bug.

If your bug is for a specific distro using cloud-init, please first consider reporting it with the upstream distro or confirm that it still occurs with the latest upstream cloud-init code. See below for details on specific distro reporting.

1.6.3 Distro Specific Issues

For issues specific to your distro please use one of the following distro specific reporting mechanisms:

Ubuntu

To report a bug on Ubuntu use the *ubuntu-bug* command on the affected system to automatically collect the necessary logs and file a bug on Launchpad:

```
$ ubuntu-bug cloud-init
```

If that does not work or is not an option, please collect the logs using the commands in the above Collect Logs section and then report the bug on the [Ubuntu bug tracker](#). Make sure to attach your collected logs!

Debian

To file a bug against the Debian package for cloud-init please use the [Debian bug tracker](#) to file against ‘Package: cloud-init’. See the [Debian bug reporting wiki](#) page for more details.

Red Hat, CentOS, & Fedora

To file a bug against the Red Hat or Fedora packages of cloud-init please use the [Red Hat bugzilla](#).

SUSE & openSUSE

To file a bug against the SuSE packages of cloud-init please use the [SUSE bugzilla](#).

Arch

To file a bug against the Arch package of cloud-init please use the [Arch Linux Bugtracker](#). See the [Arch bug reporting wiki](#) for more details.

1.7 User-Data Formats

User data that will be acted upon by cloud-init must be in one of the following types.

1.7.1 Gzip Compressed Content

Content found to be gzip compressed will be uncompressed. The uncompressed data will then be used as if it were not compressed. This is typically useful because user-data is limited to ~16384¹ bytes.

¹ See your cloud provider for applicable user-data size limitations...

1.7.2 Mime Multi Part Archive

This list of rules is applied to each part of this multi-part file. Using a mime-multi part file, the user can specify more than one type of data.

For example, both a user data script and a cloud-config type could be specified.

Supported content-types are listed from the cloud-init subcommand make-mime:

```
$ cloud-init devel make-mime --list-types
cloud-boothook
cloud-config
cloud-config-archive
cloud-config-jsonp
jinja2
part-handler
upstart-job
x-include-once-url
x-include-url
x-shellscript
x-shellscript-per-boot
x-shellscript-per-instance
x-shellscript-per-once
```

Helper subcommand to generate mime messages

The cloud-init subcommand can generate MIME multi-part files: [make-mime](#).

make-mime subcommand takes pairs of (filename, “text/” mime subtype) separated by a colon (e.g. config.yaml:cloud-config) and emits a MIME multipart message to stdout.

Examples

Create userdata containing both a cloud-config (config.yaml) and a shell script (script.sh)

```
$ cloud-init devel make-mime -a config.yaml:cloud-config -a script.sh:x-shellscript >_
↪userdata
```

Create userdata containing 3 shell scripts:

- always.sh - Run every boot
- instance.sh - Run once per instance
- once.sh - Run once

```
$ cloud-init devel make-mime -a always.sh:x-shellscript-per-boot -a instance.sh:x-
↪shellscript-per-instance -a once.sh:x-shellscript-per-once
```

1.7.3 User-Data Script

Typically used by those who just want to execute a shell script.

Begins with: `#!` or `Content-Type: text/x-shellscript` when using a MIME archive.

Note: New in cloud-init v. 18.4: User-data scripts can also render cloud instance metadata variables using jinja templating. See [Instance Metadata](#) for more information.

Example

```
$ cat myscript.sh

#!/bin/sh
echo "Hello World.  The time is now $(date -R)!" | tee /root/output.txt

$ euca-run-instances --key mykey --user-data-file myscript.sh ami-a07d95c9
```

1.7.4 Include File

This content is a `include` file.

The file contains a list of urls, one per line. Each of the URLs will be read, and their content will be passed through this same set of rules. I.e., the content read from the URL can be gzipped, mime-multi-part, or plain text. If an error occurs reading a file the remaining files will not be read.

Begins with: `#include` or `Content-Type: text/x-include-url` when using a MIME archive.

1.7.5 Cloud Config Data

Cloud-config is the simplest way to accomplish some things via user-data. Using cloud-config syntax, the user can specify certain things in a human friendly format.

These things include:

- apt upgrade should be run on first boot
- a different apt mirror should be used
- additional apt sources should be added
- certain SSH keys should be imported
- *and many more...*

Note: This file must be valid YAML syntax.

See the [Cloud config examples](#) section for a commented set of examples of supported cloud config formats.

Begins with: `#cloud-config` or `Content-Type: text/cloud-config` when using a MIME archive.

Note: New in cloud-init v. 18.4: Cloud config data can also render cloud instance metadata variables using jinja templating. See [Instance Metadata](#) for more information.

1.7.6 Upstart Job

Content is placed into a file in `/etc/init`, and will be consumed by upstart as any other upstart job.

Begins with: `#upstart-job` or `Content-Type: text/upstart-job` when using a MIME archive.

1.7.7 Cloud Boothook

This content is boothook data. It is stored in a file under `/var/lib/cloud` and then executed immediately. This is the earliest hook available. Note, that there is no mechanism provided for running only once. The boothook must take care of this itself.

It is provided with the instance id in the environment variable `INSTANCE_ID`. This could be made use of to provide a 'once-per-instance' type of functionality.

Begins with: `#cloud-boothook` or `Content-Type: text/cloud-boothook` when using a MIME archive.

1.7.8 Part Handler

This is a part-handler: It contains custom code for either supporting new mime-types in multi-part user data, or overriding the existing handlers for supported mime-types. It will be written to a file in `/var/lib/cloud/data` based on its filename (which is generated).

This must be python code that contains a `list_types` function and a `handle_part` function. Once the section is read the `list_types` method will be called. It must return a list of mime-types that this part-handler handles. Because mime parts are processed in order, a part-handler part must precede any parts with mime-types it is expected to handle in the same user data.

The `handle_part` function must be defined like:

```
def handle_part(data, ctype, filename, payload):
    # data = the cloudinit object
    # ctype = "__begin__", "__end__", or the mime-type of the part that is being_
    ↪handled.
    # filename = the filename of the part (or a generated filename if none is present_
    ↪in mime data)
    # payload = the parts' content
```

Cloud-init will then call the `handle_part` function once before it handles any parts, once per part received, and once after all parts have been handled. The `'__begin__'` and `'__end__'` sentinels allow the part handler to do initialization or teardown before or after receiving any parts.

Begins with: `#part-handler` or `Content-Type: text/part-handler` when using a MIME archive.

Example

```
1 #part-handler
2
3 def list_types():
4     # return a list of mime-types that are handled by this module
5     return(["text/plain", "text/go-cubs-go"])
6
7 def handle_part(data, ctype, filename, payload):
8     # data: the cloudinit object
```

(continues on next page)

(continued from previous page)

```
9  # ctype: '__begin__', '__end__', or the specific mime-type of the part
10 # filename: the filename for the part, or dynamically generated part if
11 #           no filename is given attribute is present
12 # payload: the content of the part (empty for begin or end)
13 if ctype == "__begin__":
14     print "my handler is beginning"
15     return
16 if ctype == "__end__":
17     print "my handler is ending"
18     return
19
20 print "==== received ctype=%s filename=%s ===" % (ctype, filename)
21 print payload
22 print "==== end ctype=%s filename=%s" % (ctype, filename)
```

Also this [blog](#) post offers another example for more advanced usage.

1.7.9 Kernel Command Line

When using the *NoCloud* datasource, users can pass user data via the kernel command line parameters. See the *NoCloud* datasource documentation for more details.

1.7.10 Disabling User-Data

Cloud-init can be configured to ignore any user-data provided to instance. This allows custom images to prevent users from accidentally breaking closed appliances. Setting `allow_userdata: false` in the configuration will disable cloud-init from processing user-data.

1.8 Cloud config examples

1.8.1 Including users and groups

```
1  #cloud-config
2  # Add groups to the system
3  # The following example adds the ubuntu group with members 'root' and 'sys'
4  # and the empty group cloud-users.
5  groups:
6  - ubuntu: [root,sys]
7  - cloud-users
8
9  # Add users to the system. Users are added after groups are added.
10 # Note: Most of these configuration options will not be honored if the user
11 #       already exists. Following options are the exceptions and they are
12 #       applicable on already-existing users:
13 #       - 'plain_text_passwd', 'hashed_passwd', 'lock_passwd', 'sudo',
14 #       'ssh_authorized_keys', 'ssh_redirect_user'.
15 users:
16 - default
17 - name: foobar
18   gecos: Foo B. Bar
19   primary_group: foobar
```

(continues on next page)

(continued from previous page)

```

20 groups: users
21 selinux_user: staff_u
22 expiredate: '2032-09-01'
23 ssh_import_id:
24     - lp:falcojr
25     - gh:TheRealFalcon
26 lock_passwd: false
27 passwd: $6$j212wezy$7H/1LT4f9/
    ↪N3wpgNunhsIqtMj62OKiS3nyNwuizouQc3u7MbYCarYeAHWYPYb2FT.1bioDm2RrkJPb9BZMN1O/
28 - name: barfoo
29   gecos: Bar B. Foo
30   sudo: ALL=(ALL) NOPASSWD:ALL
31   groups: users, admin
32   ssh_import_id:
33       - lp:falcojr
34       - gh:TheRealFalcon
35   lock_passwd: true
36   ssh_authorized_keys:
37       - <ssh pub key 1>
38       - <ssh pub key 2>
39 - name: cloudy
40   gecos: Magic Cloud App Daemon User
41   inactive: '5'
42   system: true
43 - name: fizzbuzz
44   sudo: False
45   ssh_authorized_keys:
46       - <ssh pub key 1>
47       - <ssh pub key 2>
48 - snapuser: joe@joeuser.io
49 - name: nosshlogins
50   ssh_redirect_user: true
51
52 # Valid Values:
53 #   name: The user's login name
54 #   expiredate: Date on which the user's account will be disabled.
55 #   gecos: The user name's real name, i.e. "Bob B. Smith"
56 #   homedir: Optional. Set to the local path you want to use. Defaults to
57 #             /home/<username>
58 #   primary_group: define the primary group. Defaults to a new group created
59 #                   named after the user.
60 #   groups: Optional. Additional groups to add the user to. Defaults to none
61 #   selinux_user: Optional. The SELinux user for the user's login, such as
62 #                   "staff_u". When this is omitted the system will select the default
63 #                   SELinux user.
64 #   lock_passwd: Defaults to true. Lock the password to disable password login
65 #   inactive: Number of days after password expires until account is disabled
66 #   passwd: The hash -- not the password itself -- of the password you want
67 #             to use for this user. You can generate a safe hash via:
68 #             mkpasswd --method=SHA-512 --rounds=4096
69 #             (the above command would create from stdin an SHA-512 password hash
70 #             with 4096 salt rounds)
71 #
72 #   Please note: while the use of a hashed password is better than
73 #               plain text, the use of this feature is not ideal. Also,
74 #               using a high number of salting rounds will help, but it should
75 #               not be relied upon.

```

(continues on next page)

(continued from previous page)

```

76 #
77 #         To highlight this risk, running John the Ripper against the
78 #         example hash above, with a readily available wordlist, revealed
79 #         the true password in 12 seconds on a i7-2620QM.
80 #
81 #         In other words, this feature is a potential security risk and is
82 #         provided for your convenience only. If you do not fully trust the
83 #         medium over which your cloud-config will be transmitted, then you
84 #         should use SSH authentication only.
85 #
86 #         You have thus been warned.
87 # no_create_home: When set to true, do not create home directory.
88 # no_user_group: When set to true, do not create a group named after the user.
89 # no_log_init: When set to true, do not initialize lastlog and faillog database.
90 # ssh_import_id: Optional. Import SSH ids
91 # ssh_authorized_keys: Optional. [list] Add keys to user's authorized keys file
92 # ssh_redirect_user: Optional. [bool] Set true to block ssh logins for cloud
93 #     ssh public keys and emit a message redirecting logins to
94 #     use <default_username> instead. This option only disables cloud
95 #     provided public-keys. An error will be raised if ssh_authorized_keys
96 #     or ssh_import_id is provided for the same user.
97 #
98 # sudo: Defaults to none. Accepts a sudo rule string, a list of sudo rule
99 #     strings or False to explicitly deny sudo usage. Examples:
100 #
101 #     Allow a user unrestricted sudo access.
102 #     sudo: ALL=(ALL) NOPASSWD:ALL
103 #
104 #     Adding multiple sudo rule strings.
105 #     sudo:
106 #         - ALL=(ALL) NOPASSWD:/bin/mysql
107 #         - ALL=(ALL) ALL
108 #
109 #     Prevent sudo access for a user.
110 #     sudo: False
111 #
112 #     Note: Please double check your syntax and make sure it is valid.
113 #     cloud-init does not parse/check the syntax of the sudo
114 #     directive.
115 # system: Create the user as a system user. This means no home directory.
116 # snapuser: Create a Snappy (Ubuntu-Core) user via the snap create-user
117 #     command available on Ubuntu systems. If the user has an account
118 #     on the Ubuntu SSO, specifying the email will allow snap to
119 #     request a username and any public ssh keys and will import
120 #     these into the system with username specified by SSO account.
121 #     If 'username' is not set in SSO, then username will be the
122 #     shortname before the email domain.
123 #
124 #
125 # Default user creation:
126 #
127 # Unless you define users, you will get a 'ubuntu' user on Ubuntu systems with the
128 # legacy permission (no password sudo, locked user, etc). If however, you want
129 # to have the 'ubuntu' user in addition to other users, you need to instruct
130 # cloud-init that you also want the default user. To do this use the following
131 # syntax:
132 #     users:

```

(continues on next page)

(continued from previous page)

```

133 #     - default
134 #     - bob
135 #     - ....
136 # foobar: ...
137 #
138 # users[0] (the first user in users) overrides the user directive.
139 #
140 # The 'default' user above references the distro's config:
141 # system_info:
142 #   default_user:
143 #     name: Ubuntu
144 #     plain_text_passwd: 'ubuntu'
145 #     home: /home/ubuntu
146 #     shell: /bin/bash
147 #     lock_passwd: True
148 #     geccos: Ubuntu
149 #     groups: [adm, audio, cdrom, dialout, floppy, video, plugdev, dip, netdev]

```

1.8.2 Writing out arbitrary files

```

1 #cloud-config
2 # vim: syntax=yaml
3 #
4 # This is the configuration syntax that the write_files module
5 # will know how to understand. Encoding can be given b64 or gzip or (gz+b64).
6 # The content will be decoded accordingly and then written to the path that is
7 # provided.
8 #
9 # Note: Content strings here are truncated for example purposes.
10 write_files:
11 - encoding: b64
12   content: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHROZSBzdGF0ZSBvZiBTRUxpbnV4...
13   owner: root:root
14   path: /etc/sysconfig/selinux
15   permissions: '0644'
16 - content: |
17     # My new /etc/sysconfig/samba file
18
19     SMBDOPTIONS="-D"
20   path: /etc/sysconfig/samba
21 - content: !!binary |
22     f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAwAAAAAAAAABAAAAAAAAAAJAVAAAAAAAAAAAAEAAOAI
23     AEAAHgAdAAAYAAAAFAAAQAAAAAAAAABAAEAAAAAAAAEAAQAAAAAAwAEAAAAAADAAQAAAAAAAgA
24     AAAAAAAAAAwAAAAQAAAAAAgAAAAAAAAACQAAAAAAAAAJAAAAAAAAAcAAAAAAAAABwAAAAAAAAAQAA
25     ....
26   path: /bin/arch
27   permissions: '0555'
28 - encoding: gzip
29   content: !!binary |
30     H4sIAIDb/U8C/1NW1E/KzNMvzuBKtc7IV8hIzcnJVyjpL8pJ4QIA6N+MVxsAAAA=
31   path: /usr/bin/hello
32   permissions: '0755'

```

1.8.3 Adding a yum repository

```
1 #cloud-config
2 # vim: syntax=yaml
3 #
4 # Add yum repository configuration to the system
5 #
6 # The following example adds the file /etc/yum.repos.d/epel_testing.repo
7 # which can then subsequently be used by yum for later operations.
8 yum_repos:
9   # The name of the repository
10   epel-testing:
11     # Any repository configuration options
12     # See: man yum.conf
13     #
14     # This one is required!
15     baseurl: http://download.fedoraproject.org/pub/epel/testing/5/$basearch
16     enabled: false
17     failovermethod: priority
18     gpgcheck: true
19     gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL
20     name: Extra Packages for Enterprise Linux 5 - Testing
```

1.8.4 Configure an instances trusted CA certificates

```
1 #cloud-config
2 #
3 # This is an example file to configure an instance's trusted CA certificates
4 # system-wide for SSL/TLS trust establishment when the instance boots for the
5 # first time.
6 #
7 # Make sure that this file is valid yaml before starting instances.
8 # It should be passed as user-data when starting the instance.
9
10 ca_certs:
11   # If present and set to True, the 'remove_defaults' parameter will remove
12   # all the default trusted CA certificates that are normally shipped with
13   # Ubuntu.
14   # This is mainly for paranoid admins - most users will not need this
15   # functionality.
16   remove_defaults: true
17
18   # If present, the 'trusted' parameter should contain a certificate (or list
19   # of certificates) to add to the system as trusted CA certificates.
20   # Pay close attention to the YAML multiline list syntax. The example shown
21   # here is for a list of multiline certificates.
22   trusted:
23     - |
24       -----BEGIN CERTIFICATE-----
25       YOUR-ORGS-TRUSTED-CA-CERT-HERE
26       -----END CERTIFICATE-----
27     - |
28       -----BEGIN CERTIFICATE-----
29       YOUR-ORGS-TRUSTED-CA-CERT-HERE
30       -----END CERTIFICATE-----
```

1.8.5 Configure an instances resolv.conf

Note: when using a config drive and a RHEL like system resolv.conf will also be managed ‘automatically’ due to the available information provided for dns servers in the config drive network format. For those that wish to have different settings use this module.

```

1 #cloud-config
2 #
3 # This is an example file to automatically configure resolv.conf when the
4 # instance boots for the first time.
5 #
6 # Ensure that your yaml is valid and pass this as user-data when starting
7 # the instance. Also be sure that your cloud.cfg file includes this
8 # configuration module in the appropriate section.
9 #
10 manage_resolv_conf: true
11
12 resolv_conf:
13   nameservers: ['8.8.4.4', '8.8.8.8']
14   searchdomains:
15     - foo.example.com
16     - bar.example.com
17   domain: example.com
18   options:
19     rotate: true
20     timeout: 1

```

1.8.6 Install and run chef recipes

```

1 #cloud-config
2 #
3 # This is an example file to automatically install chef-client and run a
4 # list of recipes when the instance boots for the first time.
5 # Make sure that this file is valid yaml before starting instances.
6 # It should be passed as user-data when starting the instance.
7
8 # The default is to install from packages.
9
10 # Key from https://packages.chef.io/chef.asc
11 apt:
12   sources:
13     source1:
14       source: "deb http://packages.chef.io/repos/apt/stable $RELEASE main"
15       key: |
16         -----BEGIN PGP PUBLIC KEY BLOCK-----
17         Version: GnuPG v1.4.12 (Darwin)
18         Comment: GPGTools - http://gpgtools.org
19
20         mQGIBepPC7QRBADfsOkZU6KZK+YmKw4wev5mjKJEkVGlus+NxW8wItX5sGa6kdUu
21         twAyj7Yr92rF+ICFEP3gGU6+lGo0Nve7KxkN/1W7/m3G4zuk+ccIKmjp8KS3qn99
22         dxy64vcji9jI1lVa+XXOGIp0G8GEaj7mbkixL/bMeGfdMlv8Gf2XPPp9vwCgn/GC
23         JKacfnw7MpLKUHOYS1b//JsEAJqao3ViNfav83jJKEkD8cf59Y8xKia5OpZqTK5W
24         ShVnNWS3U5IVQk10ZDH97Qn/YrK387H4CyhLE9mxPXs/ul18ioiaars/q2MEKU2I
25         XKfV21eMLO9LYd6Ny/Kqj8o5WQK2J6+NAhSwvthZcIEphcFignIuobP+B5wNFQpe
26         DbKfA/0WvN2OwFeWRcmmd3Hz7nHTpcnSF+4QX6yHRF/5BgxkG6IqBIACQbzPn6Hm
27         sMtm/SVf11izmDgSsQptCrOZILfLX/mE+YOl+CwWShhl+YsFts1W0uh1EhQD26aO

```

(continues on next page)

(continued from previous page)

```

28     Z84HuHV5HFRWjDLw9LriltBVQcXbpfSrRP5bdr7Wh8vhqJTPjrQnT3BzY29kZSBQ
29     YWNrYWdlcyA8cGFja2FnZXNAb3BzY29kZS5jb20+iGAEEeECACAFakppC7QCGwMG
30     CwkIBwMCBBUCCAMEFgIDAQIeAQIXgAAKCRApQKupg++Caj8sAKCOXmdG36gWji/K
31     +o+XtBfvdMnFyQCfTCEWxRy2BnzLoBBFCjDSK6sJqCu0IENIRUYgUGFja2FnZXMG
32     PHBhY2thZ2VzQGNoZWYuaW8+iGIEEeECACIFAlQwYFECCGwMGCwkIBwMCBhUIAgK
33     CwQWAgMBAh4BAheAAAOJEC1Aq6mD74JqX94An26z99XOHWPtLN8ahzm7cp13t4Xid
34     AJ9wVcgoUBzvvgg91lKfv/34cmemZn7kCDQRKaQu0EAga7ZLCVGVtMlqBM6njZEd
35     Zbv+mZbvWLBsSomdiqddE6u3eH0X3GuwaQfQWHUVG2yedyDMiG+EMtCdEeeRebTCz
36     SNXQ8Xvi22hrPoEsBSwWlZi8/XNg0n0f1+GER+mOKO0BxDB2DG7DA0nnEISxwFkK
37     OFJFebR3fRsrWjj0KjDxkhse2ddU/jVz1BY7Nf8toZmwpBmdozETMOTx3LJy1HZ/
38     Te9FJXJMUaB2lRyluv15MVWCKQJro4MQG/7QGcIfrIZNfAGJ32DDsjV7/YO+IprY
39     IL4CUBQ65suY4gYUG4jhrH6u7H1p99sdwsg50IpBe/v2Vbc/tbwAB+eJJAp89Zeu
40     twADBQf/ZcGoPhTGFuzbkcnRSIz+boaeWPoSxK2DyFScyCAuG41CY9+g0HIw9Sq8
41     DuxQvJ+vrEJjNvNE3EAEdK1/zkXMZDb1EXjGwDi845TxEMhhD1dDw2qpHqnJ2mtE
42     WpZ7juGwA3sGhi6Fap004tIGacCfNNHmlRGipyq5ZiKIRq9mLEndlECr8cwaKgkS
43     0wWu+xmMZe7N5/t/Tk19HXNh4tVacv0F3fYK54GUjt2FjCQV75USnmNY4KPTYLXA
44     dzC364hEMlXpN21siIFgB04w+TXn5UF3B4FAy5hevvr4DtV4MvMiGLu0oWjpaLC
45     MpmrR3Ny2wkm00h+vgri9uIP06ODWIhJBbGragAJBQJKaQu0AhsMAAOJEC1Aq6mD
46     74Jq4hIAoJ5KrYS8kCwj26SAGzglwggpvt3CAJ0bekyky56vNqoegB+y4PQVDv4K
47     zA==
48     =IxPr
49     -----END PGP PUBLIC KEY BLOCK-----

```

51 **chef:**

```

52
53     # Valid values are 'accept' and 'accept-no-persist'
54     chef_license: "accept"
55
56     # Valid values are 'gems' and 'packages' and 'omnibus'
57     install_type: "packages"
58
59     # Boolean: run 'install_type' code even if chef-client
60     #           appears already installed.
61     force_install: false
62
63     # Chef settings
64     server_url: "https://chef.yourorg.com"
65
66     # Node Name
67     # Defaults to the instance-id if not present
68     node_name: "your-node-name"
69
70     # Environment
71     # Defaults to '_default' if not present
72     environment: "production"
73
74     # Default validation name is chef-validator
75     validation_name: "yourorg-validator"
76     # if validation_cert's value is "system" then it is expected
77     # that the file already exists on the system.
78     validation_cert: |
79         -----BEGIN RSA PRIVATE KEY-----
80         YOUR-ORGS-VALIDATION-KEY-HERE
81         -----END RSA PRIVATE KEY-----
82
83     # A run list for a first boot json, an example (not required)
84     run_list:

```

(continues on next page)

(continued from previous page)

```

85     - "recipe[apache2]"
86     - "role[db]"
87
88     # Specify a list of initial attributes used by the cookbooks
89     initial_attributes:
90         apache:
91             prefork:
92                 maxclients: 100
93                 keepalive: "off"
94
95     # if install_type is 'omnibus', change the url to download
96     omnibus_url: "https://www.chef.io/chef/install.sh"
97
98     # if install_type is 'omnibus', pass pinned version string
99     # to the install script
100    omnibus_version: "12.3.0"
101
102    # If encrypted data bags are used, the client needs to have a secrets file
103    # configured to decrypt them
104    encrypted_data_bag_secret: "/etc/chef/encrypted_data_bag_secret"
105
106    # Capture all subprocess output into a logfile
107    # Useful for troubleshooting cloud-init issues
108    output: {all: '| tee -a /var/log/cloud-init-output.log'}

```

1.8.7 Setup and run puppet

```

1  #cloud-config
2  #
3  # This is an example file to automatically setup and run puppet
4  # when the instance boots for the first time.
5  # Make sure that this file is valid yaml before starting instances.
6  # It should be passed as user-data when starting the instance.
7  puppet:
8      # Boolean: whether or not to install puppet (default: true)
9      install: true
10
11     # A specific version to pass to the installer script or package manager
12     version: "7.7.0"
13
14     # Valid values are 'packages' and 'aio' (default: 'packages')
15     install_type: "packages"
16
17     # Puppet collection to install if 'install_type' is 'aio'
18     collection: "puppet7"
19
20     # Boolean: whether or not to remove the puppetlabs repo after installation
21     # if 'install_type' is 'aio' (default: true)
22     cleanup: true
23
24     # If 'install_type' is 'aio', change the url to the install script
25     aio_install_url: "https://raw.githubusercontent.com/puppetlabs/install-puppet/main/
26     ↪install.sh"
27
28     # Path to the puppet config file (default: depends on 'install_type')

```

(continues on next page)

(continued from previous page)

```

28 conf_file: "/etc/puppet/puppet.conf"
29
30 # Path to the puppet SSL directory (default: depends on 'install_type')
31 ssl_dir: "/var/lib/puppet/ssl"
32
33 # Path to the CSR attributes file (default: depends on 'install_type')
34 csr_attributes_path: "/etc/puppet/csr_attributes.yaml"
35
36 # The name of the puppet package to install (no-op if 'install_type' is 'aio')
37 package_name: "puppet"
38
39 # Boolean: whether or not to run puppet after configuration finishes
40 # (default: false)
41 exec: false
42
43 # A list of arguments to pass to 'puppet agent' if 'exec' is true
44 # (default: ['--test'])
45 exec_args: ['--test']
46
47 # Every key present in the conf object will be added to puppet.conf:
48 # [name]
49 # subkey=value
50 #
51 # For example the configuration below will have the following section
52 # added to puppet.conf:
53 # [main]
54 # server=puppetserver.example.org
55 # certname=i-0123456.ip-X-Y-Z.cloud.internal
56 #
57 # The puppetserver ca certificate will be available in
58 # /var/lib/puppet/ssl/certs/ca.pem if using distro packages
59 # or /etc/puppetlabs/puppet/ssl/certs/ca.pem if using AIO packages.
60 conf:
61   agent:
62     server: "puppetserver.example.org"
63     # certname supports substitutions at runtime:
64     #   %i: instanceid
65     #       Example: i-0123456
66     #   %f: fqdn of the machine
67     #       Example: ip-X-Y-Z.cloud.internal
68     #
69     # NB: the certname will automatically be lowercased as required by puppet
70     certname: "%i.%f"
71     # ca_cert is a special case. It won't be added to puppet.conf.
72     # It holds the puppetserver certificate in pem format.
73     # It should be a multi-line string (using the | yaml notation for
74     # multi-line strings).
75     # The puppetserver certificate is located in
76     # /var/lib/puppet/ssl/ca/ca.crt.pem on the puppetserver host if using
77     # distro packages or /etc/puppetlabs/puppet/ssl/ca/ca.crt.pem if using AIO
78     # packages.
79     #
80     ca_cert: |
81       -----BEGIN CERTIFICATE-----
82       MIICCTCCAXKgAwIBAgIBATANBgkqhkiG9w0BAQUFADANMQswCQYDVQQDDAJjYTAe
83       Fw0xMDAyMTUxNzI5MjFjFaFw0xNTAyMTQxNzI5MjFjFAMA0xCzAJBgNVBAMMamNhMIGf
84       MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCu7Q40sm47/E1Pf+r8AYb/V/FWGPgc

```

(continues on next page)

(continued from previous page)

```

85     b014OmNoX7dgCxTDvps/h8Vw555PdAFsW5+QhsGr31IJNI3kSYprFQcYf7A8tNWu
86     1MASW2CfaEiOEi9F1R3R4Q1z4ix+iNoHiUDTjazzw/tZwEdxaQXQVLwgTGRwVa+aa
87     qbutJKi93MILLwIDAQABo3kwdzA4BglghkgBhvhCAQ0EKxYpUHVwcGV0IFJ1YnkV
88     T3BlblNTTCBHZW51cmF0ZWQgQ2VydGlmaWNhdGUwDwYDVR0TAQH/BAUwAwEB/zAd
89     BgNVHQ4EFgQUu4+jHB+GYE5Vxo+ol1OAhevspjAwCwYDVR0PBAQDAgEGMA0GCSqG
90     SIb3DQEBBQUAA4GBAH/rxlUIjwNb3n7TXJcDJ6MMHULwjr03BDJXKb34Ulnkpfaf
91     +GAlzPXWa7b0908M9I8RnPfvtKnteLbvgTK+h+zX1XCty+S2EQWk29i2AdoqOTxb
92     hppiGmp0tT5Havu4aceCXiy2crVcudj3NFciy8X66SoECemW9UYDCb9T5D0d
93     -----END CERTIFICATE-----

```

1.8.8 Add primary apt repositories

```

1  #cloud-config
2
3  # Add primary apt repositories
4  #
5  # To add 3rd party repositories, see cloud-config-apt.txt or the
6  # Additional apt configuration and repositories section.
7  #
8  #
9  # Default: auto select based on cloud metadata
10 # in ec2, the default is <region>.archive.ubuntu.com
11 # apt:
12 #   primary:
13 #     - arches [default]
14 #       uri:
15 #         use the provided mirror
16 #       search:
17 #         search the list for the first mirror.
18 #         this is currently very limited, only verifying that
19 #         the mirror is dns resolvable or an IP address
20 #
21 # if neither mirror is set (the default)
22 # then use the mirror provided by the DataSource found.
23 # In EC2, that means using <region>.ec2.archive.ubuntu.com
24 #
25 # if no mirror is provided by the DataSource, but 'search_dns' is
26 # true, then search for dns names '<distro>-mirror' in each of
27 # - fqdn of this host per cloud metadata
28 # - localdomain
29 # - no domain (which would search domains listed in /etc/resolv.conf)
30 # If there is a dns entry for <distro>-mirror, then it is assumed that there
31 # is a distro mirror at http://<distro>-mirror.<domain>/<distro>
32 #
33 # That gives the cloud provider the opportunity to set mirrors of a distro
34 # up and expose them only by creating dns entries.
35 #
36 # if none of that is found, then the default distro mirror is used
37 apt:
38   primary:
39     - arches: [default]
40       uri: http://us.archive.ubuntu.com/ubuntu/
41 # or
42 apt:
43   primary:

```

(continues on next page)

(continued from previous page)

```

44     - arches: [default]
45     search:
46         - http://local-mirror.mydomain
47         - http://archive.ubuntu.com
48 # or
49 apt:
50     primary:
51         - arches: [default]
52         search_dns: True

```

1.8.9 Run commands on first boot

```

1 #cloud-config
2
3 # boot commands
4 # default: none
5 # this is very similar to runcmd, but commands run very early
6 # in the boot process, only slightly after a 'boothook' would run.
7 # bootcmd should really only be used for things that could not be
8 # done later in the boot process. bootcmd is very much like
9 # boothook, but possibly with more friendly.
10 # - bootcmd will run on every boot
11 # - the INSTANCE_ID variable will be set to the current instance id.
12 # - you can use 'cloud-init-per' command to help only run once
13 bootcmd:
14     - echo 192.168.1.130 us.archive.ubuntu.com >> /etc/hosts
15     - [ cloud-init-per, once, mymkfs, mkfs, /dev/vdb ]

```

```

1 #cloud-config
2
3 # run commands
4 # default: none
5 # runcmd contains a list of either lists or a string
6 # each item will be executed in order at rc.local like level with
7 # output to the console
8 # - runcmd only runs during the first boot
9 # - if the item is a list, the items will be properly executed as if
10 #   passed to execve(3) (with the first arg as the command).
11 # - if the item is a string, it will be simply written to the file and
12 #   will be interpreted by 'sh'
13 #
14 # Note, that the list has to be proper yaml, so you have to quote
15 # any characters yaml would eat (':' can be problematic)
16 runcmd:
17     - [ ls, -l, / ]
18     - [ sh, -xc, "echo $(date) ': hello world!'" ]
19     - [ sh, -c, echo "=====hello world'===== " ]
20     - ls -l /root
21     # Note: Don't write files to /tmp from cloud-init use /run/somedir instead.
22     # Early boot environments can race systemd-tmpfiles-clean LP: #1707222.
23     - mkdir /run/mydir
24     - [ wget, "http://slashdot.org", -O, /run/mydir/index.html ]

```

1.8.10 Alter the completion message

```

1 #cloud-config
2
3 # final_message
4 # default: cloud-init boot finished at $TIMESTAMP. Up $UPTIME seconds
5 # this message is written by cloud-final when the system is finished
6 # its first boot
7 final_message: "The system is finally up, after $UPTIME seconds"

```

1.8.11 Install arbitrary packages

```

1 #cloud-config
2
3 # Install additional packages on first boot
4 #
5 # Default: none
6 #
7 # if packages are specified, this apt_update will be set to true
8 #
9 # packages may be supplied as a single package name or as a list
10 # with the format [<package>, <version>] wherein the specific
11 # package version will be installed.
12 packages:
13 - pwgen
14 - pastebinit
15 - [libpython2.7, 2.7.3-0ubuntu3.1]

```

1.8.12 Update apt database on first boot

```

1 #cloud-config
2 # Update apt database on first boot (run 'apt-get update').
3 # Note, if packages are given, or package_upgrade is true, then
4 # update will be done independent of this setting.
5 #
6 # Default: false
7 # Aliases: apt_update
8 package_update: true

```

1.8.13 Run apt or yum upgrade

```

1 #cloud-config
2
3 # Upgrade the instance on first boot
4 # (ie run apt-get upgrade)
5 #
6 # Default: false
7 # Aliases: apt_upgrade
8 package_upgrade: true

```

1.8.14 Adjust mount points mounted

```

1 #cloud-config
2
3 # set up mount points
4 # 'mounts' contains a list of lists
5 # the inner list are entries for an /etc/fstab line
6 # ie : [ fs_spec, fs_file, fs_vfstype, fs_mntops, fs_freq, fs_passno ]
7 #
8 # default:
9 # mounts:
10 # - [ ephemeral0, /mnt ]
11 # - [ swap, none, swap, sw, 0, 0 ]
12 #
13 # in order to remove a previously listed mount (ie, one from defaults)
14 # list only the fs_spec. For example, to override the default, of
15 # mounting swap:
16 # - [ swap ]
17 # or
18 # - [ swap, null ]
19 #
20 # - if a device does not exist at the time, an entry will still be
21 # written to /etc/fstab.
22 # - '/dev' can be omitted for device names that begin with: xvd, sd, hd, vd
23 # - if an entry does not have all 6 fields, they will be filled in
24 # with values from 'mount_default_fields' below.
25 #
26 # Note, that you should set 'nofail' (see man fstab) for volumes that may not
27 # be attached at instance boot (or reboot).
28 #
29 mounts:
30 - [ ephemeral0, /mnt, auto, "defaults,noexec" ]
31 - [ sdc, /opt/data ]
32 - [ xvdh, /opt/data, "auto", "defaults,nofail", "0", "0" ]
33 - [ dd, /dev/zero ]
34
35 # mount_default_fields
36 # These values are used to fill in any entries in 'mounts' that are not
37 # complete. This must be an array, and must have 6 fields.
38 mount_default_fields: [ None, None, "auto", "defaults,nofail", "0", "2" ]
39
40
41 # swap can also be set up by the 'mounts' module
42 # default is to not create any swap files, because 'size' is set to 0
43 swap:
44 filename: /swap.img
45 size: "auto" # or size in bytes
46 maxsize: size in bytes

```

1.8.15 Call a url when finished

```

1 #cloud-config
2
3 # phone_home: if this dictionary is present, then the phone_home
4 # cloud-config module will post specified data back to the given

```

(continues on next page)

(continued from previous page)

```

5 # url
6 # default: none
7 # phone_home:
8 #   url: http://my.foo.bar/$INSTANCE/
9 #   post: all
10 #   tries: 10
11 #
12 phone_home:
13   url: http://my.example.com/$INSTANCE_ID/
14   post: [ pub_key_dsa, pub_key_rsa, pub_key_ecdsa, instance_id ]

```

1.8.16 Reboot/poweroff when finished

```

1 #cloud-config
2
3 ## poweroff or reboot system after finished
4 # default: none
5 #
6 # power_state can be used to make the system shutdown, reboot or
7 # halt after boot is finished. This same thing can be achieved by
8 # user-data scripts or by runcmd by simply invoking 'shutdown'.
9 #
10 # Doing it this way ensures that cloud-init is entirely finished with
11 # modules that would be executed, and avoids any error/log messages
12 # that may go to the console as a result of system services like
13 # syslog being taken down while cloud-init is running.
14 #
15 # If you delay '+5' (5 minutes) and have a timeout of
16 # 120 (2 minutes), then the max time until shutdown will be 7 minutes.
17 # cloud-init will invoke 'shutdown +5' after the process finishes, or
18 # when 'timeout' seconds have elapsed.
19 #
20 # delay: form accepted by shutdown. default is 'now'. other format
21 #   accepted is '+m' (m in minutes)
22 # mode: required. must be one of 'poweroff', 'halt', 'reboot'
23 # message: provided as the message argument to 'shutdown'. default is none.
24 # timeout: the amount of time to give the cloud-init process to finish
25 #   before executing shutdown.
26 # condition: apply state change only if condition is met.
27 #   May be boolean True (always met), or False (never met),
28 #   or a command string or list to be executed.
29 #   command's exit code indicates:
30 #       0: condition met
31 #       1: condition not met
32 #   other exit codes will result in 'not met', but are reserved
33 #   for future use.
34 #
35 power_state:
36   delay: "+30"
37   mode: poweroff
38   message: Bye Bye
39   timeout: 30
40   condition: True

```

1.8.17 Configure instances SSH keys

```

1  #cloud-config
2
3  # add each entry to ~/.ssh/authorized_keys for the configured user or the
4  # first user defined in the user definition directive.
5  ssh_authorized_keys:
6      - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUUK8EEAnnkhXlukKoUPND/
        ↳RRClWz2s5TCzIk3Ou5+Cyz71X0XmazM3l5WgeErvtIwQMyT1KjNoMhoJMrJnWqQPOT5Q8zWd9qG7PB19+eiH5qV7NZ_
        ↳mykey@host
7      - ssh-rsa_
        ↳AAAAB3NzaC1yc2EAAAABIwAAAEQA3I7VUF2l5gSn5uavROsc5HRDpZdQueUq5ozemNSj8T7enqKHOEaFoU2VoPgGEWC9RyzSQV
        ↳+ilD+ey3ONkZLN+LQ714cgj8fRS4Hj29SCmXp5Kt5/82cD/VN3NtHw== smoser@brickies
8
9  # Send pre-generated SSH private keys to the server
10 # If these are present, they will be written to /etc/ssh and
11 # new random keys will not be generated
12 # in addition to 'rsa' and 'dsa' as shown below, 'ecdsa' is also supported
13 ssh_keys:
14     rsa_private: |
15         -----BEGIN RSA PRIVATE KEY-----
16         MIIBxwIBAAJhAKD0YSHy73nUgysO13XsJmd4fHiFyQ+00R7VVu2iV9Qcon2LZS/x
17         lcydPZ4pQpfjEha6WxZ6o8ci/Ea/wOn+0HGPwaxLEG2Z9inNtj3pgFrYcRztfECb
18         1j6HCibZbAzYtwIBIwJgO8h72WjcmvcpZ8OvHSvTwAguO2TkR6mPgHsgSaKy6GJo
19         PUJnaZRWuba/HX0KGyhz19nPzLpzG5f0fYahlMJAYc13FV7K6kMBPXTRR6FxfHEg
20         L0MPC7cdqAwOVNcPY6A7AjEA1bNaIjOzFN2sfZX0j7OMhQuc4zP7r80zaGc5oy6W
21         p58hRancFKEvnEq2CeL3vtuZAJEAwNBHpbNsBYTRPCHM7rZuG/iBtwp8Rxhc9I5w
22         ixvzMgi+HpGLWzUIBS+P/XhekIjPAjA285rVMEP+DR255LS65QbgYhJmTzIXQ2T9
23         luLvcmFBC6l35Uc4gTgg4ALsmXLn7lMCMGMpSWspEvuGINayTCL+vEjmNBT+FAAdO
24         W7D4zCpI43jRS9U06JVOeSc9CDk2lwiA3wIwCTB/6uc8Cq85D9YqpM10FuHjKpnP
25         REPP0yrAspdeOAV+6VKRavstea7+2DZmSUGe
26         -----END RSA PRIVATE KEY-----
27
28     rsa_public: ssh-rsa_
        ↳AAAAB3NzaC1yc2EAAAABIwAAAGEAoPrHIfLvedSDKw7XdewmZ3h8eIXJD7TRhtVW7aJX1ByifYt1L/
        ↳HVzJ09nilC1+MSFrpbFnqjxyL8Rr/DSf7QcY/BrGUQbZn2Kc22PemAWthxH018QJvWPocKJt1sDNi3_
        ↳smoser@localhost
29
30     dsa_private: |
31         -----BEGIN DSA PRIVATE KEY-----
32         MIIBuwIBAAKBgQDP2HLu7pTEXL89USyM0264RCyWX/CMLmukxX0Jdbm29ax8FBJT
33         pLrO8TIXVY5rPAJmldTHnpuyJhOvU9G7M8tPUABtzSJh4GVSHlwaCfycwcpLv9TX
34         DgWIpSj+6EIHCyARlB1/CBp9RiaB+10QcFbm+lapuET+/Au6vSDp9IRt1QIVAIMR
35         8KucvUYbOEI+yv+5LW9u3z/BAoGBAI0q6JP+JvJmwZFaecMMVxXUbqiSko/P1lsa
36         LNNBH25/8MOUIm8rB2FC6ziidfueJpqTMqeQmSAIEBCwnwreUnGfRrKoJpyPNENY
37         dl5MG6N5J+z8lsEchFeprryZ+D3Ge9VjPq3Tf3NhKKwCDQ0240aPezbnjPeFm4mH
38         bYxxcZ9GAoGAXmLIFSQgiAPu459rCKxT46tHJtM0QfnNiEnQLbFluefZ/yiI4DI3
39         8UzTCOXLhUA7ybmZha+D/csj15Y9/BNFu07unzVhikCQV9DTeXX46pG4slo23JKC/
40         /QaYWNMZ7kTRv+wWow9MhGiVdML4ZN4Xnifu05krqAybnGiy66PMEoQCFEIsKKWv
41         99iziAH0KBMVbxy03Trz
42         -----END DSA PRIVATE KEY-----
43
44     dsa_public: ssh-dss AAAAB3NzaC1kc3MAAACBAM/
        ↳Ycu7ulMTEvz1RLIzTbrhELJZf8Iwua6TFfQ1lubb1rHwUElOkus7xMhdVjms8AmbV1Meem7ImE69T0bszy09QAAG3NImHgZVIEX
        ↳JzByku/
        ↳lNcOBYilKP7oSicLJpGUHX8IGn1GJoH7XRBwVub6Vqm4RP78C7q9IOhG2VAAAAFQCDEfCrnL1GGzhCPsr/
        ↳uSlvbt8/wQAAAEIAjsrok/4m8mbBkVp4IwxXFdRuqJKSj8/WWxos00Ednn/
        ↳ww5QibysHYULrOKJ1+54mmpMyp5CZICUQELCfct5ScZ9Gsggmni8Q01h3Xkwo3kn7PzWwRwcV6muvJn4PcZ71WM+rdN/
        ↳c2EorAINDTbjRo97NueM94WbiYdtjHFxn0YAAACAXmLIFSQgiAPu459rCKxT46tHJtM0QfnNiEnQLbFluefZ/yiI4DI38UzTCOXLhUA7ybmZha+D/csj15Y9/BNFu07unzVhikCQV9DTeXX46pG4slo23JKC/
        ↳yiI4DI38UzTCOXLhUA7ybmZha+D/csj15Y9/BNFu07unzVhikCQV9DTeXX46pG4slo23JKC/
        ↳QaYWNMZ7kTRv+wWow9MhGiVdML4ZN4Xnifu05krqAybnGiy66PMEoQ= smoser@localhost

```

(continues on next page)

(continued from previous page)

```

45 # By default, the fingerprints of the authorized keys for the users
46 # cloud-init adds are printed to the console. Setting
47 # no_ssh_fingerprints to true suppresses this output.
48 no_ssh_fingerprints: false
49
50
51 # By default, (most) ssh host keys are printed to the console. Setting
52 # emit_keys_to_console to false suppresses this output.
53 ssh:
54     emit_keys_to_console: false

```

1.8.18 Additional apt configuration and repositories

```

1  #cloud-config
2  # apt_pipelining (configure Acquire::http::Pipeline-Depth)
3  # Default: disables HTTP pipelining. Certain web servers, such
4  # as S3 do not pipeline properly (LP: #948461).
5  # Valid options:
6  #   False/default: Disables pipelining for APT
7  #   None/Unchanged: Use OS default
8  #   Number: Set pipelining to some number (not recommended)
9  apt_pipelining: False
10
11 ## apt config via system_info:
12 # under the 'system_info', you can customize cloud-init's interaction
13 # with apt.
14 # system_info:
15 #   apt_get_command: [command, argument, argument]
16 #   apt_get_upgrade_subcommand: dist-upgrade
17 #
18 # apt_get_command:
19 #   To specify a different 'apt-get' command, set 'apt_get_command'.
20 #   This must be a list, and the subcommand (update, upgrade) is appended to it.
21 #   default is:
22 #   ['apt-get', '--option=Dpkg::Options::--force-confold',
23 #    '--option=Dpkg::options::--force-unsafe-io', '--assume-yes', '--quiet']
24 #
25 # apt_get_upgrade_subcommand: "dist-upgrade"
26 #   Specify a different subcommand for 'upgrade. The default is 'dist-upgrade'.
27 #   This is the subcommand that is invoked for package_upgrade.
28 #
29 # apt_get_wrapper:
30 #   command: eatmydata
31 #   enabled: [True, False, "auto"]
32 #
33
34 # Install additional packages on first boot
35 #
36 # Default: none
37 #
38 # if packages are specified, this apt_update will be set to true
39
40 packages: ['pastebinit']
41
42 apt:

```

(continues on next page)

(continued from previous page)

```

43 # The apt config consists of two major "areas".
44 #
45 # On one hand there is the global configuration for the apt feature.
46 #
47 # On one hand (down in this file) there is the source dictionary which allows
48 # to define various entries to be considered by apt.
49
50 #####
51 # Section 1: global apt configuration
52 #
53 # The following examples number the top keys to ease identification in
54 # discussions.
55
56 # 1.1 preserve_sources_list
57 #
58 # Preserves the existing /etc/apt/sources.list
59 # Default: false - do overwrite sources_list. If set to true then any
60 # "mirrors" configuration will have no effect.
61 # Set to true to avoid affecting sources.list. In that case only
62 # "extra" source specifications will be written into
63 # /etc/apt/sources.list.d/*
64 preserve_sources_list: true
65
66 # 1.2 disable_suites
67 #
68 # This is an empty list by default, so nothing is disabled.
69 #
70 # If given, those suites are removed from sources.list after all other
71 # modifications have been made.
72 # Suites are even disabled if no other modification was made,
73 # but not if is preserve_sources_list is active.
74 # There is a special alias "$RELEASE" as in the sources that will be replace
75 # by the matching release.
76 #
77 # To ease configuration and improve readability the following common ubuntu
78 # suites will be automatically mapped to their full definition.
79 # updates    => $RELEASE-updates
80 # backports  => $RELEASE-backports
81 # security   => $RELEASE-security
82 # proposed   => $RELEASE-proposed
83 # release    => $RELEASE
84 #
85 # There is no harm in specifying a suite to be disabled that is not found in
86 # the source.list file (just a no-op then)
87 #
88 # Note: Lines don't get deleted, but disabled by being converted to a comment.
89 # The following example disables all usual defaults except $RELEASE-security.
90 # On top it disables a custom suite called "mysuite"
91 disable_suites: [$RELEASE-updates, backports, $RELEASE, mysuite]
92
93 # 1.3 primary/security archives
94 #
95 # Default: none - instead it is auto select based on cloud metadata
96 # so if neither "uri" nor "search", nor "search_dns" is set (the default)
97 # then use the mirror provided by the DataSource found.
98 # In EC2, that means using <region>.ec2.archive.ubuntu.com
99 #

```

(continues on next page)

(continued from previous page)

```

100 # define a custom (e.g. localized) mirror that will be used in sources.list
101 # and any custom sources entries for deb / deb-src lines.
102 #
103 # One can set primary and security mirror to different uri's
104 # the child elements to the keys primary and secondary are equivalent
105 primary:
106 # arches is list of architectures the following config applies to
107 # the special keyword "default" applies to any architecture not explicitly
108 # listed.
109 - arches: [amd64, i386, default]
110 # uri is just defining the target as-is
111 uri: http://us.archive.ubuntu.com/ubuntu
112 #
113 # via search one can define lists that are tried one by one.
114 # The first with a working DNS resolution (or if it is an IP) will be
115 # picked. That way one can keep one configuration for multiple
116 # subenvironments that select the working one.
117 search:
118 - http://cool.but-sometimes-unreachable.com/ubuntu
119 - http://us.archive.ubuntu.com/ubuntu
120 # if no mirror is provided by uri or search but 'search_dns' is
121 # true, then search for dns names '<distro>-mirror' in each of
122 # - fqdn of this host per cloud metadata
123 # - localdomain
124 # - no domain (which would search domains listed in /etc/resolv.conf)
125 # If there is a dns entry for <distro>-mirror, then it is assumed that
126 # there is a distro mirror at http://<distro>-mirror.<domain>/<distro>
127 #
128 # That gives the cloud provider the opportunity to set mirrors of a distro
129 # up and expose them only by creating dns entries.
130 #
131 # if none of that is found, then the default distro mirror is used
132 search_dns: true
133 #
134 # If multiple of a category are given
135 # 1. uri
136 # 2. search
137 # 3. search_dns
138 # the first defining a valid mirror wins (in the order as defined here,
139 # not the order as listed in the config).
140 #
141 # Additionally, if the repository requires a custom signing key, it can be
142 # specified via the same fields as for custom sources:
143 # 'keyid': providing a key to import via shortid or fingerprint
144 # 'key': providing a raw PGP key
145 # 'keyserver': specify an alternate keyserver to pull keys from that
146 # were specified by keyid
147 - arches: [s390x, arm64]
148 # as above, allowing to have one config for different per arch mirrors
149 # security is optional, if not defined it is set to the same value as primary
150 security:
151 - uri: http://security.ubuntu.com/ubuntu
152 arches: [default]
153 # If search_dns is set for security the searched pattern is:
154 # <distro>-security-mirror
155
156 # if no mirrors are specified at all, or all lookups fail it will try

```

(continues on next page)

(continued from previous page)

```

157 # to get them from the cloud datasource and if those neither provide one fall
158 # back to:
159 #   primary: http://archive.ubuntu.com/ubuntu
160 #   security: http://security.ubuntu.com/ubuntu
161
162 # 1.4 sources_list
163 #
164 # Provide a custom template for rendering sources.list
165 # without one provided cloud-init uses builtin templates for
166 # ubuntu and debian.
167 # Within these sources.list templates you can use the following replacement
168 # variables (all have sane Ubuntu defaults, but mirrors can be overwritten
169 # as needed (see above)):
170 # => $RELEASE, $MIRROR, $PRIMARY, $SECURITY
171 sources_list: | # written by cloud-init custom template
172     deb $MIRROR $RELEASE main restricted
173     deb-src $MIRROR $RELEASE main restricted
174     deb $PRIMARY $RELEASE universe restricted
175     deb $SECURITY $RELEASE-security multiverse
176
177 # 1.5 conf
178 #
179 # Any apt config string that will be made available to apt
180 # see the APT.CONF(5) man page for details what can be specified
181 conf: | # APT config
182     APT {
183         Get {
184             Assume-Yes "true";
185             Fix-Broken "true";
186         };
187     };
188
189 # 1.6 (http_/ftp_/https_)proxy
190 #
191 # Proxies are the most common apt.conf option, so that for simplified use
192 # there is a shortcut for those. Those get automatically translated into the
193 # correct Acquire::Proxy statements.
194 #
195 # note: proxy actually being a short synonym to http_proxy
196 proxy: http://[[user][:pass@]host[:port]]/
197 http_proxy: http://[[user][:pass@]host[:port]]/
198 ftp_proxy: ftp://[[user][:pass@]host[:port]]/
199 https_proxy: https://[[user][:pass@]host[:port]]/
200
201 # 1.7 add_apt_repo_match
202 #
203 # 'source' entries in apt-sources that match this python regex
204 # expression will be passed to add-apt-repository
205 # The following example is also the builtin default if nothing is specified
206 add_apt_repo_match: '^[\\w-]+:\\w'
207
208 #####
209 # Section 2: source list entries
210 #
211 # This is a dictionary (unlike most block/net which are lists)
212 #

```

(continues on next page)

(continued from previous page)

```

214 # The key of each source entry is the filename and will be prepended by
215 # /etc/apt/sources.list.d/ if it doesn't start with a '/'.
216 # If it doesn't end with .list it will be appended so that apt picks up its
217 # configuration.
218 #
219 # Whenever there is no content to be written into such a file, the key is
220 # not used as filename - yet it can still be used as index for merging
221 # configuration.
222 #
223 # The values inside the entries consist of the following optional entries:
224 # 'source': a sources.list entry (some variable replacements apply)
225 # 'keyid': providing a key to import via shortid or fingerprint
226 # 'key': providing a raw PGP key
227 # 'keyserver': specify an alternate keyserver to pull keys from that
228 #               were specified by keyid
229
230 # This allows merging between multiple input files than a list like:
231 # cloud-config1
232 # sources:
233 #   s1: {'key': 'key1', 'source': 'source1'}
234 # cloud-config2
235 # sources:
236 #   s2: {'key': 'key2'}
237 #   s1: {'keyserver': 'foo'}
238 # This would be merged to
239 # sources:
240 #   s1:
241 #     keyserver: foo
242 #     key: key1
243 #     source: source1
244 #   s2:
245 #     key: key2
246 #
247 # The following examples number the subfeatures per sources entry to ease
248 # identification in discussions.
249
250
251 sources:
252   curtin-dev-ppa.list:
253     # 2.1 source
254     #
255     # Creates a file in /etc/apt/sources.list.d/ for the sources list entry
256     # based on the key: "/etc/apt/sources.list.d/curtin-dev-ppa.list"
257     source: "deb http://ppa.launchpad.net/curtin-dev/test-archive/ubuntu bionic main
258 ↪ "
259
260     # 2.2 keyid
261     #
262     # Importing a gpg key for a given key id. Used keyserver defaults to
263     # keyserver.ubuntu.com
264     keyid: F430BBA5 # GPG key ID published on a key server
265
266   ignored1:
267     # 2.3 PPA shortcut
268     #
269     # Setup correct apt sources.list line and Auto-Import the signing key
270     # from LP

```

(continues on next page)

(continued from previous page)

```

270 #
271 # See https://help.launchpad.net/Packaging/PPA for more information
272 # this requires 'add-apt-repository'. This will create a file in
273 # /etc/apt/sources.list.d automatically, therefore the key here is
274 # ignored as filename in those cases.
275 source: "ppa:curtin-dev/test-archive"      # Quote the string
276
277 my-repo2.list:
278 # 2.4 replacement variables
279 #
280 # sources can use $MIRROR, $PRIMARY, $SECURITY, $RELEASE and $KEY_FILE
281 # replacement variables.
282 # They will be replaced with the default or specified mirrors and the
283 # running release.
284 # The entry below would be possibly turned into:
285 # source: deb http://archive.ubuntu.com/ubuntu bionic multiverse
286 source: deb [signed-by=$KEY_FILE] $MIRROR $RELEASE multiverse
287 keyid: F430BBA5
288
289 my-repo3.list:
290 # this would have the same end effect as 'ppa:curtin-dev/test-archive'
291 source: "deb http://ppa.launchpad.net/curtin-dev/test-archive/ubuntu bionic main
292 ↪"
293 keyid: F430BBA5 # GPG key ID published on the key server
294 filename: curtin-dev-ppa.list
295
296 ignored2:
297 # 2.5 key only
298 #
299 # this would only import the key without adding a ppa or other source spec
300 # since this doesn't generate a source.list file the filename key is ignored
301 keyid: F430BBA5 # GPG key ID published on a key server
302
303 ignored3:
304 # 2.6 key id alternatives
305 #
306 # Keyid's can also be specified via their long fingerprints
307 keyid: B59D 5F15 97A5 04B7 E230 6DCA 0620 BBCF 0368 3F77
308
309 ignored4:
310 # 2.7 alternative keyserver
311 #
312 # One can also specify alternative keyserver to fetch keys from.
313 keyid: B59D 5F15 97A5 04B7 E230 6DCA 0620 BBCF 0368 3F77
314 keyserver: pgp.mit.edu
315
316 ignored5:
317 # 2.8 signed-by
318 #
319 # One can specify [signed-by=$KEY_FILE] in the source definition, which
320 # will make the key be installed in the directory /etc/cloud-init.gpg.d/
321 # and the $KEY_FILE replacement variable will be replaced with the path
322 # to the specified key. If $KEY_FILE is used, but no key is specified,
323 # apt update will (rightfully) fail due to an invalid value.
324 source: deb [signed-by=$KEY_FILE] $MIRROR $RELEASE multiverse
325 keyid: B59D 5F15 97A5 04B7 E230 6DCA 0620 BBCF 0368 3F77

```

(continues on next page)

(continued from previous page)

```

326 my-repo4.list:
327     # 2.9 raw key
328     #
329     # The apt signing key can also be specified by providing a pgp public key
330     # block. Providing the PGP key this way is the most robust method for
331     # specifying a key, as it removes dependency on a remote key server.
332     #
333     # As with keyid's this can be specified with or without some actual source
334     # content.
335     key: | # The value needs to start with -----BEGIN PGP PUBLIC KEY BLOCK-----
336           -----BEGIN PGP PUBLIC KEY BLOCK-----
337           Version: SKS 1.0.10
338
339           mI0ESpA3UQEEALdZKVMq0j6qWAXAyxSlF63SvPVIgXHPb9Nk0DZUixn+akqytxG4zKCONz6
340           qLjoBBfHnynyVLfT4ihg9an1PqxRnTO+JKQx18NgKGz6Pon569GtAOdWKNw15XKinJTDLjnj
341           9y961jJqRcpV9t/WsIcdJPcKFR5voHTEoABE2aEXABEBAAG0GUxhdW5jaHBhZCBQUEEEgZm9y
342           IEFsZXN0aWoitgQTAQIAIAUCSpA3UQIbAwYLCQgHAWIEFQIIAwQWAgMBAh4BAheAAAJEA7H
343           5Qi+CcVxWZ8D/1MyYvfj3FJPZUm2Yo1zZsQ657vHI9+pPouqf1W0ayRR9jbiyUFIn0VdQBrP
344           t0FwvnOFARuovUWoKAEdqR8hPy3M3APUZjl5K4cMZr/xAMQeQRZ5CHpS4DBKURKAHC0ltS5o
345           uBJKQOZm5iltJpl5cgyIkBkGe8Mx18VFyVglAZey
346           =Y2oI
347           -----END PGP PUBLIC KEY BLOCK-----

```

1.8.19 Disk setup

```

1  #cloud-config
2  # Cloud-init supports the creation of simple partition tables and file systems
3  # on devices.
4
5  # Default disk definitions for AWS
6  # -----
7  # (Not implemented yet, but provided for future documentation)
8
9  disk_setup:
10     ephemeral0:
11         table_type: 'mbr'
12         layout: True
13         overwrite: False
14
15  fs_setup:
16     - label: None,
17       filesystem: ext3
18       device: ephemeral0
19       partition: auto
20
21  # Default disk definitions for Microsoft Azure
22  # -----
23
24  device_aliases: {'ephemeral0': '/dev/sdb'}
25  disk_setup:
26     ephemeral0:
27         table_type: mbr
28         layout: True
29         overwrite: False
30

```

(continues on next page)

(continued from previous page)

```

31 fs_setup:
32   - label: ephemeral0
33     filesystem: ext4
34     device: ephemeral0.1
35     replace_fs: ntfs
36
37
38 # Data disks definitions for Microsoft Azure
39 # -----
40
41 disk_setup:
42   /dev/disk/azure/scsi1/lun0:
43     table_type: gpt
44     layout: True
45     overwrite: True
46
47 fs_setup:
48   - device: /dev/disk/azure/scsi1/lun0
49     partition: 1
50     filesystem: ext4
51
52
53 # Default disk definitions for SmartOS
54 # -----
55
56 device_aliases: {'ephemeral0': '/dev/vdb'}
57 disk_setup:
58   ephemeral0:
59     table_type: mbr
60     layout: False
61     overwrite: False
62
63 fs_setup:
64   - label: ephemeral0
65     filesystem: ext4
66     device: ephemeral0.0
67
68 # Caveat for SmartOS: if ephemeral disk is not defined, then the disk will
69 #   not be automatically added to the mounts.
70
71
72 # The default definition is used to make sure that the ephemeral storage is
73 # setup properly.
74
75 # "disk_setup": disk partitioning
76 # -----
77
78 # The disk_setup directive instructs Cloud-init to partition a disk. The format is:
79
80 disk_setup:
81   ephemeral0:
82     table_type: 'mbr'
83     layout: true
84   /dev/xvdh:
85     table_type: 'mbr'
86     layout:
87       - 33

```

(continues on next page)

(continued from previous page)

```

88     - [33, 82]
89     - 33
90     overwrite: True
91
92     # The format is a list of dicts of dicts. The first value is the name of the
93     # device and the subsequent values define how to create and layout the
94     # partition.
95     # The general format is:
96     #     disk_setup:
97     #         <DEVICE>:
98     #             table_type: 'mbr'
99     #             layout: <LAYOUT|BOOL>
100    #             overwrite: <BOOL>
101    #
102    # Where:
103    #     <DEVICE>: The name of the device. 'ephemeralX' and 'swap' are special
104    #               values which are specific to the cloud. For these devices
105    #               Cloud-init will look up what the real devices is and then
106    #               use it.
107    #
108    #               For other devices, the kernel device name is used. At this
109    #               time only simply kernel devices are supported, meaning
110    #               that device mapper and other targets may not work.
111    #
112    #               Note: At this time, there is no handling or setup of
113    #               device mapper targets.
114    #
115    #     table_type=<TYPE>: Currently the following are supported:
116    #         'mbr': default and setups a MS-DOS partition table
117    #         'gpt': setups a GPT partition table
118    #
119    #               Note: At this time only 'mbr' and 'gpt' partition tables
120    #               are allowed. It is anticipated in the future that
121    #               we'll also have "RAID" to create a mdadm RAID.
122    #
123    #     layout={...}: The device layout. This is a list of values, with the
124    #                   percentage of disk that partition will take.
125    #                   Valid options are:
126    #                   [<SIZE>, [<SIZE>, <PART_TYPE>]]
127    #
128    #                   Where <SIZE> is the _percentage_ of the disk to use, while
129    #                   <PART_TYPE> is the numerical value of the partition type.
130    #
131    #                   The following setups two partitions, with the first
132    #                   partition having a swap label, taking 1/3 of the disk space
133    #                   and the remainder being used as the second partition.
134    #                   /dev/xvdb':
135    #                       table_type: 'mbr'
136    #                       layout:
137    #                           - [33,82]
138    #                           - 66
139    #                       overwrite: True
140    #
141    #                   When layout is "true" it means single partition the entire
142    #                   device.
143    #
144    #                   When layout is "false" it means don't partition or ignore

```

(continues on next page)

(continued from previous page)

```

145 #             existing partitioning.
146 #
147 #             If layout is set to "true" and overwrite is set to "false",
148 #             it will skip partitioning the device without a failure.
149 #
150 #             overwrite=<BOOL>: This describes whether to ride with saftey's on and
151 #             everything holstered.
152 #
153 #             'false' is the default, which means that:
154 #                 1. The device will be checked for a partition table
155 #                 2. The device will be checked for a file system
156 #                 3. If either a partition of file system is found, then
157 #                    the operation will be _skipped_.
158 #
159 #             'true' is cowboy mode. There are no checks and things are
160 #             done blindly. USE with caution, you can do things you
161 #             really, really don't want to do.
162 #
163 #
164 # fs_setup: Setup the file system
165 # -----
166 #
167 # fs_setup describes the how the file systems are supposed to look.
168
169 fs_setup:
170 - label: ephemeral0
171   filesystem: 'ext3'
172   device: 'ephemeral0'
173   partition: 'auto'
174 - label: mylabl2
175   filesystem: 'ext4'
176   device: '/dev/xvda1'
177 - cmd: mkfs -t %(filesystem)s -L %(label)s %(device)s
178   label: mylabl3
179   filesystem: 'btrfs'
180   device: '/dev/xvda1'
181
182 # The general format is:
183 #   fs_setup:
184 #     - label: <LABEL>
185 #       filesystem: <FS_TYPE>
186 #       device: <DEVICE>
187 #       partition: <PART_VALUE>
188 #       overwrite: <OVERWRITE>
189 #       replace_fs: <FS_TYPE>
190 #
191 # Where:
192 #   <LABEL>: The file system label to be used. If set to None, no label is
193 #   used.
194 #
195 #   <FS_TYPE>: The file system type. It is assumed that the there
196 #   will be a "mkfs.<FS_TYPE>" that behaves likes "mkfs". On a standard
197 #   Ubuntu Cloud Image, this means that you have the option of ext{2,3,4},
198 #   and vfat by default.
199 #
200 #   <DEVICE>: The device name. Special names of 'ephemeralX' or 'swap'
201 #   are allowed and the actual device is acquired from the cloud datasource.

```

(continues on next page)

(continued from previous page)

```

202 # When using 'ephemeralX' (i.e. ephemeral0), make sure to leave the
203 # label as 'ephemeralX' otherwise there may be issues with the mounting
204 # of the ephemeral storage layer.
205 #
206 # If you define the device as 'ephemeralX.Y' then Y will be interpreted
207 # as a partition value. However, ephemeralX.0 is the _same_ as ephemeralX.
208 #
209 # <PART_VALUE>:
210 # Partition definitions are overwritten if you use the '<DEVICE>.Y' notation.
211 #
212 # The valid options are:
213 # "auto/any": tell cloud-init not to care whether there is a partition
214 # or not. Auto will use the first partition that does not contain a
215 # file system already. In the absence of a partition table, it will
216 # put it directly on the disk.
217 #
218 # "auto": If a file system that matches the specification in terms of
219 # label, type and device, then cloud-init will skip the creation of
220 # the file system.
221 #
222 # "any": If a file system that matches the file system type and device,
223 # then cloud-init will skip the creation of the file system.
224 #
225 # Devices are selected based on first-detected, starting with partitions
226 # and then the raw disk. Consider the following:
227 # NAME      FSTYPE LABEL
228 # xvdb
229 # |-xvdb1  ext4
230 # |-xvdb2
231 # |-xvdb3  btrfs  test
232 # \-xvdb4  ext4   test
233 #
234 # If you ask for 'auto', label of 'test', and file system of 'ext4'
235 # then cloud-init will select the 2nd partition, even though there
236 # is a partition match at the 4th partition.
237 #
238 # If you ask for 'any' and a label of 'test', then cloud-init will
239 # select the 1st partition.
240 #
241 # If you ask for 'auto' and don't define label, then cloud-init will
242 # select the 1st partition.
243 #
244 # In general, if you have a specific partition configuration in mind,
245 # you should define either the device or the partition number. 'auto'
246 # and 'any' are specifically intended for formatting ephemeral storage or
247 # for simple schemes.
248 #
249 # "none": Put the file system directly on the device.
250 #
251 # <NUM>: where NUM is the actual partition number.
252 #
253 # <OVERWRITE>: Defines whether or not to overwrite any existing
254 # filesystem.
255 #
256 # "true": Indiscriminately destroy any pre-existing file system. Use at
257 # your own peril.
258 #

```

(continues on next page)

(continued from previous page)

```

259 #      "false": If an existing file system exists, skip the creation.
260 #
261 #      <REPLACE_FS>: This is a special directive, used for Microsoft Azure that
262 #      instructs cloud-init to replace a file system of <FS_TYPE>. NOTE:
263 #      unless you define a label, this requires the use of the 'any' partition
264 #      directive.
265 #
266 # Behavior Caveat: The default behavior is to _check_ if the file system exists.
267 #      If a file system matches the specification, then the operation is a no-op.

```

1.8.20 Register Red Hat Subscription

```

1  #cloud-config
2
3  # register your Red Hat Enterprise Linux based operating system
4  #
5  # this cloud-init plugin is capable of registering by username
6  # and password *or* activation and org. Following a successfully
7  # registration you can:
8  #   - auto-attach subscriptions
9  #   - set the service level
10 #   - add subscriptions based on its pool ID
11 #   - enable yum repositories based on its repo id
12 #   - disable yum repositories based on its repo id
13 #   - alter the rhsm_baseurl and server-hostname in the
14 #     /etc/rhsm/rhs.conf file
15
16 rh_subscription:
17   username: joe@foo.bar
18
19   ## Quote your password if it has symbols to be safe
20   password: '1234abcd'
21
22   ## If you prefer, you can use the activation key and
23   ## org instead of username and password. Be sure to
24   ## comment out username and password
25
26   #activation-key: foobar
27   #org: 12345
28
29   ## Uncomment to auto-attach subscriptions to your system
30   #auto-attach: True
31
32   ## Uncomment to set the service level for your
33   ##   subscriptions
34   #service-level: self-support
35
36   ## Uncomment to add pools (needs to be a list of IDs)
37   #add-pool: []
38
39   ## Uncomment to add or remove yum repos
40   ##   (needs to be a list of repo IDs)
41   #enable-repo: []
42   #disable-repo: []
43

```

(continues on next page)

(continued from previous page)

```

44  ## Uncomment to alter the baseurl in /etc/rhsm/rhsm.conf
45  #rhsm-baseurl: http://url
46
47  ## Uncomment to alter the server hostname in
48  ## /etc/rhsm/rhsm.conf
49  #server-hostname: foo.bar.com

```

1.8.21 Configure data sources

```

1  #cloud-config
2
3  # Documentation on data sources configuration options
4  datasource:
5      # Ec2
6      Ec2:
7          # timeout: the timeout value for a request at metadata service
8          timeout : 50
9          # The length in seconds to wait before giving up on the metadata
10         # service. The actual total wait could be up to
11         # len(resolvable_metadata_urls)*timeout
12         max_wait : 120
13
14         #metadata_url: a list of URLs to check for metadata services
15         metadata_urls:
16             - http://169.254.169.254:80
17             - http://instance-data:8773
18
19         MAAS:
20             timeout : 50
21             max_wait : 120
22
23             # there are no default values for metadata_url or oauth credentials
24             # If no credentials are present, non-authed attempts will be made.
25             metadata_url: http://mass-host.localdomain/source
26             consumer_key: Xh234sdkljf
27             token_key: kjfhgb3n
28             token_secret: 24uysdfx1w4
29
30         NoCloud:
31             # default seedfrom is None
32             # if found, then it should contain a url with:
33             #   <url>/user-data and <url>/meta-data
34             # seedfrom: http://my.example.com/i-abcde
35             seedfrom: None
36
37             # fs_label: the label on filesystems to be searched for NoCloud source
38             fs_label: cidata
39
40             # these are optional, but allow you to basically provide a datasource
41             # right here
42             user-data: |
43                 # This is the user-data verbatim
44             meta-data:
45                 instance-id: i-87018aed
46                 local-hostname: myhost.internal

```

(continues on next page)

(continued from previous page)

```

47
48 SmartOS:
49     # For KVM guests:
50     # Smart OS datasource works over a serial console interacting with
51     # a server on the other end. By default, the second serial console is the
52     # device. SmartOS also uses a serial timeout of 60 seconds.
53     serial_device: /dev/ttyS1
54     serial_timeout: 60
55
56     # For LX-Brand Zones guests:
57     # Smart OS datasource works over a socket interacting with
58     # the host on the other end. By default, the socket file is in
59     # the native .zoncontrol directory.
60     metadata_sockfile: /native/.zonecontrol/metadata.sock
61
62     # a list of keys that will not be base64 decoded even if base64_all
63     no_base64_decode: ['root_authorized_keys', 'motd_sys_info',
64                          'iptables_disable']
65     # a plaintext, comma delimited list of keys whose values are b64 encoded
66     base64_keys: []
67     # a boolean indicating that all keys not in 'no_base64_decode' are encoded
68     base64_all: False

```

1.8.22 Create partitions and filesystems

```

1  #cloud-config
2  # Cloud-init supports the creation of simple partition tables and file systems
3  # on devices.
4
5  # Default disk definitions for AWS
6  # -----
7  # (Not implemented yet, but provided for future documentation)
8
9  disk_setup:
10     ephemeral0:
11         table_type: 'mbr'
12         layout: True
13         overwrite: False
14
15  fs_setup:
16     - label: None,
17       filesystem: ext3
18       device: ephemeral0
19       partition: auto
20
21  # Default disk definitions for Microsoft Azure
22  # -----
23
24  device_aliases: {'ephemeral0': '/dev/sdb'}
25  disk_setup:
26     ephemeral0:
27         table_type: mbr
28         layout: True
29         overwrite: False
30

```

(continues on next page)

(continued from previous page)

```

31 fs_setup:
32   - label: ephemeral0
33     filesystem: ext4
34     device: ephemeral0.1
35     replace_fs: ntfs
36
37
38 # Data disks definitions for Microsoft Azure
39 # -----
40
41 disk_setup:
42   /dev/disk/azure/scsi1/lun0:
43     table_type: gpt
44     layout: True
45     overwrite: True
46
47 fs_setup:
48   - device: /dev/disk/azure/scsi1/lun0
49     partition: 1
50     filesystem: ext4
51
52
53 # Default disk definitions for SmartOS
54 # -----
55
56 device_aliases: {'ephemeral0': '/dev/vdb'}
57 disk_setup:
58   ephemeral0:
59     table_type: mbr
60     layout: False
61     overwrite: False
62
63 fs_setup:
64   - label: ephemeral0
65     filesystem: ext4
66     device: ephemeral0.0
67
68 # Caveat for SmartOS: if ephemeral disk is not defined, then the disk will
69 #   not be automatically added to the mounts.
70
71
72 # The default definition is used to make sure that the ephemeral storage is
73 # setup properly.
74
75 # "disk_setup": disk partitioning
76 # -----
77
78 # The disk_setup directive instructs Cloud-init to partition a disk. The format is:
79
80 disk_setup:
81   ephemeral0:
82     table_type: 'mbr'
83     layout: true
84   /dev/xvdh:
85     table_type: 'mbr'
86     layout:
87       - 33

```

(continues on next page)

(continued from previous page)

```

88     - [33, 82]
89     - 33
90     overwrite: True
91
92     # The format is a list of dicts of dicts. The first value is the name of the
93     # device and the subsequent values define how to create and layout the
94     # partition.
95     # The general format is:
96     #     disk_setup:
97     #         <DEVICE>:
98     #             table_type: 'mbr'
99     #             layout: <LAYOUT|BOOL>
100    #             overwrite: <BOOL>
101    #
102    # Where:
103    #     <DEVICE>: The name of the device. 'ephemeralX' and 'swap' are special
104    #               values which are specific to the cloud. For these devices
105    #               Cloud-init will look up what the real devices is and then
106    #               use it.
107    #
108    #               For other devices, the kernel device name is used. At this
109    #               time only simply kernel devices are supported, meaning
110    #               that device mapper and other targets may not work.
111    #
112    #               Note: At this time, there is no handling or setup of
113    #               device mapper targets.
114    #
115    #     table_type=<TYPE>: Currently the following are supported:
116    #         'mbr': default and setups a MS-DOS partition table
117    #         'gpt': setups a GPT partition table
118    #
119    #               Note: At this time only 'mbr' and 'gpt' partition tables
120    #               are allowed. It is anticipated in the future that
121    #               we'll also have "RAID" to create a mdadm RAID.
122    #
123    #     layout={...}: The device layout. This is a list of values, with the
124    #                   percentage of disk that partition will take.
125    #                   Valid options are:
126    #                   [<SIZE>, [<SIZE>, <PART_TYPE>]]
127    #
128    #                   Where <SIZE> is the _percentage_ of the disk to use, while
129    #                   <PART_TYPE> is the numerical value of the partition type.
130    #
131    #                   The following setups two partitions, with the first
132    #                   partition having a swap label, taking 1/3 of the disk space
133    #                   and the remainder being used as the second partition.
134    #                   /dev/xvdb:
135    #                       table_type: 'mbr'
136    #                       layout:
137    #                           - [33, 82]
138    #                           - 66
139    #                       overwrite: True
140    #
141    #                   When layout is "true" it means single partition the entire
142    #                   device.
143    #
144    #                   When layout is "false" it means don't partition or ignore

```

(continues on next page)

(continued from previous page)

```

145 #           existing partitioning.
146 #
147 #           If layout is set to "true" and overwrite is set to "false",
148 #           it will skip partitioning the device without a failure.
149 #
150 #           overwrite=<BOOL>: This describes whether to ride with saftey's on and
151 #           everything holstered.
152 #
153 #           'false' is the default, which means that:
154 #               1. The device will be checked for a partition table
155 #               2. The device will be checked for a file system
156 #               3. If either a partition of file system is found, then
157 #                  the operation will be _skipped_.
158 #
159 #           'true' is cowboy mode. There are no checks and things are
160 #           done blindly. USE with caution, you can do things you
161 #           really, really don't want to do.
162 #
163 #
164 # fs_setup: Setup the file system
165 # -----
166 #
167 # fs_setup describes the how the file systems are supposed to look.
168
169 fs_setup:
170 - label: ephemeral0
171   filesystem: 'ext3'
172   device: 'ephemeral0'
173   partition: 'auto'
174 - label: mylabl2
175   filesystem: 'ext4'
176   device: '/dev/xvda1'
177 - cmd: mkfs -t %(filesystem)s -L %(label)s %(device)s
178   label: mylabl3
179   filesystem: 'btrfs'
180   device: '/dev/xvda1'
181
182 # The general format is:
183 #   fs_setup:
184 #     - label: <LABEL>
185 #       filesystem: <FS_TYPE>
186 #       device: <DEVICE>
187 #       partition: <PART_VALUE>
188 #       overwrite: <OVERWRITE>
189 #       replace_fs: <FS_TYPE>
190 #
191 # Where:
192 #   <LABEL>: The file system label to be used. If set to None, no label is
193 #   used.
194 #
195 #   <FS_TYPE>: The file system type. It is assumed that the there
196 #   will be a "mkfs.<FS_TYPE>" that behaves likes "mkfs". On a standard
197 #   Ubuntu Cloud Image, this means that you have the option of ext{2,3,4},
198 #   and vfat by default.
199 #
200 #   <DEVICE>: The device name. Special names of 'ephemeralX' or 'swap'
201 #   are allowed and the actual device is acquired from the cloud datasource.

```

(continues on next page)

(continued from previous page)

```

202 # When using 'ephemeralX' (i.e. ephemeral0), make sure to leave the
203 # label as 'ephemeralX' otherwise there may be issues with the mounting
204 # of the ephemeral storage layer.
205 #
206 # If you define the device as 'ephemeralX.Y' then Y will be interpreted
207 # as a partition value. However, ephemeralX.0 is the _same_ as ephemeralX.
208 #
209 # <PART_VALUE>:
210 # Partition definitions are overwritten if you use the '<DEVICE>.Y' notation.
211 #
212 # The valid options are:
213 # "auto/any": tell cloud-init not to care whether there is a partition
214 # or not. Auto will use the first partition that does not contain a
215 # file system already. In the absence of a partition table, it will
216 # put it directly on the disk.
217 #
218 # "auto": If a file system that matches the specification in terms of
219 # label, type and device, then cloud-init will skip the creation of
220 # the file system.
221 #
222 # "any": If a file system that matches the file system type and device,
223 # then cloud-init will skip the creation of the file system.
224 #
225 # Devices are selected based on first-detected, starting with partitions
226 # and then the raw disk. Consider the following:
227 # NAME      FSTYPE LABEL
228 # xvdb
229 # |-xvdb1  ext4
230 # |-xvdb2
231 # |-xvdb3  btrfs  test
232 # \-xvdb4  ext4   test
233 #
234 # If you ask for 'auto', label of 'test', and file system of 'ext4'
235 # then cloud-init will select the 2nd partition, even though there
236 # is a partition match at the 4th partition.
237 #
238 # If you ask for 'any' and a label of 'test', then cloud-init will
239 # select the 1st partition.
240 #
241 # If you ask for 'auto' and don't define label, then cloud-init will
242 # select the 1st partition.
243 #
244 # In general, if you have a specific partition configuration in mind,
245 # you should define either the device or the partition number. 'auto'
246 # and 'any' are specifically intended for formatting ephemeral storage or
247 # for simple schemes.
248 #
249 # "none": Put the file system directly on the device.
250 #
251 # <NUM>: where NUM is the actual partition number.
252 #
253 # <OVERWRITE>: Defines whether or not to overwrite any existing
254 # filesystem.
255 #
256 # "true": Indiscriminately destroy any pre-existing file system. Use at
257 # your own peril.
258 #

```

(continues on next page)

(continued from previous page)

```

259 # "false": If an existing file system exists, skip the creation.
260 #
261 # <REPLACE_FS>: This is a special directive, used for Microsoft Azure that
262 # instructs cloud-init to replace a file system of <FS_TYPE>. NOTE:
263 # unless you define a label, this requires the use of the 'any' partition
264 # directive.
265 #
266 # Behavior Caveat: The default behavior is to _check_ if the file system exists.
267 # If a file system matches the specification, then the operation is a no-op.

```

1.8.23 Grow partitions

```

1 #cloud-config
2 #
3 # growpart entry is a dict, if it is not present at all
4 # in config, then the default is used ({'mode': 'auto', 'devices': ['/']})
5 #
6 # mode:
7 #   values:
8 #     * auto: use any option possible (any available)
9 #             if none are available, do not warn, but debug.
10 #    * growpart: use growpart to grow partitions
11 #                if growpart is not available, this is an error.
12 #    * off, false
13 #
14 # devices:
15 #   a list of things to resize.
16 #   items can be filesystem paths or devices (in /dev)
17 #   examples:
18 #     devices: [/, /dev/vdb1]
19 #
20 # ignore_growroot_disabled:
21 #   a boolean, default is false.
22 #   if the file /etc/growroot-disabled exists, then cloud-init will not grow
23 #   the root partition. This is to allow a single file to disable both
24 #   cloud-initramfs-growroot and cloud-init's growroot support.
25 #
26 #   true indicates that /etc/growroot-disabled should be ignored
27 #
28 growpart:
29   mode: auto
30   devices: ['/']
31   ignore_growroot_disabled: false

```

1.9 Events and Updates

1.9.1 Events

Cloud-init will fetch and apply cloud and user data configuration upon several event types. The two most common events for cloud-init are when an instance first boots and any subsequent boot thereafter (reboot). In addition to boot events, cloud-init users and vendors are interested in when devices are added. cloud-init currently supports the following event types:

- **BOOT_NEW_INSTANCE**: New instance first boot
- **BOOT**: Any system boot other than ‘BOOT_NEW_INSTANCE’
- **BOOT_LEGACY**: Similar to ‘BOOT’, but applies networking config twice each boot: once during Local stage, then again in Network stage. As this behavior was previously the default behavior, this option exists to prevent regressing such behavior.
- **HOTPLUG**: Dynamic add of a system device

Future work will likely include infrastructure and support for the following events:

- **METADATA_CHANGE**: An instance’s metadata has change
- **USER_REQUEST**: Directed request to update

1.9.2 Datasource Event Support

All *Datasources* by default support the `BOOT_NEW_INSTANCE` event. Each Datasource will declare a set of these events that it is capable of handling. Datasources may not support all event types. In some cases a system may be configured to allow a particular event but may be running on a platform whose datasource cannot support the event.

1.9.3 Configuring Event Updates

Update configuration may be specified via user data, which can be used to enable or disable handling of specific events. This configuration will be honored as long as the events are supported by the datasource. However, configuration will always be applied at first boot, regardless of the user data specified.

Updates

Update policy configuration defines which events are allowed to be handled. This is separate from whether a particular platform or datasource has the capability for such events.

scope: *<name of the scope for event policy>*

The `scope` value is a string which defines under which domain does the event occur. Currently the only one known scope is `network`, though more scopes may be added in the future. Scopes are defined by convention but arbitrary values can be used.

when: *<list of events to handle for a particular scope>*

Each `scope` requires a `when` element to specify which events are to allowed to be handled.

1.9.4 Hotplug

When the hotplug event is supported by the data source and configured in user data, cloud-init will respond to the addition or removal of network interfaces to the system. In addition to fetching and updating the system metadata, cloud-init will also bring up/down the newly added interface.

Warning: Due to its use of systemd sockets, hotplug functionality is currently incompatible with SELinux. This issue is being tracked [on Launchpad](#). Additionally, hotplug support is considered experimental for non-Debian based systems.

1.9.5 Examples

apply network config every boot

On every boot, apply network configuration found in the datasource.

```
# apply network config on every boot
updates:
  network:
    when: ['boot']
```

1.10 Modules

Table of Contents

- *Modules*
 - *APK Configure*
 - *Apt Configure*
 - *Apt Pipelining*
 - *Bootcmd*
 - *Byobu*
 - *CA Certificates*
 - *Chef*
 - *Debug*
 - *Disable EC2 Metadata*
 - *Disk Setup*
 - *Emit Upstart*
 - *Fan*
 - *Final Message*
 - *Foo*
 - *Growpart*
 - *Grub Dpkg*
 - *Install Hotplug*
 - *Keyboard*
 - *Keys to Console*
 - *Landscape*
 - *Locale*
 - *LXD*
 - *Mcollective*

- *Migrator*
- *Mounts*
- *NTP*
- *Package Update Upgrade Install*
- *Phone Home*
- *Power State Change*
- *Puppet*
- *Resizefs*
- *Resolv Conf*
- *Red Hat Subscription*
- *Rightscale Userdata*
- *Rsyslog*
- *Runcmd*
- *Salt Minion*
- *Scripts Per Boot*
- *Scripts Per Instance*
- *Scripts Per Once*
- *Scripts User*
- *Scripts Vendor*
- *Seed Random*
- *Set Hostname*
- *Set Passwords*
- *Snap*
- *Spacewalk*
- *SSH*
 - * *Authorized Keys*
 - * *Host Keys*
- *SSH Authkey Fingerprints*
- *SSH Import Id*
- *Timezone*
- *Ubuntu Advantage*
- *Ubuntu Drivers*
- *Update Etc Hosts*
- *Update Hostname*
- *Users and Groups*

- *Write Files*
- *Yum Add Repo*

1.10.1 APK Configure

Summary: Configure apk repositories file

This module handles configuration of the `/etc/apk/repositories` file.

Note: To ensure that apk configuration is valid yaml, any strings containing special characters, especially `:` should be quoted.

Internal name: `cc_apk_configure`

Module frequency: once-per-instance

Supported distros: alpine

Config schema: `apk_repos:` (object)

preserve_repositories: (boolean) By default, cloud-init will generate a new repositories file `/etc/apk/repositories` based on any valid configuration settings specified within a `apk_repos` section of cloud config. To disable this behavior and preserve the repositories file from the pristine image, set `preserve_repositories` to `true`.

The `preserve_repositories` option overrides all other config keys that would alter `/etc/apk/repositories`.

alpine_repo: (object/null)

base_url: (string) The base URL of an Alpine repository, or mirror, to download official packages from. If not specified then it defaults to `https://alpine.global.ssl.fastly.net/alpine`

community_enabled: (boolean) Whether to add the Community repo to the repositories file. By default the Community repo is not included.

testing_enabled: (boolean) Whether to add the Testing repo to the repositories file. By default the Testing repo is not included. It is only recommended to use the Testing repo on a machine running the Edge version of Alpine as packages installed from Testing may have dependencies that conflict with those in non-Edge Main or Community repos.

version: (string) The Alpine version to use (e.g. `v3.12` or `edge`)

local_repo_base_url: (string) The base URL of an Alpine repository containing unofficial packages

Examples:

```
# Keep the existing /etc/apk/repositories file unaltered.
apk_repos:
  preserve_repositories: true

# --- Example2 ---
# Create repositories file for Alpine v3.12 main and community
# using default mirror site.
apk_repos:
  alpine_repo:
```

(continues on next page)

(continued from previous page)

```
community_enabled: true
version: 'v3.12'

# --- Example3 ---
# Create repositories file for Alpine Edge main, community, and
# testing using a specified mirror site and also a local repo.
apk_repos:
  alpine_repo:
    base_url: 'https://some-alpine-mirror/alpine'
    community_enabled: true
    testing_enabled: true
    version: 'edge'
  local_repo_base_url: 'https://my-local-server/local-alpine'
```

1.10.2 Apt Configure

Summary: Configure apt for the user

This module handles both configuration of apt options and adding source lists. There are configuration options such as `apt_get_wrapper` and `apt_get_command` that control how cloud-init invokes apt-get. These configuration options are handled on a per-distro basis, so consult documentation for cloud-init's distro support for instructions on using these config options.

Note: To ensure that apt configuration is valid yaml, any strings containing special characters, especially `:` should be quoted.

Note: For more information about apt configuration, see the `Additional apt configuration example`.

Internal name: `cc_apt_configure`

Module frequency: once-per-instance

Supported distros: ubuntu, debian

Config schema: `apt:` (object)

preserve_sources_list: (boolean) By default, cloud-init will generate a new sources list in `/etc/apt/sources.list.d` based on any changes specified in cloud config. To disable this behavior and preserve the sources list from the pristine image, set `preserve_sources_list` to `true`.

The `preserve_sources_list` option overrides all other config keys that would alter `sources.list` or `sources.list.d`, **except** for additional sources to be added to `sources.list.d`.

disable_suites: (array of string) Entries in the sources list can be disabled using `disable_suites`, which takes a list of suites to be disabled. If the string `$RELEASE` is present in a suite in the `disable_suites` list, it will be replaced with the release name. If a suite specified in `disable_suites` is not present in `sources.list` it will be ignored. For convenience, several aliases are provided for “`disable_suites`”:

- `updates => $RELEASE-updates`
- `backports => $RELEASE-backports`
- `security => $RELEASE-security`

- `proposed => $RELEASE-proposed`
- `release => $RELEASE.`

When a suite is disabled using `disable_suites`, its entry in `sources.list` is not deleted; it is just commented out.

primary: (array of object) The primary and security archive mirrors can be specified using the `primary` and `security` keys, respectively. Both the `primary` and `security` keys take a list of configs, allowing mirrors to be specified on a per-architecture basis. Each config is a dictionary which must have an entry for `arches`, specifying which architectures that config entry is for. The keyword `default` applies to any architecture not explicitly listed. The mirror url can be specified with the `uri` key, or a list of mirrors to check can be provided in order, with the first mirror that can be resolved being selected. This allows the same configuration to be used in different environment, with different hosts used for a local APT mirror. If no mirror is provided by `uri` or `search`, `search_dns` may be used to search for dns names in the format `<distro>-mirror` in each of the following:

- fqdn of this host per cloud metadata,
- `localdomain`,
- domains listed in `/etc/resolv.conf`.

If there is a dns entry for `<distro>-mirror`, then it is assumed that there is a distro mirror at `http://<distro>-mirror.<domain>/<distro>`. If the `primary` key is defined, but not the `security` key, then then configuration for `primary` is also used for `security`. If `search_dns` is used for the `security` key, the search pattern will be `<distro>-security-mirror`.

Each mirror may also specify a key to import via any of the following optional keys:

- `keyid`: a key to import via shortid or fingerprint.
- `key`: a raw PGP key.
- `keyserver`: alternate keyserver to pull `keyid` key from.

If no mirrors are specified, or all lookups fail, then default mirrors defined in the datasource are used. If none are present in the datasource either the following defaults are used:

- `primary => http://archive.ubuntu.com/ubuntu.`
- `security => http://security.ubuntu.com/ubuntu`

Each item in **primary** list supports the following keys:

- arches:** (array of string)
- uri:** (string)
- search:** (array of string)
- search_dns:** (boolean)
- keyid:** (string)
- key:** (string)
- keyserver:** (string)

security: (array of object) Please refer to the primary config documentation

Each item in **security** list supports the following keys:

arches: (array of string)

uri: (string)

search: (array of string)

search_dns: (boolean)

keyid: (string)

key: (string)

keyserver: (string)

add_apt_repo_match: (string) All source entries in `apt-sources` that match regex in `add_apt_repo_match` will be added to the system using `add-apt-repository`. If `add_apt_repo_match` is not specified, it defaults to `^[\w-]+: \w`

debconf_selections: (object) Debconf additional configurations can be specified as a dictionary under the `debconf_selections` config key, with each key in the dict representing a different set of configurations. The value of each key must be a string containing all the debconf configurations that must be applied. We will bundle all of the values and pass them to `debconf-set-selections`. Therefore, each value line must be a valid entry for `debconf-set-selections`, meaning that they must possess for distinct fields:

```
pkgname question type answer
```

Where:

- `pkgname` is the name of the package.
- `question` the name of the questions.
- `type` is the type of question.
- `answer` is the value used to answer the question.

For example: `ippackage ippackage/ip string 127.0.0.1`

^.+\$.: (string)

sources_list: (string) Specifies a custom template for rendering `sources.list`. If no `sources_list` template is given, cloud-init will use sane default. Within this template, the following strings will be replaced with the appropriate values:

- `$MIRROR`
- `$RELEASE`
- `$PRIMARY`
- `$SECURITY`
- `$KEY_FILE`

conf: (string) Specify configuration for apt, such as proxy configuration. This configuration is specified as a string. For multiline APT configuration, make sure to follow yaml syntax.

https_proxy: (string) More convenient way to specify https APT proxy. https proxy url is specified in the format `https://[[user][:pass]@]host[:port]/`.

http_proxy: (string) More convenient way to specify http APT proxy. http proxy url is specified in the format `http://[[user][:pass]@]host[:port]/`.

proxy: (string) Alias for defining a http APT proxy.

ftp_proxy: (string) More convenient way to specify ftp APT proxy. ftp proxy url is specified in the format `ftp://[[user][:pass]@]host[:port]/`.

sources: (object) Source list entries can be specified as a dictionary under the `sources` config key, with each key in the dict representing a different source file. The key of each source entry will be used as an id that can be referenced in other config entries, as well as the filename for the source's configuration under `/etc/apt/sources.list.d`. If the name does not end with `.list`, it will be appended. If there is no configuration for a key in `sources`, no file will be written, but the key may still be referred to as an id in other `sources` entries.

Each entry under **sources** is a dictionary which may contain any of the following optional keys:

- **source:** a `sources.list` entry (some variable replacements apply).
- **keyid:** a key to import via shortid or fingerprint.
- **key:** a raw PGP key.
- **keyserver:** alternate keyserver to pull `keyid` key from.
- **filename:** specify the name of the list file

The `source` key supports variable replacements for the following strings:

- `$MIRROR`
- `$PRIMARY`
- `$SECURITY`
- `$RELEASE`
- `$KEY_FILE`

^.+\$.: (object)

source: (string)

keyid: (string)

key: (string)

keyserver: (string)

filename: (string)

Examples:

```
apt:
  preserve_sources_list: false
  disable_suites:
    - $RELEASE-updates
    - backports
    - $RELEASE
    - mysuite
  primary:
    - arches:
        - amd64
        - i386
        - default
      uri: 'http://us.archive.ubuntu.com/ubuntu'
      search:
        - 'http://cool.but-sometimes-unreachable.com/ubuntu'
        - 'http://us.archive.ubuntu.com/ubuntu'
```

(continues on next page)

(continued from previous page)

```

    search_dns: false
  - arches:
    - s390x
    - arm64
    uri: 'http://archive-to-use-for-arm64.example.com/ubuntu'

security:
  - arches:
    - default
    search_dns: true
sources_list: |
deb $MIRROR $RELEASE main restricted
deb-src $MIRROR $RELEASE main restricted
deb $PRIMARY $RELEASE universe restricted
deb $SECURITY $RELEASE-security multiverse
debconf_selections:
  set1: the-package the-package/some-flag boolean true
conf: |
  APT {
    Get {
      Assume-Yes 'true';
      Fix-Broken 'true';
    }
  }
proxy: 'http://[[user][:pass]@]host[:port]/'
http_proxy: 'http://[[user][:pass]@]host[:port]/'
ftp_proxy: 'ftp://[[user][:pass]@]host[:port]/'
https_proxy: 'https://[[user][:pass]@]host[:port]/'
sources:
  source1:
    keyid: 'keyid'
    keyserver: 'keyserverurl'
    source: 'deb [signed-by=$KEY_FILE] http://<url>/ bionic main'
  source2:
    source: 'ppa:<ppa-name>'
  source3:
    source: 'deb $MIRROR $RELEASE multiverse'
  key: |
    -----BEGIN PGP PUBLIC KEY BLOCK-----
    <key data>
    -----END PGP PUBLIC KEY BLOCK-----

```

1.10.3 Apt Pipelining

Summary: Configure apt pipelining

This module configures apt's `Acquire::http::Pipeline-Depth` option, which controls how apt handles HTTP pipelining. It may be useful for pipelining to be disabled, because some web servers, such as S3 do not pipeline properly (LP: #948461).

Value configuration options for this module are:

- `false` (Default): disable pipelining altogether
- `none`, `unchanged`, or `os`: use distro default
- `<number>`: Manually specify pipeline depth. This is not recommended.

Internal name: cc_apt_pipelining

Module frequency: once-per-instance

Supported distros: ubuntu, debian

Config schema: apt_pipelining: (integer/boolean/string)

Examples:

```

apt_pipelining: false
# --- Example2 ---
apt_pipelining: none
# --- Example3 ---
apt_pipelining: unchanged
# --- Example4 ---
apt_pipelining: os
# --- Example5 ---
apt_pipelining: 3

```

1.10.4 Bootcmd

Summary: Run arbitrary commands early in the boot process

This module runs arbitrary commands very early in the boot process, only slightly after a boothook would run. This is very similar to a boothook, but more user friendly. The environment variable `INSTANCE_ID` will be set to the current instance id for all run commands. Commands can be specified either as lists or strings. For invocation details, see `runcmd`.

Note: bootcmd should only be used for things that could not be done later in the boot process.

Note: when writing files, do not use `/tmp` dir as it races with `systemd-tmpfiles-clean` LP: #1707222. Use `/run/somedir` instead.

Internal name: cc_bootcmd

Module frequency: always

Supported distros: all

Config schema: bootcmd: (array of (array of string)/(string))

Examples:

```

bootcmd:
- echo 192.168.1.130 us.archive.ubuntu.com > /etc/hosts
- [ cloud-init-per, once, mymkfs, mkfs, /dev/vdb ]

```

1.10.5 Byobu

Summary: Enable/disable byobu system wide and for default user

This module controls whether byobu is enabled or disabled system wide and for the default system user. If byobu is to be enabled, this module will ensure it is installed. Likewise, if it is to be disabled, it will be removed if installed.

Valid configuration options for this module are:

- `enable-system`: enable byobu system wide
- `enable-user`: enable byobu for the default user
- `disable-system`: disable byobu system wide
- `disable-user`: disable byobu for the default user
- `enable`: enable byobu both system wide and for default user
- `disable`: disable byobu for all users
- `user`: alias for `enable-user`
- `system`: alias for `enable-system`

Internal name: `cc_byobu`

Module frequency: once-per-instance

Supported distros: ubuntu, debian

Config schema: `byobu_by_default`: (string)

Examples:

```
byobu_by_default: enable-user
# --- Example2 ---
byobu_by_default: disable-system
```

1.10.6 CA Certificates

Summary: Add ca certificates

This module adds CA certificates to `/etc/ca-certificates.conf` and updates the ssl cert cache using `update-ca-certificates`. The default certificates can be removed from the system with the configuration option `remove_defaults`.

Note: certificates must be specified using valid yaml. in order to specify a multiline certificate, the yaml multiline list syntax must be used

Note: For Alpine Linux the “`remove_defaults`” functionality works if the `ca-certificates` package is installed but not if the `ca-certificates-bundle` package is installed.

Internal name: `cc_ca_certs`

Module frequency: once-per-instance

Supported distros: alpine, debian, ubuntu, rhel

Config schema: `ca_certs`: (object)

remove-defaults: (boolean) Deprecated key name. Use `remove_defaults` instead.

remove_defaults: (boolean) Remove default CA certificates if true. Default: false

trusted: (array of string) List of trusted CA certificates to add.

ca-certs: (object)

remove-defaults: (boolean) Deprecated key name. Use `remove_defaults` instead.

remove_defaults: (boolean) Remove default CA certificates if true. Default: false

trusted: (array of string) List of trusted CA certificates to add.

Examples:

```
ca_certs:
  remove_defaults: true
  trusted:
    - single_line_cert
    - |
      -----BEGIN CERTIFICATE-----
      YOUR-ORGS-TRUSTED-CA-CERT-HERE
      -----END CERTIFICATE-----
```

1.10.7 Chef

Summary: module that configures, starts and installs chef

This module enables chef to be installed (from packages, gems, or from omnibus). Before this occurs, chef configuration is written to disk (`validation.pem`, `client.pem`, `firstboot.json`, `client.rb`), and required directories are created (`/etc/chef` and `/var/log/chef` and so-on). If configured, chef will be installed and started in either daemon or non-daemon mode. If run in non-daemon mode, post run actions are executed to do finishing activities such as removing `validation.pem`.

Internal name: `cc_chef`

Module frequency: always

Supported distros: all

Config schema: `chef:` (object)

directories: (array of string) Create the necessary directories for chef to run. By default, it creates the following directories:

- `/etc/chef`
- `/var/log/chef`
- `/var/lib/chef`
- `/var/cache/chef`
- `/var/backups/chef`
- `/var/run/chef`

validation_cert: (string) Optional string to be written to file `validation_key`. Special value `system` means set use existing file.

validation_key: (string) Optional path for `validation_cert`. default to `/etc/chef/validation.pem`

firstboot_path: (string) Path to write `run_list` and `initial_attributes` keys that should also be present in this configuration, defaults to `/etc/chef/firstboot.json`

exec: (boolean) Set true if we should run or not run chef (defaults to false, unless a gem installed is requested where this will then default to true).

client_key: (string) Optional path for `client_cert`. Default to `/etc/chef/client.pem`.

encrypted_data_bag_secret: (string) Specifies the location of the secret key used by chef to encrypt data items. By default, this path is set to null, meaning that chef will have to look at the path `/etc/chef/encrypted_data_bag_secret` for it.

environment: (string) Specifies which environment chef will use. By default, it will use the `_default` configuration.

file_backup_path: (string) Specifies the location in which backup files are stored. By default, it uses the `/var/backups/chef` location.

file_cache_path: (string) Specifies the location in which chef cache files will be saved. By default, it uses the `/var/cache/chef` location.

json_attribs: (string) Specifies the location in which some chef json data is stored. By default, it uses the `/etc/chef/firstboot.json` location.

log_level: (string) Defines the level of logging to be stored in the log file. By default this value is set to `:info`.

log_location: (string) Specifies the location of the chef log file. By default, the location is specified at `/var/log/chef/client.log`.

node_name: (string) The name of the node to run. By default, we will use the instance id as the node name.

omnibus_url: (string) Omnibus URL if chef should be installed through Omnibus. By default, it uses the `https://www.chef.io/chef/install.sh`.

omnibus_url_retries: (integer) The number of retries that will be attempted to reach the Omnibus URL. Default is 5.

omnibus_version: (string) Optional version string to require for omnibus install.

pid_file: (string) The location in which a process identification number (pid) is saved. By default, it saves in the `/var/run/chef/client.pid` location.

server_url: (string) The URL for the chef server

show_time: (boolean) Show time in chef logs

ssl_verify_mode: (string) Set the verify mode for HTTPS requests. We can have two possible values for this parameter:

- `:verify_none`: No validation of SSL certificates.
- `:verify_peer`: Validate all SSL certificates.

By default, the parameter is set as `:verify_none`.

validation_name: (string) The name of the chef-validator key that Chef Infra Client uses to access the Chef Infra Server during the initial Chef Infra Client run.

force_install: (boolean) If set to `true`, forces chef installation, even if it is already installed.

initial_attributes: (object of string) Specify a list of initial attributes used by the cookbooks.

install_type: (string) The type of installation for chef. It can be one of the following values:

- `packages`
- `gems`
- `omnibus`

run_list: (array of string) A run list for a first boot json.

chef_license: (string) string that indicates if user accepts or not license related to some of chef products

Examples:

```
chef:
  directories:
    - /etc/chef
    - /var/log/chef
  validation_cert: system
  install_type: omnibus
  initial_attributes:
    apache:
      prefork:
        maxclients: 100
      keepalive: off
  run_list:
    - recipe[apache2]
    - role[db]
  encrypted_data_bag_secret: /etc/chef/encrypted_data_bag_secret
  environment: _default
  log_level: :auto
  omnibus_url_retries: 2
  server_url: https://chef.yourorg.com:4000
  ssl_verify_mode: :verify_peer
  validation_name: yourorg-validator
```

1.10.8 Debug

Summary: Helper to debug cloud-init *internal* datastructures

This module will enable for outputting various internal information that cloud-init sources provide to either a file or to the output console/log location that this cloud-init has been configured with when running.

Note: Log configurations are not output.

Internal name: cc_debug

Module frequency: once-per-instance

Supported distros: all

Config schema: debug: (object)

verbose: (boolean) Should always be true for this module

output: (string) Location to write output. Defaults to console + log

Examples:

```
debug:
  verbose: true
  output: /tmp/my_debug.log
```

1.10.9 Disable EC2 Metadata

Summary: Disable AWS EC2 Metadata

This module can disable the ec2 datasource by rejecting the route to 169.254.169.254, the usual route to the datasource. This module is disabled by default.

Internal name: `cc_disable_ec2_metadata`

Module frequency: `always`

Supported distros: `all`

Config schema: `disable_ec2_metadata`: (boolean) Set true to disable IPv4 routes to EC2 metadata. Default: `false`.

Examples:

```
disable_ec2_metadata: true
```

1.10.10 Disk Setup

Summary: Configure partitions and filesystems

This module is able to configure simple partition tables and filesystems.

Note: for more detail about configuration options for disk setup, see the disk setup example

For convenience, aliases can be specified for disks using the `device_aliases` config key, which takes a dictionary of alias: path mappings. There are automatic aliases for `swap` and `ephemeral<X>`, where `swap` will always refer to the active swap partition and `ephemeral<X>` will refer to the block device of the ephemeral image.

Disk partitioning is done using the `disk_setup` directive. This config directive accepts a dictionary where each key is either a path to a block device or an alias specified in `device_aliases`, and each value is the configuration options for the device. File system configuration is done using the `fs_setup` directive. This config directive accepts a list of filesystem configs.

Internal name: `cc_disk_setup`

Module frequency: `once-per-instance`

Supported distros: `all`

Config schema: `device_aliases`: (object)

<alias_name>: (string) Path to disk to be aliased by this name.

disk_setup: (object)

<alias name/path>: (object)

table_type: (string) Specifies the partition table type, either `mbr` or `gpt`. Default: `mbr`.

layout: (string/boolean/array) If set to `true`, a single partition using all the space on the device will be created. If set to `false`, no partitions will be created. If set to `remove`, any existing partition table will be purged. Partitions can be specified by providing a list to `layout`, where each entry in the list is either a size or a list containing a size and the numerical value for a partition type. The size for partitions is specified in **percentage** of disk space, not in bytes (e.g. a size of 33 would take up 1/3 of the disk space). Default: `false`.

overwrite: (boolean) Controls whether this module tries to be safe about writing partition tables or not. If `overwrite: false` is set, the device will be checked for a partition table and for a file system and if either is found, the operation will be skipped. If `overwrite: true` is set, no checks will be performed. Using `overwrite: true`

is **dangerous** and can lead to data loss, so double check that the correct device has been specified if using this option. Default: `false`

fs_setup: (array of object)

Each item in **fs_setup** list supports the following keys:

label: (string) Label for the filesystem.

filesystem: (string) Filesystem type to create. E.g., `ext4` or `btrfs`

device: (string) Specified either as a path or as an alias in the format `<alias name>.<y>` where `<y>` denotes the partition number on the device. If specifying device using the `<device name>.<partition number>` format, the value of `partition` will be overwritten.

partition: (string/integer) The partition can be specified by setting `partition` to the desired partition number. The `partition` option may also be set to `auto`, in which this module will search for the existence of a filesystem matching the `label`, `type` and `device` of the `fs_setup` entry and will skip creating the filesystem if one is found. The `partition` option may also be set to `any`, in which case any file system that matches `type` and `device` will cause this module to skip filesystem creation for the `fs_setup` entry, regardless of `label` matching or not. To write a filesystem directly to a device, use `partition: none`. `partition: none` will **always** write the filesystem, even when the `label` and `filesystem` are matched, and `overwrite` is `false`.

overwrite: (boolean) If `true`, overwrite any existing filesystem. Using `overwrite: true` for filesystems is **dangerous** and can lead to data loss, so double check the entry in `fs_setup`. Default: `false`

replace_fs: (string) Ignored unless `partition` is `auto` or `any`. Default `false`.

extra_opts: (array/string of string) Optional options to pass to the filesystem creation command. Ignored if you using `cmd` directly.

cmd: (array/string of string) Optional command to run to create the filesystem. Can include string substitutions of the other `fs_setup` config keys. This is only necessary if you need to override the default command.

Examples:

```
device_aliases:
  my_alias: /dev/sdb
disk_setup:
  my_alias:
    table_type: gpt
    layout: [50, 50]
    overwrite: true
fs_setup:
- label: fs1
  filesystem: ext4
  device: my_alias.1
  cmd: mkfs -t %(filesystem)s -L %(label)s %(device)s
- label: fs2
  device: my_alias.2
  filesystem: ext4
mounts:
- ["my_alias.1", "/mnt1"]
- ["my_alias.2", "/mnt2"]
```

1.10.11 Emit Upstart

Summary: emit upstart configuration

Emit upstart configuration for cloud-init modules on upstart based systems. No user configuration should be required.

Internal name: `cc_emit_upstart`

Module frequency: always

Supported distros: ubuntu, debian

1.10.12 Fan

Summary: configure ubuntu fan networking

This module installs, configures and starts the ubuntu fan network system. For more information about Ubuntu Fan, see: <https://wiki.ubuntu.com/FanNetworking>.

If cloud-init sees a fan entry in cloud-config it will:

- write `config_path` with the contents of the `config` key
- install the package `ubuntu-fan` if it is not installed
- ensure the service is started (or restarted if was previously running)

Internal name: `cc_fan`

Module frequency: per instance

Supported distros: ubuntu

Config keys:

```
fan:
  config: |
    # fan 240
    10.0.0.0/8 eth0/16 dhcp
    10.0.0.0/8 eth1/16 dhcp off
    # fan 241
    241.0.0.0/8 eth0/16 dhcp
  config_path: /etc/network/fan
```

1.10.13 Final Message

Summary: output final message when cloud-init has finished

This module configures the final message that cloud-init writes. The message is specified as a jinja template with the following variables set:

- `version`: cloud-init version
- `timestamp`: time at cloud-init finish
- `datasource`: cloud-init data source
- `uptime`: system uptime

Internal name: `cc_final_message`

Module frequency: always

Supported distros: all

Config keys:

```
final_message: <message>
```

1.10.14 Foo

Summary: example module

Example to show module structure. Does not do anything.

Internal name: `cc_foo`

Module frequency: per instance

Supported distros: all

1.10.15 Growpart

Summary: grow partitions

Growpart resizes partitions to fill the available disk space. This is useful for cloud instances with a larger amount of disk space available than the pristine image uses, as it allows the instance to automatically make use of the extra space.

The devices on which to run growpart are specified as a list under the `devices` key. Each entry in the devices list can be either the path to the device's mountpoint in the filesystem or a path to the block device in `/dev`.

The utility to use for resizing can be selected using the `mode` config key. If the `mode` key is set to `auto`, then any available utility (either `growpart` or BSD `gpart`) will be used. If neither utility is available, no error will be raised. If `mode` is set to `growpart`, then the `growpart` utility will be used. If this utility is not available on the system, this will result in an error. If `mode` is set to `off` or `false`, then `cc_growpart` will take no action.

There is some functionality overlap between this module and the `growroot` functionality of `cloud-initramfs-tools`. However, there are some situations where one tool is able to function and the other is not. The default configuration for both should work for most cloud instances. To explicitly prevent `cloud-initramfs-tools` from running `growroot`, the file `/etc/growroot-disabled` can be created. By default, both `growroot` and `cc_growpart` will check for the existence of this file and will not run if it is present. However, this file can be ignored for `cc_growpart` by setting `ignore_growroot_disabled` to `true`. For more information on `cloud-initramfs-tools` see: <https://launchpad.net/cloud-initramfs-tools>

Growpart is enabled by default on the root partition. The default config for growpart is:

```
growpart:
  mode: auto
  devices: ["/"]
  ignore_growroot_disabled: false
```

Internal name: `cc_growpart`

Module frequency: always

Supported distros: all

Config keys:

```
growpart:
  mode: <auto/growpart/off/false>
  devices:
    - "/"
    - "/dev/vdb1"
  ignore_growroot_disabled: <true/false>
```

1.10.16 Grub Dpkg

Summary: configure grub debconf installation device

Configure which device is used as the target for grub installation. This module should work correctly by default without any user configuration. It can be enabled/disabled using the `enabled` config key in the `grub_dpkg` config dict. The global config key `grub-dpkg` is an alias for `grub_dpkg`. If no installation device is specified this module will execute `grub-probe` to determine which disk the `/boot` directory is associated with.

The value which is placed into the debconf database is in the format which the grub postinstall script expects. Normally, this is a `/dev/disk/by-id/` value, but we do fallback to the plain disk name if a by-id name is not present.

If this module is executed inside a container, then the debconf database is seeded with empty values, and `install_devices_empty` is set to true.

Internal name: `cc_grub_dpkg`

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:

```
grub_dpkg:
  enabled: <true/false>
  grub-pc/install_devices: <devices>
  grub-pc/install_devices_empty: <devices>
grub-dpkg: (alias for grub_dpkg)
```

1.10.17 Install Hotplug

Summary: Install hotplug if supported and enabled

This module will install the udev rules to enable hotplug if supported by the datasource and enabled in the userdata. The udev rules will be installed as `/etc/udev/rules.d/10-cloud-init-hook-hotplug.rules`.

When hotplug is enabled, newly added network devices will be added to the system by cloud-init. After udev detects the event, cloud-init will refresh the instance metadata from the datasource, detect the device in the updated metadata, then apply the updated network configuration.

Currently supported datasources: Openstack, EC2

Internal name: `cc_install_hotplug`

Module frequency: once-per-instance

Supported distros: all

Config schema: `updates:` (object)

`network:` (object)

`when:` (array of string)

Examples:

```
# Enable hotplug of network devices
updates:
  network:
    when: ["hotplug"]

# --- Example2 ---
# Enable network hotplug alongside boot event
updates:
  network:
    when: ["boot", "hotplug"]
```

1.10.18 Keyboard

Summary: Set keyboard layout

Handle keyboard configuration.

Internal name: cc_keyboard

Module frequency: once-per-instance

Supported distros: arch, debian, ubuntu, almalinux, amazon, centos, cloudlinux, eurolinux, fedora, miraclelinux, openEuler, photon, rhel, rocky, virtuozone, opensuse, sles

Config schema: **keyboard:** (object)

layout: (string) Required. Keyboard layout. Corresponds to XKBLAYOUT.

model: (string) Optional. Keyboard model. Corresponds to XKBMODEL.

variant: (string) Optional. Keyboard variant. Corresponds to XKBVARIANT.

options: (string) Optional. Keyboard options. Corresponds to XKBOPTIONS.

Examples:

```
# Set keyboard layout to "us"
keyboard:
  layout: us

# --- Example2 ---
# Set specific keyboard layout, model, variant, options
keyboard:
  layout: de
  model: pc105
  variant: nodeadkeys
  options: compose:rwin
```

1.10.19 Keys to Console

Summary: control which SSH host keys may be written to console

For security reasons it may be desirable not to write SSH host keys and their fingerprints to the console. To avoid either being written to the console the `emit_keys_to_console` config key under the main `ssh` config key can be used. To avoid the fingerprint of types of SSH host keys being written to console the `ssh_fp_console_blacklist` config key can be used. By default all types of keys will have their fingerprints written to console. To avoid host keys

of a key type being written to console the `ssh_key_console_blacklist` config key can be used. By default `ssh-dss` host keys are not written to console.

Internal name: `cc_keys_to_console`

Module frequency: per instance

Supported distros: all

Config keys:

```
ssh:
  emit_keys_to_console: false

ssh_fp_console_blacklist: <list of key types>
ssh_key_console_blacklist: <list of key types>
```

1.10.20 Landscape

Summary: install and configure landscape client

This module installs and configures `landscape-client`. The landscape client will only be installed if the key `landscape` is present in config. Landscape client configuration is given under the `client` key under the main `landscape` config key. The config parameters are not interpreted by cloud-init, but rather are converted into a `ConfigObj` formatted file and written out to `/etc/landscape/client.conf`.

The following default client config is provided, but can be overridden:

```
landscape:
  client:
    log_level: "info"
    url: "https://landscape.canonical.com/message-system"
    ping_url: "http://landscape.canonical.com/ping"
    data_path: "/var/lib/landscape/client"
```

Note: see landscape documentation for client config keys

Note: if `tags` is defined, its contents should be a string delimited with `,` rather than a list

Internal name: `cc_landscape`

Module frequency: per instance

Supported distros: ubuntu

Config keys:

```
landscape:
  client:
    url: "https://landscape.canonical.com/message-system"
    ping_url: "http://landscape.canonical.com/ping"
    data_path: "/var/lib/landscape/client"
    http_proxy: "http://my.proxy.com/foobar"
    https_proxy: "https://my.proxy.com/foobar"
    tags: "server,cloud"
```

(continues on next page)

(continued from previous page)

```
computer_title: "footitle"
registration_key: "fookey"
account_name: "fooaccount"
```

1.10.21 Locale

Summary: Set system locale

Configure the system locale and apply it system wide. By default use the locale specified by the datasource.

Internal name: cc_locale

Module frequency: once-per-instance

Supported distros: all

Config schema: **locale:** (string) The locale to set as the system's locale (e.g. ar_PS)

locale_configfile: (string) The file in which to write the locale configuration (defaults to the distro's default location)

Examples:

```
# Set the locale to ar_AE
locale: ar_AE

# --- Example2 ---
# Set the locale to fr_CA in /etc/alternate_path/locale
locale: fr_CA
locale_configfile: /etc/alternate_path/locale
```

1.10.22 LXD

Summary: configure lxd with `lxd init` and optionally `lxd-bridge`

This module configures lxd with user specified options using `lxd init`. If lxd is not present on the system but lxd configuration is provided, then lxd will be installed. If the selected storage backend is zfs, then zfs will be installed if missing. If network bridge configuration is provided, then lxd-bridge will be configured accordingly.

Internal name: cc_lxd

Module frequency: per instance

Supported distros: ubuntu

Config keys:

```
lxd:
  init:
    network_address: <ip addr>
    network_port: <port>
    storage_backend: <zfs/dir>
    storage_create_device: <dev>
    storage_create_loop: <size>
    storage_pool: <name>
    trust_password: <password>
  bridge:
```

(continues on next page)

(continued from previous page)

```
mode: <new, existing or none>
name: <name>
ipv4_address: <ip addr>
ipv4_netmask: <cidr>
ipv4_dhcp_first: <ip addr>
ipv4_dhcp_last: <ip addr>
ipv4_dhcp_leases: <size>
ipv4_nat: <bool>
ipv6_address: <ip addr>
ipv6_netmask: <cidr>
ipv6_nat: <bool>
domain: <domain>
```

1.10.23 Mcollective

Summary: install, configure and start mcollective

This module installs, configures and starts mcollective. If the `mcollective` key is present in config, then mcollective will be installed and started.

Configuration for mcollective can be specified in the `conf` key under `mcollective`. Each config value consists of a key value pair and will be written to `/etc/mcollective/server.cfg`. The `public-cert` and `private-cert` keys, if present in `conf` may be used to specify the public and private certificates for mcollective. Their values will be written to `/etc/mcollective/ssl/server-public.pem` and `/etc/mcollective/ssl/server-private.pem`.

Note: The ec2 metadata service is readable by non-root users. If security is a concern, use `include-once` and `ssl` urls.

Internal name: `cc_mcollective`

Module frequency: per instance

Supported distros: all

Config keys:

```
mcollective:
  conf:
    <key>: <value>
    public-cert: |
      -----BEGIN CERTIFICATE-----
      <cert data>
      -----END CERTIFICATE-----
    private-cert: |
      -----BEGIN CERTIFICATE-----
      <cert data>
      -----END CERTIFICATE-----
```

1.10.24 Migrator

Summary: migrate old versions of cloud-init data to new

This module handles moving old versions of cloud-init data to newer ones. Currently, it only handles renaming cloud-init's per-frequency semaphore files to canonicalized name and renaming legacy semaphore names to newer ones. This module is enabled by default, but can be disabled by specifying `migrate: false` in config.

Internal name: `cc_migrator`

Module frequency: `always`

Supported distros: `all`

Config keys:

```
migrate: <true/false>
```

1.10.25 Mounts

Summary: configure mount points and swap files

This module can add or remove mountpoints from `/etc/fstab` as well as configure swap. The `mounts` config key takes a list of fstab entries to add. Each entry is specified as a list of `[fs_spec, fs_file, fs_vfstype, fs_mntops, fs_freq, fs_passno]`. For more information on these options, consult the manual for `/etc/fstab`. When specifying the `fs_spec`, if the device name starts with one of `xvd`, `sd`, `hd`, or `vd`, the leading `/dev` may be omitted.

In order to remove a previously listed mount, an entry can be added to the `mounts` list containing `fs_spec` for the device to be removed but no mountpoint (i.e. `[sda1]` or `[sda1, null]`).

The `mount_default_fields` config key allows default options to be specified for the values in a `mounts` entry that are not specified, aside from the `fs_spec` and the `fs_file`. If specified, this must be a list containing 6 values. It defaults to:

```
mount_default_fields: [none, none, "auto", "defaults,nobootwait", "0", "2"]
```

On a systemd booted system that default is the mostly equivalent:

```
mount_default_fields: [none, none, "auto",
    "defaults,nofail,x-systemd.requires=cloud-init.service", "0", "2"]
```

Note that *nobootwait* is an upstart specific boot option that somewhat equates to the more standard *nofail*.

Swap files can be configured by setting the path to the swap file to create with `filename`, the size of the swap file with `size` maximum size of the swap file if using an `size: auto` with `maxsize`. By default no swap file is created.

Internal name: `cc_mounts`

Module frequency: `per instance`

Supported distros: `all`

Config keys:

```
mounts:
- [ /dev/ephemeral0, /mnt, auto, "defaults,noexec" ]
- [ sdc, /opt/data ]
- [ xvdh, /opt/data, "auto", "defaults,nofail", "0", "0" ]
mount_default_fields: [None, None, "auto", "defaults,nofail", "0", "2"]
swap:
    filename: <file>
```

(continues on next page)

(continued from previous page)

```
size: <"auto"/size in bytes>
maxsize: <size in bytes>
```

1.10.26 NTP

Summary: enable and configure ntp

Handle ntp configuration. If ntp is not installed on the system and ntp configuration is specified, ntp will be installed. If there is a default ntp config file in the image or one is present in the distro's ntp package, it will be copied to a file with `.dist` appended to the filename before any changes are made. A list of ntp pools and ntp servers can be provided under the ntp config key. If no ntp servers or pools are provided, 4 pools will be used in the format `{0-3}.{distro}.pool.ntp.org`.

Internal name: `cc_ntp`

Module frequency: once-per-instance

Supported distros: almalinux, alpine, centos, cloudlinux, debian, eurolinux, fedora, miraclelinux, openEuler, open-suse, photon, rhel, rocky, sles, ubuntu, virtuoizzo

Config schema: `ntp`: (object/null)

pools: (array of string) List of ntp pools. If both pools and servers are empty, 4 default pool servers will be provided of the format `{0-3}.{distro}.pool.ntp.org`. NOTE: for Alpine Linux when using the Busybox NTP client this setting will be ignored due to the limited functionality of Busybox's ntpd.

servers: (array of string) List of ntp servers. If both pools and servers are empty, 4 default pool servers will be provided with the format `{0-3}.{distro}.pool.ntp.org`.

ntp_client: (string) Name of an NTP client to use to configure system NTP. When unprovided or 'auto' the default client preferred by the distribution will be used. The following built-in client names can be used to override existing configuration defaults: `chrony`, `ntp`, `ntpdate`, `systemd-timesyncd`.

enabled: (boolean) Attempt to enable ntp clients if set to True. If set to False, ntp client will not be configured or installed

config: (object) Configuration settings or overrides for the `ntp_client` specified.

confpath: (string) The path to where the `ntp_client` configuration is written.

check_exe: (string) The executable name for the `ntp_client`. For example, ntp service `check_exe` is 'ntpd' because it runs the ntpd binary.

packages: (array of string) List of packages needed to be installed for the selected `ntp_client`.

service_name: (string) The systemd or sysvinit service name used to start and stop the `ntp_client` service.

template: (string) Inline template allowing users to define their own `ntp_client` configuration template. The value must start with '`## template:jinja`' to enable use of templating support.

Examples:

```
# Override ntp with chrony configuration on Ubuntu
ntp:
  enabled: true
```

(continues on next page)

(continued from previous page)

```

ntp_client: chrony # Uses cloud-init default chrony configuration

# --- Example2 ---
# Provide a custom ntp client configuration
ntp:
  enabled: true
  ntp_client: myntpclient
  config:
    confpath: /etc/myntpclient/myntpclient.conf
    check_exe: myntpclientd
    packages:
      - myntpclient
    service_name: myntpclient
    template: |
      ## template:jinja
      # My NTP Client config
      {% if pools -%}# pools{% endif %}
      {% for pool in pools -%}
      pool {{pool}} iburst
      {% endfor %}
      {%- if servers %}# servers
      {% endif %}
      {% for server in servers -%}
      server {{server}} iburst
      {% endfor %}
  pools: [0.int.pool.ntp.org, 1.int.pool.ntp.org, ntp.myorg.org]
  servers:
    - ntp.server.local
    - ntp.ubuntu.com
    - 192.168.23.2

```

1.10.27 Package Update Upgrade Install

Summary: update, upgrade, and install packages

This module allows packages to be updated, upgraded or installed during boot. If any packages are to be installed or an upgrade is to be performed then the package cache will be updated first. If a package installation or upgrade requires a reboot, then a reboot can be performed if `package_reboot_if_required` is specified. A list of packages to install can be provided. Each entry in the list can be either a package name or a list with two entries, the first being the package name and the second being the specific package version to install.

Internal name: `cc_package_update_upgrade_install`

Module frequency: per instance

Supported distros: all

Config keys:

```

packages:
  - pwgen
  - pastebinit
  - [libpython2.7, 2.7.3-0ubuntu3.1]
package_update: <true/false>
package_upgrade: <true/false>
package_reboot_if_required: <true/false>

```

(continues on next page)

(continued from previous page)

```
apt_update: (alias for package_update)
apt_upgrade: (alias for package_upgrade)
apt_reboot_if_required: (alias for package_reboot_if_required)
```

1.10.28 Phone Home

Summary: post data to url

This module can be used to post data to a remote host after boot is complete. If the post url contains the string `$INSTANCE_ID` it will be replaced with the id of the current instance. Either all data can be posted or a list of keys to post. Available keys are:

- `pub_key_dsa`
- `pub_key_rsa`
- `pub_key_ecdsa`
- `pub_key_ed25519`
- `instance_id`
- `hostname`
- `fqdn`

Data is sent as `x-www-form-urlencoded` arguments.

Example HTTP POST:

```
POST / HTTP/1.1
Content-Length: 1337
User-Agent: Cloud-Init/21.4
Accept-Encoding: gzip, deflate
Accept: */*
Content-Type: application/x-www-form-urlencoded

pub_key_dsa=dsa_contents&pub_key_rsa=rsa_contents&pub_key_ecdsa=ecdsa_contents&pub_
↪key_ed25519=ed25519_contents&instance_id=i-87018aed&hostname=myhost&fqdn=myhost.
↪internal
```

Internal name: `cc_phone_home`

Module frequency: per instance

Supported distros: all

Config keys:

```
phone_home:
  url: http://example.com/$INSTANCE_ID/
  post:
    - pub_key_dsa
    - instance_id
    - fqdn
  tries: 10
```

1.10.29 Power State Change

Summary: change power state

This module handles shutdown/reboot after all config modules have been run. By default it will take no action, and the system will keep running unless a package installation/upgrade requires a system reboot (e.g. installing a new kernel) and `package_reboot_if_required` is `true`. The `power_state` config key accepts a dict of options. If mode is any value other than `poweroff`, `halt`, or `reboot`, then no action will be taken.

The system can be shutdown before cloud-init has finished using the `timeout` option. The `delay` key specifies a duration to be added onto any shutdown command used. Therefore, if a 5 minute delay and a 120 second shutdown are specified, the maximum amount of time between cloud-init starting and the system shutting down is 7 minutes, and the minimum amount of time is 5 minutes. The `delay` key must have an argument in either the form `'+5'` for 5 minutes or `now` for immediate shutdown.

Optionally, a command can be run to determine whether or not the system should shut down. The command to be run should be specified in the `condition` key. For command formatting, see the documentation for `cc_runcmd`. The specified shutdown behavior will only take place if the `condition` key is omitted or the command specified by the `condition` key returns 0.

Note: With Alpine Linux any message value specified is ignored as Alpine's `halt`, `poweroff`, and `reboot` commands do not support broadcasting a message.

Internal name: `cc_power_state_change`

Module frequency: per instance

Supported distros: all

Config keys:

```
power_state:
  delay: <now/'+minutes'>
  mode: <poweroff/halt/reboot>
  message: <shutdown message>
  timeout: <seconds>
  condition: <true/false/command>
```

1.10.30 Puppet

Summary: install, configure and start puppet

This module handles puppet installation and configuration. If the `puppet` key does not exist in global configuration, no action will be taken. If a config entry for `puppet` is present, then by default the latest version of puppet will be installed. If `install` is set to `false`, puppet will not be installed. However, this will result in an error if puppet is not already present on the system. The version of puppet to be installed can be specified under `version`, and defaults to `none`, which selects the latest version in the repos. If the `puppet` config key exists in the config archive, this module will attempt to start puppet even if no installation was performed.

The module also provides keys for configuring the new puppet 4 paths and installing the puppet package from the puppetlabs repositories: https://docs.puppet.com/puppet/4.2/reference/whered_it_go.html. The keys are `package_name`, `conf_file`, `ssl_dir` and `csr_attributes_path`. If unset, their values will default to ones that work with puppet 3.x and with distributions that ship modified puppet 4.x that uses the old paths.

Agent packages from the puppetlabs repositories can be installed by setting `install_type` to `aio`. Based on this setting, the default config/SSL/CSR paths will be adjusted accordingly. To maintain backwards compatibility this setting defaults to `packages` which will install puppet from the distro packages.

If installing aio packages, `collection` can also be set to one of `puppet` (rolling release), `puppet6`, `puppet7` (or their nightly counterparts) in order to install specific release streams. By default, the puppetlabs repository will be purged after installation finishes; set `cleanup` to `false` to prevent this. AIO packages are installed through a shell script which is downloaded on the machine and then executed; the path to this script can be overridden using the `aio_install_url` key.

Puppet configuration can be specified under the `conf` key. The configuration is specified as a dictionary containing high-level <section> keys and lists of <key>=<value> pairs within each section. Each section name and <key>=<value> pair is written directly to `puppet.conf`. As such, section names should be one of: `main`, `server`, `agent` or `user` and keys should be valid puppet configuration options. The `certname` key supports string substitutions for `%i` and `%f`, corresponding to the instance id and fqdn of the machine respectively. If `ca_cert` is present, it will not be written to `puppet.conf`, but instead will be used as the puppetserver certificate. It should be specified in pem format as a multi-line string (using the `| yml` notation).

Additionally it's possible to create a `csr_attributes.yaml` file for CSR attributes and certificate extension requests. See https://puppet.com/docs/puppet/latest/config_file_csr_attributes.html

By default, the puppet service will be automatically enabled after installation and set to automatically start on boot. To override this in favor of manual puppet execution set `start_service` to `false`.

A single manual run can be triggered by setting `exec` to `true`, and additional arguments can be passed to puppet agent via the `exec_args` key (by default the agent will execute with the `--test` flag).

Internal name: `cc_puppet`

Module frequency: per instance

Supported distros: all

Config keys:

```
puppet:
  install: <true/false>
  version: <version>
  collection: <aio collection>
  install_type: <packages/aio>
  aio_install_url: 'https://git.io/JBhoQ'
  cleanup: <true/false>
  conf_file: '/etc/puppet/puppet.conf'
  ssl_dir: '/var/lib/puppet/ssl'
  csr_attributes_path: '/etc/puppet/csr_attributes.yaml'
  package_name: 'puppet'
  exec: <true/false>
  exec_args: ['--test']
  start_service: <true/false>
  conf:
    agent:
      server: "puppetserver.example.org"
      certname: "%i.%f"
      ca_cert: |
        -----BEGIN CERTIFICATE-----
        <cert data>
        -----END CERTIFICATE-----
    csr_attributes:
      custom_attributes:
        1.2.840.113549.1.9.7: 342thbjkt82094y0uthhor289jnqthpc2290
      extension_requests:
        pp_uuid: ED803750-E3C7-44F5-BB08-41A04433FE2E
        pp_image_name: my_ami_image
        pp_preshared_key: 342thbjkt82094y0uthhor289jnqthpc2290
```


1.10.31 Resizefs

Summary: Resize filesystem

Resize a filesystem to use all available space on partition. This module is useful along with `cc_growpart` and will ensure that if the root partition has been resized the root filesystem will be resized along with it. By default, `cc_resizefs` will resize the root partition and will block the boot process while the `resize` command is running. Optionally, the `resize` operation can be performed in the background while cloud-init continues running modules. This can be enabled by setting `resize_rootfs` to `true`. This module can be disabled altogether by setting `resize_rootfs` to `false`.

Internal name: `cc_resizefs`

Module frequency: always

Supported distros: all

Config schema: `resize_rootfs`: (true/false/noblock) Whether to resize the root partition. Default: 'true'

Examples:

```
resize_rootfs: false # disable root filesystem resize operation
```

1.10.32 Resolv Conf

Summary: configure `resolv.conf`

This module is intended to manage `resolv.conf` in environments where early configuration of `resolv.conf` is necessary for further bootstrapping and/or where configuration management such as puppet or chef own dns configuration. As Debian/Ubuntu will, by default, utilize `resolvconf`, and similarly Red Hat will use `sysconfig`, this module is likely to be of little use unless those are configured correctly.

Note: For Red Hat with `sysconfig`, be sure to set `PEERDNS=no` for all DHCP enabled NICs.

Note: And, in Ubuntu/Debian it is recommended that DNS be configured via the standard `/etc/network/interfaces` configuration file.

Internal name: `cc_resolv_conf`

Module frequency: per instance

Supported distros: alpine, fedora, photon, rhel, sles

Config keys:

```
manage_resolv_conf: <true/false>
resolv_conf:
  nameservers: ['8.8.4.4', '8.8.8.8']
  searchdomains:
    - foo.example.com
    - bar.example.com
  domain: example.com
  options:
    rotate: <true/false>
    timeout: 1
```

1.10.33 Red Hat Subscription

Summary: register red hat enterprise linux based system

Register a Red Hat system either by username and password *or* activation and org. Following a successful registration, you can auto-attach subscriptions, set the service level, add subscriptions based on pool id, enable/disable yum repositories based on repo id, and alter the rhsm_baseurl and server-hostname in `/etc/rhsm/rhsm.conf`. For more details, see the Register Red Hat Subscription example config.

Internal name: `cc_rh_subscription`

Module frequency: per instance

Supported distros: rhel, fedora

Config keys:

```
rh_subscription:
  username: <username>
  password: <password>
  activation-key: <activation key>
  org: <org number>
  auto-attach: <true/false>
  service-level: <service level>
  add-pool: <list of pool ids>
  enable-repo: <list of yum repo ids>
  disable-repo: <list of yum repo ids>
  rhsm-baseurl: <url>
  server-hostname: <hostname>
```

1.10.34 Rightscale Userdata

Summary: support rightscale configuration hooks

This module adds support for RightScale configuration hooks to cloud-init. RightScale adds a entry in the format `CLOUD_INIT_REMOTE_HOOK=http://...` to ec2 user-data. This module checks for this line in the raw userdata and retrieves any scripts linked by the RightScale user data and places them in the user scripts configuration directory, to be run later by `cc_scripts_user`.

Note: the `CLOUD_INIT_REMOTE_HOOK` config variable is present in the raw ec2 user data only, not in any cloud-config parts

Internal name: `cc_rightscale_userdata`

Module frequency: per instance

Supported distros: all

Config keys:

```
CLOUD_INIT_REMOTE_HOOK=<url>
```

1.10.35 Rsyslog

Summary: configure system logging via rsyslog

This module configures remote system logging using rsyslog.

The rsyslog config file to write to can be specified in `config_filename`, which defaults to `20-cloud-config.conf`. The rsyslog config directory to write config files to may be specified in `config_dir`, which defaults to `/etc/rsyslog.d`.

A list of configurations for rsyslog can be specified under the `configs` key in the rsyslog config. Each entry in `configs` is either a string or a dictionary. Each config entry contains a configuration string and a file to write it to. For config entries that are a dictionary, `filename` sets the target filename and `content` specifies the config string to write. For config entries that are only a string, the string is used as the config string to write. If the filename to write the config to is not specified, the value of the `config_filename` key is used. A file with the selected filename will be written inside the directory specified by `config_dir`.

The command to use to reload the rsyslog service after the config has been updated can be specified in `service_reload_command`. If this is set to `auto`, then an appropriate command for the distro will be used. This is the default behavior. To manually set the command, use a list of command args (e.g. `[systemctl, restart, rsyslog]`).

Configuration for remote servers can be specified in `configs`, but for convenience it can be specified as key value pairs in `remotes`. Each key is the name for an rsyslog remote entry. Each value holds the contents of the remote config for rsyslog. The config consists of the following parts:

- filter for log messages (defaults to `*.*`)
- optional leading `@` or `@@`, indicating `udp` and `tcp` respectively (defaults to `@`, for `udp`)
- `ipv4` or `ipv6` hostname or address. `ipv6` addresses must be in `[: : 1]` format, (e.g. `@ [fd00 : : 1] : 514`)
- optional port number (defaults to `514`)

This module will provide sane defaults for any part of the remote entry that is not specified, so in most cases remote hosts can be specified just using `<name>: <address>`.

For backwards compatibility, this module still supports legacy names for the config entries. Legacy to new mappings are as follows:

- `rsyslog -> rsyslog/configs`
- `rsyslog_filename -> rsyslog/config_filename`
- `rsyslog_dir -> rsyslog/config_dir`

Note: The legacy config format does not support specifying `service_reload_command`.

Internal name: `cc_rsyslog`

Module frequency: per instance

Supported distros: all

Config keys:

```
rsyslog:
  config_dir: config_dir
  config_filename: config_filename
  configs:
    - " *.* @@192.158.1.1"
    - content: " *.* @@192.0.2.1:10514"
      filename: 01-example.conf
    - content: |
        *.* @@syslogd.example.com
  remotes:
    maas: "192.168.1.1"
```

(continues on next page)

(continued from previous page)

```
juju: "10.0.4.1"
service_reload_command: [your, syslog, restart, command]
```

Legacy config keys:

```
rsyslog:
- "*. * @@192.158.1.1"
rsyslog_dir: /etc/rsyslog-config.d/
rsyslog_filename: 99-local.conf
```

1.10.36 Runcmd

Summary: Run arbitrary commands

Run arbitrary commands at a rc.local like level with output to the console. Each item can be either a list or a string. If the item is a list, it will be properly quoted. Each item is written to `/var/lib/cloud/instance/runcmd` to be later interpreted using `sh`.

Note that the `runcmd` module only writes the script to be run later. The module that actually runs the script is `scripts-user` in the *Final* boot stage.

Note: all commands must be proper yaml, so you have to quote any characters yaml would eat (‘:’ can be problematic)

Note: when writing files, do not use `/tmp` dir as it races with `systemd-tmpfiles-clean` LP: #1707222. Use `/run/somedir` instead.

Internal name: `cc_runcmd`**Module frequency:** once-per-instance**Supported distros:** all**Config schema:** `runcmd`: (array of (array of string)/(string)/(null))**Examples:**

```
runcmd:
- [ ls, -l, / ]
- [ sh, -xc, "echo $(date) ': hello world!'" ]
- [ sh, -c, echo "====hello world'====" ]
- ls -l /root
- [ wget, "http://example.org", -O, /tmp/index.html ]
```

1.10.37 Salt Minion

Summary: set up and run salt minion

This module installs, configures and starts salt minion. If the `salt_minion` key is present in the config parts, then salt minion will be installed and started. Configuration for salt minion can be specified in the `conf` key under `salt_minion`. Any conf values present there will be assigned in `/etc/salt/minion`. The public and

private keys to use for salt minion can be specified with `public_key` and `private_key` respectively. Optionally if you have a custom package name, service name or config directory you can specify them with `pkg_name`, `service_name` and `config_dir`.

Internal name: `cc_salt_minion`

Module frequency: per instance

Supported distros: all

Config keys:

```
salt_minion:
  pkg_name: 'salt-minion'
  service_name: 'salt-minion'
  config_dir: '/etc/salt'
  conf:
    master: salt.example.com
  grains:
    role:
      - web
  public_key: |
    -----BEGIN PUBLIC KEY-----
    <key data>
    -----END PUBLIC KEY-----
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key data>
    -----END PRIVATE KEY-----
```

1.10.38 Scripts Per Boot

Summary: run per boot scripts

Any scripts in the `scripts/per-boot` directory on the datasource will be run every time the system boots. Scripts will be run in alphabetical order. This module does not accept any config keys.

Internal name: `cc_scripts_per_boot`

Module frequency: always

Supported distros: all

1.10.39 Scripts Per Instance

Summary: run per instance scripts

Any scripts in the `scripts/per-instance` directory on the datasource will be run when a new instance is first booted. Scripts will be run in alphabetical order. This module does not accept any config keys.

Some cloud platforms change instance-id if a significant change was made to the system. As a result per-instance scripts will run again.

Internal name: `cc_scripts_per_instance`

Module frequency: per instance

Supported distros: all

1.10.40 Scripts Per Once

Summary: run one time scripts

Any scripts in the `scripts/per-once` directory on the datasource will be run only once. Changes to the instance will not force a re-run. The only way to re-run these scripts is to run the `clean` subcommand and reboot. Scripts will be run in alphabetical order. This module does not accept any config keys.

Internal name: `cc_scripts_per_once`

Module frequency: per once

Supported distros: all

1.10.41 Scripts User

Summary: run user scripts

This module runs all user scripts. User scripts are not specified in the `scripts` directory in the datasource, but rather are present in the `scripts` dir in the instance configuration. Any cloud-config parts with a `#!` will be treated as a script and run. Scripts specified as cloud-config parts will be run in the order they are specified in the configuration. This module does not accept any config keys.

Internal name: `cc_scripts_user`

Module frequency: per instance

Supported distros: all

1.10.42 Scripts Vendor

Summary: run vendor scripts

Any scripts in the `scripts/vendor` directory in the datasource will be run when a new instance is first booted. Scripts will be run in alphabetical order. Vendor scripts can be run with an optional prefix specified in the `prefix` entry under the `vendor_data` config key.

Internal name: `cc_scripts_vendor`

Module frequency: per instance

Supported distros: all

Config keys:

```
vendor_data:
  prefix: <vendor data prefix>
```

1.10.43 Seed Random

Summary: provide random seed data

Since all cloud instances started from the same image will produce very similar data when they are first booted, as they are all starting with the same seed for the kernel's entropy keyring. To avoid this, random seed data can be provided to the instance either as a string or by specifying a command to run to generate the data.

Configuration for this module is under the `random_seed` config key. The `file` key specifies the path to write the data to, defaulting to `/dev/urandom`. Data can be passed in directly with `data`, and may optionally be specified in encoded form, with the encoding specified in `encoding`.

If the cloud provides its own random seed data, it will be appended to `data` before it is written to `file`.

Note: when using a multiline value for `data` or specifying binary data, be sure to follow yaml syntax and use the `|` and `!binary` yaml format specifiers when appropriate

If the `command` key is specified, the given command will be executed. This will happen after `file` has been populated. That command's environment will contain the value of the `file` key as `RANDOM_SEED_FILE`. If a command is specified that cannot be run, no error will be reported unless `command_required` is set to `true`.

For example, to use `pollinate` to gather data from a remote entropy server and write it to `/dev/urandom`, the following could be used:

```
random_seed:
  file: /dev/urandom
  command: ["pollinate", "--server=http://local.pollinate.server"]
  command_required: true
```

Internal name: `cc_seed_random`

Module frequency: per instance

Supported distros: all

Config keys:

```
random_seed:
  file: <file>
  data: <random string>
  encoding: <raw/base64/b64/gzip/gz>
  command: [<cmd name>, <arg1>, <arg2>...]
  command_required: <true/false>
```

1.10.44 Set Hostname

Summary: set hostname and fqdn

This module handles setting the system hostname and fqdn. If `preserve_hostname` is set, then the hostname will not be altered.

A hostname and fqdn can be provided by specifying a full domain name under the `fqdn` key. Alternatively, a hostname can be specified using the `hostname` key, and the fqdn of the cloud will be used. If a fqdn specified with the `hostname` key, it will be handled properly, although it is better to use the `fqdn` config key. If both `fqdn` and `hostname` are set, it is distro dependent whether `hostname` or `fqdn` is used, unless the `prefer_fqdn_over_hostname` option is true and `fqdn` is set it will force the use of FQDN in all distros, and if false then it will force the hostname use.

This module will run in the `init-local` stage before networking is configured if the hostname is set by metadata or user data on the local system.

This will occur on datasources like `nocloud` and `ovf` where metadata and user data are available locally. This ensures that the desired hostname is applied before any DHCP requests are performed on these platforms where dynamic DNS is based on initial hostname.

Internal name: `cc_set_hostname`

Module frequency: always

Supported distros: all

Config keys:

```
preserve_hostname: <true/false>
prefer_fqdn_over_hostname: <true/false>
fqdn: <fqdn>
hostname: <fqdn/hostname>
```

1.10.45 Set Passwords

Summary: Set user passwords and enable/disable SSH password authentication

This module consumes three top-level config keys: `ssh_pwauth`, `chpasswd` and `password`.

The `ssh_pwauth` config key determines whether or not `sshd` will be configured to accept password authentication. True values will enable password auth, false values will disable password auth, and the literal string `unchanged` will leave it unchanged. Setting no value will also leave the current setting on-disk unchanged.

The `chpasswd` config key accepts a dictionary containing either or both of `expire` and `list`.

If the `list` key is provided, it should contain a list of `username:password` pairs. This can be either a YAML list (of strings), or a multi-line string with one pair per line. Each user will have the corresponding password set. A password can be randomly generated by specifying `RANDOM` or `R` as a user's password. A hashed password, created by a tool like `mkpasswd`, can be specified; a regex (`r'\$(1|2a|2y|5|6)(\$.+){2}'`) is used to determine if a password value should be treated as a hash.

Note: The users specified must already exist on the system. Users will have been created by the `cc_users_groups` module at this point.

By default, all users on the system will have their passwords expired (meaning that they will have to be reset the next time the user logs in). To disable this behaviour, set `expire` under `chpasswd` to a false value.

If a list of user/password pairs is not specified under `chpasswd`, then the value of the `password` config key will be used to set the default user's password.

Internal name: `cc_set_passwords`

Module frequency: per instance

Supported distros: all

Config keys:

```
ssh_pwauth: <yes/no/unchanged>

password: password1
chpasswd:
  expire: <true/false>

chpasswd:
  list: |
    user1:password1
    user2:RANDOM
    user3:password3
    user4:R

##
# or as yaml list
```

(continues on next page)

(continued from previous page)

```
##
chpasswd:
  list:
    - user1:password1
    - user2:RANDOM
    - user3:password3
    - user4:R
    - user4:$6$rL..$ej...
```

1.10.46 Snap

Summary: Install, configure and manage snapd and snap packages

This module provides a simple configuration namespace in cloud-init to both setup snapd and install snaps.

Note: Both `assertions` and `commands` values can be either a dictionary or a list. If these configs are provided as a dictionary, the keys are only used to order the execution of the assertions or commands and the dictionary is merged with any vendor-data snap configuration provided. If a list is provided by the user instead of a dict, any vendor-data snap configuration is ignored.

The `assertions` configuration option is a dictionary or list of properly-signed snap assertions which will run before any snap commands. They will be added to snapd's assertion database by invoking `snap ack <aggregate_assertion_file>`.

`Snap commands` is a dictionary or list of individual snap commands to run on the target system. These commands can be used to create snap users, install snaps and provide snap configuration.

Note: If 'side-loading' private/unpublished snaps on an instance, it is best to create a snap seed directory and seed.yaml manifest in `/var/lib/snapd/seed/` which snapd automatically installs on startup.

Development only: The `squashfuse_in_container` boolean can be set true to install squashfuse package when in a container to enable snap installs. Default is false.

Internal name: `cc_snap`

Module frequency: once-per-instance

Supported distros: ubuntu

Config schema: `snap:` (object)

assertions: (object/array of string)

commands: (object/array of (array of string)/(string))

squashfuse_in_container: (boolean)

Examples:

```
snap:
  assertions:
    00: |
      signed_assertion_blob_here
    02: |
      signed_assertion_blob_here
```

(continues on next page)

(continued from previous page)

```

commands:
  00: snap create-user --sudoer --known <snap-user>@mydomain.com
  01: snap install canonical-livepatch
  02: canonical-livepatch enable <AUTH_TOKEN>

# --- Example2 ---
# LXC-based containers require squashfuse before snaps can be installed
snap:
  commands:
    00: apt-get install squashfuse -y
    11: snap install emoji

# --- Example3 ---
# Convenience: the snap command can be omitted when specifying commands
# as a list and 'snap' will automatically be prepended.
# The following commands are equivalent:
snap:
  commands:
    00: ['install', 'vlc']
    01: ['snap', 'install', 'vlc']
    02: snap install vlc
    03: 'snap install vlc'

# --- Example4 ---
# You can use a list of commands
snap:
  commands:
    - ['install', 'vlc']
    - ['snap', 'install', 'vlc']
    - snap install vlc
    - 'snap install vlc'

# --- Example5 ---
# You can use a list of assertions
snap:
  assertions:
    - signed_assertion_blob_here
    - |
      signed_assertion_blob_here

```

1.10.47 Spacewalk

Summary: install and configure spacewalk

This module installs spacewalk and applies basic configuration. If the `spacewalk` config key is present spacewalk will be installed. The server to connect to after installation must be provided in the `server` in spacewalk configuration. A proxy to connect through and a activation key may optionally be specified.

For more information about spacewalk see: <https://fedorahosted.org/spacewalk/>

Internal name: `cc_spacewalk`

Module frequency: per instance

Supported distros: redhat, fedora

Config keys:

```
spacewalk:
  server: <url>
  proxy: <proxy host>
  activation_key: <key>
```

1.10.48 SSH

Summary: configure SSH and SSH keys (host and authorized)

This module handles most configuration for SSH and both host and authorized SSH keys.

Authorized Keys

Authorized keys are a list of public SSH keys that are allowed to connect to a user account on a system. They are stored in `.ssh/authorized_keys` in that account's home directory. Authorized keys for the default user defined in `users` can be specified using `ssh_authorized_keys`. Keys should be specified as a list of public keys.

Note: see the `cc_set_passwords` module documentation to enable/disable SSH password authentication

Root login can be enabled/disabled using the `disable_root` config key. Root login options can be manually specified with `disable_root_opts`. If `disable_root_opts` is specified and contains the string `$USER`, it will be replaced with the username of the default user. By default, root login is disabled, and root login opts are set to:

```
no-port-forwarding,no-agent-forwarding,no-X11-forwarding
```

Supported public key types for the `ssh_authorized_keys` are:

- dsa
- rsa
- ecdsa
- ed25519
- `ecdsa-sha2-nistp256-cert-v01@openssh.com`
- `ecdsa-sha2-nistp256`
- `ecdsa-sha2-nistp384-cert-v01@openssh.com`
- `ecdsa-sha2-nistp384`
- `ecdsa-sha2-nistp521-cert-v01@openssh.com`
- `ecdsa-sha2-nistp521`
- `sk-ecdsa-sha2-nistp256-cert-v01@openssh.com`
- `sk-ecdsa-sha2-nistp256@openssh.com`
- `sk-ssh-ed25519-cert-v01@openssh.com`
- `sk-ssh-ed25519@openssh.com`
- `ssh-dss-cert-v01@openssh.com`
- `ssh-dss`
- `ssh-ed25519-cert-v01@openssh.com`

- `ssh-ed25519`
- `ssh-rsa-cert-v01@openssh.com`
- `ssh-rsa`
- `ssh-xmss-cert-v01@openssh.com`
- `ssh-xmss@openssh.com`

Note: this list has been filtered out from the supported keytypes of [OpenSSH](#) source, where the signonly keys are removed. Please see `ssh_util` for more information.

`dsa`, `rsa`, `ecdsa` and `ed25519` are added for legacy, as they are valid public keys in some old distros. They can possibly be removed in the future when support for the older distros are dropped

Host Keys

Host keys are for authenticating a specific instance. Many images have default host SSH keys, which can be removed using `ssh_deletekeys`. This prevents re-use of a private host key from an image on multiple machines. Since removing default host keys is usually the desired behavior this option is enabled by default.

Host keys can be added using the `ssh_keys` configuration key. The argument to this config key should be a dictionary entries for the public and private keys of each desired key type. Entries in the `ssh_keys` config dict should have keys in the format `<key type>_private`, `<key type>_public`, and, optionally, `<key type>_certificate`, e.g. `rsa_private: <key>`, `rsa_public: <key>`, and `rsa_certificate: <key>`. See below for supported key types. Not all key types have to be specified, ones left unspecified will not be used. If this config option is used, then no keys will be generated.

When host keys are generated the output of the `ssh-keygen` command(s) can be displayed on the console using the `ssh_quiet_keygen` configuration key. This settings defaults to `False` which displays the keygen output.

Note: when specifying private host keys in cloud-config, care should be taken to ensure that the communication between the data source and the instance is secure

Note: to specify multiline private host keys and certificates, use yaml multiline syntax

If no host keys are specified using `ssh_keys`, then keys will be generated using `ssh-keygen`. By default one public/private pair of each supported host key type will be generated. The key types to generate can be specified using the `ssh_genkeytypes` config flag, which accepts a list of host key types to use. For each host key type for which this module has been instructed to create a keypair, if a key of the same type is already present on the system (i.e. if `ssh_deletekeys` was `false`), no key will be generated.

Supported host key types for the `ssh_keys` and the `ssh_genkeytypes` config flags are:

- `rsa`
- `dsa`
- `ecdsa`
- `ed25519`

Internal name: `cc_ssh`

Module frequency: per instance

Supported distros: all

Config keys:

```
ssh_deletekeys: <true/false>
ssh_keys:
  rsa_private: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIBxwIBAAJhAKD0YSHy73nUgysO13XsJmd4fHiFyQ+00R7VVu2iV9Qco
    ...
    -----END RSA PRIVATE KEY-----
  rsa_public: ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEAoPRhIfLvedSDKw7Xd ...
  rsa_certificate: |
    ssh-rsa-cert-v01@openssh.com AAAAIHNzaC1lZDIiNTE5LWNlcnQt ...
  dsa_private: |
    -----BEGIN DSA PRIVATE KEY-----
    MIIBxwIBAAJhAKD0YSHy73nUgysO13XsJmd4fHiFyQ+00R7VVu2iV9Qco
    ...
    -----END DSA PRIVATE KEY-----
  dsa_public: ssh-dsa AAAAB3NzaC1yc2EAAAABIwAAAGEAoPRhIfLvedSDKw7Xd ...
  dsa_certificate: |
    ssh-dsa-cert-v01@openssh.com AAAAIHNzaC1lZDIiNTE5LWNlcnQt ...

ssh_genkeytypes: <key type>
disable_root: <true/false>
disable_root_opts: <disable root options string>
ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUU ...
  - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAQE3I7VUf2l5gSn5uavROsc5HRDpZ ...
allow_public_ssh_keys: <true/false>
ssh_publish_hostkeys:
  enabled: <true/false> (Defaults to true)
  blacklist: <list of key types> (Defaults to [dsa])
ssh_quiet_keygen: <true/false>
```

1.10.49 SSH Authkey Fingerprints

Summary: log fingerprints of user SSH keys

Write fingerprints of authorized keys for each user to log. This is enabled by default, but can be disabled using `no_ssh_fingerprints`. The hash type for the keys can be specified, but defaults to sha256.

Internal name: `cc_ssh_authkey_fingerprints`

Module frequency: per instance

Supported distros: all

Config keys:

```
no_ssh_fingerprints: <true/false>
authkey_hash: <hash type>
```

1.10.50 SSH Import Id

Summary: import SSH id

This module imports SSH keys from either a public keyserver, usually launchpad or github using `ssh-import-id`. Keys are referenced by the username they are associated with on the keyserver. The keyserver can be specified by prepending either `lp:` for launchpad or `gh:` for github to the username.

Internal name: `cc_ssh_import_id`

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:

```
ssh_import_id:
- user
- gh:user
- lp:user
```

1.10.51 Timezone

Summary: set system timezone

Set the system timezone. If any args are passed to the module then the first will be used for the timezone. Otherwise, the module will attempt to retrieve the timezone from cloud config.

Internal name: `cc_timezone`

Module frequency: per instance

Supported distros: all

Config keys:

```
timezone: <timezone>
```

1.10.52 Ubuntu Advantage

Summary: Configure Ubuntu Advantage support services

Attach machine to an existing Ubuntu Advantage support contract and enable or disable support services such as Livepatch, ESM, FIPS and FIPS Updates. When attaching a machine to Ubuntu Advantage, one can also specify services to enable. When the 'enable' list is present, any named service will be enabled and all absent services will remain disabled.

Note that when enabling FIPS or FIPS updates you will need to schedule a reboot to ensure the machine is running the FIPS-compliant kernel. See *Power State Change* for information on how to configure cloud-init to perform this reboot.

Internal name: `cc_ubuntu_advantage`

Module frequency: once-per-instance

Supported distros: ubuntu

Config schema: `ubuntu_advantage:` (object)

enable: (array of string)

token: (string) A contract token obtained from <https://ubuntu.com/advantage>.

Examples:

```

# Attach the machine to an Ubuntu Advantage support contract with a
# UA contract token obtained from https://ubuntu.com/advantage.
ubuntu_advantage:
  token: <ua_contract_token>

# --- Example2 ---
# Attach the machine to an Ubuntu Advantage support contract enabling
# only fips and esm services. Services will only be enabled if
# the environment supports said service. Otherwise warnings will
# be logged for incompatible services specified.
ubuntu-advantage:
  token: <ua_contract_token>
  enable:
    - fips
    - esm

# --- Example3 ---
# Attach the machine to an Ubuntu Advantage support contract and enable
# the FIPS service. Perform a reboot once cloud-init has
# completed.
power_state:
  mode: reboot
ubuntu-advantage:
  token: <ua_contract_token>
  enable:
    - fips

```

1.10.53 Ubuntu Drivers

Summary: Interact with third party drivers in Ubuntu.

This module interacts with the ‘ubuntu-drivers’ command to install third party driver packages.

Internal name: cc_ubuntu_drivers

Module frequency: once-per-instance

Supported distros: ubuntu

Config schema: **drivers:** (object)

nvidia: (object)

license-accepted: (boolean) Do you accept the NVIDIA driver license?

version: (string) The version of the driver to install (e.g. “390”, “410”). Defaults to the latest version.

Examples:

```

drivers:
  nvidia:
    license-accepted: true

```

1.10.54 Update Etc Hosts

Summary: update the hosts file (usually /etc/hosts)

This module will update the contents of the local hosts database (hosts file; usually `/etc/hosts`) based on the hostname/fqdn specified in config. Management of the hosts file is controlled using `manage_etc_hosts`. If this is set to false, cloud-init will not manage the hosts file at all. This is the default behavior.

If set to `true` or `template`, cloud-init will generate the hosts file using the template located in `/etc/cloud/templates/hosts.tmpl`. In the `/etc/cloud/templates/hosts.tmpl` template, the strings `$hostname` and `$fqdn` will be replaced with the hostname and fqdn respectively.

If `manage_etc_hosts` is set to `localhost`, then cloud-init will not rewrite the hosts file entirely, but rather will ensure that a entry for the fqdn with a distribution dependent ip is present (i.e. `ping <hostname>` will ping `127.0.0.1` or `127.0.1.1` or other ip).

Note: if `manage_etc_hosts` is set `true` or `template`, the contents of the hosts file will be updated every boot. To make any changes to the hosts file persistent they must be made in `/etc/cloud/templates/hosts.tmpl`

Note: for instructions on specifying hostname and fqdn, see documentation for `cc_set_hostname`

Internal name: `cc_update_etc_hosts`

Module frequency: always

Supported distros: all

Config keys:

```
manage_etc_hosts: <true/"template"/false/"localhost">
fqdn: <fqdn>
hostname: <fqdn/hostname>
```

1.10.55 Update Hostname

Summary: update hostname and fqdn

This module will update the system hostname and fqdn. If `preserve_hostname` is set, then the hostname will not be altered.

Note: for instructions on specifying hostname and fqdn, see documentation for `cc_set_hostname`

Internal name: `cc_update_hostname`

Module frequency: always

Supported distros: all

Config keys:

```
preserve_hostname: <true/false>
prefer_fqdn_over_hostname: <true/false>
fqdn: <fqdn>
hostname: <fqdn/hostname>
```


1.10.56 Users and Groups

Summary: configure users and groups

This module configures users and groups. For more detailed information on user options, see the `Including users and groups config example`.

Groups to add to the system can be specified as a list under the `groups` key. Each entry in the list should either contain a the group name as a string, or a dictionary with the group name as the key and a list of users who should be members of the group as the value. **Note:** Groups are added before users, so any users in a group list must already exist on the system.

The `users` config key takes a list of users to configure. The first entry in this list is used as the default user for the system. To preserve the standard default user for the distro, the string `default` may be used as the first entry of the `users` list. Each entry in the `users` list, other than a `default` entry, should be a dictionary of options for the user. Supported config keys for an entry in `users` are as follows:

- `name`: The user's login name
- `expiredate`: Optional. Date on which the user's account will be disabled. Default: none
- `gecos`: Optional. Comment about the user, usually a comma-separated string of real name and contact information. Default: none
- `groups`: Optional. Additional groups to add the user to. Default: none
- `homedir`: Optional. Home dir for user. Default is `/home/<username>`
- `inactive`: Optional. Number of days after a password expires until the account is permanently disabled. Default: none
- `lock_passwd`: Optional. Disable password login. Default: true
- `no_create_home`: Optional. Do not create home directory. Default: false
- `no_log_init`: Optional. Do not initialize lastlog and faillog for user. Default: false
- `no_user_group`: Optional. Do not create group named after user. Default: false
- `passwd`: Hash of user password
- `primary_group`: Optional. Primary group for user. Default to new group named after user.
- `selinux_user`: Optional. SELinux user for user's login. Default to default SELinux user.
- `shell`: Optional. The user's login shell. The default is to set no shell, which results in a system-specific default being used.
- `snapuser`: Optional. Specify an email address to create the user as a Snappy user through `snap create-user`. If an Ubuntu SSO account is associated with the address, username and SSH keys will be requested from there. Default: none
- `ssh_authorized_keys`: Optional. List of SSH keys to add to user's `authkeys` file. Default: none. This key can not be combined with `ssh_redirect_user`.
- `ssh_import_id`: Optional. SSH id to import for user. Default: none. This key can not be combined with `ssh_redirect_user`.
- `ssh_redirect_user`: Optional. Boolean set to true to disable SSH logins for this user. When specified, all cloud meta-data public SSH keys will be set up in a disabled state for this username. Any SSH login as this username will timeout and prompt with a message to login instead as the configured `<default_username>` for this instance. Default: false. This key can not be combined with `ssh_import_id` or `ssh_authorized_keys`.
- `sudo`: Optional. Sudo rule to use, list of sudo rules to use or False. Default: none. An absence of sudo key, or a value of none or false will result in no sudo rules being written for the user.

- `system`: Optional. Create user as system user with no home directory. Default: `false`
- `uid`: Optional. The user's ID. Default: The next available value.

Note: Specifying a hash of a user's password with `passwd` is a security risk if the cloud-config can be intercepted. SSH authentication is preferred.

Note: If specifying a sudo rule for a user, ensure that the syntax for the rule is valid, as it is not checked by cloud-init.

Note: Most of these configuration options will not be honored if the user already exists. The following options are the exceptions; they are applied to already-existing users: `plain_text_passwd`, `hashed_passwd`, `lock_passwd`, `sudo`, `ssh_authorized_keys`, `ssh_redirect_user`.

Internal name: `cc_users_groups`

Module frequency: per instance

Supported distros: all

Config keys:

```
groups:
  - <group>: [<user>, <user>]
  - <group>

users:
  - default
    # User explicitly omitted from sudo permission; also default behavior.
  - name: <some_restricted_user>
    sudo: false
  - name: <username>
    expiredate: '<date>'
    gecost: <comment>
    groups: <additional groups>
    homedir: <home directory>
    inactive: '<number of days>'
    lock_passwd: <true/false>
    no_create_home: <true/false>
    no_log_init: <true/false>
    no_user_group: <true/false>
    passwd: <password>
    primary_group: <primary group>
    selinux_user: <selinux username>
    shell: <shell path>
    snapuser: <email>
    ssh_redirect_user: <true/false>
    ssh_authorized_keys:
      - <key>
      - <key>
    ssh_import_id: <id>
    sudo: <sudo config>
    system: <true/false>
    uid: <user id>
```

1.10.57 Write Files

Summary: write arbitrary files

Write out arbitrary content to files, optionally setting permissions. Parent folders in the path are created if absent. Content can be specified in plain text or binary. Data encoded with either base64 or binary gzip data can be specified and will be decoded before being written. For empty file creation, content can be omitted.

Note: if multiline data is provided, care should be taken to ensure that it follows yaml formatting standards. to specify binary data, use the yaml option `!!binary`

Note: Do not write files under `/tmp` during boot because of a race with `systemd-tmpfiles-clean` that can cause temp files to get cleaned during the early boot process. Use `/run/somedir` instead to avoid race LP:1707222.

Internal name: `cc_write_files`

Module frequency: once-per-instance

Supported distros: all

Config schema: `write_files`: (array of object)

Each item in `write_files` list supports the following keys:

- path:** (string) Path of the file to which `content` is decoded and written
- content:** (string) Optional content to write to the provided `path`. When content is present and encoding is not `'text/plain'`, decode the content prior to writing. Default: `''`
- owner:** (string) Optional owner:group to chown on the file. Default: `root:root`
- permissions:** (string) Optional file permissions to set on `path` represented as an octal string `'0###'`. Default: `'0644'`
- encoding:** (string) Optional encoding type of the content. Default is `text/plain` and no content decoding is performed. Supported encoding types are: `gz`, `gzip`, `gz+base64`, `gzip+base64`, `gz+b64`, `gzip+b64`, `b64`, `base64`.
- append:** (boolean) Whether to append `content` to existing file if `path` exists. Default: `false`.
- defer:** (boolean) Defer writing the file until `'final'` stage, after users were created, and packages were installed. Default: `False`.

Examples:

```
# Write out base64 encoded content to /etc/sysconfig/selinux
write_files:
- encoding: b64
  content: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHRobzSBzdGF0ZSBvZiBTRUxpbnV4...
  owner: root:root
  path: /etc/sysconfig/selinux
  permissions: '0644'

# --- Example2 ---
# Appending content to an existing file
write_files:
```

(continues on next page)

(continued from previous page)

```

- content: |
    15 * * * * root ship_logs
  path: /etc/crontab
  append: true

# --- Example3 ---
# Provide gzipped binary content
write_files:
- encoding: gzip
  content: !!binary |
    H4sIAIDb/U8C/1NW1E/KzNMvzuBKTC7IV8hIzcnJVYjPL8pJ4QIA6N+MVxsAAAA=
  path: /usr/bin/hello
  permissions: '0755'

# --- Example4 ---
# Create an empty file on the system
write_files:
- path: /root/CLOUD_INIT_WAS_HERE

# --- Example5 ---
# Defer writing the file until after the package (Nginx) is
# installed and its user is created alongside
write_files:
- path: /etc/nginx/conf.d/example.com.conf
  content: |
    server {
      server_name example.com;
      listen 80;
      root /var/www;
      location / {
        try_files $uri $uri/ $uri.html =404;
      }
    }
  owner: 'nginx:nginx'
  permissions: '0640'
  defer: true

```

1.10.58 Yum Add Repo

Summary: add yum repository configuration to the system

Add yum repository configuration to `/etc/yum.repos.d`. Configuration files are named based on the dictionary key under the `yum_repos` they are specified with. If a config file already exists with the same name as a config entry, the config entry will be skipped.

Internal name: `cc_yum_add_repo`

Module frequency: always

Supported distros: almalinux, centos, cloudlinux, eurolinux, fedora, miraclelinux, openEuler, photon, rhel, rocky, virtuoizzo

Config keys:

```

yum_repos:
  <repo-name>:

```

(continues on next page)

(continued from previous page)

```

baseurl: <repo url>
name: <repo name>
enabled: <true/false>
# any repository configuration options (see man yum.conf)

```

1.11 Merging User-Data Sections

1.11.1 Overview

This was implemented because it has been a common feature request that there be a way to specify how cloud-config YAML “dictionaries” provided as user-data are merged together when there are multiple YAML files to merge together (say when performing an `#include`).

Since previously the merging algorithm was very simple and would only overwrite and not append lists, or strings, and so on it was decided to create a new and improved way to merge dictionaries (and their contained objects) together in a way that is customizable, thus allowing for users who provide cloud-config user-data to determine exactly how their objects will be merged.

For example.

```

#cloud-config (1)
runcmd:
  - bash1
  - bash2

#cloud-config (2)
runcmd:
  - bash3
  - bash4

```

The previous way of merging the two objects above would result in a final cloud-config object that contains the following.

```

#cloud-config (merged)
runcmd:
  - bash3
  - bash4

```

Typically this is not what users want; instead they would likely prefer:

```

#cloud-config (merged)
runcmd:
  - bash1
  - bash2
  - bash3
  - bash4

```

This way makes it easier to combine the various cloud-config objects you have into a more useful list, thus reducing duplication necessary to accomplish the same result with the previous method.

1.11.2 Built-in Mergers

Cloud-init provides merging for the following built-in types:

- Dict
- List
- String

The `Dict` merger has the following options which control what is done with values contained within the config.

- `allow_delete`: Existing values not present in the new value can be deleted, defaults to `False`
- `no_replace`: Do not replace an existing value if one is already present, enabled by default.
- `replace`: Overwrite existing values with new ones.

The `List` merger has the following options which control what is done with the values contained within the config.

- `append`: Add new value to the end of the list, defaults to `False`.
- `prepend`: Add new values to the start of the list, defaults to `False`.
- `no_replace`: Do not replace an existing value if one is already present, enabled by default.
- `replace`: Overwrite existing values with new ones.

The `Str` merger has the following options which control what is done with the values contained within the config.

- `append`: Add new value to the end of the string, defaults to `False`.

Common options for all merge types which control how recursive merging is done on other types.

- `recurse_dict`: If `True` merge the new values of the dictionary, defaults to `True`.
- `recurse_list`: If `True` merge the new values of the list, defaults to `False`.
- `recurse_array`: Alias for `recurse_list`.
- `recurse_str`: If `True` merge the new values of the string, defaults to `False`.

1.11.3 Customizability

Because the above merging algorithm may not always be desired (just as the previous merging algorithm was not always the preferred one), the concept of customized merging was introduced through ‘merge classes’.

A merge class is a class definition which provides functions that can be used to merge a given type with another given type.

An example of one of these merging classes is the following:

```
class Merger(object):
    def __init__(self, merger, opts):
        self._merger = merger
        self._overwrite = 'overwrite' in opts

    # This merging algorithm will attempt to merge with
    # another dictionary, on encountering any other type of object
    # it will not merge with said object, but will instead return
    # the original value
    #
    # On encountering a dictionary, it will create a new dictionary
    # composed of the original and the one to merge with, if 'overwrite'
    # is enabled then keys that exist in the original will be overwritten
    # by keys in the one to merge with (and associated values). Otherwise
    # if not in overwrite mode the 2 conflicting keys themselves will
    # be merged.
```

(continues on next page)

(continued from previous page)

```

def _on_dict(self, value, merge_with):
    if not isinstance(merge_with, (dict)):
        return value
    merged = dict(value)
    for (k, v) in merge_with.items():
        if k in merged:
            if not self._overwrite:
                merged[k] = self._merger.merge(merged[k], v)
            else:
                merged[k] = v
        else:
            merged[k] = v
    return merged

```

As you can see there is a ‘_on_dict’ method here that will be given a source value and a value to merge with. The result will be the merged object. This code itself is called by another merging class which ‘directs’ the merging to happen by analyzing the types of the objects to merge and attempting to find a known object that will merge that type. I will avoid pasting that here, but it can be found in the *mergers/__init__.py* file (see *LookupMerger* and *UnknownMerger*).

So following the typical cloud-init way of allowing source code to be downloaded and used dynamically, it is possible for users to inject their own merging files to handle specific types of merging as they choose (the basic ones included will handle lists, dicts, and strings). Note how each merge can have options associated with it which affect how the merging is performed, for example a dictionary merger can be told to overwrite instead of attempt to merge, or a string merger can be told to append strings instead of discarding other strings to merge with.

1.11.4 How to activate

There are a few ways to activate the merging algorithms, and to customize them for your own usage.

1. The first way involves the usage of MIME messages in cloud-init to specify multipart documents (this is one way in which multiple cloud-config is joined together into a single cloud-config). Two new headers are looked for, both of which can define the way merging is done (the first header to exist wins). These new headers (in lookup order) are ‘Merge-Type’ and ‘X-Merge-Type’. The value should be a string which will satisfy the new merging format definition (see below for this format).
2. The second way is actually specifying the merge-type in the body of the cloud-config dictionary. There are 2 ways to specify this, either as a string or as a dictionary (see format below). The keys that are looked up for this definition are the following (in order), ‘merge_how’, ‘merge_type’.

String format

The string format that is expected is the following.

```
classname1(option1,option2)+classname2(option3,option4)...
```

The class name there will be connected to class names used when looking for the class that can be used to merge and options provided will be given to the class on construction of that class.

For example, the default string that is used when none is provided is the following:

```
list()+dict()+str()
```

Dictionary format

A dictionary can be used when it specifies the same information as the string format (i.e. the second option above), for example:

```
{'merge_how': [{ 'name': 'list', 'settings': ['append']},
                { 'name': 'dict', 'settings': ['no_replace', 'recurse_list']},
                { 'name': 'str', 'settings': ['append']}]}
```

This would be the equivalent format for default string format but in dictionary form instead of string form.

1.11.5 Specifying multiple types and its effect

Now you may be asking yourself, if I specify a merge-type header or dictionary for every cloud-config that I provide, what exactly happens?

The answer is that when merging, a stack of ‘merging classes’ is kept, the first one on that stack is the default merging classes, this set of mergers will be used when the first cloud-config is merged with the initial empty cloud-config dictionary. If the cloud-config that was just merged provided a set of merging classes (via the above formats) then those merging classes will be pushed onto the stack. Now if there is a second cloud-config to be merged then the merging classes from the cloud-config before the first will be used (not the default) and so on. This way a cloud-config can decide how it will merge with a cloud-config dictionary coming after it.

1.11.6 Other uses

In addition to being used for merging user-data sections, the default merging algorithm for merging ‘conf.d’ YAML files (which form an initial YAML config for cloud-init) was also changed to use this mechanism so its full benefits (and customization) can also be used there as well. Other places that used the previous merging are also, similarly, now extensible (metadata merging, for example).

Note, however, that merge algorithms are not used *across* types of configuration. As was the case before merging was implemented, user-data will overwrite conf.d configuration without merging.

1.11.7 Example cloud-config

A common request is to include multiple `runcloud` directives in different files and merge all of the commands together. To achieve this, we must modify the default merging to allow for dictionaries to join list values.

The first config

```
#cloud-config
merge_how:
- name: list
  settings: [append]
- name: dict
  settings: [no_replace, recurse_list]

runcloud:
- bash1
- bash2
```

The second config


```
#cloud-config
merge_how:
- name: list
  settings: [append]
- name: dict
  settings: [no_replace, recurse_list]

runcmd:
- bash3
- bash4
```

1.12 Instance Metadata

1.12.1 What is instance data?

Instance data is the collection of all configuration data that cloud-init processes to configure the instance. This configuration typically comes from any number of sources:

- cloud-provided metadata services (aka metadata)
- custom config-drive attached to the instance
- cloud-config seed files in the booted cloud image or distribution
- vendordata provided from files or cloud metadata services
- userdata provided at instance creation

Each cloud provider presents unique configuration metadata in different formats to the instance. Cloud-init provides a cache of any crawled metadata as well as a versioned set of standardized instance data keys which it makes available on all platforms.

Cloud-init produces a simple json object in `/run/cloud-init/instance-data.json` which represents standardized and versioned representation of the metadata it consumes during initial boot. The intent is to provide the following benefits to users or scripts on any system deployed with cloud-init:

- simple static object to query to obtain a instance's metadata
- speed: avoid costly network transactions for metadata that is already cached on the filesystem
- reduce need to recrawl metadata services for static metadata that is already cached
- leverage cloud-init's best practices for crawling cloud-metadata services
- avoid rolling unique metadata crawlers on each cloud platform to get metadata configuration values

Cloud-init stores any instance data processed in the following files:

- `/run/cloud-init/instance-data.json`: world-readable json containing standardized keys, sensitive keys redacted
- `/run/cloud-init/instance-data-sensitive.json`: root-readable unredacted json blob
- `/var/lib/cloud/instance/user-data.txt`: root-readable sensitive raw userdata
- `/var/lib/cloud/instance/vendor-data.txt`: root-readable sensitive raw vendordata

Cloud-init redacts any security sensitive content from `instance-data.json`, stores `/run/cloud-init/instance-data.json` as a world-readable json file. Because `user-data` and `vendor-data` can contain passwords both of these files are readonly for `root` as well. The `root` user can also read `/run/cloud-init/`

`instance-data-sensitive.json` which is all instance data from `instance-data.json` as well as unredacted sensitive content.

1.12.2 Format of instance-data.json

The `instance-data.json` and `instance-data-sensitive.json` files are well-formed JSON and record the set of keys and values for any metadata processed by cloud-init. Cloud-init standardizes the format for this content so that it can be generalized across different cloud platforms.

There are three basic top-level keys:

- **base64_encoded_keys**: A list of forward-slash delimited key paths into the `instance-data.json` object whose value is base64encoded for json compatibility. Values at these paths should be decoded to get the original value.
- **sensitive_keys**: A list of forward-slash delimited key paths into the `instance-data.json` object whose value is considered by the datasource as 'security sensitive'. Only the keys listed here will be redacted from `instance-data.json` for non-root users.
- **merged_cfg**: Merged cloud-init 'system_config' from `/etc/cloud/cloud.cfg` and `/etc/cloud/cloud-cfg.d`. Values under this key could contain sensitive information such as passwords, so it is included in the **sensitive-keys** list which is only readable by root.
- **ds**: Datasource-specific metadata crawled for the specific cloud platform. It should closely represent the structure of the cloud metadata crawled. The structure of content and details provided are entirely cloud-dependent. Mileage will vary depending on what the cloud exposes. The content exposed under the 'ds' key is currently **experimental** and expected to change slightly in the upcoming cloud-init release.
- **sys_info**: Information about the underlying os, python, architecture and kernel. This represents the data collected by `cloudinit.util.system_info`.
- **v1**: Standardized cloud-init metadata keys, these keys are guaranteed to exist on all cloud platforms. They will also retain their current behavior and format and will be carried forward even if cloud-init introduces a new version of standardized keys with **v2**.

The standardized keys present:

v1._beta_keys

List of standardized keys still in 'beta'. The format, intent or presence of these keys can change. Do not consider them production-ready.

Example output:

- [subplatform]

v1.cloud_name

Where possible this will indicate the 'name' of the cloud the system is running on. This is different than the 'platform' item. For example, the cloud name of Amazon Web Services is 'aws', while the platform is 'ec2'.

If determining a specific name is not possible or provided in meta-data, then this field may contain the same content as 'platform'.

Example output:

- aws
- openstack

- azure
- configdrive
- nocloud
- ovf

v1.distro, v1.distro_version, v1.distro_release

This shall be the distro name, version and release as determined by *cloudinit.util.get_linux_distro*.

Example output:

- alpine, 3.12.0, ‘
- centos, 7.5, core
- debian, 9, stretch
- freebsd, 12.0-release-p10,
- opensuse, 42.3, x86_64
- opensuse-tumbleweed, 20180920, x86_64
- redhat, 7.5, ‘maipo’
- sles, 12.3, x86_64
- ubuntu, 20.04, focal

v1.instance_id

Unique instance_id allocated by the cloud.

Examples output:

- i-<hash>

v1.kernel_release

This shall be the running kernel *uname -r*

Example output:

- 5.3.0-1010-aws

v1.local_hostname

The internal or local hostname of the system.

Examples output:

- ip-10-41-41-70
- <user-provided-hostname>

v1.machine

This shall be the running cpu machine architecture *uname -m*

Example output:

- x86_64
- i686
- ppc64le
- s390x

v1.platform

An attempt to identify the cloud platform instance that the system is running on.

Examples output:

- ec2
- openstack
- lxd
- gce
- nocloud
- ovf

v1.subplatform

Additional platform details describing the specific source or type of metadata used. The format of subplatform will be:

`<subplatform_type> (<url_file_or_dev_path>)`

Examples output:

- metadata (<http://168.254.169.254>)
- seed-dir (/path/to/seed-dir/)
- config-disk (/dev/cd0)
- configdrive (/dev/sr0)

v1.public_ssh_keys

A list of SSH keys provided to the instance by the datasource metadata.

Examples output:

- ['ssh-rsa AA...', ...]

v1.python_version

The version of python that is running cloud-init as determined by *cloudinit.util.system_info*

Example output:

- 3.7.6

v1.region

The physical region/data center in which the instance is deployed.

Examples output:

- us-east-2

v1.availability_zone

The physical availability zone in which the instance is deployed.

Examples output:

- us-east-2b
- nova
- null

Example Output

Below is an example of `/run/cloud-init/instance-data-sensitive.json` on an EC2 instance:

```
{
  "_beta_keys": [
    "subplatform"
  ],
  "availability_zone": "us-east-1b",
  "base64_encoded_keys": [],
  "merged_cfg": {
    "_doc": "Merged cloud-init system config from /etc/cloud/cloud.cfg and /etc/cloud/
↳ cloud.cfg.d/",
    "_log": [
      "[loggers]\nkeys=root,cloudinit\n\n[handlers]\nkeys=consoleHandler,
↳ cloudLogHandler\n\n[formatters]\nkeys=simpleFormatter,arg0Formatter\n\n[logger_
↳ root]\nlevel=DEBUG\nhandlers=consoleHandler,cloudLogHandler\n\n[logger_
↳ cloudinit]\nlevel=DEBUG\nqualname=cloudinit\nhandlers=\npropagate=1\n\n[handler_
↳ consoleHandler]\nclass=StreamHandler\nlevel=WARNING\nformatter=arg0Formatter\nargs=(sys.
↳ stderr,)\n\n[formatter_arg0Formatter]\nformat=%(asctime)s - %(filename)s[
↳ %(levelname)s]: %(message)s\n\n[formatter_simpleFormatter]\nformat=[CLOUDINIT]
↳ %(filename)s[%(levelname)s]: %(message)s\n",
      "[handler_
↳ cloudLogHandler]\nclass=FileHandler\nlevel=DEBUG\nformatter=arg0Formatter\nargs=(' /
↳ var/log/cloud-init.log',)\n",
      "[handler_cloudLogHandler]\nclass=handlers.
↳ SysLogHandler\nlevel=DEBUG\nformatter=simpleFormatter\nargs=(\"/dev/log\", handlers.
↳ SysLogHandler.LOG_USER)\n"
    ],
  },
}
```

(continues on next page)

(continued from previous page)

```

"cloud_config_modules": [
    "emit_upstart",
    "snap",
    "ssh-import-id",
    "locale",
    "set-passwords",
    "grub-dpkg",
    "apt-pipelining",
    "apt-configure",
    "ubuntu-advantage",
    "ntp",
    "timezone",
    "disable-ec2-metadata",
    "runcmd",
    "byobu"
],
"cloud_final_modules": [
    "package-update-upgrade-install",
    "fan",
    "landscape",
    "lxd",
    "ubuntu-drivers",
    "puppet",
    "chef",
    "mcollective",
    "salt-minion",
    "rightscale_userdata",
    "scripts-vendor",
    "scripts-per-once",
    "scripts-per-boot",
    "scripts-per-instance",
    "scripts-user",
    "ssh-authkey-fingerprints",
    "keys-to-console",
    "phone-home",
    "final-message",
    "power-state-change"
],
"cloud_init_modules": [
    "migrator",
    "seed_random",
    "bootcmd",
    "write-files",
    "growpart",
    "resizefs",
    "disk_setup",
    "mounts",
    "set_hostname",
    "update_hostname",
    "update_etc_hosts",
    "ca-certs",
    "rsyslog",
    "users-groups",
    "ssh"
],
"datasource_list": [
    "Ec2",

```

(continues on next page)

(continued from previous page)

```

    "None"
  ],
  "def_log_file": "/var/log/cloud-init.log",
  "disable_root": true,
  "log_cfgs": [
    [
      [loggers]\nkeys=root,cloudinit\n\n[handlers]\nkeys=consoleHandler,
↪cloudLogHandler\n\n[formatters]\nkeys=simpleFormatter,arg0Formatter\n\n[logger_
↪root]\nlevel=DEBUG\nhandlers=consoleHandler,cloudLogHandler\n\n[logger_
↪cloudinit]\nlevel=DEBUG\nqualname=cloudinit\nhandlers=\npropagate=1\n\n[handler_
↪consoleHandler]\ncls=StreamHandler\nlevel=WARNING\nformatter=arg0Formatter\nargs=(sys.
↪stderr,)\n\n[formatter_arg0Formatter]\nformat=%(asctime)s - %(filename)s[
↪%(levelname)s]: %(message)s\n\n[formatter_simpleFormatter]\nformat=[CLOUDINIT]
↪%(filename)s[%(levelname)s]: %(message)s\n",
      [handler_
↪cloudLogHandler]\ncls=FileHandler\nlevel=DEBUG\nformatter=arg0Formatter\nargs=('/
↪var/log/cloud-init.log',)\n"
    ]
  ],
  "output": {
    "all": "| tee -a /var/log/cloud-init-output.log"
  },
  "preserve_hostname": false,
  "syslog_fix_perms": [
    "syslog:adm",
    "root:adm",
    "root:wheel",
    "root:root"
  ],
  "users": [
    "default"
  ],
  "vendor_data": {
    "enabled": true,
    "prefix": []
  }
},
"cloud_name": "aws",
"distro": "ubuntu",
"distro_release": "focal",
"distro_version": "20.04",
"ds": {
  "_doc": "EXPERIMENTAL: The structure and format of content scoped under the 'ds'_
↪key may change in subsequent releases of cloud-init.",
  "_metadata_api_version": "2016-09-02",
  "dynamic": {
    "instance_identity": {
      "document": {
        "accountId": "329910648901",
        "architecture": "x86_64",
        "availabilityZone": "us-east-1b",
        "billingProducts": null,
        "devpayProductCodes": null,
        "imageId": "ami-02e8aa396f8be3b6d",
        "instanceId": "i-0929128ff2f73a2f1",
        "instanceType": "t2.micro",
        "kernelId": null,

```

(continues on next page)

(continued from previous page)

```

    "marketplaceProductCodes": null,
    "pendingTime": "2020-02-27T20:46:18Z",
    "privateIp": "172.31.81.43",
    "ramdiskId": null,
    "region": "us-east-1",
    "version": "2017-09-30"
  },
  "pkcs7": [
    "MIAGCSqGSib3DQ...",
    "REDACTED",
    "AhQUgq0iPWqPTVnT96tZE6L1XjjLHQAAAAAAA=="
  ],
  "rsa2048": [
    "MIAGCSqGSib...",
    "REDACTED",
    "clYQvuE45xXm7Yreg3QtQbrP//owlleZHj6s350AAAAAAA=="
  ],
  "signature": [
    "dA+QV+LLCWCNRddnrKleYmh2GvYo+t8urDkdgmDSsPi",
    "REDACTED",
    "kDT4ygyJLFkd3b4qjAs="
  ]
},
"meta_data": {
  "ami_id": "ami-02e8aa396f8be3b6d",
  "ami_launch_index": "0",
  "ami_manifest_path": "(unknown)",
  "block_device_mapping": {
    "ami": "/dev/sda1",
    "root": "/dev/sda1"
  },
  "hostname": "ip-172-31-81-43.ec2.internal",
  "instance_action": "none",
  "instance_id": "i-0929128ff2f73a2f1",
  "instance_type": "t2.micro",
  "local_hostname": "ip-172-31-81-43.ec2.internal",
  "local_ipv4": "172.31.81.43",
  "mac": "12:7e:c9:93:29:af",
  "metrics": {
    "vhostmd": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
  },
  "network": {
    "interfaces": {
      "macs": {
        "12:7e:c9:93:29:af": {
          "device_number": "0",
          "interface_id": "eni-0c07a0474339b801d",
          "ipv4_associations": {
            "3.89.187.177": "172.31.81.43"
          },
          "local_hostname": "ip-172-31-81-43.ec2.internal",
          "local_ipv4s": "172.31.81.43",
          "mac": "12:7e:c9:93:29:af",
          "owner_id": "329910648901",
          "public_hostname": "ec2-3-89-187-177.compute-1.amazonaws.com",
          "public_ipv4s": "3.89.187.177",

```

(continues on next page)

(continued from previous page)

```

        "security_group_ids": "sg-0100038b68aa79986",
        "security_groups": "launch-wizard-3",
        "subnet_id": "subnet-04e2d12a",
        "subnet_ipv4_cidr_block": "172.31.80.0/20",
        "vpc_id": "vpc-210b4b5b",
        "vpc_ipv4_cidr_block": "172.31.0.0/16",
        "vpc_ipv4_cidr_blocks": "172.31.0.0/16"
    }
}
},
"placement": {
    "availability_zone": "us-east-1b"
},
"profile": "default-hvm",
"public_hostname": "ec2-3-89-187-177.compute-1.amazonaws.com",
"public_ipv4": "3.89.187.177",
"reservation_id": "r-0c481643d15766a02",
"security_groups": "launch-wizard-3",
"services": {
    "domain": "amazonaws.com",
    "partition": "aws"
}
},
},
"instance_id": "i-0929128ff2f73a2f1",
"kernel_release": "5.3.0-1010-aws",
"local_hostname": "ip-172-31-81-43",
"machine": "x86_64",
"platform": "ec2",
"public_ssh_keys": [],
"python_version": "3.7.6",
"region": "us-east-1",
"sensitive_keys": [],
"subplatform": "metadata (http://169.254.169.254)",
"sys_info": {
    "dist": [
        "ubuntu",
        "20.04",
        "focal"
    ],
    "platform": "Linux-5.3.0-1010-aws-x86_64-with-Ubuntu-20.04-focal",
    "python": "3.7.6",
    "release": "5.3.0-1010-aws",
    "system": "Linux",
    "uname": [
        "Linux",
        "ip-172-31-81-43",
        "5.3.0-1010-aws",
        "#11-Ubuntu SMP Thu Jan 16 07:59:32 UTC 2020",
        "x86_64",
        "x86_64"
    ],
    "variant": "ubuntu"
},
"system_platform": "Linux-5.3.0-1010-aws-x86_64-with-Ubuntu-20.04-focal",
"userdata": "#cloud-config\nssh_import_id: [<my-launchpad-id>]\n...",

```

(continues on next page)

(continued from previous page)

```

"v1": {
  "_beta_keys": [
    "subplatform"
  ],
  "availability_zone": "us-east-1b",
  "cloud_name": "aws",
  "distro": "ubuntu",
  "distro_release": "focal",
  "distro_version": "20.04",
  "instance_id": "i-0929128ff2f73a2f1",
  "kernel": "5.3.0-1010-aws",
  "local_hostname": "ip-172-31-81-43",
  "machine": "x86_64",
  "platform": "ec2",
  "public_ssh_keys": [],
  "python": "3.7.6",
  "region": "us-east-1",
  "subplatform": "metadata (http://169.254.169.254)",
  "system_platform": "Linux-5.3.0-1010-aws-x86_64-with-Ubuntu-20.04-focal",
  "variant": "ubuntu"
},
"variant": "ubuntu",
"vendordata": ""
}

```

1.12.3 Using instance-data

As of cloud-init v. 18.4, any instance-data can be used in:

- User-data scripts
- Cloud config data
- Command line interface via **cloud-init query** or **cloud-init devel render**

This means that any variable present in `/run/cloud-init/instance-data-sensitive.json` can be used, unless a non-root user is using the command line interface. In the non-root user case, `/run/cloud-init/instance-data.json` will be used instead.

Many clouds allow users to provide user-data to an instance at the time the instance is launched. Cloud-init supports a number of *User-Data Formats*.

Both user-data scripts and **#cloud-config** data support jinja template rendering. When the first line of the provided user-data begins with, **## template: jinja** cloud-init will use jinja to render that file. Any instance-data-sensitive.json variables are surfaced as jinja template variables because cloud-config modules are run as 'root' user.

Note: cloud-init also provides jinja-safe key aliases for any instance-data.json keys which contain jinja operator characters such as `+`, `-`, `.`, `/`, etc. Any jinja operator will be replaced with underscores in the jinja-safe key alias. This allows for cloud-init templates to use aliased variable references which allow for jinja's dot-notation reference such as `{{ ds.v1_0.my_safe_key }}` instead of `{{ ds["v1.0"]["my/safe-key"] }}`.

Below are some other examples of using jinja templates in user-data:

- Cloud config calling home with the ec2 public hostname and availability-zone

```
## template: jinja
#cloud-config
runcmd:
- echo 'EC2 public hostname allocated to instance: {{
  ds.meta_data.public_hostname }}' > /tmp/instance_metadata
- echo 'EC2 availability zone: {{ v1.availability_zone }}' >>
  /tmp/instance_metadata
- curl -X POST -d '{"hostname": "{{ds.meta_data.public_hostname }}",
  "availability-zone": "{{ v1.availability_zone }}"}'
  https://example.com
```

- Custom user-data script performing different operations based on region

```
## template: jinja
#!/bin/bash
{% if v1.region == 'us-east-2' -%}
echo 'Installing custom proxies for {{ v1.region }}'
sudo apt-get install my-xtra-fast-stack
{%- endif %}
...
```

One way to easily explore what Jinja variables are available on your machine is to use the `cloud-init query --format (-f)` commandline option which will render any Jinja syntax you use. Warnings or exceptions will be raised on invalid instance-data keys, paths or invalid syntax.

```
# List all instance-data keys and values as root user
% sudo cloud-init query --all
{...}

# Introspect nested keys on an object
% cloud-init query -f "{{ds.keys()}}"
dict_keys(['meta_data', '_doc'])

# Test your Jinja rendering syntax on the command-line directly

# Failure to reference valid top-level instance-data key
% cloud-init query -f "{{invalid.instance-data.key}}"
WARNING: Ignoring jinja template for query commandline: 'invalid' is undefined

# Failure to reference valid dot-delimited key path on a known top-level key
% cloud-init query -f "{{v1.not_here}}"
WARNING: Could not render jinja template variables in file 'query commandline': 'not_
↪here'
CI_MISSING_JINJA_VAR/not_here

# Test expected value using valid instance-data key path
% cloud-init query -f "My AMI: {{ds.meta_data.ami_id}}"
My AMI: ami-0fecc35d3c8ba8d60
```

Note: Trying to reference jinja variables that don't exist in instance-data will result in warnings in `/var/log/cloud-init.log` and the following string in your rendered user-data: `CI_MISSING_JINJA_VAR/<your_varname>`.

Cloud-init also surfaces a command line tool **cloud-init query** which can assist developers or scripts with obtaining instance metadata easily. See [query](#) for more information.

To cut down on keystrokes on the command line, cloud-init also provides top-level key aliases for any standardized `v#` keys present. The preceding `v1` is not required of `v1.var_name`. These aliases will represent the value of the highest versioned standard key. For example, `cloud_name` value will be `v2.cloud_name` if both `v1` and `v2` keys are present in `instance-data.json`. The **query** command also publishes `userdata` and `vendordata` keys to the root user which will contain the decoded user and vendor data provided to this instance. Non-root users referencing `userdata` or `vendordata` keys will see only redacted values.

```
# List all top-level instance-data keys available
% cloud-init query --list-keys

# Find your EC2 ami-id
% cloud-init query ds.metadata.ami_id

# Format your cloud_name and region using jinja template syntax
% cloud-init query --format 'cloud: {{ v1.cloud_name }} myregion: {{
% v1.region }}'

# Locally test that your template userdata provided to the vm was rendered as
# intended.
% cloud-init query --format "$(sudo cloud-init query userdata)"

# The --format command renders jinja templates, this can also be used
# to develop and test jinja template constructs
% cat > test-templating.yaml <<EOF
  {% for val in ds.meta_data.keys() %}
  - {{ val }}
  {% endfor %}
EOF
% cloud-init query --format="$( cat test-templating.yaml )"
- instance_id
- dsmode
- local_hostname
```

Note: To save time designing a user-data template for a specific cloud's `instance-data.json`, use the 'render' cloud-init command on an instance booted on your favorite cloud. See [devel](#) for more information.

1.13 Datasources

Datasources are sources of configuration data for cloud-init that typically come from the user (i.e. `userdata`) or come from the cloud that created the configuration drive (i.e. `metadata`). Typical `userdata` would include files, YAML, and shell scripts while typical `metadata` would include server name, instance id, display name and other cloud specific details.

Since there are multiple ways to provide this data (each cloud solution seems to prefer its own way) internally a datasource abstract class was created to allow for a single way to access the different cloud systems methods to provide this data through the typical usage of subclasses.

Any metadata processed by cloud-init's datasources is persisted as `/run/cloud-init/instance-data.json`. Cloud-init provides tooling to quickly introspect some of that data. See [Instance Metadata](#) for more information.

1.13.1 Known Sources

The following is a list of documents for each supported datasource:

Alibaba Cloud (AliYun)

The `AliYun` datasource reads data from Alibaba Cloud ECS. Support is present in cloud-init since 0.7.9.

Metadata Service

The Alibaba Cloud metadata service is available at the well known url `http://100.100.100.200/`. For more information see Alibaba Cloud ECS on [metadata](#).

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

An example configuration with the default values is provided below:

```
datasource:
  AliYun:
    metadata_urls: ["http://100.100.100.200"]
    timeout: 50
    max_wait: 120
```

Versions

Like the EC2 metadata service, Alibaba Cloud's metadata service provides versioned data under specific paths. As of April 2018, there are only `2016-01-01` and `latest` versions.

It is expected that the dated version will maintain a stable interface but `latest` may change content at a future date.

Cloud-init uses the `2016-01-01` version.

You can list the versions available to your instance with:

```
$ curl http://100.100.100.200/
2016-01-01
latest
```

Metadata

Instance metadata can be queried at `http://100.100.100.200/2016-01-01/meta-data`

```
$ curl http://100.100.100.200/2016-01-01/meta-data
dns-conf/
eipv4
hostname
image-id
instance-id
instance/
mac
network-type
network/
ntp-conf/
```

(continues on next page)

(continued from previous page)

```
owner-account-id
private-ipv4
public-keys/
region-id
serial-number
source-address
sub-private-ipv4-list
vpc-cidr-block
vpc-id
```

Userdata

If provided, user-data will appear at `http://100.100.100.200/2016-01-01/user-data`. If no user-data is provided, this will return a 404.

```
$ curl http://100.100.100.200/2016-01-01/user-data
#!/bin/sh
echo "Hello World."
```

Alt Cloud

The datasource altcloud will be used to pick up user data on **RHEVm** and **vSphere**.

RHEVm

For **RHEVm** v3.0 the userdata is injected into the VM using floppy injection via the **RHEVm** dashboard “Custom Properties”.

The format of the Custom Properties entry must be:

```
floppyinject=user-data.txt:<base64 encoded data>
```

For example to pass a simple bash script:

```
% cat simple_script.bash
#!/bin/bash
echo "Hello Joe!" >> /tmp/JJV_Joe_out.txt

% base64 < simple_script.bash
IyEvYmluL2Jhc2gKZWNoYAiSGVsbG8gSm9lISIgPj4gL3RtcC9KS1ZfSm9lX291dC50eHQK
```

To pass this example script to cloud-init running in a **RHEVm** v3.0 VM set the “Custom Properties” when creating the **RHEMv** v3.0 VM to:

```
floppyinject=user-data.
→txt:IyEvYmluL2Jhc2gKZWNoYAiSGVsbG8gSm9lISIgPj4gL3RtcC9KS1ZfSm9lX291dC50eHQK
```

NOTE: The prefix with file name must be: `floppyinject=user-data.txt:`

It is also possible to launch a **RHEVm** v3.0 VM and pass optional user data to it using the Delta Cloud.

For more information on Delta Cloud see: <http://deltacloud.apache.org>

vSphere

For VMWare's **vSphere** the userdata is injected into the VM as an ISO via the cdrom. This can be done using the **vSphere** dashboard by connecting an ISO image to the CD/DVD drive.

To pass this example script to cloud-init running in a **vSphere** VM set the CD/DVD drive when creating the vSphere VM to point to an ISO on the data store.

Note: The ISO must contain the user data.

For example, to pass the same `simple_script.bash` to vSphere:

Create the ISO

```
% mkdir my-iso
```

NOTE: The file name on the ISO must be: `user-data.txt`

```
% cp simple_script.bash my-iso/user-data.txt
% genisoimage -o user-data.iso -r my-iso
```

Verify the ISO

```
% sudo mkdir /media/vsphere_iso
% sudo mount -o loop user-data.iso /media/vsphere_iso
% cat /media/vsphere_iso/user-data.txt
% sudo umount /media/vsphere_iso
```

Then, launch the **vSphere** VM the ISO `user-data.iso` attached as a CDRROM.

It is also possible to launch a **vSphere** VM and pass optional user data to it using the Delta Cloud.

For more information on Delta Cloud see: <http://deltacloud.apache.org>

Amazon EC2

The EC2 datasource is the oldest and most widely used datasource that cloud-init supports. This datasource interacts with a *magic* ip that is provided to the instance by the cloud provider. Typically this ip is `169.254.169.254` of which at this ip a http server is provided to the instance so that the instance can make calls to get instance userdata and instance metadata.

Metadata is accessible via the following URL:

```
GET http://169.254.169.254/2009-04-04/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-id
instance-type
local-hostname
local-ipv4
placement/
```

(continues on next page)

(continued from previous page)

```
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

Userdata is accessible via the following URL:

```
GET http://169.254.169.254/2009-04-04/user-data
1234,fred,reboot,true | 4512,jimbo, | 173,,,
```

Note that there are multiple versions of this data provided, cloud-init by default uses **2009-04-04** but newer versions can be supported with relative ease (newer versions have more data exposed, while maintaining backward compatibility with the previous versions). Version **2016-09-02** is required for secondary IP address support.

To see which versions are supported from your cloud provider use the following URL:

```
GET http://169.254.169.254/
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
...
latest
```

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

The settings that may be configured are:

- **metadata_urls**: This list of urls will be searched for an Ec2 metadata service. The first entry that successfully returns a 200 response for `<url>/<version>/meta-data/instance-id` will be selected. (default: [`'http://169.254.169.254'`, `'http://instance-data:8773'`]).
- **max_wait**: the maximum amount of clock time in seconds that should be spent searching metadata_urls. A value less than zero will result in only one request being made, to the first in the list. (default: 120)
- **timeout**: the timeout value provided to `urlopen` for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 50)
- **apply_full_imds_network_config**: Boolean (default: True) to allow cloud-init to configure any secondary NICs and secondary IPs described by the metadata service. All network interfaces are configured with DHCP (v4) to obtain an primary IPv4 address and route. Interfaces which have a non-empty 'ipv6s' list will also enable DHCPv6 to obtain a primary IPv6 address and route. The DHCP response (v4 and v6) return an IP that matches the first element of `local-ipv4s` and `ipv6s` lists respectively. All additional values (secondary addresses) in the static ip lists will be added to interface.

An example configuration with the default values is provided below:


```
datasource:
  Ec2:
    metadata_urls: ["http://169.254.169.254:80", "http://instance-data:8773"]
    max_wait: 120
    timeout: 50
    apply_full_imds_network_config: true
```

Notes

- There are 2 types of EC2 instances network-wise: VPC ones (Virtual Private Cloud) and Classic ones (also known as non-VPC). One major difference between them is that Classic instances have their MAC address changed on stop/restart operations, so cloud-init will recreate the network config file for EC2 Classic instances every boot. On VPC instances this file is generated only in the first boot of the instance. The check for the instance type is performed by `is_classic_instance()` method.
- For EC2 instances with multiple network interfaces (NICs) attached, dhcp4 will be enabled to obtain the primary private IPv4 address of those NICs. Wherever dhcp4 or dhcp6 is enabled for a NIC, a dhcp route-metric will be added with the value of `<device-number + 1> * 100` to ensure dhcp routes on the primary NIC are preferred to any secondary NICs. For example: the primary NIC will have a DHCP route-metric of 100, the next NIC will be 200.

Azure

This datasource finds metadata and user-data from the Azure cloud platform.

The Azure cloud platform provides initial data to an instance via an attached CD formatted in UDF. That CD contains a 'ovf-env.xml' file that provides some information. Additional information is obtained via interaction with the "endpoint".

To find the endpoint, we now leverage the dhcp client's ability to log its known values on exit. The endpoint server is special DHCP option 245. Depending on your networking stack, this can be done by calling a script in `/etc/dhcp/dhclient-exit-hooks` or a file in `/etc/NetworkManager/dispatcher.d`. Both of these call a sub-command 'dhclient_hook' of cloud-init itself. This sub-command will write the client information in json format to `/run/cloud-init/dhclient.hook/<interface>.json`.

If those files are not available, the fallback is to check the leases file for the endpoint server (again option 245).

You can define the path to the lease file with the 'dhclient_lease_file' configuration.

IMDS

Azure provides the [instance metadata service \(IMDS\)](#) which is a REST service on `169.254.169.254` providing additional configuration information to the instance. Cloud-init uses the IMDS for:

- network configuration for the instance which is applied per boot
- a preprovisioning gate which blocks instance configuration until Azure fabric is ready to provision
- retrieving SSH public keys. Cloud-init will first try to utilize SSH keys returned from IMDS, and if they are not provided from IMDS then it will fallback to using the OVF file provided from the CD-ROM. There is a large performance benefit to using IMDS for SSH key retrieval, but in order to support environments where IMDS is not available then we must continue to all for keys from OVF

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

The settings that may be configured are:

- **apply_network_config**: Boolean set to True to use network configuration described by Azure's IMDS endpoint instead of fallback network config of dhcp on eth0. Default is True. For Ubuntu 16.04 or earlier, default is False.
- **data_dir**: Path used to read metadata files and write crawled data.
- **dhclient_lease_file**: The fallback lease file to source when looking for custom DHCP option 245 from Azure fabric.
- **disk_aliases**: A dictionary defining which device paths should be interpreted as ephemeral images. See `cc_disk_setup` module for more info.

Configuration for the datasource can also be read from a `dscfg` entry in the `LinuxProvisioningConfigurationSet`. Content in `dscfg` node is expected to be base64 encoded yaml content, and it will be merged into the 'datasource: Azure' entry.

An example configuration with the default values is provided below:

```
datasource:
  Azure:
    apply_network_config: true
    data_dir: /var/lib/waagent
    dhclient_lease_file: /var/lib/dhcp/dhclient.eth0.leases
    disk_aliases:
      ephemeral0: /dev/disk/cloud/azure_resource
```

Userdata

Userdata is provided to cloud-init inside the `ovf-env.xml` file. Cloud-init expects that user-data will be provided as base64 encoded value inside the text child of a element named `UserData` or `CustomData` which is a direct child of the `LinuxProvisioningConfigurationSet` (a sibling to `UserName`) If both `UserData` and `CustomData` are provided behavior is undefined on which will be selected.

In the example below, user-data provided is 'this is my userdata'

Example:

```
<wa:ProvisioningSection>
  <wa:Version>1.0</wa:Version>
  <LinuxProvisioningConfigurationSet
    xmlns="http://schemas.microsoft.com/windowsazure"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <ConfigurationSetType>LinuxProvisioningConfiguration</ConfigurationSetType>
    <HostName>myHost</HostName>
    <UserName>myuser</UserName>
    <UserPassword/>
    <CustomData>dGhpYBpcyBteSB1c2VyZGF0YQ===</CustomData>
    <dscfg>eyJhZ2VudF9jb21tYW5kIjogWyJzdGFydCI6ICJ3YWxpbnV4YWdlbnQiXX0=</dscfg>
    <DisableSshPasswordAuthentication>true</DisableSshPasswordAuthentication>
    <SSH>
      <PublicKeys>
        <PublicKey>
```

(continues on next page)

(continued from previous page)

```

    <Fingerprint>6BE7A7C3C8A8F4B123CCA5D0C2F1BE4CA7B63ED7</Fingerprint>
    <Path>this-value-unused</Path>
  </PublicKey>
</PublicKeys>
</SSH>
</LinuxProvisioningConfigurationSet>
</wa:ProvisioningSection>

```

hostname

When the user launches an instance, they provide a hostname for that instance. The hostname is provided to the instance in the `ovf-env.xml` file as `HostName`.

Whatever value the instance provides in its dhcp request will resolve in the domain returned in the ‘search’ request.

A generic image will already have a hostname configured. The ubuntu cloud images have ‘ubuntu’ as the hostname of the system, and the initial dhcp request on `eth0` is not guaranteed to occur after the datasource code has been run. So, on first boot, that initial value will be sent in the dhcp request and *that* value will resolve.

In order to make the `HostName` provided in the `ovf-env.xml` resolve, a dhcp request must be made with the new value. cloud-init handles this by setting the hostname in the DataSource’s ‘get_data’ method via ‘hostname \$HostName’, and then bouncing the interface. This behavior can be configured or disabled in the datasource config. See ‘Configuration’ above.

CloudSigma

This datasource finds metadata and user-data from the [CloudSigma](#) cloud platform. Data transfer occurs through a virtual serial port of the [CloudSigma](#)’s VM and the presence of network adapter is **NOT** a requirement, See [server context](#) in the public documentation for more information.

Setting a hostname

By default the name of the server will be applied as a hostname on the first boot.

Providing user-data

You can provide user-data to the VM using the dedicated [meta field](#) in the [server context](#) `cloudinit-user-data`. By default *cloud-config* format is expected there and the `#cloud-config` header could be omitted. However since this is a raw-text field you could provide any of the valid [config formats](#).

You have the option to encode your user-data using Base64. In order to do that you have to add the `cloudinit-user-data` field to the `base64_fields`. The latter is a comma-separated field with all the meta fields whit base64 encoded values.

If your user-data does not need an internet connection you can create a [meta field](#) in the [server context](#) `cloudinit-dsmode` and set “local” as value. If this field does not exist the default value is “net”.

CloudStack

Apache CloudStack expose user-data, meta-data, user password and account SSH key thru the Virtual-Router. The datasource obtains the VR address via dhcp lease information given to the instance. For more details on meta-data and user-data, refer the [CloudStack Administrator Guide](#).

URLs to access user-data and meta-data from the Virtual Machine. *data-server* is a well-known hostname provided by the CloudStack virtual router that points to the next UserData server (which is usually also the virtual router).

```
http://data-server./latest/user-data
http://data-server./latest/meta-data
http://data-server./latest/meta-data/{metadata type}
```

If *data-server* cannot be resolved, cloud-init will try to obtain the virtual router's address from the system's DHCP leases. If that fails, it will use the system's default gateway.

Configuration

The following configuration can be set for the datasource in system configuration (in */etc/cloud/cloud.cfg* or */etc/cloud/cloud.cfg.d/*).

The settings that may be configured are:

- **max_wait**: the maximum amount of clock time in seconds that should be spent searching metadata_urls. A value less than zero will result in only one request being made, to the first in the list. (default: 120)
- **timeout**: the timeout value provided to urlopen for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 50)

An example configuration with the default values is provided below:

```
datasource:
  CloudStack:
    max_wait: 120
    timeout: 50
```

Config Drive

The configuration drive datasource supports the [OpenStack](#) configuration drive disk.

See the [config drive extension](#) and [metadata introduction](#) in the public documentation for more information.

By default, cloud-init does *always* consider this source to be a full-fledged datasource. Instead, the typical behavior is to assume it is really only present to provide networking information. Cloud-init will copy off the network information, apply it to the system, and then continue on. The “full” datasource could then be found in the EC2 metadata service. If this is not the case then the files contained on the located drive must provide equivalents to what the EC2 metadata service would provide (which is typical of the version 2 support listed below)

Version 1

Note: Version 1 is legacy and should be considered deprecated. Version 2 has been supported in OpenStack since 2012.2 (Folsom).

The following criteria are required to as a config drive:

1. Must be formatted with **vfat** filesystem
2. Must contain *one* of the following files

```
/etc/network/interfaces
/root/.ssh/authorized_keys
/meta.js
```

```
/etc/network/interfaces
```

This file is laid down by nova in order to pass static networking information to the guest. Cloud-init will copy it off of the config-drive and into /etc/network/interfaces (or convert it to RH format) as soon as it can, and then attempt to bring up all network interfaces.

```
/root/.ssh/authorized_keys
```

This file is laid down by nova, and contains the ssh keys that were provided to nova on instance creation (nova-boot -key)

```
/meta.js
```

meta.js is populated on the config-drive in response to the user passing “meta flags” (nova boot -meta key=value ...). It is expected to be json formatted.

Version 2

The following criteria are required to as a config drive:

1. Must be formatted with **vfat** or **iso9660** filesystem or have a *filesystem* label of **config-2**
2. The files that will typically be present in the config drive are:

```
openstack/
- 2012-08-10/ or latest/
- meta_data.json
- user_data (not mandatory)
- content/
- 0000 (referenced content files)
- 0001
- ....
ec2
- latest/
- meta-data.json (not mandatory)
```

Keys and values

Cloud-init’s behavior can be modified by keys found in the meta.js (version 1 only) file in the following ways.

```
dsmode:
values: local, net, pass
default: pass
```

This is what indicates if configdrive is a final data source or not. By default it is ‘pass’, meaning this datasource should not be read. Set it to ‘local’ or ‘net’ to stop cloud-init from continuing on to search for other data sources after network config.

The difference between ‘local’ and ‘net’ is that local will not require networking to be up before user-data actions (or boothooks) are run.

```
instance-id:
  default: iid-dsconfigdrive
```

This is utilized as the metadata's instance-id. It should generally be unique, as it is what is used to determine “is this a new instance”.

```
public-keys:
  default: None
```

If present, these keys will be used as the public keys for the instance. This value overrides the content in `authorized_keys`.

Note: it is likely preferable to provide keys via user-data

```
user-data:
  default: None
```

This provides cloud-init user-data. See *examples* for what all can be present here.

DigitalOcean

The [DigitalOcean](#) datasource consumes the content served from DigitalOcean's [metadata service](#). This metadata service serves information about the running droplet via HTTP over the link local address 169.254.169.254. The metadata API endpoints are fully described at <https://developers.digitalocean.com/metadata/>.

Configuration

DigitalOcean's datasource can be configured as follows:

datasource:

DigitalOcean: retries: 3 timeout: 2

- *retries*: Determines the number of times to attempt to connect to the metadata service
- *timeout*: Determines the timeout in seconds to wait for a response from the metadata service

E24Cloud

[E24Cloud](#) platform provides an AWS Ec2 metadata service clone. It identifies itself to guests using the dmi system-manufacturer (`/sys/class/dmi/id/sys_vendor`).

Exoscale

This datasource supports reading from the metadata server used on the [Exoscale platform](#).

Use of the Exoscale datasource is recommended to benefit from new features of the Exoscale platform.

The datasource relies on the availability of a compatible metadata server (`http://169.254.169.254` is used by default) and its companion password server, reachable at the same address (by default on port 8080).

Crawling of metadata

The metadata service and password server are crawled slightly differently:

- The “metadata service” is crawled every boot.
- The password server is also crawled every boot (the Exoscale datasource forces the password module to run with “frequency always”).

In the password server case, the following rules apply in order to enable the “restore instance password” functionality:

- If a password is returned by the password server, it is then marked “saved” by the cloud-init datasource. Subsequent boots will skip setting the password (the password server will return “saved_password”).
- When the instance password is reset (via the Exoscale UI), the password server will return the non-empty password at next boot, therefore causing cloud-init to reset the instance’s password.

Configuration

Users of this datasource are discouraged from changing the default settings unless instructed to by Exoscale support.

The following settings are available and can be set for the datasource in system configuration (in */etc/cloud/cloud.cfg.d/*).

The settings available are:

- **metadata_url**: The URL for the metadata service (defaults to `http://169.254.169.254`)
- **api_version**: The API version path on which to query the instance metadata (defaults to `1.0`)
- **password_server_port**: The port (on the metadata server) on which the password server listens (defaults to `8080`).
- **timeout**: the timeout value provided to `urlopen` for each individual http request. (defaults to `10`)
- **retries**: The number of retries that should be done for an http request (defaults to `6`)

An example configuration with the default values is provided below:

```
datasource:
  Exoscale:
    metadata_url: "http://169.254.169.254"
    api_version: "1.0"
    password_server_port: 8080
    timeout: 10
    retries: 6
```

Fallback/None

This is the fallback datasource when no other datasource can be selected. It is the equivalent of a empty datasource in that it provides a empty string as userdata and a empty dictionary as metadata. It is useful for testing as well as for when you do not have a need to have an actual datasource to meet your instance requirements (ie you just want to run modules that are not concerned with any external data). It is typically put at the end of the datasource search list so that if all other datasources are not matched, then this one will be so that the user is not left with an inaccessible instance.

Note: the instance id that this datasource provides is `iid-datasource-none`.

Google Compute Engine

The GCE datasource gets its data from the internal compute metadata server. Metadata can be queried at the URL `'http://metadata.google.internal/computeMetadata/v1/'` from within an instance. For more information see the [GCE metadata docs](#).

Currently the default project and instance level metadatakeys keys `project/attributes/sshKeys` and `instance/attributes/ssh-keys` are merged to provide `public-keys`.

`user-data` and `user-data-encoding` can be provided to cloud-init by setting those custom metadata keys for an *instance*.

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

The settings that may be configured are:

- **retries**: The number of retries that should be done for an http request. This value is used only after `metadata_url` is selected. (default: 5)
- **sec_between_retries**: The amount of wait time between the retries when crawling the metadata service. (default: 1)

An example configuration with the default values is provided below:

```
datasource:  
  GCE:  
    retries: 5  
    sec_between_retries: 1
```

LXD

The data source LXD allows the user to provide custom user-data, vendor-data, meta-data and network-config to the instance without running a network service (or even without having a network at all). This datasource performs HTTP GETs against the [LXD socket device](#) which is provided to each running LXD container and VM as `/dev/lxd/sock` and represents all instance-metadata as versioned HTTP routes such as:

- `1.0/meta-data`
- `1.0/config/user.meta-data`
- `1.0/config/user.vendor-data`
- `1.0/config/user.user-data`
- `1.0/config/user.<any-custom-key>`

The LXD socket device `/dev/lxd/sock` is only present on containers and VMs when the instance configuration has `security.devlxd=true` (default). Disabling `security.devlxd` configuration setting at initial launch will ensure that cloud-init uses the *NoCloud* datasource. Disabling `security.devlxd` over the life of the container will result in warnings from cloud-init and cloud-init will keep the originally detected LXD datasource.

The LXD datasource provides cloud-init the opportunity to react to meta-data, vendor-data, user-data and network-config changes and render the updated configuration across a system reboot.

One can manipulate what meta-data, vendor-data or user-data is provided to the launched container using the LXD profiles or `lxc launch ... -c <key>=<value>` at initial container launch using one of the following keys:

- `user.meta-data`: YAML metadata which will be appended to base meta-data
- `user.vendor-data`: YAML which overrides any meta-data values
- `user.network-config`: YAML representing either *Networking Config Version 1* or *Networking Config Version 2* format
- `user.user-data`: YAML which takes preference and overrides both meta-data and vendor-data values
- `user.any-key`: Custom user configuration key and value pairs can be passed to cloud-init. Those keys/values will be present in instance-data which can be used by both `#template: jinja` `#cloud-config` templates and the `cloud-init query` command.

By default, network configuration from this datasource will be:

```
version: 1
config:
  - type: physical
    name: eth0
    subnets:
      - type: dhcp
        control: auto
```

This datasource is intended to replace *NoCloud* datasource for LXD instances with a more direct support for LXD APIs instead of static NoCloud seed files.

MAAS

TODO

For now see: <https://maas.io/docs>

NoCloud

The data source NoCloud allows the user to provide user-data and meta-data to the instance without running a network service (or even without having a network at all).

You can provide meta-data and user-data to a local vm boot via files on a `vfat` or `iso9660` filesystem. The filesystem volume label must be `cidata` or `CIDATA`.

Alternatively, you can provide meta-data via kernel command line or SMBIOS “serial number” option. The data must be passed in the form of a string:

```
ds=nocloud[;key=val;key=val]
```

or

```
ds=nocloud-net[;key=val;key=val]
```

The permitted keys are:

- `h` or `local-hostname`
- `i` or `instance-id`
- `s` or `seedfrom`

With `ds=nocloud`, the `seedfrom` value must start with `/` or `file://`. With `ds=nocloud-net`, the `seedfrom` value must start with `http://` or `https://`.

e.g. you can pass this option to QEMU:

```
-smbios type=1,serial=ds=nocloud-net;s=http://10.10.0.1:8000/
```

to cause NoCloud to fetch the full meta-data from <http://10.10.0.1:8000/meta-data> after the network initialization is complete.

These user-data and meta-data files are expected to be in the following format.

```
/user-data
/meta-data
```

Both files are required to be present for it to be considered a valid seed ISO.

Basically, user-data is simply user-data and meta-data is a YAML formatted file representing what you'd find in the EC2 metadata service.

You may also optionally provide a vendor-data file in the following format.

```
/vendor-data
```

Given a disk ubuntu 12.04 cloud image in 'disk.img', you can create a sufficient disk by following the example below.

```
## create user-data and meta-data files that will be used
## to modify image on first boot
$ { echo instance-id: iid-local01; echo local-hostname: cloudimg; } > meta-data

$ printf "#cloud-config\npassword: passw0rd\nchpasswd: { expire: False }\nssh_pwauth: \n\nTrue\n" > user-data

## create a disk to attach with some user-data and meta-data
$ genisoimage -output seed.iso -volid cidata -joliet -rock user-data meta-data

## alternatively, create a vfat filesystem with same files
## $ truncate --size 2M seed.img
## $ mkfs.vfat -n cidata seed.img
## $ mcopy -oi seed.img user-data meta-data ::

## create a new qcow image to boot, backed by your original image
$ qemu-img create -f qcow2 -b disk.img boot-disk.img

## boot the image and login as 'ubuntu' with password 'passw0rd'
## note, passw0rd was set as password through the user-data above,
## there is no password set on these images.
$ kvm -m 256 \
    -net nic -net user,hostfwd=tcp::2222-:22 \
    -drive file=boot-disk.img,if=virtio \
    -drive file=seed.iso,if=virtio
```

Note: that the instance-id provided (iid-local01 above) is what is used to determine if this is “first boot”. So if you are making updates to user-data you will also have to change that, or start the disk fresh.

Also, you can inject an `/etc/network/interfaces` file by providing the content for that file in the `network-interfaces` field of metadata.

Example metadata:

```
instance-id: iid-abcdefg
network-interfaces: |
  iface eth0 inet static
  address 192.168.1.10
  network 192.168.1.0
  netmask 255.255.255.0
  broadcast 192.168.1.255
  gateway 192.168.1.254
hostname: myhost
```

Network configuration can also be provided to cloud-init in either *Networking Config Version 1* or *Networking Config Version 2* by providing that YAML formatted data in a file named `network-config`. If found, this file will override a `network-interfaces` file.

See an example below. Note specifically that this file does not have a top level `network` key as it is already assumed to be network configuration based on the filename.

```
version: 1
config:
  - type: physical
    name: interface0
    mac_address: "52:54:00:12:34:00"
    subnets:
      - type: static
        address: 192.168.1.10
        netmask: 255.255.255.0
        gateway: 192.168.1.254
```

```
version: 2
ethernets:
  interface0:
    match:
      mac_address: "52:54:00:12:34:00"
    set-name: interface0
    addresses:
      - 192.168.1.10/255.255.255.0
    gateway4: 192.168.1.254
```

OpenNebula

The **OpenNebula** (ON) datasource supports the contextualization disk.

See [contextualization overview](#), [contextualizing VMs](#) and [network configuration](#) in the public documentation for more information.

OpenNebula's virtual machines are contextualized (parametrized) by CD-ROM image, which contains a shell script `context.sh` with custom variables defined on virtual machine start. There are no fixed contextualization variables, but the datasource accepts many used and recommended across the documentation.

Datasource configuration

Datasource accepts following configuration options.

```
dsmode:
  values: local, net, disabled
  default: net
```

Tells if this datasource will be processed in ‘local’ (pre-networking) or ‘net’ (post-networking) stage or even completely ‘disabled’.

```
parseuser:
  default: nobody
```

Unprivileged system user used for contextualization script processing.

Contextualization disk

The following criteria are required:

1. Must be formatted with [iso9660](#) filesystem or have a *filesystem* label of **CONTEXT** or **CDROM**
2. Must contain file *context.sh* with contextualization variables. File is generated by OpenNebula, it has a KEY=‘VALUE’ format and can be easily read by bash

Contextualization variables

There are no fixed contextualization variables in OpenNebula, no standard. Following variables were found on various places and revisions of the OpenNebula documentation. Where multiple similar variables are specified, only first found is taken.

```
DSMODE
```

Datasource mode configuration override. Values: local, net, disabled.

```
DNS
ETH<x>_IP
ETH<x>_NETWORK
ETH<x>_MASK
ETH<x>_GATEWAY
ETH<x>_GATEWAY6
ETH<x>_DOMAIN
ETH<x>_DNS
ETH<x>_SEARCH_DOMAIN
ETH<x>_MTU
ETH<x>_IP6
ETH<x>_IP6_ULA
ETH<x>_IP6_PREFIX_LENGTH
ETH<x>_IP6_GATEWAY
```

Static [network configuration](#).

```
SET_HOSTNAME
HOSTNAME
```

Instance hostname.

```
PUBLIC_IP
IP_PUBLIC
ETH0_IP
```

If no hostname has been specified, cloud-init will try to create hostname from instance's IP address in 'local' dsmode. In 'net' dsmode, cloud-init tries to resolve one of its IP addresses to get hostname.

```
SSH_KEY
SSH_PUBLIC_KEY
```

One or multiple SSH keys (separated by newlines) can be specified.

```
USER_DATA
USERDATA
```

cloud-init user data.

Example configuration

This example cloud-init configuration (*cloud.cfg*) enables OpenNebula datasource only in 'net' mode.

```
disable_ec2_metadata: True
datasource_list: ['OpenNebula']
datasource:
  OpenNebula:
    dsmode: net
    parseuser: nobody
```

Example VM's context section

```
CONTEXT=[
  SSH_KEY="$USER[SSH_KEY]
  $USER[SSH_KEY1]
  $USER[SSH_KEY2]",
  PUBLIC_IP="$NIC[IP]",
  USER_DATA="#cloud-config
# see https://help.ubuntu.com/community/CloudInit

packages: []

mounts:
- [vdc, none, swap, sw, 0, 0]
runcmd:
- echo 'Instance has been configured by cloud-init.' | wall
" ]
```

OpenStack

This datasource supports reading data from the [OpenStack Metadata Service](#).

Discovery

To determine whether a platform looks like it may be OpenStack, cloud-init checks the following environment attributes as a potential OpenStack platform:

- Maybe OpenStack if:
 - **non-x86 cpu architecture**: because DMI data is buggy on some arches
- Is OpenStack **if x86 architecture and ANY** of the following:
 - **/proc/1/environ**: Nova-lxd contains *product_name=OpenStack Nova*
 - **DMI product_name**: Either *Openstack Nova* or *OpenStack Compute*
 - **DMI chassis_asset_tag** is *OpenTelekomCloud*, *SAP CCloud VM*, *OpenStack Nova* (since 19.2) or *OpenStack Compute* (since 19.2)

Configuration

The following configuration can be set for the datasource in system configuration (in */etc/cloud/cloud.cfg* or */etc/cloud/cloud.cfg.d/*).

The settings that may be configured are:

- **metadata_urls**: This list of urls will be searched for an OpenStack metadata service. The first entry that successfully returns a 200 response for <url>/openstack will be selected. (default: [*'http://169.254.169.254'*])
- **max_wait**: the maximum amount of clock time in seconds that should be spent searching metadata_urls. A value less than zero will result in only one request being made, to the first in the list. (default: -1)
- **timeout**: the timeout value provided to urlopen for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 10)
- **retries**: The number of retries that should be done for an http request. This value is used only after metadata_url is selected. (default: 5)
- **apply_network_config**: A boolean specifying whether to configure the network for the instance based on network_data.json provided by the metadata service. When False, only configure dhcp on the primary nic for this instances. (default: True)

An example configuration with the default values is provided below:

```
datasource:
  OpenStack:
    metadata_urls: ["http://169.254.169.254"]
    max_wait: -1
    timeout: 10
    retries: 5
    apply_network_config: True
```

Vendor Data

The OpenStack metadata server can be configured to serve up vendor data which is available to all instances for consumption. OpenStack vendor data is, generally, a JSON object.

cloud-init will look for configuration in the `cloud-init` attribute of the vendor data JSON object. cloud-init processes this configuration using the same handlers as user data, so any formats that work for user data should work for vendor data.

For example, configuring the following as vendor data in OpenStack would upgrade packages and install `htop` on all instances:

```
{ "cloud-init": "#cloud-config\npackage_upgrade: True\npackages:\n - htop" }
```

For more general information about how cloud-init handles vendor data, including how it can be disabled by users on instances, see [Vendor Data](#).

OpenStack can also be configured to provide ‘dynamic vendordata’ which is provided by the DynamicJSON provider and appears under a different metadata path, `/vendor_data2.json`.

Cloud-init will look for a `cloud-init` at the `vendor_data2` path; if found, settings are applied after (and, hence, overriding) the settings from static vendor data. Both sets of vendor data can be overridden by user data.

Oracle

This datasource reads metadata, vendor-data and user-data from [Oracle Compute Infrastructure \(OCI\)](#).

Oracle Platform

OCI provides bare metal and virtual machines. In both cases, the platform identifies itself via DMI data in the chassis asset tag with the string ‘OracleCloud.com’.

Oracle’s platform provides a metadata service that mimics the 2013-10-17 version of OpenStack metadata service. Initially support for Oracle was done via the OpenStack datasource.

Cloud-init has a specific datasource for Oracle in order to:

- a. allow and support future growth of the OCI platform.
- b. address small differences between OpenStack and Oracle metadata implementation.

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

The settings that may be configured are:

- **configure_secondary_nics**: A boolean, defaulting to False. If set to True on an OCI Virtual Machine, cloud-init will fetch networking metadata from Oracle’s IMDS and use it to configure the non-primary network interface controllers in the system. If set to True on an OCI Bare Metal Machine, it will have no effect (though this may change in the future).

An example configuration with the default values is provided below:

```
datasource:
  Oracle:
    configure_secondary_nics: false
```

OVF

The OVF Datasource provides a datasource for reading data from on an [Open Virtualization Format](#) ISO transport.

For further information see a full working example in cloud-init’s source code tree in `doc/sources/ovf`

Configuration

The following configuration can be set for the datasource in system configuration (in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d/`).

The settings that may be configured are:

- `disable_vmware_customization`: disable or enable the vmware customization based on vmware customization files. (default: True)
- `allow_raw_data`: enable or disable the vmware customization based on raw cloud-init data including metadata and userdata. (default: True)
- `vmware_cust_file_max_wait`: the maximum amount of clock time in seconds that should be spent waiting for vmware customization files. (default: 15)

On VMware platforms, VMTools use is required for OVF datasource configuration settings as well as vCloud and vSphere admin configuration. User could change the VMTools configuration options with command:

```
vmware-toolbox-cmd config set <section> <key> <value>
```

The following VMTools configuration options affect cloud-init's behavior on a booted VM:

- **a: [deploypkg] enable-custom-scripts** If this option is absent in VMTools configuration, the custom script is disabled by default for security reasons. Some VMware products could change this default behavior (for example: enabled by default) via customization specification settings.

VMWare admin can refer to (<https://github.com/canonical/cloud-init/blob/main/cloudinit/sources/helpers/vmware/imc/config.py>) and set the customization specification settings.

For more information, see [VMware vSphere Product Documentation](#) and specific VMTools parameters consumed.

Rbx Cloud

The Rbx datasource consumes the metadata drive available on platform [HyperOne](#) and [Rootbox](#) platform.

Datasource supports, in particular, network configurations, hostname, user accounts and user metadata.

Metadata drive

Drive metadata is a [FAT](#)-formatted partition with the ``CLOUDMD`` label on the system disk. Its contents are refreshed each time the virtual machine is restarted, if the partition exists. For more information see [HyperOne Virtual Machine docs](#).

SmartOS Datasource

This datasource finds metadata and user-data from the SmartOS virtualization platform (i.e. Joyent).

Please see <http://smartos.org/> for information about SmartOS.

SmartOS Platform

The SmartOS virtualization platform uses meta-data to the instance via the second serial console. On Linux, this is `/dev/ttyS1`. The data is provided via a simple protocol: something queries for the data, the console responds with the status and if "SUCCESS" returns until a single ".n".

New versions of the SmartOS tooling will include support for base64 encoded data.

Meta-data channels

Cloud-init supports three modes of delivering user/meta-data via the flexible channels of SmartOS.

- user-data is written to `/var/db/user-data`
 - per the spec, user-data is for consumption by the end-user, not provisioning tools
 - cloud-init entirely ignores this channel other than writing it to disk
 - removal of the meta-data key means that `/var/db/user-data` gets removed
 - a backup of previous meta-data is maintained as `/var/db/user-data.<timestamp>`. `<timestamp>` is the epoch time when cloud-init ran
- user-script is written to `/var/lib/cloud/scripts/per-boot/99_user_data`
 - this is executed each boot
 - a link is created to `/var/db/user-script`
 - previous versions of the user-script is written to `/var/lib/cloud/scripts/per-boot.backup/99_user_script.<timestamp>`. - `<timestamp>` is the epoch time when cloud-init ran.
 - when the ‘user-script’ meta-data key goes missing, the user-script is removed from the file system, although a backup is maintained.
 - if the script does not start with a shebang (i.e. starts with `#!<executable>`), then or is not an executable, cloud-init will add a shebang of `“#!/bin/bash”`
- cloud-init:user-data is treated like on other Clouds.
 - this channel is used for delivering `_all_` cloud-init instructions
 - scripts delivered over this channel must be well formed (i.e. must have a shebang)

Cloud-init supports reading the traditional meta-data fields supported by the SmartOS tools. These are:

- `root_authorized_keys`
- `hostname`
- `enable_motd_sys_info`
- `iptables_disable`

Note: At this time `iptables_disable` and `enable_motd_sys_info` are read but are not actioned.

Disabling user-script

Cloud-init uses the per-boot script functionality to handle the execution of the user-script. If you want to prevent this use a cloud-config of:

```
#cloud-config
cloud_final_modules:
- scripts-per-once
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
```

(continues on next page)

(continued from previous page)

- phone-home
- final-message
- power-state-change

Alternatively you can use the json patch method

```
#cloud-config-jsonp
[
  {
    "op": "replace",
    "path": "/cloud_final_modules",
    "value": [
      "scripts-per-once",
      "scripts-per-instance",
      "scripts-user",
      "ssh-authkey-fingerprints",
      "keys-to-console",
      "phone-home",
      "final-message",
      "power-state-change"
    ]
  }
]
```

The default cloud-config includes “script-per-boot”. Cloud-init will still ingest and write the user-data but will not execute it, when you disable the per-boot script handling.

Note: Unless you have an explicit use-case, it is recommended that you not disable the per-boot script execution, especially if you are using any of the life-cycle management features of SmartOS.

The cloud-config needs to be delivered over the cloud-init:user-data channel in order for cloud-init to ingest it.

base64

The following are exempt from base64 encoding, owing to the fact that they are provided by SmartOS:

- root_authorized_keys
- enable_motd_sys_info
- iptables_disable
- user-data
- user-script

This list can be changed through system config of variable ‘no_base64_decode’.

This means that user-script and user-data as well as other values can be base64 encoded. Since Cloud-init can only guess as to whether or not something is truly base64 encoded, the following meta-data keys are hints as to whether or not to base64 decode something:

- base64_all: Except for excluded keys, attempt to base64 decode the values. If the value fails to decode properly, it will be returned in its text
- base64_keys: A comma delimited list of which keys are base64 encoded.
- b64-<key>: for any key, if there exists an entry in the metadata for ‘b64-<key>’ Then ‘b64-<key>’ is expected to be a plaintext boolean indicating whether or not its value is encoded.
- no_base64_decode: This is a configuration setting (i.e. /etc/cloud/cloud.cfg.d) that sets which values should not be base64 decoded.

disk_aliases and ephemeral disk

By default, SmartOS only supports a single ephemeral disk. That disk is completely empty (un-partitioned with no filesystem).

The SmartOS datasource has built-in cloud-config which instructs the ‘disk_setup’ module to partition and format the ephemeral disk.

You can control the disk_setup then in 2 ways:

1. through the datasource config, you can change the ‘alias’ of ephemeral0 to reference another device. The default is:

```
‘disk_aliases’: { ‘ephemeral0’: ‘/dev/vdb’ },
```

Which means anywhere disk_setup sees a device named ‘ephemeral0’ then /dev/vdb will be substituted.

2. you can provide disk_setup or fs_setup data in user-data to overwrite the datasource’s built-in values.

See doc/examples/cloud-config-disk-setup.txt for information on disk_setup.

UpCloud

The UpCloud datasource consumes information from UpCloud’s [metadata service](https://developers.upcloud.com/1.3/8-servers/#metadata-service). This metadata service serves information about the running server via HTTP over the address 169.254.169.254 available in every DHCP-configured interface. The metadata API endpoints are fully described in UpCloud API documentation at <https://developers.upcloud.com/1.3/8-servers/#metadata-service>.

Providing user-data

When creating a server, user-data is provided by specifying it as *user_data* in the API or via the server creation tool in the control panel. User-data is immutable during server’s lifetime and can be removed by deleting the server.

VMware

This datasource is for use with systems running on a VMware platform such as vSphere and currently supports the following data transports:

- [GuestInfo](#) keys

Configuration

The configuration method is dependent upon the transport:

GuestInfo Keys

One method of providing meta, user, and vendor data is by setting the following key/value pairs on a VM’s `extraConfig` property:

Property	Description
guestinfo.metadata	A YAML or JSON document containing the cloud-init metadata.
guestinfo.metadata.encoding	The encoding type for guestinfo.metadata.
guestinfo.userdata	A YAML document containing the cloud-init user data.
guestinfo.userdata.encoding	The encoding type for guestinfo.userdata.
guestinfo.vendordata	A YAML document containing the cloud-init vendor data.
guestinfo.vendordata.encoding	The encoding type for guestinfo.vendordata.

All `guestinfo.*.encoding` values may be set to `base64` or `gzip+base64`.

Features

This section reviews several features available in this datasource, regardless of how the meta, user, and vendor data was discovered.

Instance data and lazy networks

One of the hallmarks of cloud-init is [its use of instance-data and JINJA queries](#) – the ability to write queries in user and vendor data that reference runtime information present in `/run/cloud-init/instance-data.json`. This works well when the metadata provides all of the information up front, such as the network configuration. For systems that rely on DHCP, however, this information may not be available when the metadata is persisted to disk.

This datasource ensures that even if the instance is using DHCP to configure networking, the same details about the configured network are available in `/run/cloud-init/instance-data.json` as if static networking was used. This information collected at runtime is easy to demonstrate by executing the datasource on the command line. From the root of this repository, run the following command:

```
PYTHONPATH="$(pwd)" python3 cloudinit/sources/DataSourceVMware.py
```

The above command will result in output similar to the below JSON:

```
{
  "hostname": "akutz.localhost",
  "local-hostname": "akutz.localhost",
  "local-ipv4": "192.168.0.188",
  "local_hostname": "akutz.localhost",
  "network": {
    "config": {
      "dhcp": true
    },
    "interfaces": {
      "by-ipv4": {
        "172.0.0.2": {
          "netmask": "255.255.255.255",
          "peer": "172.0.0.2"
        },
        "192.168.0.188": {
          "broadcast": "192.168.0.255",
          "mac": "64:4b:f0:18:9a:21",
          "netmask": "255.255.255.0"
        }
      },
      "by-ipv6": {
```

(continues on next page)

(continued from previous page)

```

        "fd8e:d25e:c5b6:1:1f5:b2fd:8973:22f2": {
            "flags": 208,
            "mac": "64:4b:f0:18:9a:21",
            "netmask": "ffff:ffff:ffff:ffff::/64"
        },
        "by-mac": {
            "64:4b:f0:18:9a:21": {
                "ipv4": [
                    {
                        "addr": "192.168.0.188",
                        "broadcast": "192.168.0.255",
                        "netmask": "255.255.255.0"
                    }
                ],
                "ipv6": [
                    {
                        "addr": "fd8e:d25e:c5b6:1:1f5:b2fd:8973:22f2",
                        "flags": 208,
                        "netmask": "ffff:ffff:ffff:ffff::/64"
                    }
                ]
            },
            "ac:de:48:00:11:22": {
                "ipv6": []
            }
        }
    },
    "wait-on-network": {
        "ipv4": true,
        "ipv6": "false"
    }
}

```

Redacting sensitive information

Sometimes the cloud-init userdata might contain sensitive information, and it may be desirable to have the `guestinfo.userdata` key (or other `guestinfo` keys) redacted as soon as its data is read by the datasource. This is possible by adding the following to the metadata:

```

redact: # formerly named cleanup-guestinfo, which will also work
- userdata
- vendordata

```

When the above snippet is added to the metadata, the datasource will iterate over the elements in the `redact` array and clear each of the keys. For example, when the `guestinfo` transport is used, the above snippet will cause the following commands to be executed:

```

vmware-rpctool "info-set guestinfo.userdata ---"
vmware-rpctool "info-set guestinfo.userdata.encoding "
vmware-rpctool "info-set guestinfo.vendordata ---"
vmware-rpctool "info-set guestinfo.vendordata.encoding "

```

Please note that keys are set to the valid YAML string `---` as it is not possible to remove an existing key from the

guestinfo key-space. A key's analogous encoding property will be set to a single white-space character, causing the datasource to treat the actual key value as plain-text, thereby loading it as an empty YAML doc (hence the aforementioned ---).

Reading the local IP addresses

This datasource automatically discovers the local IPv4 and IPv6 addresses for a guest operating system based on the default routes. However, when inspecting a VM externally, it's not possible to know what the *default* IP address is for the guest OS. That's why this datasource sets the discovered, local IPv4 and IPv6 addresses back in the guestinfo namespace as the following keys:

- `guestinfo.local-ipv4`
- `guestinfo.local-ipv6`

It is possible that a host may not have any default, local IP addresses. It's also possible the reported, local addresses are link-local addresses. But these two keys may be used to discover what this datasource determined were the local IPv4 and IPv6 addresses for a host.

Waiting on the network

Sometimes cloud-init may bring up the network, but it will not finish coming online before the datasource's `setup` function is called, resulting in an `/var/run/cloud-init/instance-data.json` file that does not have the correct network information. It is possible to instruct the datasource to wait until an IPv4 or IPv6 address is available before writing the instance data with the following metadata properties:

```
wait-on-network:  
  ipv4: true  
  ipv6: true
```

If either of the above values are true, then the datasource will sleep for a second, check the network status, and repeat until one or both addresses from the specified families are available.

Walkthrough

The following series of steps is a demonstration on how to configure a VM with this datasource:

1. Create the metadata file for the VM. Save the following YAML to a file named `metadata.yaml`:

```
instance-id: cloud-vm  
local-hostname: cloud-vm  
network:  
  version: 2  
  ethernets:  
    nics:  
      match:  
        name: ens*  
      dhcp4: yes
```

2. Create the userdata file `userdata.yaml`:

```
#cloud-config  
  
users:
```

(continues on next page)

(continued from previous page)

```

- default
- name: akutz
  primary_group: akutz
  sudo: ALL=(ALL) NOPASSWD:ALL
  groups: sudo, wheel
  lock_passwd: true
  ssh_authorized_keys:
    - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDE0c5FcZvcGSh/tG4iw+Fhfi/O5/EvUM/
    ↪ 96js65tly4++YTXK1d9jcznPS5ruD1bIZ30oveCBd3kT8LLVFwzh6hepYTf0YmCTpF4eDunyqmpCXDvVscQYRXyasEm5o1
    ↪ mt2PcPjooeX00vAj81jjU2f3XKrjjz2u2+Kit9eba+vOQ6HiC8c2IzRkUAJ5i1atLy8RIbejo23+0P4N2jjk17QySFOVHw
    ↪ 0M/4ideeU74EN/
    ↪ CgVsvO6JrLsPBR4dojkV5qNbMNxIVv5cUwIy2Th1LgqpNcEFIDLCWNZEfKlEuNeSQ2mPtIO7ETxEL2Cz5y/
    ↪ 7AIuILDzYMc6wi2bofRC8HmQ7rMXRWdWlKWsR0L7SKjHblIwarxOGqLnUI+k2E71YoP7SZSlxaKi17pgkr0OMCF+kKqvcv
    ↪ tZKSpjYtjMb5+RonfhaFRNzv7cCE1f3Kp8UVqAdcGBtReoE8eRUT63qIxjw03a7VwAyB2w+9culR9/
    ↪ vAo8SBerQw== sakutz@gmail.com

```

3. Please note this step requires that the VM be powered off. All of the commands below use the VMware CLI tool, `govc`.

Go ahead and assign the path to the VM to the environment variable `VM`:

```
export VM="/inventory/path/to/the/vm"
```

4. Power off the VM:

To ensure the next power-on operation results in a first-boot scenario for cloud-init, it may be necessary to run the following command just before powering off the VM:

```
cloud-init clean
```

Otherwise cloud-init may not run in first-boot mode. For more information on how the boot mode is determined, please see the [First Boot Documentation](#).

```
govc vm.power -off "${VM}"
```

5. Export the environment variables that contain the cloud-init metadata and userdata:

```

export METADATA=$(gzip -c9 <metadata.yaml | { base64 -w0 2>/dev/null || base64; }
↪ ) \
  USERDATA=$(gzip -c9 <userdata.yaml | { base64 -w0 2>/dev/null || base64; })

```

6. Assign the metadata and userdata to the VM:

```

govc vm.change -vm "${VM}" \
-e guestinfo.metadata="${METADATA}" \
-e guestinfo.metadata.encoding="gzip+base64" \
-e guestinfo.userdata="${USERDATA}" \
-e guestinfo.userdata.encoding="gzip+base64"

```

Please note the above commands include specifying the encoding for the properties. This is important as it informs the datasource how to decode the data for cloud-init. Valid values for `metadata.encoding` and `userdata.encoding` include:

- base64
- gzip+base64

7. Power on the VM:

```
govc vm.power -vm "${VM}" -on
```

If all went according to plan, the CentOS box is:

- Locked down, allowing SSH access only for the user in the userdata
- Configured for a dynamic IP address via DHCP
- Has a hostname of `cloud-vm`

Examples

This section reviews common configurations:

Setting the hostname

The hostname is set by way of the metadata key `local-hostname`.

Setting the instance ID

The instance ID may be set by way of the metadata key `instance-id`. However, if this value is absent then the instance ID is read from the file `/sys/class/dmi/id/product_uuid`.

Providing public SSH keys

The public SSH keys may be set by way of the metadata key `public-keys-data`. Each newline-terminated string will be interpreted as a separate SSH public key, which will be placed in distro's default user's `~/.ssh/authorized_keys`. If the value is empty or absent, then nothing will be written to `~/.ssh/authorized_keys`.

Configuring the network

The network is configured by setting the metadata key `network` with a value consistent with Network Config Versions 1 or 2, depending on the Linux distro's version of cloud-init.

The metadata key `network.encoding` may be used to indicate the format of the metadata key “network”. Valid encodings are `base64` and `gzip+base64`.

Vultr

The **Vultr** datasource retrieves basic configuration values from the locally accessible **metadata service**. All data is served over HTTP from the address 169.254.169.254. The endpoints are documented in <https://www.vultr.com/metadata/>

Configuration

Vultr's datasource can be configured as follows:

datasource:

Vultr: url: 'http://169.254.169.254' retries: 3 timeout: 2 wait: 2

- *url*: The URL used to acquire the metadata configuration from
- *retries*: Determines the number of times to attempt to connect to the metadata service
- *timeout*: Determines the timeout in seconds to wait for a response from the metadata service
- *wait*: Determines the timeout in seconds to wait before retrying after accessible failure

ZStack

ZStack platform provides a AWS Ec2 metadata service, but with different datasource identity. More information about ZStack can be found at [ZStack](#).

Discovery

To determine whether a vm running on ZStack platform, cloud-init checks DMI information by 'dmidecode -s chassis-asset-tag', if the output ends with '.zstack.io', it's running on ZStack platform:

Metadata

Same as EC2, instance metadata can be queried at

```
GET http://169.254.169.254/2009-04-04/meta-data/
instance-id
local-hostname
```

Userdata

Same as EC2, instance userdata can be queried at

```
GET http://169.254.169.254/2009-04-04/user-data/
meta_data.json
user_data
password
```

1.13.2 Creation

The datasource objects have a few touch points with cloud-init. If you are interested in adding a new datasource for your cloud platform you will need to take care of the following items:

- **Identify a mechanism for positive identification of the platform:** It is good practice for a cloud platform to positively identify itself to the guest. This allows the guest to make educated decisions based on the platform on which it is running. On the x86 and arm64 architectures, many clouds identify themselves through DMI data. For example, Oracle's public cloud provides the string 'OracleCloud.com' in the DMI chassis-asset field.

cloud-init enabled images produce a log file with details about the platform. Reading through this log in /run/cloud-init/ds-identify.log may provide the information needed to uniquely identify the platform. If the log is not present, you can generate it by running from source ./tools/ds-identify or the installed location /usr/lib/cloud-init/ds-identify.

The mechanism used to identify the platform will be required for the ds-identify and datasource module sections below.

- **Add datasource module “cloudinit/sources/DataSource<CloudPlatform>.py”:** It is suggested that you start by copying one of the simpler datasources such as DataSourceHetzner.
- **Add tests for datasource module:** Add a new file with some tests for the module to cloudinit/sources/test_<yourplatform>.py. For example see cloudinit/sources/tests/test_oracle.py
- **Update ds-identify:** In systemd systems, ds-identify is used to detect which datasource should be enabled or if cloud-init should run at all. You’ll need to make changes to tools/ds-identify.
- **Add tests for ds-identify:** Add relevant tests in a new class to tests/unittests/test_ds_identify.py. You can use TestOracle as an example.
- **Add your datasource name to the builtin list of datasources:** Add your datasource module name to the end of the datasource_list entry in cloudinit/settings.py.
- **Add your cloud platform to apport collection prompts:** Update the list of cloud platforms in cloudinit/apport.py. This list will be provided to the user who invokes ubuntu-bug cloud-init.
- **Enable datasource by default in ubuntu packaging branches:** Ubuntu packaging branches contain a template file debian/cloud-init.templates that ultimately sets the default datasource_list when installed via package. This file needs updating when the commit gets into a package.
- **Add documentation for your datasource:** You should add a new file in doc/datasources/<cloudplatform>.rst

1.13.3 API

The current interface that a datasource object must provide is the following:

```
# returns a mime multipart message that contains
# all the various fully-expanded components that
# were found from processing the raw user data string
# - when filtering only the mime messages targeting
#   this instance id will be returned (or messages with
#   no instance id)
def get_userdata(self, apply_filter=False)

# returns the raw userdata string (or none)
def get_userdata_raw(self)

# returns a integer (or none) which can be used to identify
# this instance in a group of instances which are typically
# created from a single command, thus allowing programmatic
# filtering on this launch index (or other selective actions)
@property
def launch_index(self)

# the data sources' config_obj is a cloud-config formatted
# object that came to it from ways other than cloud-config
# because cloud-config content would be handled elsewhere
def get_config_obj(self)

# returns a list of public SSH keys
def get_public_ssh_keys(self)

# translates a device 'short' name into the actual physical device
```

(continues on next page)

(continued from previous page)

```

# fully qualified name (or none if said physical device is not attached
# or does not exist)
def device_name_to_device(self, name)

# gets the locale string this instance should be applying
# which typically used to adjust the instances locale settings files
def get_locale(self)

@property
def availability_zone(self)

# gets the instance id that was assigned to this instance by the
# cloud provider or when said instance id does not exist in the backing
# metadata this will return 'iid-datasource'
def get_instance_id(self)

# gets the fully qualified domain name that this host should be using
# when configuring network or hostname related settings, typically
# assigned either by the cloud provider or the user creating the vm
def get_hostname(self, fqdn=False)

def get_package_mirror_info(self)

```

1.14 Vendor Data

1.14.1 Overview

Vendordata is data provided by the entity that launches an instance (for example, the cloud provider). This data can be used to customize the image to fit into the particular environment it is being run in.

Vendordata follows the same rules as user-data, with the following caveats:

1. Users have ultimate control over vendordata. They can disable its execution or disable handling of specific parts of multipart input.
2. By default it only runs on first boot
3. Vendordata can be disabled by the user. If the use of vendordata is required for the instance to run, then vendordata should not be used.
4. user supplied cloud-config is merged over cloud-config from vendordata.

Users providing cloud-config data can use the '#cloud-config-jsonp' method to more finely control their modifications to the vendor supplied cloud-config. For example, if both vendor and user have provided 'runcmd' then the default merge handler will cause the user's runcmd to override the one provided by the vendor. To append to 'runcmd', the user could better provide multipart input with a cloud-config-jsonp part like:

```

#cloud-config-jsonp
[{"op": "add", "path": "/runcmd", "value": ["my", "command", "here"]}

```

Further, we strongly advise vendors to not 'be evil'. By evil, we mean any action that could compromise a system. Since users trust you, please take care to make sure that any vendordata is safe, atomic, idempotent and does not put your users at risk.

1.14.2 Input Formats

cloud-init will download and cache to filesystem any vendor-data that it finds. Vendordata is handled exactly like user-data. That means that the vendor can supply multipart input and have those parts acted on in the same way as user-data.

The only differences are:

- vendor-data-defined scripts are stored in a different location than user-data-defined scripts (to avoid namespace collision)
- user can disable part handlers by cloud-config settings. For example, to disable handling of ‘part-handlers’ in vendor-data, the user could provide user-data like this:

```
#cloud-config
vendordata: {excluded: 'text/part-handler'}
```

1.14.3 Examples

There are examples in the examples subdirectory.

Additionally, the ‘tools’ directory contains ‘write-mime-multipart’, which can be used to easily generate mime-multi-part files from a list of input files. That data can then be given to an instance.

See ‘write-mime-multipart –help’ for usage.

1.15 Network Configuration

- Default Behavior
- Disabling Network Configuration
- Fallback Networking
- Network Configuration Sources
- Network Configuration Outputs
- Network Output Policy
- Network Configuration Tools
- Examples

1.15.1 Default Behavior

Cloud-init ‘s searches for network configuration in order of increasing precedence; each item overriding the previous.

Datasource

For example, OpenStack may provide network config in the MetaData Service.

System Config

A `network:` entry in `/etc/cloud/cloud.cfg.d/*` configuration files.

Kernel Command Line

`ip=` or `network-config=<Base64 encoded YAML config string>`

User-data cannot change an instance’s network configuration. In the absence of network configuration in any of the above sources, Cloud-init will write out a network configuration that will issue a DHCP request on a “first” network interface.

Note: The network-config value is expected to be a Base64 encoded YAML string in *Networking Config Version 1* or *Networking Config Version 2* format. Optionally it can be compressed with `gzip` prior to Base64 encoding.

1.15.2 Disabling Network Configuration

Users may disable Cloud-init’s network configuration capability and rely on other methods, such as embedded configuration or other customizations.

Cloud-init supports the following methods for disabling cloud-init.

Kernel Command Line

Cloud-init will additionally check for the parameter `network-config=disabled` which will automatically disable any network configuration.

Example disabling kernel command line entry:

```
network-config=disabled
```

cloud config

In the combined cloud-init configuration dictionary, merged from `/etc/cloud/cloud.cfg` and `/etc/cloud/cloud.cfg.d/*`:

```
network:
  config: disabled
```

If Cloud-init’s networking config has not been disabled, and no other network information is found, then it will proceed to generate a fallback networking configuration.

Disabling Network Activation

Some datasources may not be initialized until after network has been brought up. In this case, cloud-init will attempt to bring up the interfaces specified by the datasource metadata.

This behavior can be disabled in the cloud-init configuration dictionary, merged from `/etc/cloud/cloud.cfg` and `/etc/cloud/cloud.cfg.d/*`:

```
disable_network_activation: true
```

1.15.3 Fallback Network Configuration

Cloud-init will attempt to determine which of any attached network devices is most likely to have a connection and then generate a network configuration to issue a DHCP request on that interface.

Cloud-init runs during early boot and does not expect composed network devices (such as Bridges) to be available. Cloud-init does not consider the following interface devices as likely ‘first’ network interfaces for fallback configuration; they are filtered out from being selected.

- **loopback:** `name=lo`

- **Virtual Ethernet:** `name=veth*`
- **Software Bridges:** `type=bridge`
- **Software VLANs:** `type=vlan`

Cloud-init will prefer network interfaces that indicate they are connected via the Linux `carrier` flag being set. If no interfaces are marked connected, then all unfiltered interfaces are potential connections.

Of the potential interfaces, Cloud-init will attempt to pick the “right” interface given the information it has available.

Finally after selecting the “right” interface, a configuration is generated and applied to the system.

Note: PhotonOS disables fallback networking configuration by default leaving network unrendered when no other network config is provided. If fallback config is still desired on PhotonOS, it can be enabled by providing `disable_fallback_netcfg: false` in `/etc/cloud/cloud.cfg:sys_config` settings.

1.15.4 Network Configuration Sources

Cloud-init accepts a number of different network configuration formats in support of different cloud substrates. The Datasource for these clouds in Cloud-init will detect and consume Datasource-specific network configuration formats for use when writing an instance’s network configuration.

The following Datasources optionally provide network configuration:

- *Config Drive*
 - OpenStack Metadata Service Network
 - *Network Configuration ENI (Legacy)*
- *DigitalOcean*
 - DigitalOcean JSON metadata
- *NoCloud*
 - *Networking Config Version 1*
 - *Networking Config Version 2*
 - *Network Configuration ENI (Legacy)*
- *OpenNebula*
 - *Network Configuration ENI (Legacy)*
- *OpenStack*
 - *Network Configuration ENI (Legacy)*
 - OpenStack Metadata Service Network
- *SmartOS Datasource*
 - SmartOS JSON Metadata
- *UpCloud*
 - UpCloud JSON metadata
- *Vultr*
 - Vultr JSON metadata

For more information on network configuration formats

Network Configuration ENI (Legacy)

Cloud-init supports reading and writing network config in the ENI format which is consumed by the `ifupdown` tool to parse and apply network configuration.

As an input format this is **legacy**. In cases where ENI format is available and another format is also available, it will prefer to use the other format. This can happen in either *NoCloud* or *OpenStack* datasources.

Please reference existing [documentation](#) for the `/etc/network/interfaces (5)` format.

Networking Config Version 1

This network configuration format lets users customize their instance's networking interfaces by assigning subnet configuration, virtual device creation (bonds, bridges, vlans) routes and DNS configuration.

Required elements of a Network Config Version 1 are `config` and `version`.

Cloud-init will read this format from system config. For example the following could be present in `/etc/cloud/cloud.cfg.d/custom-networking.cfg`:

```
network:
  version: 1
  config:
    - type: physical
      name: eth0
      subnets:
        - type: dhcp
```

The *NoCloud* datasource can also provide cloud-init networking configuration in this Format.

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`)

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. Letters must be lowercase.

Note: MAC addresses must be strings. As MAC addresses which consist of only the digits 0-9 (i.e. no hex a-f) can be interpreted as a base 60 integer per the [YAML 1.1 spec](#) it is best practice to quote all MAC addresses to ensure they are parsed as strings regardless of value.

Note: Cloud-init will handle the 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
```


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 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.

Note: MAC addresses must be strings. As MAC addresses which consist of only the digits 0-9 (i.e. no hex a-f) can be interpreted as a base 60 integer per the [YAML 1.1 spec](#) it is best practice to quote all MAC addresses to ensure they are parsed as strings regardless of value.

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.

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.

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:
```

(continues on next page)

(continued from previous page)

```

- gbe0
- gbe1
params:
  bond-mode: active-backup

```

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

```

(continues on next page)

(continued from previous page)

```
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
```

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.

The following optional keys are supported:

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.

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
```

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.
- **interface:** Optional. Ties the nameserver definition to the specified interface. The value specified here must match the *name* of an interface defined in this config. If unspecified, this nameserver will be considered a global nameserver.

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
      interface: interface0 # Ties nameserver to interface0 only
      address:
        - 192.168.23.2
        - 8.8.8.8
      search:
        - exemplary
```

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
```

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_nameservers`: 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`.
- `routes`: Specify a list of routes for a given interface

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 .
- `ipv6_dhcpv6-stateful`: Configure this interface with `dhcp6`
- `ipv6_dhcpv6-stateless`: Configure this interface with SLAAC and DHCP
- `ipv6_slaac`: Configure address with SLAAC

When making use of `dhcp` or either of the `ipv6_dhcpv6` types, no additional configuration is needed in the subnet dictionary.

Using `ipv6_dhcpv6-stateless` or `ipv6_slaac` allows the IPv6 address to be automatically configured with StateLess Address AutoConfiguration (SLAAC). SLAAC requires support from the network, so verify that your cloud or network offering has support before trying it out. With `ipv6_dhcpv6-stateless`, DHCPv6 is still used to fetch other subnet details such as gateway or DNS servers. If you only want to discover the address, use `ipv6_slaac`.

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'
```

(continues on next page)

(continued from previous page)

```

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

```

Subnet with routes Example:

```

network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: '00:11:22:33:44:55'
      subnets:
        - type: dhcp
        - type: static
          address: 10.184.225.122
          netmask: 255.255.255.252
          routes:
            - gateway: 10.184.225.121
              netmask: 255.240.0.0
              network: 10.176.0.0
            - gateway: 10.184.225.121
              netmask: 255.240.0.0
              network: 10.208.0.0

```

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
```

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
```

(continues on next page)

(continued from previous page)

```

    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

```

Networking Config Version 2

Cloud-init's support for Version 2 network config is a subset of the version 2 format defined for the [netplan](#) tool. Cloud-init supports both reading and writing of Version 2; the latter support requires a distro with [netplan](#) present.

Netplan Passthrough

On a system with netplan present, cloud-init will pass Version 2 configuration through to netplan without modification. On such systems, you do not need to limit yourself to the below subset of netplan's configuration format.

Warning: If you are writing or generating network configuration that may be used on non-netplan systems, you **must** limit yourself to the subset described in this document, or you will see network configuration failures on non-netplan systems.

Version 2 Configuration Format

The `network` key has at least two required elements. First it must include `version: 2` and one or more of possible device types.

Cloud-init will read this format from system config. For example the following could be present in `/etc/cloud/cloud.cfg.d/custom-networking.cfg`:

```
network:
  version: 2
  ethernets: []
```

It may also be provided in other locations including the *NoCloud*, see *Default Behavior* for other places.

Supported device types values are as follows:

- Ethernets (`ethernets`)
- Bonds (`bonds`)
- Bridges (`bridges`)
- VLANs (`vlan`s)

Each type block contains device definitions as a map where the keys (called “configuration IDs”). Each entry under the types may include IP and/or device configuration.

Device configuration IDs

The key names below the per-device-type definition maps (like `ethernets:`) are called “ID”s. They must be unique throughout the entire set of configuration files. Their primary purpose is to serve as anchor names for composite devices, for example to enumerate the members of a bridge that is currently being defined.

There are two physically/structurally different classes of device definitions, and the ID field has a different interpretation for each:

Physical devices (Examples: ethernet, wifi): These can dynamically come and go between reboots and even during runtime (hotplugging). In the generic case, they can be selected by `match:` rules on desired properties, such as name/name pattern, MAC address, driver, or device paths. In general these will match any number of devices (unless they refer to properties which are unique such as the full path or MAC address), so without further knowledge about the hardware these will always be considered as a group.

It is valid to specify no match rules at all, in which case the ID field is simply the interface name to be matched. This is mostly useful if you want to keep simple cases simple, and it's how network device configuration has been done for a long time.

If there are `match:` rules, then the ID field is a purely opaque name which is only being used for references from definitions of compound devices in the config.

Virtual devices (Examples: veth, bridge, bond): These are fully under the control of the config file(s) and the network stack. I. e. these devices are being created instead of matched. Thus `match:` and `set-name:` are not applicable for these, and the ID field is the name of the created virtual device.

Common properties for physical device types

match: *<(mapping)>*

This selects a subset of available physical devices by various hardware properties. The following configuration will then apply to all matching devices, as soon as they appear. *All* specified properties must match. The following properties for creating matches are supported:

name: *<(scalar)>*

Current interface name. Globs are supported, and the primary use case for matching on names, as selecting one fixed name can be more easily achieved with having no `match:` at all and just using the ID (see above). Note that currently only networkd supports globbing, NetworkManager does not.

macaddress: *<(scalar)>*

Device's MAC address in the form `xx:xx:xx:xx:xx:xx`. Globs are not allowed. Letters must be lowercase.

Note: MAC addresses must be strings. As MAC addresses which consist of only the digits 0-9 (i.e. no hex a-f) can be interpreted as a base 60 integer per the [YAML 1.1 spec](#) it is best practice to quote all MAC addresses to ensure they are parsed as strings regardless of value.

driver: *<(scalar)>*

Kernel driver name, corresponding to the `DRIVER` udev property. Globs are supported. Matching on driver is *only* supported with networkd.

Examples:

```
# all cards on second PCI bus
match:
  name: enp2*

# fixed MAC address
match:
  macaddress: 11:22:33:aa:bb:ff

# first card of driver `ixgbe`
match:
  driver: ixgbe
  name: en*s0
```

set-name: *<(scalar)>*

When matching on unique properties such as path or MAC, or with additional assumptions such as “there will only ever be one wifi device”, match rules can be written so that they only match one device. Then this property can be used to give that device a more specific/desirable/nicer name than the default from udev's ifnames. Any additional device that satisfies the match rules will then fail to get renamed and keep the original kernel name (and dmesg will show an error).

wakeonlan: *<(bool)>*

Enable wake on LAN. Off by default.

Common properties for all device types

renderer: *<(scalar)>*

Use the given networking backend for this definition. Currently supported are `networkd` and `NetworkManager`. This property can be specified globally in `networks:`, for a device type (in e. g. `ethernets:`) or for a particular device definition. Default is `networkd`.

Note: Cloud-init only supports `networkd` backend if rendering version2 config to the instance.

dhcp4: *<(bool)>*

Enable DHCP for IPv4. Off by default.

dhcp6: *<(bool)>*

Enable DHCP for IPv6. Off by default.

addresses: *<(sequence of scalars)>*

Add static addresses to the interface in addition to the ones received through DHCP or RA. Each sequence entry is in CIDR notation, i. e. of the form `addr/prefixlen`. `addr` is an IPv4 or IPv6 address as recognized by `inet_pton` `(3)` and ``prefixlen` the number of bits of the subnet.

Example: `addresses: [192.168.14.2/24, 2001:1::1/64]`

gateway4: or **gateway6:** *<(scalar)>*

Set default gateway for IPv4/6, for manual address configuration. This requires setting `addresses` too. Gateway IPs must be in a form recognized by `inet_pton(3)`

Example for IPv4: `gateway4: 172.16.0.1` Example for IPv6: `gateway6: 2001:4::1`

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.

nameservers: *<(mapping)>*

Set DNS servers and search domains, for manual address configuration. There are two supported fields: `addresses:` is a list of IPv4 or IPv6 addresses similar to `gateway*`, and `search:` is a list of search domains.

Example:

```
nameservers:
  search: [lab, home]
  addresses: [8.8.8.8, FEDC::1]
```

routes: *<(sequence of mapping)>*

Add device specific routes. Each mapping includes a `to`, `via` key with an IPv4 or IPv6 address as value. `metric` is an optional value.

Example:

```

routes:
- to: 0.0.0.0/0
  via: 10.23.2.1
  metric: 3

```

Ethernets

Ethernet device definitions do not support any specific properties beyond the common ones described above.

Bonds

interfaces *<(sequence of scalars)>*

All devices matching this ID list will be added to the bond.

Example:

```

ethernets:
  switchports:
    match: {name: "enp2*"}
  [...]
bonds:
  bond0:
    interfaces: [switchports]

```

parameters: *<(mapping)>*

Customization parameters for special bonding options. Time values are specified in seconds unless otherwise specified.

mode: *<(scalar)>*

Set the bonding mode used for the interfaces. The default is `balance-rr` (round robin). Possible values are `balance-rr`, `active-backup`, `balance-xor`, `broadcast`, `802.3ad`, `balance-tlb`, and `balance-alb`.

lacp-rate: *<(scalar)>*

Set the rate at which LACPDU's are transmitted. This is only useful in `802.3ad` mode. Possible values are `slow` (30 seconds, default), and `fast` (every second).

mii-monitor-interval: *<(scalar)>*

Specifies the interval for MII monitoring (verifying if an interface of the bond has carrier). The default is 0; which disables MII monitoring.

min-links: *<(scalar)>*

The minimum number of links up in a bond to consider the bond interface to be up.

transmit-hash-policy: *<(scalar)>*

Specifies the transmit hash policy for the selection of slaves. This is only useful in `balance-xor`, `802.3ad` and `balance-tlb` modes. Possible values are `layer2`, `layer3+4`, `layer2+3`, `encap2+3`, and `encap3+4`.

ad-select: *<(scalar)>*

Set the aggregation selection mode. Possible values are `stable`, `bandwidth`, and `count`. This option is only used in `802.3ad` mode.

all-slaves-active: *<(bool)>*

If the bond should drop duplicate frames received on inactive ports, set this option to `false`. If they should be delivered, set this option to `true`. The default value is `false`, and is the desirable behavior in most situations.

arp-interval: *<(scalar)>*

Set the interval value for how frequently ARP link monitoring should happen. The default value is 0, which disables ARP monitoring.

arp-ip-targets: *<(sequence of scalars)>*

IPs of other hosts on the link which should be sent ARP requests in order to validate that a slave is up. This option is only used when `arp-interval` is set to a value other than 0. At least one IP address must be given for ARP link monitoring to function. Only IPv4 addresses are supported. You can specify up to 16 IP addresses. The default value is an empty list.

arp-validate: *<(scalar)>*

Configure how ARP replies are to be validated when using ARP link monitoring. Possible values are `none`, `active`, `backup`, and `all`.

arp-all-targets: *<(scalar)>*

Specify whether to use any ARP IP target being up as sufficient for a slave to be considered up; or if all the targets must be up. This is only used for `active-backup` mode when `arp-validate` is enabled. Possible values are `any` and `all`.

up-delay: *<(scalar)>*

Specify the delay before enabling a link once the link is physically up. The default value is 0.

down-delay: *<(scalar)>*

Specify the delay before disabling a link once the link has been lost. The default value is 0.

fail-over-mac-policy: *<(scalar)>*

Set whether to set all slaves to the same MAC address when adding them to the bond, or how else the system should handle MAC addresses. The possible values are `none`, `active`, and `follow`.

gratuitious-arp: *<(scalar)>*

Specify how many ARP packets to send after failover. Once a link is up on a new slave, a notification is sent and possibly repeated if this value is set to a number greater than 1. The default value is 1 and valid values are between 1 and 255. This only affects `active-backup` mode.

packets-per-slave: *<(scalar)>*

In `balance-rr` mode, specifies the number of packets to transmit on a slave before switching to the next. When this value is set to 0, slaves are chosen at random. Allowable values are between 0 and 65535. The default value is 1. This setting is only used in `balance-rr` mode.

primary-reselect-policy: *<(scalar)>*

Set the reselection policy for the primary slave. On failure of the active slave, the system will use this policy to decide how the new active slave will be chosen and how recovery will be handled. The possible values are `always`, `better`, and `failure`.

learn-packet-interval: *<(scalar)>*

Specify the interval between sending learning packets to each slave. The value range is between 1 and `0x7fffffff`. The default value is 1. This option only affects `balance-tlb` and `balance-alb` modes.

Bridges

interfaces: *<(sequence of scalars)>*

All devices matching this ID list will be added to the bridge.

Example:

```
ethernets:
  switchports:
    match: {name: "enp2*"}
  [...]
bridges:
  br0:
    interfaces: [switchports]
```

parameters: *<(mapping)>*

Customization parameters for special bridging options. Time values are specified in seconds unless otherwise specified.

ageing-time: *<(scalar)>*

Set the period of time to keep a MAC address in the forwarding database after a packet is received.

priority: *<(scalar)>*

Set the priority value for the bridge. This value should be an number between 0 and 65535. Lower values mean higher priority. The bridge with the higher priority will be elected as the root bridge.

forward-delay: *<(scalar)>*

Specify the period of time the bridge will remain in Listening and Learning states before getting to the Forwarding state. This value should be set in seconds for the systemd backend, and in milliseconds for the NetworkManager backend.

hello-time: *<(scalar)>*

Specify the interval between two hello packets being sent out from the root and designated bridges. Hello packets communicate information about the network topology.

max-age: *<(scalar)>*

Set the maximum age of a hello packet. If the last hello packet is older than that value, the bridge will attempt to become the root bridge.

path-cost: *<(scalar)>*

Set the cost of a path on the bridge. Faster interfaces should have a lower cost. This allows a finer control on the network topology so that the fastest paths are available whenever possible.

stp: *<(bool)>*

Define whether the bridge should use Spanning Tree Protocol. The default value is “true”, which means that Spanning Tree should be used.

VLANs

id: *<(scalar)>*

VLAN ID, a number between 0 and 4094.

link: *<(scalar)>*

ID of the underlying device definition on which this VLAN gets created.

Example:

```
ethernets:
  eno1: {...}
vlangs:
  en-intra:
    id: 1
    link: eno1
    dhcp4: yes
  en-vpn:
    id: 2
    link: eno1
    address: ...
```

Examples

Configure an ethernet device with networkd, identified by its name, and enable DHCP:

```
network:
  version: 2
  ethernets:
    eno1:
      dhcp4: true
```

This is a complex example which shows most available features:

```
network:
  version: 2
  ethernets:
    # opaque ID for physical interfaces, only referred to by other stanzas
    id0:
      match:
        macaddress: '00:11:22:33:44:55'
      wakeonlan: true
      dhcp4: true
      addresses:
        - 192.168.14.2/24
        - 2001:1::1/64
      gateway4: 192.168.14.1
      gateway6: 2001:1::2
      nameservers:
        search: [foo.local, bar.local]
        addresses: [8.8.8.8]
    # static routes
    routes:
      - to: 192.0.2.0/24
        via: 11.0.0.1
        metric: 3
  lom:
    match:
      driver: ixgbe
    # you are responsible for setting tight enough match rules
    # that only match one device if you use set-name
    set-name: lom1
    dhcp6: true
```

(continues on next page)

(continued from previous page)

```

switchports:
    # all cards on second PCI bus; unconfigured by themselves, will be added
    # to br0 below
    match:
        name: enp2*
        mtu: 1280
bonds:
    bond0:
        interfaces: [id0, lom]
bridges:
    # the key name is the name for virtual (created) interfaces; no match: and
    # set-name: allowed
    br0:
        # IDs of the components; switchports expands into multiple interfaces
        interfaces: [wlpls0, switchports]
        dhcp4: true
vlans:
    en-intra:
        id: 1
        link: id0
        dhcp4: yes

```

1.15.5 Network Configuration Outputs

Cloud-init converts various forms of user supplied or automatically generated configuration into an internal network configuration state. From this state Cloud-init delegates rendering of the configuration to Distro supported formats. The following renderers are supported in cloud-init:

- **ENI**

/etc/network/interfaces or ENI is supported by the `ifupdown` package found in Alpine Linux, Debian and Ubuntu.

- **Netplan**

Introduced in Ubuntu 16.10 (Yakkety Yak), `netplan` has been the default network configuration tool in Ubuntu since 17.10 (Artful Aardvark). `netplan` consumes *Networking Config Version 2* input and renders network configuration for supported backends such as `systemd-networkd` and `NetworkManager`.

- **Sysconfig**

Sysconfig format is used by RHEL, CentOS, Fedora and other derivatives.

1.15.6 Network Output Policy

The default policy for selecting a network `renderer` in order of preference is as follows:

- ENI
- Sysconfig
- Netplan

When applying the policy, Cloud-init checks if the current instance has the correct binaries and paths to support the renderer. The first renderer that can be used is selected. Users may override the network renderer policy by supplying an updated configuration in `cloud-config`.

```
system_info:
network:
  renderers: ['netplan', 'eni', 'sysconfig', 'freebsd', 'netbsd', 'openbsd']
```

1.15.7 Network Configuration Tools

Cloud-init contains one tool used to test input/output conversion between formats. The `tools/net-convert.py` in the Cloud-init source repository is helpful for examining expected output for a given input format.

CLI Interface :

```
% tools/net-convert.py --help
usage: net-convert.py [-h] --network-data PATH --kind
                    {eni,network_data.json,yaml} -d PATH [-m name,mac]
                    --output-kind {eni,netplan,sysconfig}

optional arguments:
  -h, --help            show this help message and exit
  --network-data PATH, -p PATH
                        {eni,network_data.json,yaml}, -k {eni,network_data.json,yaml}
  -d PATH, --directory PATH
                        directory to place output in
  -m name,mac, --mac name,mac
                        interface name to mac mapping
  --output-kind {eni,netplan,sysconfig}, -ok {eni,netplan,sysconfig}
```

Example output converting V2 to sysconfig:

```
% tools/net-convert.py --network-data v2.yaml --kind yaml \
  --output-kind sysconfig -d target
% cat target/etc/sysconfig/network-scripts/ifcfg-eth*
# Created by cloud-init on instance boot automatically, do not edit.
#
BOOTPROTO=static
DEVICE=eth7
IPADDR=192.168.1.5/255.255.255.0
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
USERCTL=no
# Created by cloud-init on instance boot automatically, do not edit.
#
BOOTPROTO=dhcp
DEVICE=eth9
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
USERCTL=no
```

1.16 Contributing to cloud-init

This document describes how to contribute changes to cloud-init. It assumes you have a [GitHub](#) account, and refers to your GitHub user as `GH_USER` throughout.

1.16.1 Submitting your first pull request

Summary

Before any pull request can be accepted, you must do the following:

- Sign the Canonical [contributor license agreement](#)
- Add yourself (alphabetically) to the in-repository list that we use to track CLA signatures: [tools/.github-cla-signers](#)
- Add or update any [unit tests](#) accordingly
- Add or update any [integration tests](#) (if applicable)
- Format code (using black and isort) with `tox -e do_format`
- Ensure unit tests and linting pass using `tox`
- Submit a PR against the *main* branch of the *cloud-init* repository

The detailed instructions

Follow these steps to submit your first pull request to cloud-init:

- To contribute to cloud-init, you must sign the Canonical [contributor license agreement](#)
 - If you have already signed it as an individual, your Launchpad user will be listed in the [contributor-agreement-canonical](#) group. (Unfortunately there is no easy way to check if an organization or company you are doing work for has signed.)
 - When signing it:
 - * ensure that you fill in the GitHub username field.
 - * when prompted for ‘Project contact’ or ‘Canonical Project Manager’, enter ‘James Falcon’.
 - If your company has signed the CLA for you, please contact us to help in verifying which Launchpad/GitHub accounts are associated with the company.
 - For any questions or help with the process, please email [James Falcon](#) with the subject, “Cloud-Init CLA”
 - You also may contact user `falcojr` in the `#cloud-init` channel on the Libera IRC network.
- Configure git with your email and name for commit messages.

Your name will appear in commit messages and will also be used in changelogs or release notes. Give yourself credit!:

```
git config user.name "Your Name"
git config user.email "Your Email"
```

- Sign into your [GitHub](#) account
- Fork the upstream [repository](#) on Github and clicking on the `Fork` button
- Create a new remote pointing to your personal GitHub repository.

```
git clone git://github.com/canonical/cloud-init
cd cloud-init
git remote add GH_USER git@github.com:GH_USER/cloud-init.git
git push GH_USER main
```

- Read through the cloud-init [Code Review Process](#), so you understand how your changes will end up in cloud-init's codebase.
- Submit your first cloud-init pull request, adding yourself to the in-repository list that we use to track CLA signatures: `tools/.github-cla-signers`
 - See [PR #344](#) and [PR #345](#) for examples of what this pull request should look like.
 - Note that `.github-cla-signers` is sorted alphabetically.
 - (If you already have a change that you want to submit, you can also include the change to `tools/.github-cla-signers` in that pull request, there is no need for two separate PRs.)

Transferring CLA Signatures from Launchpad to Github

For existing contributors who have signed the agreement in Launchpad before the Github username field was included, we need to verify the link between your [Launchpad](#) account and your [GitHub](#) account. To enable us to do this, we ask that you create a branch with both your Launchpad and GitHub usernames against both the Launchpad and GitHub cloud-init repositories. We've added a tool (`tools/migrate-lp-user-to-github`) to the cloud-init repository to handle this migration as automatically as possible.

The cloud-init team will review the two merge proposals and verify that the CLA has been signed for the Launchpad user and record the associated GitHub account.

Note: If you are a first time contributor, you will not need to touch Launchpad to contribute to cloud-init: all new CLA signatures are handled as part of the GitHub pull request process described above.

1.16.2 Do these things for each feature or bug

- Create a new topic branch for your work:

```
git checkout -b my-topic-branch
```

- Make and commit your changes (note, you can make multiple commits, fixes, more commits.):

```
git commit
```

- Apply black and isort formatting rules with `tox`:

```
tox -e format
```

- Run unit tests and lint/formatting checks with `tox`:

```
tox
```

- Push your changes to your personal GitHub repository:

```
git push -u GH_USER my-topic-branch
```

- Use your browser to create a pull request:

- Open the branch on GitHub

- * You can see a web view of your repository and navigate to the branch at:

```
https://github.com/GH_USER/cloud-init/tree/my-topic-branch
```

- Click ‘Pull Request’
- Fill out the pull request title, summarizing the change and a longer message indicating important details about the changes included, like

```
Activate the frobnicator.
```

```
The frobnicator was previously inactive and now runs by default.
This may save the world some day. Then, list the bugs you fixed
as footers with syntax as shown here.
```

```
The commit message should be one summary line of less than
70 characters followed by a blank line, and then one or more
paragraphs wrapped at 72 characters describing the change and why
it was needed.
```

```
This is the message that will be used on the commit when it
is squashed and merged into main. If there is a related launchpad
bug, specify it at the bottom of the commit message.
```

```
LP: #NNNNNNN (replace with the appropriate bug reference or remove
this line entirely if there is no associated bug)
```

Note that the project continues to use LP: #NNNNN format for closing launchpad bugs rather than GitHub Issues.

- Click ‘Create Pull Request’

Then, a cloud-init committer will review your changes and follow up in the pull request. Look at the [Code Review Process](#) doc to understand the following steps.

Feel free to ping and/or join #cloud-init on Libera irc if you have any questions.

1.16.3 Design

This section captures design decisions that are helpful to know when hacking on cloud-init.

Python Support

Cloud-init upstream currently supports Python 3.6 and above.

Cloud-init upstream will stay compatible with a particular python version for 6 years after release. After 6 years, we will stop testing upstream changes against the unsupported version of python and may introduce breaking changes. This policy may change as needed.

The following table lists the cloud-init versions in which the minimum python version changed:

Cloud-init version	Python version
22.1	3.6+
20.3	3.5+
19.4	2.7+

Cloud Config Modules

- Any new modules should use underscores in any new config options and not hyphens (e.g. *new_option* and *not new-option*).

Tests

Submissions to cloud-init must include testing. See [Testing](#) for details on these requirements.

Type Annotations

The cloud-init codebase uses Python’s annotation support for storing type annotations in the style specified by [PEP-484](#) and [PEP-526](#). Their use in the codebase is encouraged.

Feature Flags

Feature flags are used as a way to easily toggle configuration **at build time**. They are provided to accommodate feature deprecation and downstream configuration changes.

Currently used upstream values for feature flags are set in `cloudinit/features.py`. Overrides to these values (typically via quilt patch) can be placed in a file called `feature_overrides.py` in the same directory. Any value set in `feature_overrides.py` will override the original value set in `features.py`.

Each flag should include a short comment regarding the reason for the flag and intended lifetime.

Tests are required for new feature flags, and tests must verify all valid states of a flag, not just the default state.

`cloudinit.features.ALLOW_EC2_MIRRORS_ON_NON_AWS_INSTANCE_TYPES = False`

When configuring apt mirrors, if `ALLOW_EC2_MIRRORS_ON_NON_AWS_INSTANCE_TYPES` is `True` cloud-init will detect that a datasource’s `availability_zone` property looks like an EC2 availability zone and set the `ec2_region` variable when generating mirror URLs; this can lead to incorrect mirrors being configured in clouds whose AZs follow EC2’s naming pattern.

As of 20.3, `ALLOW_EC2_MIRRORS_ON_NON_AWS_INSTANCE_TYPES` is `False` so we no longer include `ec2_region` in mirror determination on non-AWS cloud platforms.

If the old behavior is desired, users can provide the appropriate mirrors via `apt:` directives in cloud-config.

`cloudinit.features.ERROR_ON_USER_DATA_FAILURE = True`

If there is a failure in obtaining user data (i.e., `#include` or `decompress` fails) and `ERROR_ON_USER_DATA_FAILURE` is `False`, cloud-init will log a warning and proceed. If it is `True`, cloud-init will instead raise an exception.

As of 20.3, `ERROR_ON_USER_DATA_FAILURE` is `True`.

(This flag can be removed after Focal is no longer supported.)

1.16.4 Ongoing Refactors

This captures ongoing refactoring projects in the codebase. This is intended as documentation for developers involved in the refactoring, but also for other developers who may interact with the code being refactored in the meantime.

`cloudinit.net` -> `cloudinit.distros.networking` Hierarchy

`cloudinit.net` was imported from the curtin codebase as a chunk, and then modified enough that it integrated with the rest of the cloud-init codebase. Over the ~4 years since, the fact that it is not fully integrated into the `Distro` hierarchy has caused several issues.

The common pattern of these problems is that the commands used for networking are different across distributions and operating systems. This has lead to `cloudinit.net` developing its own “distro determination” logic: `get_interfaces_by_mac` is probably the clearest example of this. Currently, these differences are primarily split along

Linux/BSD lines. However, it would be short-sighted to only refactor in a way that captures this difference: we can anticipate that differences will develop between Linux-based distros in future, or there may already be differences in tooling that we currently work around in less obvious ways.

The high-level plan is to introduce a hierarchy of networking classes in `cloudinit.distros.networking`, which each `Distro` subclass will reference. These will capture the differences between networking on our various distros, while still allowing easy reuse of code between distros that share functionality (e.g. most of the Linux networking behaviour). `Distro` objects will instantiate the networking classes at `self.networking`, so callers will call `distro.networking.<func>` instead of `cloudinit.net.<func>`; this will necessitate access to an instantiated `Distro` object.

An implementation note: there may be external consumers of the `cloudinit.net` module. We don't consider this a public API, so we will be removing it as part of this refactor. However, we will ensure that the new API is complete from its introduction, so that any such consumers can move over to it wholesale. (Note, however, that this new API is still not considered public or stable, and may not replicate the existing API exactly.)

In more detail:

- The root of this hierarchy will be the `cloudinit.distros.networking.Networking` class. This class will have a corresponding method for every `cloudinit.net` function that we identify to be involved in refactoring. Initially, these methods' implementations will simply call the corresponding `cloudinit.net` function. (This gives us the complete API from day one, for existing consumers.)
- As the biggest differentiator in behaviour, the next layer of the hierarchy will be two subclasses: `LinuxNetworking` and `BSDNetworking`. These will be introduced in the initial PR.
- When a difference in behaviour for a particular distro is identified, a new `Networking` subclass will be created. This new class should generally subclass either `LinuxNetworking` or `BSDNetworking`.
- To be clear: `Networking` subclasses will only be created when needed, we will not create a full hierarchy of per-`Distro` subclasses up-front.
- Each `Distro` class will have a class variable (`cls.networking_cls`) which points at the appropriate networking class (initially this will be either `LinuxNetworking` or `BSDNetworking`).
- When `Distro` classes are instantiated, they will instantiate `cls.networking_cls` and store the instance at `self.networking`. (This will be implemented in `cloudinit.distros.Distro.__init__`.)
- A helper function will be added which will determine the appropriate `Distro` subclass for the current system, instantiate it and return its `networking` attribute. (This is the entry point for existing consumers to migrate to.)
- Callers of refactored functions will change from calling `cloudinit.net.<func>` to `distro.networking.<func>`, where `distro` is an instance of the appropriate `Distro` class for this system. (This will require making such an instance available to callers, which will constitute a large part of the work in this project.)

After the initial structure is in place, the work in this refactor will consist of replacing the `cloudinit.net.some_func` call in each `cloudinit.distros.networking.Networking` method with the actual implementation. This can be done incrementally, one function at a time:

- pick an unmigrated `cloudinit.distros.networking.Networking` method
- find it in the [the list of bugs tagged net-refactor](#) and assign yourself to it (see *Managing Work/Tracking Progress* below for more details)
- refactor all of its callers to call the `distro.networking.<func>` method on `Distro` instead of the `cloudinit.net.<func>` function. (This is likely to be the most time-consuming step, as it may require plumbing `Distro` objects through to places that previously have not consumed them.)
- refactor its implementation from `cloudinit.net` into the `Networking` hierarchy (e.g. if it has an if/else on BSD, this is the time to put the implementations in their respective subclasses)

- if part of the method contains distro-independent logic, then you may need to create new methods to capture this distro-specific logic; we don't want to replicate common logic in different `Networking` subclasses
- if after the refactor, the method on the root `Networking` class no longer has any implementation, it should be converted to an `abstractmethod`
- ensure that the new implementation has unit tests (either by moving existing tests, or by writing new ones)
- ensure that the new implementation has a docstring
- add any appropriate type annotations
 - note that we must follow the constraints described in the “Type Annotations” section above, so you may not be able to write complete annotations
 - we have `type aliases` defined in `cloudinit.distros.networking` which should be used when applicable
- finally, remove it (and any other now-unused functions) from `cloudinit.net` (to avoid having two parallel implementations)

cloudinit.net Functions/Classes

The functions/classes that need refactoring break down into some broad categories:

- helpers for accessing `/sys` (that should not be on the top-level `Networking` class as they are Linux-specific):
 - `get_sys_class_path`
 - `sys_dev_path`
 - `read_sys_net`
 - `read_sys_net_safe`
 - `read_sys_net_int`
- those that directly access `/sys` (via helpers) and should (IMO) be included in the API of the `Networking` class:
 - `generate_fallback_config`
 - * the `config_driver` parameter is used and passed as a boolean, so we can change the default value to `False` (instead of `None`)
 - `get_ib_interface_hwaddr`
 - `get_interface_mac`
 - `interface_has_own_mac`
 - `is_bond`
 - `is_bridge`
 - `is_physical`
 - `is_renamed`
 - `is_up`
 - `is_vlan`
 - `wait_for_physdevs`

- those that directly access `/sys` (via helpers) but may be Linux-specific concepts or names:
 - `get_master`
 - `device_devid`
 - `device_driver`
- those that directly use `ip`:
 - `_get_current_rename_info`
 - * this has non-distro-specific logic so should potentially be refactored to use helpers on `self` instead of `ip` directly (rather than being wholesale reimplemented in each of `BSDNetworking` or `LinuxNetworking`)
 - * we can also remove the `check_downable` argument, it's never specified so is always `True`
 - `_rename_interfaces`
 - * this has several internal helper functions which use `ip` directly, and it calls `_get_current_rename_info`. That said, there appears to be a lot of non-distro-specific logic that could live in a function on `Networking`, so this will require some careful refactoring to avoid duplicating that logic in each of `BSDNetworking` and `LinuxNetworking`.
 - * only the `renames` and `current_info` parameters are ever passed in (and `current_info` only by tests), so we can remove the others from the definition
 - `EphemeralIPv4Network`
 - * this is another case where it mixes distro-specific and non-specific functionality. Specifically, `__init__`, `__enter__` and `__exit__` are non-specific, and the remaining methods are distro-specific.
 - * when refactoring this, the need to track `cleanup_cmds` likely means that the distro-specific behaviour cannot be captured only in the `Networking` class. See [this comment in PR #363](#) for more thoughts.
- those that implicitly use `/sys` via their call dependencies:
 - `master_is_bridge_or_bond`
 - * appends to `get_master` return value, which is a `/sys` path
 - `extract_physdevs`
 - * calls `device_driver` and `device_devid` in both `_version_*` impls
 - `apply_network_config_names`
 - * calls `extract_physdevs`
 - * there is already a `Distro.apply_network_config_names` which in the default implementation calls this function; this and its `BSD` subclass implementations should be refactored at the same time
 - * the `strict_present` and `strict_busy` parameters are never passed, nor are they used in the function definition, so they can be removed
 - `get_interfaces`
 - * calls `device_driver`, `device_devid` amongst others
 - `get_ib_hwaddrs_by_interface`
 - * calls `get_interfaces`

- those that may fall into the above categories, but whose use is only related to netfailover (which relies on a Linux-specific network driver, so is unlikely to be relevant elsewhere without a substantial refactor; these probably only need implementing in `LinuxNetworking`):
 - `get_dev_features`
 - `has_netfail_standby_feature`
 - * calls `get_dev_features`
 - `is_netfailover`
 - `is_netfail_master`
 - * this is called from `generate_fallback_config`
 - `is_netfail_primary`
 - `is_netfail_standby`
 - N.B. all of these take an optional `driver` argument which is used to pass around a value to avoid having to look it up by calling `device_driver` every time. This is something of a leaky abstraction, and is better served by caching on `device_driver` or storing the cached value on `self`, so we can drop the parameter from the new API.
- those that use `/sys` (via helpers) and have non-exhaustive BSD logic:
 - `get_devicelist`
- those that already have separate Linux/BSD implementations:
 - `find_fallback_nic`
 - `get_interfaces_by_mac`
- those that have no OS-specific functionality (so do not need to be refactored):
 - `ParserError`
 - `RendererNotFoundError`
 - `has_url_connectivity`
 - `is_ip_address`
 - `is_ipv4_address`
 - `natural_sort_key`

Note that the functions in `cloudinit.net` use inconsistent parameter names for “string that contains a device name”; we can standardise on `devname` (the most common one) in the refactor.

Managing Work/Tracking Progress

To ensure that we won’t have multiple people working on the same part of the refactor at the same time, there is a bug for each function. You can see the current status by looking at [the list of bugs tagged net-refactor](#).

When you’re working on refactoring a particular method, ensure that you have assigned yourself to the corresponding bug, to avoid duplicate work.

Generally, when considering what to pick up to refactor, it is best to start with functions in `cloudinit.net` which are not called by anything else in `cloudinit.net`. This allows you to focus only on refactoring that function and its callsites, rather than having to update the other `cloudinit.net` function also.

References

- [Mina Galić's email to the cloud-init ML in 2018](#) (plus its thread)
- [Mina Galić's email to the cloud-init ML in 2019](#) (plus its thread)
- [PR #363](#), the discussion which prompted finally starting this refactor (and where a lot of the above details were hashed out)

1.17 Code Review Process

In order to manage incoming pull requests effectively, and provide timely feedback and/or acceptance this document serves as a guideline for the review process and outlines the expectations for those submitting code to the project as well as those reviewing the code. Code is reviewed for acceptance by at least one core team member (later referred to as committers), but comments and suggestions from others are encouraged and welcome.

The process is intended to provide timely and actionable feedback for any submission.

1.17.1 Asking For Help

cloud-init contributors, potential contributors, community members and users are encouraged to ask for any help that they need. If you have questions about the code review process, or at any point during the code review process, these are the available avenues:

- if you have an open Pull Request, comment on that pull request
- join the `#cloud-init` channel on the Libera IRC network and ask away
- send an email to the cloud-init mailing list, cloud-init@lists.launchpad.net

These are listed in rough order of preference, but use whichever of them you are most comfortable with.

1.17.2 Goals

This process has the following goals:

- Ensure code reviews occur in a timely fashion and provide actionable feedback if changes are desired.
- Ensure the minimization of ancillary problems to increase the efficiency for those reviewing the submitted code

1.17.3 Role Definitions

Any code review process will have (at least) two involved parties. For our purposes, these parties are referred to as **Proposer** and **Reviewer**. (We also have the **Committer** role which is a special case of the **Reviewer** role.) The terms are defined here (and the use of the singular form is not meant to imply that they refer to a single person):

Proposer The person proposing a pull request (hereafter known as a PR).

Reviewer A person who is reviewing a PR.

Committer A cloud-init core developer (i.e. a person who has permission to merge PRs into **main**).

1.17.4 Prerequisites For Landing Pull Requests

Before a PR can be landed into **main**, the following conditions *must* be met:

- the CLA has been signed by the **Proposer** (or is covered by an entity-level CLA signature)
- all required status checks are passing
- at least one “Approve” review from a **Committer**
- no “Request changes” reviews from any **Committer**

The following conditions *should* be met:

- any Python functions/methods/classes have docstrings added/updated
- any changes to config module behaviour are captured in the documentation of the config module
- any Python code added has corresponding unit tests
- no “Request changes” reviews from any **Reviewer**

These conditions can be relaxed at the discretion of the **Committers** on a case-by-case basis. Generally, for accountability, this should not be the decision of a single **Committer**, and the decision should be documented in comments on the PR.

(To take a specific example, the `cc_phone_home` module had no tests at the time [PR #237](#) was submitted, so the **Proposer** was not expected to write a full set of tests for their minor modification, but they were expected to update the config module docs.)

1.17.5 Non-Committer Reviews

Reviews from non-**Committers** are *always* welcome. Please feel empowered to review PRs and leave your thoughts and comments on any submitted PRs, regardless of the **Proposer**.

Much of the below process is written in terms of the **Committers**. This is not intended to reflect that reviews should only come from that group, but acknowledges that we are ultimately responsible for maintaining the standards of the codebase. It would be entirely reasonable (and very welcome) for a **Reviewer** to only examine part of a PR, but it would not be appropriate for a **Committer** to merge a PR without full scrutiny.

1.17.6 Opening Phase

In this phase, the **Proposer** is responsible for opening a pull request and meeting the prerequisites laid out above.

If they need help understanding the prerequisites, or help meeting the prerequisites, then they can (and should!) ask for help. See the [Asking For Help](#) section above for the ways to do that.

These are the steps that comprise the opening phase:

1. The **Proposer** opens PR
2. CI runs automatically, and if

CI fails The **Proposer** is expected to fix CI failures. If the **Proposer** doesn’t understand the nature of the failures they are seeing, they should comment in the PR to request assistance, or use another way of [Asking For Help](#).

(Note that if assistance is not requested, the **Committers** will assume that the **Proposer** is working on addressing the failures themselves. If you require assistance, please do ask for help!)

CI passes Move on to the [Review Phase](#).

1.17.7 Review Phase

In this phase, the **Proposer** and the **Reviewers** will iterate together to, hopefully, get the PR merged into the cloud-init codebase. There are three potential outcomes: merged, rejected permanently, and temporarily closed. (The first two are covered in this section; see *Inactive Pull Requests* for details about temporary closure.)

(In the below, when the verbs “merge” or “squash merge” are used, they should be understood to mean “squash merged using the GitHub UI”, which is the only way that changes can land in cloud-init’s **main** branch.)

These are the steps that comprise the review phase:

1. **The Committers** assign a **Committer** to the PR

This **Committer** is expected to shepherd the PR to completion (and merge it, if that is the outcome reached). This means that they will perform an initial review, and monitor the PR to ensure that the **Proposer** is receiving any assistance that they require. The **Committers** will perform this assignment on a daily basis.

This assignment is intended to ensure that the **Proposer** has a clear point of contact with a cloud-init core developer, and that they get timely feedback after submitting a PR. It *is not* intended to preclude reviews from any other **Reviewers**, nor to imply that the **Committer** has ownership over the review process.

The assigned **Committer** may choose to delegate the code review of a PR to another **Reviewer** if they think that they would be better suited.

(Note that, in GitHub terms, this is setting an Assignee, not requesting a review.)

2. That **Committer** performs an initial review of the PR, resulting in one of the following:

Approve If the submitted PR meets all of the *Prerequisites For Landing Pull Requests* and passes code review, then the **Committer** will squash merge immediately.

There may be circumstances where a PR should not be merged immediately. The `wip` label will be applied to PRs for which this is true. Only **Committers** are able to apply labels to PRs, so anyone who believes that this label should be applied to a PR should request its application in a comment on the PR.

The review process is **DONE**.

Approve (with nits) If the **Proposer** submits their PR with “Allow edits from maintainer” enabled, and the only changes the **Committer** requests are minor “nits”, the **Committer** can push fixes for those nits and *immediately* squash merge. If the **Committer** does not wish to fix these nits but believes they should block a straight-up Approve, then their review should be “Needs Changes” instead.

A nit is understood to be something like a minor style issue or a spelling error, generally confined to a single line of code.

If a **Committer** is unsure as to whether their requested change is a nit, they should not treat it as a nit.

(If a **Proposer** wants to opt-out of this, then they should uncheck “Allow edits from maintainer” when submitting their PR.)

The review process is **DONE**.

Outright rejection The **Committer** will close the PR, with useful messaging for the **Proposer** as to why this has happened.

This is reserved for cases where the proposed change is completely unfit for landing, and there is no reasonable path forward. This should only be used sparingly, as there are very few cases where proposals are completely unfit.

If a different approach to the same problem is planned, it should be submitted as a separate PR. The **Committer** should include this information in their message when the PR is closed.

The review process is **DONE**.

Needs Changes The **Committer** will give the **Proposer** a clear idea of what is required for an Approve vote or, for more complex PRs, what the next steps towards an Approve vote are.

The **Proposer** will ask questions if they don't understand, or disagree with, the **Committer's** review comments.

Once consensus has been reached, the **Proposer** will address the review comments.

Once the review comments are addressed (as well as, potentially, in the interim), CI will run. If CI fails, the **Proposer** is expected to fix CI failures. If CI passes, the **Proposer** should indicate that the PR is ready for re-review (by @ing the assigned reviewer), effectively moving back to the start of this section.

1.17.8 Inactive Pull Requests

PRs will be temporarily closed if they have been waiting on **Proposer** action for a certain amount of time without activity. A PR will be marked as stale (with an explanatory comment) after 14 days of inactivity. It will be closed after a further 7 days of inactivity.

These closes are not considered permanent, and the closing message should reflect this for the **Proposer**. However, if a PR is reopened, it should effectively enter the *Opening Phase* again, as it may need some work done to get CI passing again.

1.18 Security Policy

The following documents the upstream cloud-init security policy.

1.18.1 Reporting

If a user finds a security issue, they are requested to file a [private security bug on Launchpad](#). To ensure the information stays private, change the "This bug contains information that is:" from "Public" to "Private Security" when filing.

After the bug is received, the issue is triaged within 2 working days of being reported and a response is sent to the reporter.

1.18.2 cloud-init-security

The cloud-init-security Launchpad team is a private, invite-only team used to discuss and coordinate security issues with the project.

Any issues disclosed to the cloud-init-security mailing list are considered embargoed and should only be discussed with other members of the cloud-init-security mailing list before the coordinated release date, unless specific exception is granted by the administrators of the mailing list. This includes disclosure of any details related to the vulnerability or the presence of a vulnerability itself. Violation of this policy may result in removal from the list for the company or individual involved.

1.18.3 Evaluation

If the reported bug is deemed a real security issue a CVE is assigned by the Canonical Security Team as CVE Numbering Authority (CNA).

If it is deemed a regular, non-security, issue, the reporter will be asked to follow typical bug reporting procedures.

In addition to the disclosure timeline, the core Canonical cloud-init team will enlist the expertise of the Ubuntu Security team for guidance on industry-standard disclosure practices as necessary.

If an issue specifically involves another distro or cloud vendor, additional individuals will be informed of the issue to help in evaluation.

1.18.4 Disclosure

Disclosure of security issues will be made with a public statement. Once the determined time for disclosure has arrived the following will occur:

- A public bug is filed/made public with vulnerability details, CVE, mitigations and where to obtain the fix
- An email is sent to the [public cloud-init mailing list](#)

The disclosure timeframe is coordinated with the reporter and members of the cloud-init-security list. This depends on a number of factors:

- The reporter might have their own disclosure timeline (e.g. Google Project Zero and many others use a 90-days after initial report OR when a fix becomes public)
- It might take time to decide upon and develop an appropriate fix
- A distros might want extra time to backport any possible fixes before the fix becomes public
- A cloud may need additional time to prepare to help customers or impliment a fix
- The issue might be deemed low priority
- May wish to align with an upcoming planned release

1.19 Debugging cloud-init

1.19.1 Overview

This topic will discuss general approaches for test and debug of cloud-init on deployed instances.

1.19.2 Boot Time Analysis - cloud-init analyze

Occasionally instances don't appear as performant as we would like and cloud-init packages a simple facility to inspect what operations took cloud-init the longest during boot and setup.

The script `/usr/bin/cloud-init` has an `analyze` sub-command **analyze** which parses any `cloud-init.log` file into formatted and sorted events. It allows for detailed analysis of the most costly cloud-init operations are to determine the long-pole in cloud-init configuration and setup. These subcommands default to reading `/var/log/cloud-init.log`.

- `analyze show` Parse and organize cloud-init.log events by stage and include each sub-stage granularity with time delta reports.

```
$ cloud-init analyze show -i my-cloud-init.log
-- Boot Record 01 --
The total time elapsed since completing an event is printed after the "@"
character.
The time the event takes is printed after the "+" character.

Starting stage: modules-config
```

(continues on next page)

(continued from previous page)

```
|`->config-emit_upstart ran successfully @05.47600s +00.00100s
|`->config-snap_config ran successfully @05.47700s +00.00100s
|`->config-ssh-import-id ran successfully @05.47800s +00.00200s
|`->config-locale ran successfully @05.48000s +00.00100s
...
```

- **analyze dump** Parse cloud-init.log into event records and return a list of dictionaries that can be consumed for other reporting needs.

```
$ cloud-init analyze dump -i my-cloud-init.log
[
  {
    "description": "running config modules",
    "event_type": "start",
    "name": "modules-config",
    "origin": "cloudinit",
    "timestamp": 1510807493.0
  }, ...
```

- **analyze blame** Parse cloud-init.log into event records and sort them based on highest time cost for quick assessment of areas of cloud-init that may need improvement.

```
$ cloud-init analyze blame -i my-cloud-init.log
-- Boot Record 11 --
    00.01300s (modules-final/config-scripts-per-boot)
    00.00400s (modules-final/config-final-message)
    00.00100s (modules-final/config-rightscales_userdata)
    ...
```

- **analyze boot** Make subprocess calls to the kernel in order to get relevant pre-cloud-init timestamps, such as the kernel start, kernel finish boot, and cloud-init start.

```
$ cloud-init analyze boot
-- Most Recent Boot Record --
    Kernel Started at: 2019-06-13 15:59:55.809385
    Kernel ended boot at: 2019-06-13 16:00:00.944740
    Kernel time to boot (seconds): 5.135355
    Cloud-init start: 2019-06-13 16:00:05.738396
    Time between Kernel boot and Cloud-init start (seconds): 4.793656
```

Analyze quickstart - LXC

To quickly obtain a cloud-init log try using lxc on any ubuntu system:

```
$ lxc init ubuntu-daily:focal x1
$ lxc start x1
$ # Take lxc's cloud-init.log and pipe it to the analyzer
$ lxc file pull x1/var/log/cloud-init.log - | cloud-init analyze dump -i -
$ lxc file pull x1/var/log/cloud-init.log - | \
  python3 -m cloudinit.analyze dump -i -
```

Analyze quickstart - KVM

To quickly analyze a KVM a cloud-init log:

1. Download the current cloud image

```
$ wget https://cloud-images.ubuntu.com/daily/server/focal/current/focal-server-
↳ cloudimg-amd64.img
```

2. Create a snapshot image to preserve the original cloud-image

```
$ qemu-img create -b focal-server-cloudimg-amd64.img -f qcow2 \
test-cloudinit.qcow2
```

3. Create a seed image with metadata using *cloud-localds*

```
$ cat > user-data <<EOF
#cloud-config
password: passw0rd
chpasswd: { expire: False }
EOF
$ cloud-localds my-seed.img user-data
```

4. Launch your modified VM

```
$ kvm -m 512 -net nic -net user -redir tcp:2222::22 \
-drive file=test-cloudinit.qcow2,if=virtio,format=qcow2 \
-drive file=my-seed.img,if=virtio,format=raw
```

5. Analyze the boot (blame, dump, show)

```
$ ssh -p 2222 ubuntu@localhost 'cat /var/log/cloud-init.log' | \
cloud-init analyze blame -i -
```

1.19.3 Running single cloud config modules

This subcommand is not called by the init system. It can be called manually to load the configured datasource and run a single cloud-config module once using the cached userdata and metadata after the instance has booted. Each cloud-config module has a module FREQUENCY configured: PER_INSTANCE, PER_BOOT, PER_ONCE or PER_ALWAYS. When a module is run by cloud-init, it stores a semaphore file in `/var/lib/cloud/instance/sem/config_<module_name>.<frequency>` which marks when the module last successfully ran. Presence of this semaphore file prevents a module from running again if it has already been run. To ensure that a module is run again, the desired frequency can be overridden on the commandline:

```
$ sudo cloud-init single --name cc_ssh --frequency always
...
Generating public/private ed25519 key pair
...
```

Inspect `cloud-init.log` for output of what operations were performed as a result.

1.19.4 Stable Release Updates (SRU) testing for cloud-init

Once an Ubuntu release is stable (i.e. after it is released), updates for it must follow a special procedure called a “stable release update” (or [SRU](#)).

The cloud-init project has a specific process it follows when validating a cloud-init SRU, documented in the [CloudinitUpdates](#) wiki page.

Generally an SRU test of cloud-init performs the following:

- Install a pre-release version of cloud-init from the **-proposed** APT pocket (e.g. **bionic-proposed**)
- Upgrade cloud-init and attempt a clean run of cloud-init to assert the new version of cloud-init works properly the specific platform and Ubuntu series
- Check for tracebacks or errors in behavior

Manual SRU verification procedure

Below are steps to manually test a pre-release version of cloud-init from **-proposed**

Note: For each Ubuntu SRU, the Ubuntu Server team manually validates the new version of cloud-init on these platforms: **Amazon EC2, Azure, GCE, OpenStack, Oracle, Softlayer (IBM), LXD, KVM**

1. Launch a VM on your favorite platform, providing this cloud-config user-data and replacing `<YOUR_LAUNCHPAD_USERNAME>` with your username:

```
## template: jinja
#cloud-config
ssh_import_id: [<YOUR_LAUNCHPAD_USERNAME>]
hostname: SRU-worked-{{v1.cloud_name}}
```

2. Wait for current cloud-init to complete, replace `<YOUR_VM_IP>` with the IP address of the VM that you launched in step 1:

```
CI_VM_IP=<YOUR_VM_IP>
# Make note of the datasource cloud-init detected in --long output.
# In step 5, you will use this to confirm the same datasource is detected after
↪ upgrade.
ssh ubuntu@$CI_VM_IP -- cloud-init status --wait --long
```

3. Set up the **-proposed** pocket on your VM and upgrade to the **-proposed** cloud-init:

```
# Create a script that will add the -proposed pocket to APT's sources
# and install cloud-init from that pocket
cat > setup_proposed.sh <<EOF
#!/bin/bash
mirror=http://archive.ubuntu.com/ubuntu
echo deb $mirror $(lsb_release -sc)-proposed main | tee \
    /etc/apt/sources.list.d/proposed.list
apt-get update -q
apt-get install -qy cloud-init
EOF

scp setup_proposed.sh ubuntu@$CI_VM_IP:.
ssh ubuntu@$CI_VM_IP -- sudo bash setup_proposed.sh
```

4. Change hostname, clean cloud-init's state, and reboot to run cloud-init from scratch:

```
ssh ubuntu@$CI_VM_IP -- sudo hostname something-else
ssh ubuntu@$CI_VM_IP -- sudo cloud-init clean --logs --reboot
```

5. Validate **-proposed** cloud-init came up without error

```
# Block until cloud-init completes and verify from --long the datasource
# from step 1. Errors would show up in --long

ssh ubuntu@$CI_VM_IP -- cloud-init status --wait --long
# Make sure hostname was set properly to SRU-worked-<cloud name>
ssh ubuntu@$CI_VM_IP -- hostname
# Check for any errors or warnings in cloud-init logs.
# (This should produce no output if successful.)
ssh ubuntu@$CI_VM_IP -- grep Trace "/var/log/cloud-init*"
```

6. If you encounter an error during SRU testing:

- Create a new [cloud-init bug](#) reporting the version of cloud-init affected
- Ping upstream cloud-init on Libera's [#cloud-init IRC channel](#)

1.20 Logging

Cloud-init supports both local and remote logging configurable through python's built-in logging configuration and through the cloud-init rsyslog module.

1.20.1 Command Output

Cloud-init can redirect its stdout and stderr based on config given under the `output` config key. The output of any commands run by cloud-init and any user or vendor scripts provided will also be included here. The `output` key accepts a dictionary for configuration. Output files may be specified individually for each stage (`init`, `config`, and `final`), or a single key `all` may be used to specify output for all stages.

The output for each stage may be specified as a dictionary of `output` and `error` keys, for stdout and stderr respectively, as a tuple with stdout first and stderr second, or as a single string to use for both. The strings passed to all of these keys are handled by the system shell, so any form of redirection that can be used in bash is valid, including piping cloud-init's output to `tee`, or `logger`. If only a filename is provided, cloud-init will append its output to the file as though `>>` was specified.

By default, cloud-init loads its output configuration from `/etc/cloud/cloud.cfg.d/05_logging.cfg`. The default config directs both stdout and stderr from all cloud-init stages to `/var/log/cloud-init-output.log`. The default config is given as

```
output: { all: "| tee -a /var/log/cloud-init-output.log" }
```

For a more complex example, the following configuration would output the `init` stage to `/var/log/cloud-init.out` and `/var/log/cloud-init.err`, for stdout and stderr respectively, replacing anything that was previously there. For the `config` stage, it would pipe both stdout and stderr through `tee -a /var/log/cloud-config.log`. For the `final` stage it would append the output of stdout and stderr to `/var/log/cloud-final.out` and `/var/log/cloud-final.err` respectively.

```
output:
  init:
    output: "> /var/log/cloud-init.out"
    error: "> /var/log/cloud-init.err"
  config: "tee -a /var/log/cloud-config.log"
  final:
    - ">> /var/log/cloud-final.out"
    - "/var/log/cloud-final.err"
```

Python Logging

Cloud-init uses the python logging module, and can accept config for this module using the standard python fileConfig format. Cloud-init looks for config for the logging module under the `logcfg` key.

Note: the logging configuration is not YAML, it is python fileConfig format, and is passed through directly to the python logging module. please use the correct syntax for a multi-line string in YAML.

By default, cloud-init uses the logging configuration provided in `/etc/cloud/cloud.cfg.d/05_logging.cfg`. The default python logging configuration writes all cloud-init events with a priority of `WARNING` or higher to console, and writes all events with a level of `DEBUG` or higher to `/var/log/cloud-init.log` and via `syslog`.

Python's fileConfig format consists of sections with headings in the format `[title]` and key value pairs in each section. Configuration for python logging must contain the sections `[loggers]`, `[handlers]`, and `[formatters]`, which name the entities of their respective types that will be defined. The section name for each defined logger, handler and formatter will start with its type, followed by an underscore (`_`) and the name of the entity. For example, if a logger was specified with the name `log01`, config for the logger would be in the section `[logger_log01]`.

Logger config entries contain basic logging set up. They may specify a list of handlers to send logging events to as well as the lowest priority level of events to handle. A logger named `root` must be specified and its configuration (under `[logger_root]`) must contain a level and a list of handlers. A level entry can be any of the following: `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`, or `NOTSET`. For the `root` logger the `NOTSET` option will allow all logging events to be recorded.

Each configured handler must specify a class under the python's logging package namespace. A handler may specify a message formatter to use, a priority level, and arguments for the handler class. Common handlers are `StreamHandler`, which handles stream redirects (i.e. logging to `stderr`), and `FileHandler` which outputs to a log file. The logging module also supports logging over net sockets, over http, via smtp, and additional complex configurations. For full details about the handlers available for python logging, please see the documentation for [python logging handlers](#).

Log messages are formatted using the `logging.Formatter` class, which is configured using `formatter` config entities. A default format of `%(message)s` is given if no formatter configs are specified. Formatter config entities accept a format string which supports variable replacements. These may also accept a `datefmt` string which may be used to configure the timestamp used in the log messages. The format variables `%(asctime)s`, `%(levelname)s` and `%(message)s` are commonly used and represent the timestamp, the priority level of the event and the event message. For additional information on logging formatters see [python logging formatters](#).

Note: by default the format string used in the logging formatter are in python's old style `%s` form. the `str.format()` and `string.Template` styles can also be used by using `{` or `$` in place of `%` by setting the `style` parameter in formatter config.

A simple, but functional python logging configuration for cloud-init is below. It will log all messages of priority `DEBUG` or higher both `stderr` and `/tmp/my.log` using a `StreamHandler` and a `FileHandler`, using the default format string `%(message)s`:

```
logcfg: |
  [loggers]
  keys=root,cloudinit
  [handlers]
  keys=ch,cf
  [formatters]
  keys=
  [logger_root]
```

(continues on next page)

(continued from previous page)

```

level=DEBUG
handlers=
[logger_cloudinit]
level=DEBUG
qualname=cloudinit
handlers=ch,cf
[handler_ch]
class=StreamHandler
level=DEBUG
args=(sys.stderr,)
[handler_cf]
class=FileHandler
level=DEBUG
args=('/tmp/my.log',)

```

For additional information about configuring python's logging module, please see the documentation for [python logging config](#).

Rsyslog Module

Cloud-init's `cc_rsyslog` module allows for fully customizable rsyslog configuration under the `rsyslog` config key. The simplest way to use the rsyslog module is by specifying remote servers under the `remotes` key in `rsyslog` config. The `remotes` key takes a dictionary where each key represents the name of an rsyslog server and each value is the configuration for that server. The format for server config is:

- optional filter for log messages (defaults to `*.*`)
- optional leading `@` or `@@`, indicating udp and tcp respectively (defaults to `@`, for udp)
- ipv4 or ipv6 hostname or address. ipv6 addresses must be in `[::1]` format, (e.g. `@[fd00::1]:514`)
- optional port number (defaults to 514)

For example, to send logging to an rsyslog server named `log_serv` with address `10.0.4.1`, using port number 514, over udp, with all log messages enabled one could use either of the following.

With all options specified:

```

rsyslog:
  remotes:
    log_serv: " *.* @10.0.4.1:514"

```

With defaults used:

```

rsyslog:
  remotes:
    log_serv: "10.0.4.1"

```

For more information on rsyslog configuration, see *Rsyslog*.

1.21 Directory layout

Cloud-init's directory structure is somewhat different from a regular application:

```
/var/lib/cloud/
- data/
  - instance-id
  - previous-instance-id
  - datasource
  - previous-datasource
  - previous-hostname
- handlers/
- instance
- instances/
  i-00000XYZ/
    - boot-finished
    - cloud-config.txt
    - datasource
    - handlers/
    - obj.pkl
    - scripts/
    - sem/
    - user-data.txt
    - user-data.txt.i
- scripts/
  - per-boot/
  - per-instance/
  - per-once/
- seed/
- sem/
```

`/var/lib/cloud`

The main directory containing the cloud-init specific subdirectories. It is typically located at `/var/lib` but there are certain configuration scenarios where this can be altered.

TBD, describe this overriding more.

`data/`

Contains information related to instance ids, datasources and hostnames of the previous and current instance if they are different. These can be examined as needed to determine any information related to a previous boot (if applicable).

`handlers/`

Custom `part-handlers` code is written out here. Files that end up here are written out with in the scheme of `part-handler-XYZ` where `XYZ` is the handler number (the first handler found starts at 0).

`instance`

A symlink to the current `instances/` subdirectory that points to the currently active instance (which is active is dependent on the datasource loaded).

`instances/`

All instances that were created using this image end up with instance identifier subdirectories (and corresponding data for each instance). The currently active instance will be symlinked the `instance` symlink file defined previously.

`scripts/`

Scripts that are downloaded/created by the corresponding `part-handler` will end up in one of these subdirectories.

`seed/`

TBD

sem/

Cloud-init has a concept of a module semaphore, which basically consists of the module name and its frequency. These files are used to ensure a module is only ran *per-once*, *per-instance*, *per-always*. This folder contains semaphore *files* which are only supposed to run *per-once* (not tied to the instance id).

1.22 Analyze

The analyze subcommand was added to cloud-init in order to help analyze cloud-init boot time performance. It is loosely based on systemd-analyze where there are four subcommands:

- blame
- show
- dump
- boot

1.22.1 Usage

The analyze command requires one of the four subcommands:

```
$ cloud-init analyze blame
$ cloud-init analyze show
$ cloud-init analyze dump
$ cloud-init analyze boot
```

1.22.2 Availability

The analyze subcommand is generally available across all distributions with the exception of Gentoo and FreeBSD.

1.22.3 Subcommands

Blame

The blame action matches `systemd-analyze blame` where it prints, in descending order, the units that took the longest to run. This output is highly useful for examining where cloud-init is spending its time during execution.

```
$ cloud-init analyze blame
-- Boot Record 01 --
 00.80300s (init-network/config-growpart)
 00.64300s (init-network/config-resizefs)
 00.62100s (init-network/config-ssh)
 00.57300s (modules-config/config-grub-dpkg)
 00.40300s (init-local/search-NoCloud)
 00.38200s (init-network/config-users-groups)
 00.19800s (modules-config/config-apt-configure)
 00.03700s (modules-final/config-keys-to-console)
 00.02100s (init-network/config-update_etc_hosts)
 00.02100s (init-network/check-cache)
```

(continues on next page)

(continued from previous page)

```

00.00800s (modules-final/config-ssh-authkey-fingerprints)
00.00800s (init-network/consume-vendor-data)
00.00600s (modules-config/config-timezone)
00.00500s (modules-final/config-final-message)
00.00400s (init-network/consume-user-data)
00.00400s (init-network/config-mounts)
00.00400s (init-network/config-disk_setup)
00.00400s (init-network/config-bootcmd)
00.00400s (init-network/activate-datasource)
00.00300s (init-network/config-update_hostname)
00.00300s (init-network/config-set_hostname)
00.00200s (modules-final/config-snappy)
00.00200s (init-network/config-rsyslog)
00.00200s (init-network/config-ca-certs)
00.00200s (init-local/check-cache)
00.00100s (modules-final/config-scripts-vendor)
00.00100s (modules-final/config-scripts-per-once)
00.00100s (modules-final/config-salt-minion)
00.00100s (modules-final/config-rightscales_userdata)
00.00100s (modules-final/config-phone-home)
00.00100s (modules-final/config-package-update-upgrade-install)
00.00100s (modules-final/config-fan)
00.00100s (modules-config/config-ubuntu-advantage)
00.00100s (modules-config/config-ssh-import-id)
00.00100s (modules-config/config-snap)
00.00100s (modules-config/config-set-passwords)
00.00100s (modules-config/config-runcmd)
00.00100s (modules-config/config-locale)
00.00100s (modules-config/config-byobu)
00.00100s (modules-config/config-apt-pipelining)
00.00100s (init-network/config-write-files)
00.00100s (init-network/config-seed_random)
00.00100s (init-network/config-migrator)
00.00000s (modules-final/config-ubuntu-drivers)
00.00000s (modules-final/config-scripts-user)
00.00000s (modules-final/config-scripts-per-instance)
00.00000s (modules-final/config-scripts-per-boot)
00.00000s (modules-final/config-puppet)
00.00000s (modules-final/config-power-state-change)
00.00000s (modules-final/config-mcollective)
00.00000s (modules-final/config-lxd)
00.00000s (modules-final/config-landscape)
00.00000s (modules-final/config-chef)
00.00000s (modules-config/config-snap_config)
00.00000s (modules-config/config-ntp)
00.00000s (modules-config/config-emit_upstart)
00.00000s (modules-config/config-disable-ec2-metadata)
00.00000s (init-network/setup-datasource)

1 boot records analyzed

```

Show

The `show` action is similar to `systemd-analyze critical-chain` which prints a list of units, the time they started and how long they took. Cloud-init has four stages and within each stage a number of modules may run depending on configuration. `cloudinit-analyze show` will, for each boot, print this information and a summary

total time, per boot.

The following is an abbreviated example of the show output:

```
$ cloud-init analyze show
-- Boot Record 01 --
The total time elapsed since completing an event is printed after the "@" character.
The time the event takes is printed after the "+" character.

Starting stage: init-local
|`->no cache found @00.01700s +00.00200s
|`->found local data from DataSourceNoCloud @00.11000s +00.40300s
Finished stage: (init-local) 00.94200 seconds

Starting stage: init-network
|`->restored from cache with run check: DataSourceNoCloud [seed=/dev/sr0][dsmode=net]_
↪@04.79500s +00.02100s
|`->setting up datasource @04.88900s +00.00000s
|`->reading and applying user-data @04.90100s +00.00400s
|`->reading and applying vendor-data @04.90500s +00.00800s
|`->activating datasource @04.95200s +00.00400s
Finished stage: (init-network) 02.72100 seconds

Starting stage: modules-config
|`->config-emit_upstart ran successfully @15.43100s +00.00000s
|`->config-snap ran successfully @15.43100s +00.00100s
...
|`->config-runcmd ran successfully @16.22300s +00.00100s
|`->config-byobu ran successfully @16.23400s +00.00100s
Finished stage: (modules-config) 00.83500 seconds

Starting stage: modules-final
|`->config-snappy ran successfully @16.87400s +00.00200s
|`->config-package-update-upgrade-install ran successfully @16.87600s +00.00100s
...
|`->config-final-message ran successfully @16.93700s +00.00500s
|`->config-power-state-change ran successfully @16.94300s +00.00000s
Finished stage: (modules-final) 00.10300 seconds

Total Time: 4.60100 seconds

1 boot records analyzed
```

If additional boot records are detected then they are printed out from oldest to newest.

Dump

The dump action simply dumps the cloud-init logs that the analyze module is performing the analysis on and returns a list of dictionaries that can be consumed for other reporting needs. Each element in the list is a boot entry.

```
$ cloud-init analyze dump
[
{
  "description": "starting search for local datasources",
  "event_type": "start",
  "name": "init-local",
  "origin": "cloudinit",
```

(continues on next page)

(continued from previous page)

```

    "timestamp": 1567057578.037
  },
  {
    "description": "attempting to read from cache [check]",
    "event_type": "start",
    "name": "init-local/check-cache",
    "origin": "cloudinit",
    "timestamp": 1567057578.054
  },
  {
    "description": "no cache found",
    "event_type": "finish",
    "name": "init-local/check-cache",
    "origin": "cloudinit",
    "result": "SUCCESS",
    "timestamp": 1567057578.056
  },
  {
    "description": "searching for local data from DataSourceNoCloud",
    "event_type": "start",
    "name": "init-local/search-NoCloud",
    "origin": "cloudinit",
    "timestamp": 1567057578.147
  },
  {
    "description": "found local data from DataSourceNoCloud",
    "event_type": "finish",
    "name": "init-local/search-NoCloud",
    "origin": "cloudinit",
    "result": "SUCCESS",
    "timestamp": 1567057578.55
  },
  {
    "description": "searching for local datasources",
    "event_type": "finish",
    "name": "init-local",
    "origin": "cloudinit",
    "result": "SUCCESS",
    "timestamp": 1567057578.979
  },
  {
    "description": "searching for network datasources",
    "event_type": "start",
    "name": "init-network",
    "origin": "cloudinit",
    "timestamp": 1567057582.814
  },
  {
    "description": "attempting to read from cache [trust]",
    "event_type": "start",
    "name": "init-network/check-cache",
    "origin": "cloudinit",
    "timestamp": 1567057582.832
  },
  ...
  {
    "description": "config-power-state-change ran successfully",

```

(continues on next page)

(continued from previous page)

```

    "event_type": "finish",
    "name": "modules-final/config-power-state-change",
    "origin": "cloudinit",
    "result": "SUCCESS",
    "timestamp": 1567057594.98
  },
  {
    "description": "running modules for final",
    "event_type": "finish",
    "name": "modules-final",
    "origin": "cloudinit",
    "result": "SUCCESS",
    "timestamp": 1567057594.982
  }
]

```

Boot

The `boot` action prints out kernel related timestamps that are not included in any of the cloud-init logs. There are three different timestamps that are presented to the user:

- kernel start
- kernel finish boot
- cloud-init start

This was added for additional clarity into the boot process that cloud-init does not have control over, to aid in debugging of performance issues related to cloud-init startup, and tracking regression.

```

$ cloud-init analyze boot
-- Most Recent Boot Record --
  Kernel Started at: 2019-08-29 01:35:37.753790
  Kernel ended boot at: 2019-08-29 01:35:38.807407
  Kernel time to boot (seconds): 1.053617000579834
  Cloud-init activated by systemd at: 2019-08-29 01:35:43.992460
  Time between Kernel end boot and Cloud-init activation (seconds): 5.
↪185053110122681
  Cloud-init start: 2019-08-29 08:35:45.867000
successful

```

Timestamp Gathering

The following boot related timestamps are gathered on demand when `cloud-init analyze boot` runs:

- Kernel startup gathered from system uptime
- Kernel finishes initialization from systemd `UserSpaceMonotonicTimestamp` property
- Cloud-init activation from the property `InactiveExitTimestamp` of the cloud-init local systemd unit

In order to gather the necessary timestamps using systemd, running the commands below will gather the `Userspace-Timestamp` and `InactiveExitTimestamp`:

```
$ systemctl show -p UserspaceTimestampMonotonic
UserspaceTimestampMonotonic=989279
$ systemctl show cloud-init-local -p InactiveExitTimestampMonotonic
InactiveExitTimestampMonotonic=4493126
```

The `UserspaceTimestamp` tracks when the init system starts, which is used as an indicator of kernel finishing initialization. The `InactiveExitTimestamp` tracks when a particular `systemd` unit transitions from the Inactive to Active state, which can be used to mark the beginning of `systemd`'s activation of `cloud-init`.

Currently this only works for distros that use `systemd` as the init process. We will be expanding support for other distros in the future and this document will be updated accordingly.

If `systemd` is not present on the system, `dmesg` is used to attempt to find an event that logs the beginning of the init system. However, with this method only the first two timestamps are able to be found; `dmesg` does not monitor userspace processes, so no `cloud-init` start timestamps are emitted like when using `systemd`.

1.23 Docs

These docs are hosted on Read the Docs. The following will explain how to contribute to and build these docs locally. The documentation is primarily written in `reStructuredText`.

1.23.1 Building

There is a makefile target to build the documentation for you:

```
$ tox -e doc
```

This will do two things:

- Build the documentation using `sphinx`
- Run `doc8` against the documentation source code

Once build the HTML files will be viewable in `doc/rtd_html`. Use your web browser to open `index.html` to view and navigate the site.

1.23.2 Style Guide

Headings

The headings used across the documentation use the following hierarchy:

- `*****`: used once atop of a new page
- `=====`: each sections on the page
- `-----`: subsections
- `^^^^^`: sub-subsections
- `" " " "`: paragraphs

The top level header `#####` is reserved for the first page.

If under and overline are used, their length must be identical. The length of the underline must be at least as long as the title itself

Line Length

Please keep the line lengths to a maximum of **79** characters. This ensures that the pages and tables do not get too wide that side scrolling is required.

Header

Adding a link at the top of the page allows for the page to be referenced by other pages. For example for the FAQ page this would be:

```
.. _faq:
```

Footer

The footer should include the textwidth

```
.. vi: textwidth=79
```

Vertical Whitespace

One newline between each section helps ensure readability of the documentation source code.

Common Words

There are some common words that should follow specific usage:

- `cloud-init`: always lower case with a hyphen, unless starting a sentence in which case only the 'C' is capitalized (e.g. `Cloud-init`).
- `metadata`: one word
- `user data`: two words, not to be combined
- `vendor data`: like user data, it is two words

1.24 Testing

cloud-init has both unit tests and integration tests. Unit tests can be found at `tests/unittests`. Integration tests can be found at `tests/integration_tests`. Documentation specifically for integration tests can be found on the [Integration Testing](#) page, but the guidelines specified below apply to both types of tests.

cloud-init uses `pytest` to run its tests, and has tests written both as `unittest.TestCase` sub-classes and as un-subclassed `pytest` tests.

1.24.1 Guidelines

The following guidelines should be followed.

Test Layout

- For ease of organisation and greater accessibility for developers not familiar with pytest, all cloud-init unit tests must be contained within test classes
 - Put another way, module-level test functions should not be used
- As all tests are contained within classes, it is acceptable to mix `TestCase` test classes and pytest test classes within the same test file
 - These can be easily distinguished by their definition: pytest classes will not use inheritance at all (e.g. `TestGetPackageMirrorInfo`), whereas `TestCase` classes will subclass (indirectly) from `TestCase` (e.g. `TestPrependBaseCommands`)
- Unit tests and integration tests are located under `cloud-init/tests`
 - For consistency, unit test files should have a matching name and directory location under `tests/unittests`
 - For example: the expected test file for code in `cloudinit/path/to/file.py` is `tests/unittests/path/to/test_file.py`

pytest Tests

- pytest test classes should use `pytest fixtures` to share functionality instead of inheritance
- pytest tests should use bare `assert` statements, to take advantage of pytest's `assertion introspection`

pytest Version Gotchas

As we still support Ubuntu 18.04 (Bionic Beaver), we can only use pytest features that are available in v3.3.2. This is an inexhaustive list of ways in which this may catch you out:

- Only the following built-in fixtures are available¹:
 - `cache`
 - `capfd`
 - `capfdbinary`
 - `caplog`
 - `capsys`
 - `capsysbinary`
 - `doctest_namespace`
 - `monkeypatch`
 - `pytestconfig`
 - `record_xml_property`
 - `recwarn`
 - `tmpdir_factory`
 - `tmpdir`

¹ This list of fixtures (with markup) can be reproduced by running:

```
python3 -m pytest --fixtures -q | grep "^[^ -]" | grep -v 'no tests ran in' | sort | sed 's/↩️\n[session scope]\\//g;s/.*/`'\0`'/g'
```

in an ubuntu lxd container with python3-pytest installed.

Mocking and Assertions

- Variables/parameter names for `Mock` or `MagicMock` instances should start with `m_` to clearly distinguish them from non-mock variables
 - For example, `m_readurl` (which would be a mock for `readurl`)
- The `assert_*` methods that are available on `Mock` and `MagicMock` objects should be avoided, as typos in these method names may not raise `AttributeError` (and so can cause tests to silently pass). An important exception: if a `Mock` is `autospecced` then misspelled assertion methods *will* raise an `AttributeError`, so these assertion methods may be used on `autospecced` `Mock` objects.

For non-`autospecced` `Mock`s, these substitutions can be used (`m` is assumed to be a `Mock`):

- `m.assert_any_call(*args, **kwargs) => assert mock.call(*args, **kwargs) in m.call_args_list`
- `m.assert_called() => assert 0 != m.call_count`
- `m.assert_called_once() => assert 1 == m.call_count`
- `m.assert_called_once_with(*args, **kwargs) => assert [mock.call(*args, **kwargs)] == m.call_args_list`
- `m.assert_called_with(*args, **kwargs) => assert mock.call(*args, **kwargs) == m.call_args_list[-1]`
- `m.assert_has_calls(call_list, any_order=True) => for call in call_list: assert call in m.call_args_list`
 - `* m.assert_has_calls(...)` and `m.assert_has_calls(..., any_order=False)` are not easily replicated in a single statement, so their use when appropriate is acceptable.
- `m.assert_not_called() => assert 0 == m.call_count`

- When there are multiple patch calls in a test file for the module it is testing, it may be desirable to capture the shared string prefix for these patch calls in a module-level variable. If used, such variables should be named `M_PATH` or, for datasource tests, `DS_PATH`.

Test Argument Ordering

- Test arguments should be ordered as follows:
 - `mock.patch` arguments. When used as a decorator, `mock.patch` partially applies its generated `Mock` object as the first argument, so these arguments must go first.
 - `pytest.mark.parametrize` arguments, in the order specified to the `parametrize` decorator. These arguments are also provided by a decorator, so it's natural that they sit next to the `mock.patch` arguments.
 - Fixture arguments, alphabetically. These are not provided by a decorator, so they are last, and their order has no defined meaning, so we default to alphabetical.
- It follows from this ordering of test arguments (so that we retain the property that arguments left-to-right correspond to decorators bottom-to-top) that test decorators should be ordered as follows:
 - `pytest.mark.parametrize`
 - `mock.patch`

1.25 Integration Testing

1.25.1 Overview

Integration tests are written using pytest and are located at `tests/integration_tests`. General design principles laid out in *Testing* should be followed for integration tests.

Setup is accomplished via a set of fixtures located in `tests/integration_tests/conftest.py`.

1.25.2 Test Definition

Tests are defined like any other pytest test. The `user_data` mark can be used to supply the cloud-config user data. Platform specific marks can be used to limit tests to particular platforms. The client fixture can be used to interact with the launched test instance.

See *Examples* section for examples.

1.25.3 Test Execution

Test execution happens via pytest. A tox definition exists to run integration tests. To run all integration tests, you would run:

```
$ tox -e integration-tests
```

Pytest arguments may also be passed. For example:

```
$ tox -e integration-tests tests/integration_tests/modules/test_combined.py
```

1.25.4 Configuration

All possible configuration values are defined in `tests/integration_tests/integration_settings.py`. Defaults can be overridden by supplying values in `tests/integration_tests/user_settings.py` or by providing an environment variable of the same name prepended with `CLOUD_INIT_`. For example, to set the `PLATFORM` setting:

```
CLOUD_INIT_PLATFORM='ec2' pytest tests/integration_tests/
```

1.25.5 Cloud Interaction

Cloud interaction happens via the `pycloudlib` library. In order to run integration tests, `pycloudlib` must first be configured.

For a minimal setup using LXD, write the following to `~/config/pycloudlib.toml`:

```
[lxd]
```

1.25.6 Image Selection

Each integration testing run uses a single image as its basis. This image is configured using the `OS_IMAGE` variable; see *Configuration* for details of how configuration works.

OS_IMAGE can take two types of value: an Ubuntu series name (e.g. “focal”), or an image specification. If an Ubuntu series name is given, then the most recent image for that series on the target cloud will be used. For other use cases, an image specification is used.

In its simplest form, an image specification can simply be a cloud’s image ID (e.g. “ami-deadbeef”, “ubuntu:focal”). In this case, the image so-identified will be used as the basis for this testing run.

This has a drawback, however: as we do not know what OS or release is within the image, the integration testing framework will run *all* tests against the image in question. If it’s a RHEL8 image, then we would expect Ubuntu-specific tests to fail (and vice versa).

To address this, a full image specification can be given. This is of the form: `<image_id>[:<os>[:<release>]]` where `image_id` is a cloud’s image ID, `os` is the OS name, and `release` is the OS release name. So, for example, Ubuntu 18.04 (Bionic Beaver) on LXD is `ubuntu:bionic::ubuntu::bionic` or RHEL 8 on Amazon is `ami-justanexample::rhel::8`. When a full specification is given, only tests which are intended for use on that OS and release will be executed.

1.25.7 Image Setup

Image setup occurs once when a test session begins and is implemented via fixture. Image setup roughly follows these steps:

- Launch an instance on the specified test platform
- Install the version of cloud-init under test
- Run `cloud-init clean` on the instance so subsequent boots resemble out of the box behavior
- Take a snapshot of the instance to be used as a new image from which new instances can be launched

1.25.8 Test Setup

Test setup occurs between image setup and test execution. Test setup is implemented via one of the `client` fixtures. When a client fixture is used, a test instance from which to run tests is launched prior to test execution and torn down after.

1.25.9 Continuous Integration

A subset of the integration tests are run when a pull request is submitted on Github. The tests run on these continuous integration (CI) runs are given a pytest mark:

```
@pytest.mark.ci
```

Most new tests should *not* use this mark, so be aware that having a successful CI run does not necessarily mean that your test passed successfully.

1.25.10 Fixtures

Integration tests rely heavily on fixtures to do initial test setup. One or more of these fixtures will be used in almost every integration test.

Details such as the cloud platform or initial image to use are determined via what is specified in the [Configuration](#).

client

The `client` fixture should be used for most test cases. It ensures:

- All setup performed by `session_cloud` and `setup_image`
- `Pytest marks` used during instance creation are obtained and applied
- The test instance is launched
- Test failure status is determined after test execution
- Logs are collected (if configured) after test execution
- The test instance is torn down after test execution

`module_client` and `class_client` fixtures also exist for the purpose of running multiple tests against a single launched instance. They provide the exact same functionality as `client`, but are scoped to the module or class respectively.

session_cloud

The `session_cloud` session-scoped fixture will provide an `IntegrationCloud` instance for the currently configured cloud. The fixture also ensures that any custom cloud session cleanup is performed.

setup_image

The `setup_image` session-scope fixture will create a new image to launch all further cloud instances during this test run. It ensures:

- A cloud instance is launched on the configured platform
- The version of cloud-init under test is installed on the instance
- `cloud-init clean --logs` is run on the instance
- A snapshot of the instance is taken to be used as the basis for future instance launches
- The originally launched instance is torn down
- The custom created image is torn down after all tests finish

Examples

A simple test case using the `client` fixture:

```
USER_DATA = """\
#cloud-config
bootcmd:
- echo 'hello!' > /var/tmp/hello.txt
"""

@pytest.mark.user_data(USER_DATA)
def test_bootcmd(client):
    log = client.read_from_file("/var/log/cloud-init.log")
    assert "Shellified 1 commands." in log
    assert client.execute('cat /var/tmp/hello.txt').strip() == "hello!"
```

Customizing the launch arguments before launching an instance manually:

```
def test_launch(session_cloud: IntegrationCloud, setup_image):  
    with session_cloud.launch(launch_kwargs={"wait": False}) as client:  
        client.instance.wait()  
        assert client.execute("echo hello world").strip() == "hello world"
```


C

cloudinit.config.cc_apk_configure, 59
cloudinit.config.cc_apt_configure, 60
cloudinit.config.cc_apt_pipelining, 64
cloudinit.config.cc_bootcmd, 65
cloudinit.config.cc_byobu, 65
cloudinit.config.cc_ca_certs, 66
cloudinit.config.cc_chef, 67
cloudinit.config.cc_debug, 69
cloudinit.config.cc_disable_ec2_metadata, 69
cloudinit.config.cc_disk_setup, 70
cloudinit.config.cc_emit_upstart, 71
cloudinit.config.cc_fan, 72
cloudinit.config.cc_final_message, 72
cloudinit.config.cc_foo, 73
cloudinit.config.cc_growpart, 73
cloudinit.config.cc_grub_dpkg, 74
cloudinit.config.cc_install_hotplug, 74
cloudinit.config.cc_keyboard, 75
cloudinit.config.cc_keys_to_console, 75
cloudinit.config.cc_landscape, 76
cloudinit.config.cc_locale, 77
cloudinit.config.cc_lxd, 77
cloudinit.config.cc_mcollective, 78
cloudinit.config.cc_migrator, 78
cloudinit.config.cc_mounts, 79
cloudinit.config.cc_ntp, 80
cloudinit.config.cc_package_update_upgrade_install, 81
cloudinit.config.cc_phone_home, 82
cloudinit.config.cc_power_state_change, 82
cloudinit.config.cc_puppet, 83
cloudinit.config.cc_resizefs, 84
cloudinit.config.cc_resolv_conf, 85
cloudinit.config.cc_rh_subscription, 85
cloudinit.config.cc_rightscale_userdata, 86
cloudinit.config.cc_rsyslog, 86
cloudinit.config.cc_runcmd, 88
cloudinit.config.cc_salt_minion, 88
cloudinit.config.cc_scripts_per_boot, 89
cloudinit.config.cc_scripts_per_instance, 89
cloudinit.config.cc_scripts_per_once, 89
cloudinit.config.cc_scripts_user, 90
cloudinit.config.cc_scripts_vendor, 90
cloudinit.config.cc_seed_random, 90
cloudinit.config.cc_set_hostname, 91
cloudinit.config.cc_set_passwords, 92
cloudinit.config.cc_snap, 93
cloudinit.config.cc_spacewalk, 94
cloudinit.config.cc_ssh, 95
cloudinit.config.cc_ssh_authkey_fingerprints, 97
cloudinit.config.cc_ssh_import_id, 97
cloudinit.config.cc_timezone, 98
cloudinit.config.cc_ubuntu_advantage, 98
cloudinit.config.cc_ubuntu_drivers, 99
cloudinit.config.cc_update_etc_hosts, 99
cloudinit.config.cc_update_hostname, 100
cloudinit.config.cc_users_groups, 100
cloudinit.config.cc_write_files, 102
cloudinit.config.cc_yum_add_repo, 104
cloudinit.features, 178

A

ALLOW_EC2_MIRRORS_ON_NON_AWS_INSTANCE_TYPES
(in module *cloudinit.features*), 178

C

cloudinit.config.cc_apk_configure (module), 59
cloudinit.config.cc_apt_configure (module), 60
cloudinit.config.cc_apt_pipelining (module), 64
cloudinit.config.cc_bootcmd (module), 65
cloudinit.config.cc_byobu (module), 65
cloudinit.config.cc_ca_certs (module), 66
cloudinit.config.cc_chef (module), 67
cloudinit.config.cc_debug (module), 69
cloudinit.config.cc_disable_ec2_metadata (module), 69
cloudinit.config.cc_disk_setup (module), 70
cloudinit.config.cc_emit_upstart (module), 71
cloudinit.config.cc_fan (module), 72
cloudinit.config.cc_final_message (module), 72
cloudinit.config.cc_foo (module), 73
cloudinit.config.cc_growpart (module), 73
cloudinit.config.cc_grub_dpkg (module), 74
cloudinit.config.cc_install_hotplug (module), 74
cloudinit.config.cc_keyboard (module), 75
cloudinit.config.cc_keys_to_console (module), 75
cloudinit.config.cc_landscape (module), 76
cloudinit.config.cc_locale (module), 77
cloudinit.config.cc_lxd (module), 77
cloudinit.config.cc_mcollective (module), 78
cloudinit.config.cc_migrator (module), 78
cloudinit.config.cc_mounts (module), 79
cloudinit.config.cc_ntp (module), 80
cloudinit.config.cc_package_update_upgrade_install (module), 81
cloudinit.config.cc_phone_home (module), 82
cloudinit.config.cc_power_state_change (module), 82
cloudinit.config.cc_puppet (module), 83
cloudinit.config.cc_resizefs (module), 84
cloudinit.config.cc_resolv_conf (module), 85
cloudinit.config.cc_rh_subscription (module), 85
cloudinit.config.cc_rightscale_userdata (module), 86
cloudinit.config.cc_rsyslog (module), 86
cloudinit.config.cc_runcmd (module), 88
cloudinit.config.cc_salt_minion (module), 88
cloudinit.config.cc_scripts_per_boot (module), 89
cloudinit.config.cc_scripts_per_instance (module), 89
cloudinit.config.cc_scripts_per_once (module), 89
cloudinit.config.cc_scripts_user (module), 90
cloudinit.config.cc_scripts_vendor (module), 90
cloudinit.config.cc_seed_random (module), 90
cloudinit.config.cc_set_hostname (module), 91
cloudinit.config.cc_set_passwords (module), 92
cloudinit.config.cc_snap (module), 93
cloudinit.config.cc_spacewalk (module), 94
cloudinit.config.cc_ssh (module), 95
cloudinit.config.cc_ssh_authkey_fingerprints

(module), 97
cloudinit.config.cc_ssh_import_id (*module*), 97
cloudinit.config.cc_timezone (*module*), 98
cloudinit.config.cc_ubuntu_advantage (*module*), 98
cloudinit.config.cc_ubuntu_drivers (*module*), 99
cloudinit.config.cc_update_etc_hosts (*module*), 99
cloudinit.config.cc_update_hostname (*module*), 100
cloudinit.config.cc_users_groups (*module*), 100
cloudinit.config.cc_write_files (*module*), 102
cloudinit.config.cc_yum_add_repo (*module*), 104
cloudinit.features (*module*), 178

E

ERROR_ON_USER_DATA_FAILURE (*in module cloudinit.features*), 178