cloud-init

Release 23.1.2

unknown

Apr 26, 2023

CONTENTS

1 Having trouble? We would like to help! 3

2 Project and community 5

2.1 Tutorials . . . 5
2.2 How-to guides . . . 13
2.3 Explanation . . . 17
2.4 Reference . . . 50
2.5 Development . . . 245

Python Module Index 275

Index 277

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

During boot, `cloud-init` identifies the cloud it is running on and initialises the system accordingly. Cloud instances will automatically be provisioned during first boot with networking, storage, SSH keys, packages and various other system aspects already configured.

`Cloud-init` provides the necessary glue between launching a cloud instance and connecting to it so that it works as expected.

For cloud users, `cloud-init` provides no-install first-boot configuration management of a cloud instance. For cloud providers, it provides instance setup that can be integrated with your cloud.

Tutorials Get started - a hands-on introduction to `cloud-init` for new users

How-to guides Step-by-step guides covering key operations and common tasks

Reference Technical information - specifications, APIs, architecture

Explanation Discussion and clarification of key topics

CHAPTER

ONE

HAVING TROUBLE? WE WOULD LIKE TO HELP!

- *Check out our tutorials* if you're new to `cloud-init`
- *Try the FAQ* for answers to some common questions
- You can also search the `cloud-init` mailing list archive
- Find a bug? Report bugs on Launchpad

CHAPTER

TWO

PROJECT AND COMMUNITY

`Cloud-init` is an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

- Read our Code of Conduct
- Ask questions in the `#cloud-init` IRC channel on Libera
- Join the cloud-init mailing list
- *Contribute on GitHub*
- Release schedule

2.1 Tutorials

This section contains step-by-step tutorials to help you get started with `cloud-init`. We hope our tutorials make as few assumptions as possible and are accessible to anyone with an interest in `cloud-init`. They should be a great place to start learning about `cloud-init`, how it works, and what it's capable of.

2.1.1 Core tutorial

This tutorial, which we recommend if you are completely new to `cloudinit`, uses the QEMU emulator to introduce you to all of the key concepts, tools, processes and operations that you will need to get started.

Core tutorial with QEMU

QEMU tutorial debugging

You may wish to test out the commands in this tutorial as a `script` to check for copy-paste mistakes.

If you successfully launched the virtual machine, but couldn't log in, there are a few places to check to debug your setup.

To debug, answer the following questions:

Did `cloud-init` discover the IMDS webserver?

The webserver should print a message in the terminal for each request it receives. If it didn't print out any messages when the virtual machine booted, then `cloud-init` was unable to obtain the config. Make sure that the webserver can be locally accessed using **`curl`** or **`wget`**.

```
$ curl 0.0.0.0:8000/user-data
$ curl 0.0.0.0:8000/meta-data
$ curl 0.0.0.0:8000/vendor-data
```

Did the IMDS webserver serve the expected files?

If the webserver prints out `404 errors` when launching QEMU, then check that you started the server in the temp directory.

Were the configurations inside the file correct?

When launching QEMU, if the webserver shows that it succeeded in serving `user-data`, `meta-data` and `vendor-data`, but you cannot log in, then you may have provided incorrect cloud-config files. If you can mount a copy of the virtual machine's filesystem locally to inspect the logs, it should be possible to get clues about what went wrong.

In this tutorial, we will launch an Ubuntu cloud image in a virtual machine that uses `cloud-init` to pre-configure the system during boot.

The goal of this tutorial is to provide a minimal demonstration of `cloud-init`, which you can then use as a development environment to test your `cloud-init` configurations locally before launching to the cloud.

Why QEMU?

QEMU is a cross-platform emulator capable of running performant virtual machines. QEMU is used at the core of a broad range of production operating system deployments and open source software projects (including libvirt, LXD, and vagrant) and is capable of running Windows, Linux, and Unix guest operating systems. While QEMU is flexibile and feature-rich, we are using it because of the broad support it has due to its broad adoption and ability to run on *nix-derived operating systems.

How to use this tutorial

In this tutorial, the commands in each code block can be copied and pasted directly into the terminal. Omit the prompt (`$`) before each command, or use the "copy code" button on the right-hand side of the block, which will copy the command for you without the prompt.

Each code block is preceded by a description of what the command does, and followed by an example of the type of output you should expect to see.

Install QEMU

```
$ sudo apt install qemu-system-x86
```

If you are not using Ubuntu, you can visit QEMU's install instructions for additional information.

Create a temporary directory

This directory will store our cloud image and configuration files for *user data*, *metadata*, and *vendor data*.

You should run all commands from this temporary directory. If you run the commands from anywhere else, your virtual machine will not be configured.

Let's create a temporary directory and make it our current working directory with **cd**:

```
$ mkdir temp
$ cd temp
```

Download a cloud image

Cloud images typically come with `cloud-init` pre-installed and configured to run on first boot. You will not need to worry about installing `cloud-init` for now, since we are not manually creating our own image in this tutorial.

In our case, we want to select the latest Ubuntu LTS. Let's download the server image using **wget**:

```
$ wget https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64.img
```

Define our user data

Now we need to create our `user-data` file. This user data cloud-config sets the password of the default user, and sets that password to never expire. For more details you can refer to the *Set Passwords module page*.

Run the following command, which creates a file named `user-data` containing our configuration data.

```
$ cat << EOF > user-data
#cloud-config
password: password
chpasswd:
  expire: False

EOF
```

What is user data?

Before moving forward, let's inspect our `user-data` file.

```
$ cat user-data
```

You should see the following contents:

```
#cloud-config
password: password
chpasswd:
  expire: False
```

The first line starts with `#cloud-config`, which tells `cloud-init` what type of user data is in the config. Cloud-config is a YAML-based configuration type that tells `cloud-init` how to configure the virtual machine instance. Multiple different format types are supported by `cloud-init`. For more information, see the *documentation describing different formats*.

The second line, `password: password`, as per *the Users and Groups module docs*, sets the default user's password to `password`.

The third and fourth lines direct `cloud-init` to not require a password reset on first login.

Define our metadata

Now let's run the following command, which creates a file named `meta-data` containing configuration data.

```
$ cat << EOF > meta-data
instance-id: someid/somehostname
local-hostname: jammy

EOF
```

Define our vendor data

Now we will create the empty file `vendor-data` in our temporary directory. This will speed up the retry wait time.

```
$ touch vendor-data
```

Start an ad hoc IMDS webserver

Open up a second terminal window, change to your temporary directory and then start the built-in Python webserver:

```
$ cd temp
$ python3 -m http.server --directory .
```

What is an IMDS?

Instance Metadata Service (IMDS) is a service provided by most cloud providers as a means of providing information to virtual machine instances. This service is used by cloud providers to expose information to a virtual machine. This service is used for many different things, and is the primary mechanism for some clouds to expose `cloud-init` configuration data to the instance.

How does `cloud-init` use the IMDS?

The IMDS uses a private http webserver to provide metadata to each operating system instance. During early boot, `cloud-init` sets up network access and queries this webserver to gather configuration data. This allows `cloud-init` to configure your operating system while it boots.

In this tutorial we are emulating this workflow using QEMU and a simple Python webserver. This workflow is suitable for developing and testing `cloud-init` configurations prior to cloud deployments.

Launch a virtual machine with our user data

Switch back to your original terminal, and run the following command so we can launch our virtual machine. By default, QEMU will print the kernel logs and `systemd` logs to the terminal while the operating system boots. This may take a few moments to complete.

```
$ qemu-system-x86_64                                                    \
    -net nic                                                            \
    -net user                                                           \
    -machine accel=kvm:tcg                                              \
    -cpu host                                                           \
    -m 512                                                              \
    -nographic                                                          \
    -hda jammy-server-cloudimg-amd64.img                                \
    -smbios type=1,serial=ds='nocloud-net;s=http://10.0.2.2:8000/'
```

Note: If the output stopped scrolling but you don't see a prompt yet, press `Enter` to get to the login prompt.

How is QEMU configured for `cloud-init`?

When launching QEMU, our machine configuration is specified on the command line. Many things may be configured: memory size, graphical output, networking information, hard drives and more.

Let us examine the final two lines of our previous command. The first of them, **`-hda jammy-server-cloudimg-amd64.img`**, tells QEMU to use the cloud image as a virtual hard drive. This will cause the virtual machine to boot Ubuntu, which already has `cloud-init` installed.

The second line tells `cloud-init` where it can find user data, using the *NoCloud datasource*. During boot, `cloud-init` checks the `SMBIOS` serial number for `ds=nocloud-net`. If found, `cloud-init` will use the specified URL to source its user data config files.

In this case, we use the default gateway of the virtual machine (`10.0.2.2`) and default port number of the Python webserver (`8000`), so that `cloud-init` will, inside the virtual machine, query the server running on host.

Verify that `cloud-init` ran successfully

After launching the virtual machine, we should be able to connect to our instance using the default distro username.

In this case the default username is `ubuntu` and the password we configured is `password`.

If you can log in using the configured password, it worked!

If you couldn't log in, see *this page for debug information*.

Check `cloud-init` status

Run the following command, which will allow us to check if `cloud-init` has finished running:

```
$ cloud-init status --wait
```

If you see `status: done` in the output, it succeeded!

If you see a failed status, you'll want to check `/var/log/cloud-init.log` for warning/error messages.

Tear down

In our main terminal, let's exit the QEMU shell using `ctrl-a x` (that's `ctrl` and `a` simultaneously, followed by `x`).

In the second terminal, where the Python webserver is running, we can stop the server using (`ctrl-c`).

What's next?

In this tutorial, we configured the default user's password and ran `cloud-init` inside our QEMU virtual machine.

The full list of modules available can be found in *our modules documentation*. The documentation for each module contains examples of how to use it.

You can also head over to the *examples page* for examples of more common use cases.

2.1.2 Quick-start tutorial

This tutorial is recommended if you have some familiarity with `cloud-init` or the concepts around it, and are looking to get started as quickly as possible. Here, you will use an LXD container to deploy a `cloud-init` user data script.

Quick-start tutorial with LXD

In this tutorial, we will create our first `cloud-init` user data script and deploy it into an LXD container.

Why LXD?

We'll be using LXD for this tutorial because it provides first class support for `cloud-init` user data, as well as `systemd` support. Because it is container based, it allows us to quickly test and iterate upon our user data definition.

How to use this tutorial

In this tutorial, the commands in each code block can be copied and pasted directly into the terminal. Omit the prompt (`$`) before each command, or use the "copy code" button on the right-hand side of the block, which will copy the command for you without the prompt.

Each code block is preceded by a description of what the command does, and followed by an example of the type of output you should expect to see.

Install and initialise LXD

If you already have LXD set up, you can skip this section. Otherwise, let's install LXD:

```
$ sudo snap install lxd
```

If you don't have snap, you can install LXD using one of the other installation options.

Now we need to initialise LXD. The minimal configuration will be enough for the purposes of this tutorial. If you need to, you can always change the configuration at a later time.

```
$ lxd init --minimal
```

Define our user data

Now that LXD is set up, we can define our user data. Create the following file on your local filesystem at `/tmp/my-user-data`:

```
#cloud-config
runcmd:
  - echo 'Hello, World!' > /var/tmp/hello-world.txt
```

Here, we are defining our `cloud-init` user data in the *#cloud-config* format, using the *runcmd module* to define a command to run. When applied, it will write `Hello, World!` to `/var/tmp/hello-world.txt` (as we shall see later!).

Launch a LXD container with our user data

Now that we have LXD set up and our user data defined, we can launch an instance with our user data:

```
$ lxc launch ubuntu:focal my-test --config=user.user-data="$(cat /tmp/my-user-data)"
```

Verify that `cloud-init` ran successfully

After launching the container, we should be able to connect to our instance using:

```
$ lxc shell my-test
```

You should now be in a shell inside the LXD instance.

Before validating the user data, let's wait for `cloud-init` to complete successfully:

```
$ cloud-init status --wait
```

Which provides the following output:

```
status: done
```

Verify our user data

Now we know that `cloud-init` has been successfully run, we can verify that it received the expected user data we provided earlier:

```
$ cloud-init query userdata
```

Which should print the following to the terminal window:

```
#cloud-config
runcmd:
  - echo 'Hello, World!' > /var/tmp/hello-world.txt
```

We can also assert the user data we provided is a valid cloud-config:

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

Which should print the following:

```
Valid cloud-config: system userdata
```

Finally, let us verify that our user data was applied successfully:

```
$ cat /var/tmp/hello-world.txt
```

Which should then print:

```
Hello, World!
```

We can see that `cloud-init` has received and consumed our user data successfully!

Tear down

Exit the container shell (by typing **exit** or pressing ctrl-d). Once we have exited the container, we can stop the container using:

```
$ lxc stop my-test
```

We can then remove the container completely using:

```
$ lxc rm my-test
```

What's next?

In this tutorial, we used the *runcmd module* to execute a shell command. The full list of modules available can be found in our *modules documentation*. Each module contains examples of how to use it.

You can also head over to the *examples page* for examples of more common use cases.

2.2 How-to guides

If you have a specific goal in mind and are already familiar with the basics of cloud-init, our how-to guides cover some of the more common operations and tasks that you may need to complete.

They will help you to achieve a particular end result, but may require you to understand and adapt the steps to fit your specific requirements.

2.2.1 How do I...?

How to test cloud-init locally before deploying

It's very likely that you will want to test cloud-init locally before deploying it to the cloud. Fortunately, there are several different virtual machines (VMs) and container tools that are ideal for this sort of local testing.

In this guide, we will show how to use three of the most popular tools: Multipass, LXD and QEMU.

Multipass

Multipass is a cross-platform tool for launching 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 the 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 which saves time when you're experimenting and 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, and
- network configuration.

The following command initialises 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 set up with the specific data if you need 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	cloud-init.user-data
vendor data	cloud-init.vendor-data
network config	cloud-init.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.

QEMU

The **cloud-localds** command from the cloud-utils package generates a disk with user-supplied data. The NoCloud datasouce 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
```

(continues on next page)

(continued from previous page)

```
ssh_authorized_keys:
  - ssh-rsa AAAA...UlIsqdaO+w==
EOF
$ cloud-localds seed.img user-data
```

The resulting `seed.img` can then be 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.

How to change a module's run frequency

You may want to change the default frequency at which a module runs, for example, to make the module run on every boot.

To override the default frequency, you will need to modify the module list in `/etc/cloud/cloud.cfg`:

1. Change the module from a string (default) to a list.
2. Set the first list item to the module name and the second item to the frequency.

Example

The following example demonstrates how to log boot times to a file every boot.

Update `/etc/cloud/cloud.cfg`:

```
    cloud_final_modules:
    # list shortened for brevity
     - [phone-home, always]
     - final-message
     - power-state-change
```

Then your user data could then be:

```
#cloud-config
phone_home:
    url: http://example.com/$INSTANCE_ID/
    post: all
```

Reporting bugs

In this guide, we will show you how to:

1) Collect logs to support your bug report.
2) File bugs to the upstream `cloud-init` project via Launchpad.
3) Report issues for distro-specific packages.

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
```

Example output:

```
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 running 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
```

Report upstream bugs

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 the following section for details on specific distro reporting.

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 of `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 and Fedora

To file a bug against the Red Hat or Fedora packages of `cloud-init` please use the Red Hat bugzilla.

SUSE and openSUSE

To file a bug against the SUSE packages of `cloud-init` please use the SUSE bugzilla.

Arch Linux

To file a bug against the Arch package of `cloud-init` please use the Arch Linux Bugtracker. See the Arch Linux bug reporting wiki for more details.

2.3 Explanation

Our explanatory and conceptual guides are written to provide a better understanding of how `cloud-init` works. They enable you to expand your knowledge and become better at using and configuring `cloud-init`.

2.3.1 Configuration sources

Internally, `cloud-init` builds a single configuration that is then referenced throughout the life of `cloud-init`. The configuration is built from multiple sources such that if a key is defined in multiple sources, the higher priority source overwrites the lower priority source.

Base configuration

From lowest priority to highest, configuration sources are:

- **Hardcoded config**: Config that lives within the source of `cloud-init` and cannot be changed.
- **Configuration directory**: Anything defined in `/etc/cloud/cloud.cfg` and `/etc/cloud/cloud.cfg.d`.
- **Runtime config**: Anything defined in `/run/cloud-init/cloud.cfg`.
- **Kernel command line**: On the kernel command line, anything found between `cc:` and `end_cc` will be interpreted as cloud-config user data.

These four sources make up the base configuration.

Vendor and user data

Added to the base configuration are *vendor data* and *user data* which are both provided by the datasource.

These get fetched from the datasource and are defined at instance launch.

Note: While much of what is defined in the base configuration can be overridden by vendor data and user data, base configuration sources do not conform to *#cloud-config*.

Network configuration

Network configuration happens independently from other `cloud-init` configuration. See *network configuration documentation* for more information.

Specifying configuration

End users

Pass *user data* to the cloud provider. Every platform supporting `cloud-init` will provide a method of supplying user data. If you're unsure how to do this, reference the documentation provided by the cloud platform you're on. Additionally, there may be related `cloud-init` documentation in the *datasource* section.

Once an instance has been initialised, the user data may not be edited. It is sourced directly from the cloud, so even if you find a local file that contains user data, it will likely be overwritten in the next boot.

Distro providers

Modify the base config. This often involves submitting a PR to modify the base cloud.cfg template, which is used to customise `/etc/cloud/cloud.cfg` per distro. Additionally, a file can be added to `/etc/cloud/cloud.cfg.d` to override a piece of the base configuration.

Cloud providers

Pass vendor data. This is the preferred method for clouds to provide their own customisation. In some cases, it may make sense to modify the base config in the same manner as distro providers on cloud-supported images.

2.3.2 Boot stages

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

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

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, or
- 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.

Note: These mechanisms for disabling `cloud-init` at runtime currently only exist in `systemd`.

Local

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

The purpose of the local stage is to:

- Locate "local" data sources, and
- 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 datasources 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.

Network

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

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

- retrieve any `#include` or `#include-once` (recursively) including http,
- decompress any compressed content, and
- 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 the 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 in `/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 boothooks 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.

Config

systemd service	`cloud-config.service`
runs	after network
blocks	nothing
modules	*cloud_config_modules* in `/etc/cloud/cloud.cfg`

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

Final

systemd service	`cloud-final.service`
runs	as final part of boot (traditional "rc.local")
blocks	nothing
modules	*cloud_final_modules* in `/etc/cloud/cloud.cfg`

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 (Ansible, Puppet, Chef, salt-minion), and
- 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 --wait`** 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.

First boot determination

`Cloud-init` has to determine whether or not the current boot is the first boot of a new instance, 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 the 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 behaviour as `check`.

[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.

This behaviour is required for images captured from launched instances to behave correctly, and so is the default that 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 behaviour 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 behaviour is referred to as `trust`.

To configure which of these behaviours 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).

Manual cache cleaning

`Cloud-init` ships a command for manually cleaning the cache: **`cloud-init clean`**. See *clean*'s documentation for further details.

Reverting `manual_cache_clean` setting

Currently there is no support for switching an instance that is launched with `manual_cache_clean: true` from `trust` behaviour to `check` behaviour, 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.

2.3.3 User data formats

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

[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 behaviour 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).

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.

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.

User data scripts can optionally render cloud instance metadata variables using jinja templating. See *Instance metadata* for more information.

Example script

Create a script file `myscript.sh` that contains the following:

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

Now run:

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

Kernel command line

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

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 ~16384[1] bytes.

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
```

Example output:

```
cloud-boothook
cloud-config
cloud-config-archive
cloud-config-jsonp
jinja2
part-handler
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` make-mime subcommand can also generate MIME multi-part files.

The **`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`.

[1] See your cloud provider for applicable user-data size limitations…

Examples

Create user data 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 user data 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
```

`include` file

This content is an `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.

`cloud-boothook`

This content is *boothook* data. It is stored in a file under `/var/lib/cloud` and 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.

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. Since 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.
```

(continues on next page)

(continued from previous page)

```
  # 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 initialisation or teardown before or after receiving any parts.

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

Example

```
#part-handler

def list_types():
    # return a list of mime-types that are handled by this module
    return(["text/plain", "text/go-cubs-go"])

def handle_part(data, ctype, filename, payload):
    # data: the cloudinit object
    # ctype: '__begin__', '__end__', or the specific mime-type of the part
    # filename: the filename for the part, or dynamically generated part if
    #           no filename is given attribute is present
    # payload: the content of the part (empty for begin or end)
    if ctype == "__begin__":
       print("my handler is beginning")
       return
    if ctype == "__end__":
       print("my handler is ending")
       return

    print(f"==== received ctype={ctype} filename={filename} ====")
    print(payload)
    print(f"==== end ctype={ctype} filename={filename}")
```

Also, this blog post offers another example for more advanced usage.

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.

2.3.4 Events and updates

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 the *Local stage*, then again in the *Network stage*. As this behaviour was previously the default behaviour, this option exists to prevent regressing such behaviour.
- `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 changed.
- `USER_REQUEST`: Directed request to update.

Datasource event support

All *datasources* support the `BOOT_NEW_INSTANCE` event by default. 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.

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 which domain the event occurs under. Currently, the only 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.

Hotplug

When the `hotplug` event is supported by the datasource 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.

Example

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']
```

2.3.5 Instance metadata

Kernel command line

In order to allow an ephemeral, or otherwise pristine image to receive some configuration, `cloud-init` will read a URL directed by the kernel command line and proceed as if its data had previously existed.

This allows for configuring a metadata service, or some other data.

> **Note:** Usage of the kernel command line is somewhat of a last resort, as it requires knowing in advance the correct command line or modifying the boot loader to append data.

For example, when **`cloud-init init --local`** runs, it will check to see if `cloud-config-url` appears in key/value fashion in the kernel command line, as in:

```
root=/dev/sda ro cloud-config-url=http://foo.bar.zee/abcde
```

`Cloud-init` will then read the contents of the given URL. If the content starts with `#cloud-config`, it will store that data to the local filesystem in a static filename `/etc/cloud/cloud.cfg.d/91_kernel_cmdline_url.cfg`, and consider it as part of the config from that point forward.

If that file exists already, it will not be overwritten, and the `cloud-config-url` parameter is completely ignored.

Then, when the datasource runs, it will find that config already available.

So, to be able to configure the MAAS datasource by controlling the kernel command line from outside the image, you can append:

```
cloud-config-url=http://your.url.here/abcdefg
```

Then, have the following content at that url:

```
#cloud-config
datasource:
  MAAS:
    metadata_url: http://mass-host.localdomain/source
    consumer_key: Xh234sdkljf
    token_key: kjfhgb3n
    token_secret: 24uysdfx1w4
```

Warning: `url` kernel command line key is deprecated. Please use `cloud-config-url` parameter instead.

Note: Since `cloud-config-url=` is so generic, in order to avoid false positives, `cloud-init` requires the content to start with `#cloud-config` for it to be considered.

Note: The `cloud-config-url=` is un-authed http GET, and contains credentials. It could be set up to be randomly generated and also to check the source address in order to be more secure.

What is `instance-data`?

Each cloud provider presents unique configuration metadata to a launched cloud instance. `Cloud-init` crawls this metadata and then caches and exposes this information as a standardised and versioned JSON object known as `instance-data`. This `instance-data` may then be queried or later used by `cloud-init` in templated configuration and scripts.

An example of a small subset of instance-data on a launched EC2 instance:


```
{
  "v1": {
    "cloud_name": "aws",
    "distro": "ubuntu",
    "distro_release": "jammy",
    "distro_version": "22.04",
    "instance_id": "i-06b5687b4d7b8595d",
    "machine": "x86_64",
    "platform": "ec2",
    "python_version": "3.10.4",
    "region": "us-east-2",
    "variant": "ubuntu"
  }
}
```

Discovery

One way to easily explore which `instance-data` variables are available on your machine is to use the *cloud-init query* tool. Warnings or exceptions will be raised on invalid `instance-data` keys, paths or invalid syntax.

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.

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

Using `instance-data`

`instance-data` can be used in:

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

The aforementioned configuration sources support jinja template rendering. When the first line of the provided configuration begins with **`## template: jinja`**, `cloud-init` will use jinja to render that file. Any `instance-data` variables are surfaced as jinja template variables.

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>`.

Sensitive data such as user passwords may be contained in `instance-data`. `Cloud-init` separates this sensitive data such that is it only readable by root. In the case that a non-root user attempts to read sensitive `instance-data`, they will receive redacted data or the same warnings and text that occur if a variable does not exist.

Example: Cloud config with `instance-data`

```
## 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
```

Example: User data script with `instance-data`

```
## 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 %}
...
```

Example: CLI discovery of `instance-data`

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

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

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

# Failure to reference valid dot-delimited key path on a known top-level key
$ cloud-init query v1.not_here
ERROR: instance-data 'v1' has no '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

# 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
```

Reference

Storage locations

- `/run/cloud-init/instance-data.json`: world-readable JSON containing standardised keys, sensitive keys redacted.
- `/run/cloud-init/instance-data-sensitive.json`: root-readable unredacted JSON blob.

`instance-data.json` 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` *Base configuration* 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

Standardised `cloud-init` metadata keys, these keys are guaranteed to exist on all cloud platforms. They will also retain their current behaviour and format, and will be carried forward even if `cloud-init` introduces a new version of standardised keys with `v2`.

To cut down on keystrokes on the command line, `cloud-init` also provides top-level key aliases for any standardised `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`.

`Cloud-init` also provides jinja-safe key aliases for any `instance-data` 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"] }}`.

Standardised `instance-data.json` v1 keys

v1._beta_keys

List of standardised 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 filed 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.

Example 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.

Example 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.

Example 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>)`

Example 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.

Example 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 centre in which the instance is deployed.

Example output:

- us-east-2

v1.availability_zone

The physical availability zone in which the instance is deployed.

Example 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 base 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": [
 "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",
```

(continues on next page)

(continued from previous page)


```
   "users-groups",
   "ssh"
  ],
  "datasource_list": [
   "Ec2",
   "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]\
↪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"
   ]
  ],
  "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": {
```


(continues on next page)

(continued from previous page)


```
    "accountId": "329910648901",
    "architecture": "x86_64",
    "availabilityZone": "us-east-1b",
    "billingProducts": null,
    "devpayProductCodes": null,
    "imageId": "ami-02e8aa396f8be3b6d",
    "instanceId": "i-0929128ff2f73a2f1",
    "instanceType": "t2.micro",
    "kernelId": null,
    "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",
    "AhQUgq0iPWqPTVnT96tZE6L1XjjLHQAAAAAAAA=="
   ],
   "rsa2048": [
    "MIAGCSqGSIb...",
    "REDACTED",
    "clYQvuE45xXm7Yreg3QtQbrP//owl1eZHj6s350AAAAAAAA="
   ],
   "signature": [
    "dA+QV+LLCWCRNddnrKleYmh2GvYo+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": {
```


(continues on next page)

(continued from previous page)


```
    "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",
       "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",
```


(continues on next page)

(continued from previous page)


```
    "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...",
  "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": ""
}
```

2.3.6 Vendor data

Overview

Vendor data is data provided by the entity that launches an instance (e.g., the cloud provider). This data can be used to customise the image to fit into the particular environment it is being run in.

Vendor data follows the same rules as user data, with the following caveats:

1. Users have ultimate control over vendor data. They can disable its execution or disable handling of specific parts of multi-part input.
2. By default it only runs on first boot.
3. Vendor data can be disabled by the user. If the use of vendor data is required for the instance to run, then vendor data should not be used.
4. User-supplied cloud-config is merged over cloud-config from vendor data.

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 multi-part 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 vendor data is safe, atomic, idempotent and does not put your users at risk.

Input formats

`Cloud-init` will download and cache to filesystem any vendor data that it finds. Vendor data is handled exactly like user data. This means that the vendor can supply multi-part input and have those parts acted on in the same way as with 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).
- The user can disable part handlers via the 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'}
```

Examples

You can find 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.

2.3.7 Security

Security policy

The following documents the upstream cloud-init security policy.

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.

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.

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.

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 to align with an upcoming planned release

2.3.8 Performance

The **`analyze`** subcommand was added to `cloud-init` to help analyze `cloud-init` boot time performance. It is loosely based on `systemd-analyze`, where there are four subcommands:

- **`blame`**
- **`show`**
- **`dump`**
- **`boot`**

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
```

Availability

The **`analyze`** subcommand is generally available across all distributions, with the exception of Gentoo and FreeBSD.

Subcommands

Blame

The **blame** subcommand matches **systemd-analyze blame** where it prints, in descending order, the units that took the longest time to run. This output is highly useful for examining where `cloud-init` is spending its time.

```
$ cloud-init analyze blame
```

Example output:

```
-- 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)
    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-rightscale_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)
```

(continues on next page)

(continued from previous page)

```
    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-disable-ec2-metadata)
    00.00000s (init-network/setup-datasource)

1 boot records analyzed
```

Show

The **show** subcommand 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 five *boot 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 of the total time.

The following is an abbreviated example of the **show** subcommand:

```
$ cloud-init analyze show
```

Example output:

```
-- 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
```

(continues on next page)

(continued from previous page)

```
|`->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** subcommand simply dumps the `cloud-init` logs that the **analyze** module is performing its 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
```

Example output:

```
[
{
  "description": "starting search for local datasources",
  "event_type": "start",
  "name": "init-local",
  "origin": "cloudinit",
  "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
```

(continues on next page)

(continued from previous page)


```
},
{
  "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",
  "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",
```


(continues on next page)

(continued from previous page)

```
  "result": "SUCCESS",
  "timestamp": 1567057594.982
}
]
```

Boot

The **boot** subcommand 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 performance issues related to `cloud-init` startup, and tracking regression.

```
$ cloud-init analyze boot
```

Example output:

```
-- 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 following command will gather the `UserspaceTimestamp`:

```
$ systemctl show -p UserspaceTimestampMonotonic
```

Example output:

```
UserspaceTimestampMonotonic=989279
```

The `UserspaceTimestamp` tracks when the init system starts, which is used as an indicator of the kernel finishing initialisation.

Running the following command will gather the `InactiveExitTimestamp`:

```
$ systemctl show cloud-init-local -p InactiveExitTimestampMonotonic
```

Example output:

```
InactiveExitTimestampMonotonic=4493126
```

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 – unlike when using systemd.

2.4 Reference

Our reference section contains support information for `cloud-init`. This includes details on the network requirements, API definitions, support matrices and so on.

2.4.1 Module reference

Deprecation schedule and versions

Keys may be documented as `deprecated`, `new`, or `changed`. This allows cloud-init to evolve as requirements change, and to adopt better practices without maintaining design decisions indefinitely.

Keys that have been marked as deprecated or changed may be removed or changed 5 years from the date of deprecation. For example, a key that is deprecated in version `22.1` (which is the first release in 2022) is scheduled to be removed in `27.1` (first release in 2027). Use of deprecated keys may cause warnings in the logs. In the case that a key's expected value changes, the key will be marked `changed` with a date. A 5 year timeline may also be expected for changed keys.

Ansible

Summary: Configure ansible for instance

This module provides `ansible` integration for augmenting cloud-init's configuration of the local node.

This module installs ansible during boot and then uses `ansible-pull` to run the playbook repository at the remote URL.

Internal name: `cc_ansible`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `ansible`

Config schema:
ansible: (object)

install_method: (`distro/pip`) The type of installation for ansible. It can be one of the following values:

- `distro`
- `pip`

run_user: (string) User to run module commands as. If install_method: pip, the pip install runs as this user as well.

ansible_config: (string) Sets the ANSIBLE_CONFIG environment variable. If set, overrides default config.

setup_controller: (object)

repositories: (array of object)

Each object in **repositories** list supports the following keys:

path: (string)

source: (string)

run_ansible: (array)

Each object in **run_ansible** list supports the following keys:

playbook_name: (string)

playbook_dir: (string)

become_password_file: (string)

connection_password_file: (string)

list_hosts: (boolean)

syntax_check: (boolean)

timeout: (number)

vault_id: (string)

vault_password_file: (string)

background: (number)

check: (boolean)

diff: (boolean)

module_path: (string)

poll: (number)

args: (string)

extra_vars: (string)

forks: (number)

inventory: (string)

scp_extra_args: (string)

sftp_extra_args: (string)

private_key: (string)

connection: (string)

module_name: (string)

sleep: (string)

tags: (string)

skip_tags: (string)

galaxy: (object)

actions: (array of array)

package_name: (string)

pull: (object)

accept_host_key: (boolean)

clean: (boolean)

full: (boolean)

diff: (boolean)

ssh_common_args: (string)

scp_extra_args: (string)

sftp_extra_args: (string)

private_key: (string)

checkout: (string)

module_path: (string)

timeout: (string)

url: (string)

connection: (string)

vault_id: (string)

vault_password_file: (string)

module_name: (string)

sleep: (string)

tags: (string)

skip_tags: (string)

playbook_name: (string)

Examples:

```
ansible:
  install_method: distro
  pull:
    url: "https://github.com/holmanb/vmboot.git"
    playbook_name: ubuntu.yml

# --- Example2 ---
ansible:
  package_name: ansible-core
```

(continues on next page)

(continued from previous page)

```
  install_method: pip
  pull:
    url: "https://github.com/holmanb/vmboot.git"
    playbook_name: ubuntu.yml
```

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

Activate only on keys: `apk_repos`

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: (null/object)

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
```

(continues on next page)

(continued from previous page)

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

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 object 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 object 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 127.0.01`

^.+$: (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.
- `append`: If `true`, append to sources file, otherwise overwrite it. Default: `true`.

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)

append: (boolean)

Examples:

```
apt:
  preserve_sources_list: false
  disable_suites:
    - $RELEASE-updates
    - backports
    - $RELEASE
    - mysuite
  primary:
    - arches:
        - amd64
```

(continues on next page)

(continued from previous page)

```
      - i386
      - default
    uri: 'http://us.archive.ubuntu.com/ubuntu'
    search:
      - 'http://cool.but-sometimes-unreachable.com/ubuntu'
      - 'http://us.archive.ubuntu.com/ubuntu'
    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-------
    source4:
        source: 'deb $MIRROR $RELEASE multiverse'
        append: false
        key: |
            ------BEGIN PGP PUBLIC KEY BLOCK-------
```

(continues on next page)

(continued from previous page)

```
            <key data>
            ------END PGP PUBLIC KEY BLOCK-------
```

Apt Pipelining

Summary: Configure apt pipelining

This module configures apt's `Acquite::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

Activate only on keys: `apt_pipelining`

Config schema:
apt_pipelining: (integer/boolean/`none`/`unchanged`/`os`)

Examples:

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

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

Activate only on keys: `bootcmd`

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 ]
```

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: (`enable-system/enable-user/disable-system/disable-user/enable/disable/user/system`)

Examples:

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

CA Certificates

Summary: Add ca certificates

This module adds CA certificates to the system's CA store and updates any related files using the appropriate OS-specific utility. The default CA certificates can be disabled/deleted from use by 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: Alpine Linux requires the ca-certificates package to be installed in order to provide the `update-ca-certificates` command.

Internal name: `cc_ca_certs`

Module frequency: once-per-instance

Supported distros: alpine, debian, rhel, ubuntu

Activate only on keys: `ca_certs`, `ca-certs`

Config schema:

ca_certs: (object)

remove-defaults: (boolean)

Deprecated in version 22.3. 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)

Deprecated in version 22.3. Use ``ca_certs`` instead.

remove-defaults: (boolean)

Deprecated in version 22.3. 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-----
```

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

Activate only on keys: `chef`

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 lof 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 th 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: (`packages`/`gems`/`omnibus`) 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
```

(continues on next page)

(continued from previous page)

```
  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
```

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

Activate only on keys: `disable_ec2_metadata`

Config schema:

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

Examples:

```
disable_ec2_metadata: true
```

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

Note: if a swap partition is being created via `disk_setup` then a `fs_entry` entry is also needed in order for mkswap to be run, otherwise when swap activation is later attempted it will fail.

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

Activate only on keys: `disk_setup`, `fs_setup`

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

layout: (`remove`/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). The partition type defaults to '83' (Linux partition), for other types of partition, such as Linux swap, the type must be passed as part of a list along with the size. 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 object 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 `<alias name>.<partition number>` format, the value of `partition` will be overwritten.

partition: (string/integer/`auto`/`any`/`none`) 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`, `filesystem` 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 filesystem that matches `filesystem` 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: (string/array of string) Optional options to pass to the filesystem creation command. Ignored if you using `cmd` directly.

cmd: (string/array 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
  swap_disk: /dev/sdc
disk_setup:
  my_alias:
    table_type: gpt
    layout: [50, 50]
    overwrite: true
  swap_disk:
    table_type: gpt
    layout: [[100, 82]]
    overwrite: true
  /dev/sdd:
    table_type: mbr
    layout: true
    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
- label: swap
  device: swap_disk.1
  filesystem: swap
- label: fs3
  device: /dev/sdd1
  filesystem: ext4
mounts:
- ["my_alias.1", "/mnt1"]
- ["my_alias.2", "/mnt2"]
- ["swap_disk.1", "none", "swap", "sw", "0", "0"]
- ["/dev/sdd1", "/mnt3"]
```

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)

Additionally, the `ubuntu-fan` package will be automatically installed if not present.

Internal name: `cc_fan`

Module frequency: once-per-instance

Supported distros: ubuntu

Activate only on keys: `fan`

Config schema:

fan: (object)

config: (string) The fan configuration to use as a single multi-line string

config_path: (string) The path to write the fan configuration to. Default: `/etc/network/fan`

Examples:

```
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
```

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

This message is written to the cloud-init log (usually /var/log/cloud-init.log) as well as stderr (which usually redirects to /var/log/cloud-init-output.log).

Upon exit, this module writes the system uptime, timestamp, and cloud-init version to `/var/lib/cloud/instance/boot-finished` independent of any user data specified for this module.

Internal name: `cc_final_message`

Module frequency: always

Supported distros: all

Config schema:

final_message: (string) The message to display at the end of the run

Examples:

```
final_message: |
  cloud-init has finished
  version: $version
  timestamp: $timestamp
  datasource: $datasource
  uptime: $uptime
```

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.

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 schema:

growpart: (object)

mode: (`auto`/`growpart`/`gpart`/`off`/`false`) The utility to use for resizing. Default: `auto`

Possible options:

- `auto` - Use any available utility
- `growpart` - Use growpart utility
- `gpart` - Use BSD gpart utility

- `off` - Take no action..

Changed in version 22.3. Specifying a boolean ``false`` value for ``mode`` is deprecated. Use ``off`` instead.

devices: (array of string) The devices to resize. Each entry can either be the path to the device's mountpoint in the filesystem or a path to the block device in '/dev'. Default: `[/]`

ignore_growroot_disabled: (boolean) If `true`, ignore the presence of `/etc/growroot-disabled`. If `false` and the file exists, then don't resize. Default: `false`

Examples:

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

# --- Example2 ---
growpart:
  mode: growpart
  devices:
    - "/"
    - "/dev/vdb1"
  ignore_growroot_disabled: true
```

Grub Dpkg

Summary: Configure grub debconf installation device

Configure which device is used as the target for grub installation. This module 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: once-per-instance

Supported distros: ubuntu, debian

Config schema:

grub_dpkg: (object)

enabled: (boolean) Whether to configure which device is used as the target for grub installation. Default: `true`

grub-pc/install_devices: (string) Device to use as target for grub installation. If unspecified, `grub-probe` of `/boot` will be used to find the device

grub-pc/install_devices_empty: (boolean/string) Sets values for `grub-pc/install_devices_empty`. If unspecified, will be set to `true` if `grub-pc/install_devices` is empty, otherwise `false`..

Changed in version 22.3. Use a boolean value instead.

grub-dpkg: (object)

Deprecated in version 22.2. Use ``grub_dpkg`` instead.

Examples:

```
grub_dpkg:
  enabled: true
  grub-pc/install_devices: /dev/sda
  grub-pc/install_devices_empty: false
```

Install Hotplug

Summary: Install hotplug udev rules 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 referesh 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"]
```

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, mariner, miraclelinux, openmandriva, photon, rhel, rocky, virtuozzo, opensuse, opensuse-leap, opensuse-microos, opensuse-tumbleweed, sle_hpc, sle-micro, sles

Activate only on keys: `keyboard`

Config schema:

keyboard: (object)

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

model: (string) Optional. Keyboard model. Corresponds to XKBMODEL. Default: `pc105`.

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
```

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: once-per-instance

Supported distros: all

Config schema:

ssh: (object)

emit_keys_to_console: (boolean) Set false to avoid printing SSH keys to system console. Default: `true`.

ssh_key_console_blacklist: (array of string) Avoid printing matching SSH key types to the system console.

ssh_fp_console_blacklist: (array of string) Avoid printing matching SSH fingerprints to the system console.

Examples:

```
# Do not print any SSH keys to system console
ssh:
  emit_keys_to_console: false

# --- Example2 ---
# Do not print certain ssh key types to console
ssh_key_console_blacklist: [dsa, ssh-dss]

# --- Example3 ---
# Do not print specific ssh key fingerprints to console
ssh_fp_console_blacklist:
- E25451E0221B5773DEBFF178ECDACB160995AA89
- FE76292D55E8B28EE6DB2B34B2D8A784F8C0AAB0
```

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 the *[client]* section in `/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.canoncial.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: once-per-instance

Supported distros: ubuntu

Activate only on keys: `landscape`

Config schema:

landscape: (object)

client: (object)

url: (string) The Landscape server URL to connect to. Default: `https://landscape.canonical.com/message-system`.

ping_url: (string) The URL to perform lightweight exchange initiation with. Default: `https://landscape.canonical.com/ping`.

data_path: (string) The directory to store data files in. Default: `/var/lib/land-scape/client/`.

log_level: (`debug/info/warning/error/critical`) The log level for the client. Default: `info`.

computer_title: (string) The title of this computer.

account_name: (string) The account this computer belongs to.

registration_key: (string) The account-wide key used for registering clients.

tags: (string) Comma separated list of tag names to be sent to the server.

http_proxy: (string) The URL of the HTTP proxy, if one is needed.

https_proxy: (string) The URL of the HTTPS proxy, if one is needed.

Examples:

```
# To discover additional supported client keys, run
# man landscape-config.
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"
        computer_title: "footitle"
        registration_key: "fookey"
        account_name: "fooaccount"

# --- Example2 ---
# Any keys below `client` are optional and the default values will
# be used.
landscape:
    client: {}
```

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
```

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 userspace utility is not installed, it will be installed. If network bridge configuration is provided, then lxd-bridge will be configured accordingly.

Internal name: `cc_lxd`

Module frequency: once-per-instance

Supported distros: ubuntu

Activate only on keys: `lxd`

Config schema:

lxd: (object)

init: (object) LXD init configuration values to provide to *lxd init –auto* command. Can not be combined with `lxd.preseed`.

network_address: (string) IP address for LXD to listen on

network_port: (integer) Network port to bind LXD to.

storage_backend: (`zfs/dir/lvm/btrfs`) Storage backend to use. Default: `dir`.

storage_create_device: (string) Setup device based storage using DEVICE

storage_create_loop: (integer) Setup loop based storage with SIZE in GB

storage_pool: (string) Name of storage pool to use or create

trust_password: (string) The password required to add new clients

bridge: (object) LXD bridge configuration provided to setup the host lxd bridge. Can not be combined with `lxd.preseed`.

mode: (`none/existing/new`) Whether to setup LXD bridge, use an existing bridge by `name` or create a new bridge. *none* will avoid bridge setup, *existing* will configure lxd to use the bring matching `name` and *new* will create a new bridge.

name: (string) Name of the LXD network bridge to attach or create. Default: `lxdbr0`.

mtu: (integer) Bridge MTU, defaults to LXD's default value

ipv4_address: (string) IPv4 address for the bridge. If set, `ipv4_netmask` key required.

ipv4_netmask: (integer) Prefix length for the `ipv4_address` key. Required when `ipv4_address` is set.

ipv4_dhcp_first: (string) First IPv4 address of the DHCP range for the network created. This value will combined with `ipv4_dhcp_last` key to set LXC `ipv4.dhcp.ranges`.

ipv4_dhcp_last: (string) Last IPv4 address of the DHCP range for the network created. This value will combined with `ipv4_dhcp_first` key to set LXC `ipv4.dhcp.ranges`.

ipv4_dhcp_leases: (integer) Number of DHCP leases to allocate within the range. Automatically calculated based on *ipv4_dhcp_first* and *ipv4_dchp_last* when unset.

ipv4_nat: (boolean) Set `true` to NAT the IPv4 traffic allowing for a routed IPv4 network. Default: `false`.

ipv6_address: (string) IPv6 address for the bridge (CIDR notation). When set, `ipv6_netmask` key is required. When absent, no IPv6 will be configured.

ipv6_netmask: (integer) Prefix length for `ipv6_address` provided. Required when `ipv6_address` is set.

ipv6_nat: (boolean) Whether to NAT. Default: `false`.

domain: (string) Domain to advertise to DHCP clients and use for DNS resolution.

preseed: (string) Opaque LXD preseed YAML config passed via stdin to the command: lxd init –preseed. See: https://linuxcontainers.org/lxd/docs/master/preseed/ or lxd init –dump for viable config. Can not be combined with either `lxd.init` or `lxd.bridge`.

Examples:

```
# Simplest working directory backed LXD configuration
lxd:
  init:
    storage_backend: dir

# --- Example2 ---
# LXD init showcasing cloud-init's LXD config options
lxd:
  init:
    network_address: 0.0.0.0
    network_port: 8443
    storage_backend: zfs
    storage_pool: datapool
    storage_create_loop: 10
  bridge:
    mode: new
    mtu: 1500
```

(continues on next page)

(continued from previous page)

```
    name: lxdbr0
    ipv4_address: 10.0.8.1
    ipv4_netmask: 24
    ipv4_dhcp_first: 10.0.8.2
    ipv4_dhcp_last: 10.0.8.3
    ipv4_dhcp_leases: 250
    ipv4_nat: true
    ipv6_address: fd98:9e0:3744::1
    ipv6_netmask: 64
    ipv6_nat: true
    domain: lxd

# --- Example3 ---
# For more complex non-interactive LXD configuration of networks,
# storage_pools, profiles, projects, clusters and core config,
# `lxd:preseed` config will be passed as stdin to the command:
#  lxd init --preseed
# See https://linuxcontainers.org/lxd/docs/master/preseed/ or
# run: lxd init --dump to see viable preseed YAML allowed.
#
# Preseed settings configuring the LXD daemon for HTTPS connections
# on 192.168.1.1 port 9999, a nested profile which allows for
# LXD nesting on containers and a limited project allowing for
# RBAC approach when defining behavior for sub projects.
lxd:
  preseed: |
    config:
      core.https_address: 192.168.1.1:9999
    networks:
      - config:
          ipv4.address: 10.42.42.1/24
          ipv4.nat: true
          ipv6.address: fd42:4242:4242:4242::1/64
          ipv6.nat: true
        description: ""
        name: lxdbr0
        type: bridge
        project: default
    storage_pools:
      - config:
          size: 5GiB
          source: /var/snap/lxd/common/lxd/disks/default.img
        description: ""
        name: default
        driver: zfs
    profiles:
      - config: {}
        description: Default LXD profile
        devices:
          eth0:
            name: eth0
            network: lxdbr0
```

(continues on next page)

(continued from previous page)

```
          type: nic
        root:
          path: /
          pool: default
          type: disk
      name: default
    - config: {}
      security.nesting: true
      devices:
        eth0:
          name: eth0
          network: lxdbr0
          type: nic
        root:
          path: /
          pool: default
          type: disk
      name: nested
  projects:
    - config:
        features.images: true
        features.networks: true
        features.profiles: true
        features.storage.volumes: true
      description: Default LXD project
      name: default
    - config:
        features.images: false
        features.networks: true
        features.profiles: false
        features.storage.volumes: false
      description: Limited Access LXD project
      name: limited
```

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: once-per-instance

Supported distros: all

Activate only on keys: `mcollective`

Config schema:

mcollective: (object)

conf: (object)

public-cert: (string) Optional value of server public certificate which will be written to `/etc/mcollective/ssl/server-public.pem`

private-cert: (string) Optional value of server private certificate which will be written to `/etc/mcollective/ssl/server-private.pem`

^.+$: (boolean/integer/string) Optional config key: value pairs which will be appended to `/etc/mcollective/server.cfg`.

Examples:

```
# Provide server private and public key and provide the following
# config settings in /etc/mcollective/server.cfg:
# loglevel: debug
# plugin.stomp.host: dbhost

# WARNING WARNING WARNING
# The ec2 metadata service is a network service, and thus is
# readable by non-root users on the system
# (ie: 'ec2metadata --user-data')
# If you want security for this, please use include-once + SSL urls
mcollective:
  conf:
    loglevel: debug
    plugin.stomp.host: dbhost
    public-cert: |
        -------BEGIN CERTIFICATE--------
        <cert data>
        -------END CERTIFICATE--------
    private-cert: |
        -------BEGIN CERTIFICATE--------
        <cert data>
        -------END CERTIFICATE--------
```

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 schema:

migrate: (boolean) Whether to migrate legacy cloud-init semaphores to new format. Default: `true`

Examples:

```
# Do not migrate cloud-init semaphores
migrate: false
```

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.

Any mounts that do not appear to either an attached block device or network resource will be skipped with a log like "Ignoring nonexistent mount ...".

Cloud-init will attempt to add the following mount directives if available and unconfigured in */etc/fstab*:

```
mounts:
    - ["ephemeral0", "/mnt", "auto","defaults,nofail,x-systemd.requires=cloud-init.
↪service", "0", "2"]
    - ["swap", "none", "swap", "sw", "0", "0"]
```

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. `[ swap ]` or `[ swap, 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,nofail,x-systemd.requires=cloud-init.
↪service", "0", "2"]
```

Non-systemd init systems will vary in `mount_default_fields`.

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: once-per-instance

Supported distros: all

Config schema:

mounts: (array of array) List of lists. Each inner list entry is a list of `/etc/fstab` mount declarations of the format: [fs_spec, fs_file, fs_vfstype, fs_mntops, fs-freq, fs_passno]. A mount declaration with less than 6 items will get remaining values from `mount_default_fields`. A mount declaration with only *fs_spec* and no *fs_file* mountpoint will be skipped.

mount_default_fields: (array of (string/null)) Default mount configuration for any mount entry with less than 6 options provided. When specified, 6 items are required and represent `/etc/fstab` entries. Default: `defaults,nofail,x-systemd.requires=cloud-init.service,_netdev`

swap: (object)

filename: (string) Path to the swap file to create

size: (auto/integer/string) The size in bytes of the swap file, 'auto' or a human-readable size abbreviation of the format <float_size><units> where units are one of B, K, M, G or T. **WARNING: Attempts to use IEC prefixes in your configuration prior to cloud-init version 23.1 will result in unexpected behavior. SI prefixes names (KB, MB) are required on pre-23.1 cloud-init, however IEC values are used. In summary, assume 1KB == 1024B, not 1000B**

maxsize: (integer/string) The maxsize in bytes of the swap file

Examples:

```
# Mount ephemeral0 with "noexec" flag, /dev/sdc with mount_default_fields,
# and /dev/xvdh with custom fs_passno "0" to avoid fsck on the mount.
# Also provide an automatically sized swap with a max size of 10485760
# bytes.
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: /my/swapfile
    size: auto
    maxsize: 10485760

# --- Example2 ---
# Create a 2 GB swap file at /swapfile using human-readable values
swap:
    filename: /swapfile
    size: 2G
    maxsize: 2G
```

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, cos, debian, eurolinux, fedora, freebsd, mariner, miraclelinux, openbsd, openEuler, OpenCloudOS, openmandriva, opensuse, opensuse-microos, opensuse-tumbleweed, opensuse-leap, photon, rhel, rocky, sle_hpc, sle-micro, sles, TencentOS, ubuntu, virtuozzo

Activate only on keys: `ntp`

Config schema:

ntp: (null/object)

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, openntpd, 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
  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 %}
```

(continues on next page)

(continued from previous page)

```
        {% 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
```

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.

Internal name: `cc_package_update_upgrade_install`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `apt_update`, `package_update`, `apt_upgrade`, `package_upgrade`, `packages`

Config schema:

packages: (array of (array of string/string)) A list of packages to install. 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.

package_update: (boolean) Set `true` to update packages. Happens before upgrade or install. Default: `false`

package_upgrade: (boolean) Set `true` to upgrade packages. Happens before install. Default: `false`

package_reboot_if_required: (boolean) Set `true` to reboot the system if required by presence of */var/run/reboot-required*. Default: `false`

apt_update: (boolean) Default: `false`.

Deprecated in version 22.2. Use ``package_update`` instead.

apt_upgrade: (boolean) Default: `false`.

Deprecated in version 22.2. Use ``package_upgrade`` instead.

apt_reboot_if_required: (boolean) Default: `false`.

Deprecated in version 22.2. Use ``package_reboot_if_required`` instead.

Examples:

```
packages:
  - pwgen
  - pastebinit
  - [libpython3.8, 3.8.10-0ubuntu1~20.04.2]
package_update: true
package_upgrade: true
package_reboot_if_required: true
```

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`
- `fdqn`

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: once-per-instance

Supported distros: all

Activate only on keys: `phone_home`

Config schema:

phone_home: (object)

url: (string) The URL to send the phone home data to.

post: (`all`/array) A list of keys to post or `all`. Default: `all`

tries: (integer) The number of times to try sending the phone home data. Default: `10`

Examples:

```
phone_home:
    url: http://example.com/$INSTANCE_ID/
    post: all

# --- Example2 ---
phone_home:
    url: http://example.com/$INSTANCE_ID/
```

(continues on next page)

(continued from previous page)

```
    post:
        - pub_key_dsa
        - pub_key_rsa
        - pub_key_ecdsa
        - pub_key_ed25519
        - instance_id
        - hostname
        - fqdn
    tries: 5
```

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.

Using this module ensures that cloud-init is entirely finished with modules that would be executed.

An example to distinguish delay from timeout:

If you delay 5 (5 minutes) and have a timeout of 120 (2 minutes), then the max time until shutdown will be 7 minutes, though it could be as soon as 5 minutes. Cloud-init will invoke 'shutdown +5' after the process finishes, or when 'timeout' seconds have elapsed.

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: once-per-instance

Supported distros: all

Activate only on keys: `power_state`

Config schema:

power_state: (object)

delay: (integer/string/`now`) Time in minutes to delay after cloud-init has finished. Can be `now` or an integer specifying the number of minutes to delay. Default: `now`.

Changed in version 22.3. Use of type string for this value is deprecated. Use ``now`` or integer type.

mode: (`poweroff`/`reboot`/`halt`) Must be one of `poweroff`, `halt`, or `reboot`.

message: (string) Optional message to display to the user when the system is powering off or rebooting.

timeout: (integer) Time in seconds to wait for the cloud-init process to finish before executing shutdown. Default: `30`

condition: (string/boolean/array) Apply state change only if condition is met. May be boolean true (always met), false (never met), or a command string or list to be executed. For command formatting, see the documentation for `cc_runcmd`. If exit code is 0, condition is met, otherwise not. Default: `true`

Examples:

```
power_state:
    delay: now
    mode: poweroff
    message: Powering off
    timeout: 2
    condition: true

# --- Example2 ---
power_state:
    delay: 30
    mode: reboot
    message: Rebooting machine
    condition: test -f /var/tmp/reboot_me
```

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 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.

Internal name: `cc_puppet`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `puppet`

Config schema:

puppet: (object)

install: (boolean) Whether or not to install puppet. Setting to `false` will result in an error if puppet is not already present on the system. Default: `true`

version: (string) Optional version to pass to the installer script or package manager. If unset, the latest version from the repos will be installed.

install_type: (`packages`/`aio`) Valid values are `packages` and `aio`. Agent packages from the puppetlabs repositories can be installed by setting `aio`. Based on this setting, the default config/SSL/CSR paths will be adjusted accordingly. Default: `packages`

collection: (string) Puppet collection to install if `install_type` is `aio`. This can be set to one of `puppet` (rolling release), `puppet6`, `puppet7` (or their nightly counterparts) in order to install specific release streams.

aio_install_url: (string) If `install_type` is `aio`, change the url of the install script.

cleanup: (boolean) Whether to remove the puppetlabs repo after installation if `install_type` is `aio` Default: `true`

conf_file: (string) The path to the puppet config file. Default depends on `install_type`

ssl_dir: (string) The path to the puppet SSL directory. Default depends on `install_type`

csr_attributes_path: (string) The path to the puppet csr attributes file. Default depends on `install_type`

package_name: (string) Name of the package to install if `install_type` is `packages`. Default: `puppet`

exec: (boolean) Whether or not to run puppet after configuration finishes. 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). Default: `false`

exec_args: (array of string) A list of arguments to pass to 'puppet agent' if 'exec' is true Default: `['--test']`

start_service: (boolean) 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`

conf: (object) Every key present in the conf object will be added 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 configuration is specified as a dictionary containing high-level `<section>` keys and lists of `<key>=<value>` pairs within each section. The `certname` key supports string substitutions for `%i` and `%f`, corresponding to the instance id and fqdn of the machine respectively.

`ca_cert` is a special case. It won't be added to puppet.conf. It holds the puppetserver certificate in pem format. It should be a multi-line string (using the | yaml notation for multi-line strings).

main: (object)

server: (object)

agent: (object)

user: (object)

ca_cert: (string)

csr_attributes: (object) 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

custom_attributes: (object)

extension_requests: (object)

Examples:

```
puppet:
    install: true
    version: "7.7.0"
    install_type: "aio"
    collection: "puppet7"
    aio_install_url: 'https://git.io/JBhoQ'
    cleanup: true
    conf_file: "/etc/puppet/puppet.conf"
    ssl_dir: "/var/lib/puppet/ssl"
    csr_attributes_path: "/etc/puppet/csr_attributes.yaml"
    exec: true
    exec_args: ['--test']
    conf:
```

(continues on next page)

(continued from previous page)

```
        agent:
            server: "puppetserver.example.org"
            certname: "%i.%f"
        ca_cert: |
            -----BEGIN CERTIFICATE-----
            MIICCTCCAXKgAwIBAgIBATANBgkqhkiG9w0BAQUFADANMQswCQYDVQQDDAJjYTAe
            Fw0xMDAyMTUxNzI5MjFaFw0xNTAyMTQxNzI5MjFaMA0xCzAJBgNVBAMMAmNhMIGf
            MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCu7Q40sm47/E1Pf+r8AYb/V/FWGPgc
            b014OmNoX7dgCxTDvps/h8Vw555PdAFsW5+QhsGr31IJNI3kSYprFQcYf7A8tNWu
            1MASW2CfaEiOEi9F1R3R4Qlz4ix+iNoHiUDTjazw/tZwEdxaQXQVLwgTGRwVa+aA
            qbutJKi93MILLwIDAQABo3kwdzA4BglghkgBhvhCAQ0EKxYpUHVwcGV0IFJ1Ynkv
            T3BlblNTTCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwDwYDVR0TAQH/BAUwAwEB/zAd
            BgNVHQ4EFgQUu4+jHB+GYE5Vxo+ol1OAhevspjAwCwYDVR0PBAQDAgEGMA0GCSqG
            SIb3DQEBBQUAA4GBAH/rxlUIjwNb3n7TXJcDJ6MMHUlwjr03BDJXKb34Ulndkpaf
            +GAlzPXWa7bO908M9I8RnPfvtKnteLbvgTK+h+zX1XCty+S2EQWk29i2AdoqOTxb
            hppiGMp0tT5Havu4aceCXiy2crVcudj3NFciy8X66SoECemW9UYDCb9T5D0d
            -----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

# --- Example2 ---
puppet:
    install_type: "packages"
    package_name: "puppet"
    exec: false
```

Resizefs

Summary: Resize filesystem

Resize a filesystem to use all avaliable 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 `noblock`. 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. `noblock` will resize in the background. Default: `true`

Examples:

```
resize_rootfs: false  # disable root filesystem resize operation
# --- Example2 ---
resize_rootfs: noblock  # runs resize operation in the background
```

Resolv Conf

Summary: Configure resolv.conf

Unless manually editing `/etc/resolv.conf` is the correct way to manage nameserver information on your operating system, you do not want to use this module. Many distros have moved away from manually editing `resolv.conf` so please verify that this is the preferred nameserver management method for your distro before using this module.

Note that using *Network configuration* is preferred, rather than using this module, when possible.

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.

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 *Networking config Version 2* format. For those that wish to have different settings, use this module.

In order for the `resolv_conf` section to be applied, `manage_resolv_conf` must be set `true`.

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

Internal name: `cc_resolv_conf`

Module frequency: once-per-instance

Supported distros: alpine, fedora, mariner, opensuse, opensuse-leap, opensuse-microos, opensuse-tumbleweed, photon, rhel, sle_hpc, sle-micro, sles

Activate only on keys: `manage_resolv_conf`

Config schema:

manage_resolv_conf: (boolean) Whether to manage the resolv.conf file. `resolv_conf` block will be ignored unless this is set to `true`. Default: `false`

resolv_conf: (object)

nameservers: (array) A list of nameservers to use to be added as `nameserver` lines

searchdomains: (array) A list of domains to be added `search` line

domain: (string) The domain to be added as `domain` line

sortlist: (array) A list of IP addresses to be added to `sortlist` line

options: (object) Key/value pairs of options to go under `options` heading. A unary option should be specified as `true`

Examples:

```
manage_resolv_conf: true
resolv_conf:
  nameservers:
    - 8.8.8.8
    - 8.8.4.4
```

(continues on next page)

(continued from previous page)

```
searchdomains:
  - foo.example.com
  - bar.example.com
domain: example.com
sortlist:
  - 10.0.0.1/255
  - 10.0.0.2
options:
  rotate: true
  timeout: 1
```

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
- alter the rhsm_baseurl and server-hostname in `/etc/rhsm/rhs.conf`.

Internal name: `cc_rh_subscription`

Module frequency: once-per-instance

Supported distros: fedora, rhel

Activate only on keys: `rh_subscription`

Config schema:

rh_subscription: (object)

username: (string) The username to use. Must be used with password. Should not be used with `activation-key` or `org`

password: (string) The password to use. Must be used with username. Should not be used with `activation-key` or `org`

activation-key: (string) The activation key to use. Must be used with `org`. Should not be used with `username` or `password`

org: (integer) The organization number to use. Must be used with `activation-key`. Should not be used with `username` or `password`

auto-attach: (boolean) Whether to attach subscriptions automatically

service-level: (string) The service level to use when subscribing to RH repositories. `auto-attach` must be true for this to be used

add-pool: (array of string) A list of pools ids add to the subscription

enable-repo: (array of string) A list of repositories to enable

disable-repo: (array of string) A list of repositories to disable

rhsm-baseurl: (string) Sets the baseurl in `/etc/rhsm/rhsm.conf`

server-hostname: (string) Sets the serverurl in `/etc/rhsm/rhsm.conf`

Examples:

```
rh_subscription:
    username: joe@foo.bar
    ## Quote your password if it has symbols to be safe
    password: '1234abcd'

# --- Example2 ---
rh_subscription:
    activation-key: foobar
    org: 12345

# --- Example3 ---
rh_subscription:
    activation-key: foobar
    org: 12345
    auto-attach: true
    service-level: self-support
    add-pool:
      - 1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a
      - 2b2b2b2b2b2b2b2b2b2b2b2b2b2b2b2b
    enable-repo:
      - repo-id-to-enable
      - other-repo-id-to-enable
    disable-repo:
      - repo-id-to-disable
      - other-repo-id-to-disable
    # Alter the baseurl in /etc/rhsm/rhsm.conf
    rhsm-baseurl: http://url
    # Alter the server hostname in /etc/rhsm/rhsm.conf
    server-hostname: foo.bar.com
```

RightScale Userdata

Summary: Support rightscale configuration hooks

This module adds support for RightScale configuration hooks to cloud-init. RightScale adds an 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

---

Raw user data schema:

```
CLOUD_INIT_REMOTE_HOOK=<url>
```

Internal name: `cc_rightscale_userdata`

Module frequency: once-per-instance

Supported distros: all

Rsyslog

Summary: Configure system logging via rsyslog

This module configures remote system logging using rsyslog.

Configuration for remote servers can be specified in `configs`, but for convenience it can be specified as key value pairs in `remotes`.

Internal name: `cc_rsyslog`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `rsyslog`

Config schema:

rsyslog: (object)

config_dir: (string) The directory where rsyslog configuration files will be written. Default: `/etc/rsyslog.d`

config_filename: (string) The name of the rsyslog configuration file. Default: `20-cloud-config.conf`

configs: (array of (string/object)) Each entry in `configs` is either a string or an object. Each config entry contains a configuration string and a file to write it to. For config entries that are an object, `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`.

Each object in **configs** list supports the following keys:

filename: (string)

content: (string)

remotes: (object) 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>`.

service_reload_command: (`auto`/array) The command to use to reload the rsyslog service after the config has been updated. 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]`).

Examples:

```
rsyslog:
    remotes:
        maas: 192.168.1.1
        juju: 10.0.4.1
    service_reload_command: auto

# --- Example2 ---
rsyslog:
    config_dir: /opt/etc/rsyslog.d
    config_filename: 99-late-cloud-config.conf
    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
        juju: 10.0.4.1
    service_reload_command: [your, syslog, restart, command]
```

Runcmd

Summary: Run arbitrary commands

Run arbitrary commands at a rc.local like time-frame with output to the console. Each item can be either a list or a string. The item type affects how it is executed:

- If the item is a string, it will be interpreted by `sh`.
- If the item is a list, the items will be executed as if passed to execve(3) (with the first arg as the command).

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

Activate only on keys: `runcmd`

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 ]
```

Salt Minion

Summary: Setup 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`.

Salt keys can be manually generated by: `salt-key --gen-keys=GEN_KEYS`, where `GEN_KEYS` is the name of the keypair, e.g. 'minion'. The keypair will be copied to `/etc/salt/pki` on the minion instance.

Internal name: `cc_salt_minion`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `salt_minion`

Config schema:

salt_minion: (object)

pkg_name: (string) Package name to install. Default: `salt-minion`

service_name: (string) Service name to enable. Default: `salt-minion`

config_dir: (string) Directory to write config files to. Default: `/etc/salt`

conf: (object) Configuration to be written to *config_dir*/minion

grains: (object) Configuration to be written to *config_dir*/grains

public_key: (string) Public key to be used by the salt minion

private_key: (string) Private key to be used by salt minion

pki_dir: (string) Directory to write key files. Default: *config_dir*/pki/minion

Examples:

```
salt_minion:
    pkg_name: salt-minion
    service_name: salt-minion
    config_dir: /etc/salt
    conf:
        master: salt.example.com
    grains:
        role:
            - web
    public_key: |
```

(continues on next page)

(continued from previous page)

```
    ------BEGIN PUBLIC KEY-------
    <key data>
    ------END PUBLIC KEY-------
private_key: |
    ------BEGIN PRIVATE KEY------
    <key data>
    ------END PRIVATE KEY-------
pki_dir: /etc/salt/pki/minion
```

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

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: once-per-instance

Supported distros: all

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: once

Supported distros: all

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: once-per-instance

Supported distros: all

Scripts Vendor

Summary: Run vendor scripts

On select Datasources, vendors have a channel for the consumption of all supported user data types via a special channel called vendor data. 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. This module allows control over the execution of vendor data.

Internal name: `cc_scripts_vendor`

Module frequency: once-per-instance

Supported distros: all

Config schema:

vendor_data: (object)

enabled: (boolean) Whether vendor data is enabled or not. Default: `true`.

Deprecated in version 22.3. Use of type string for this value is deprecated. Use a boolean instead.

prefix: (string/array of (string/integer)) The command to run before any vendor scripts. Its primary use case is for profiling a script, not to prevent its run

Examples:

```
vendor_data:
  enabled: true
  prefix: /usr/bin/ltrace

# --- Example2 ---
vendor_data:
  enabled: true
  prefix: [timeout, 30]

# --- Example3 ---
# Vendor data will not be processed
vendor_data:
  enabled: false
```

Seed Random

Summary: Provide random seed data

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

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.

Internal name: `cc_seed_random`

Module frequency: once-per-instance

Supported distros: all

Config schema:

random_seed: (object)

file: (string) File to write random data to. Default: `/dev/urandom`

data: (string) This data will be written to `file` before data from the datasource. When using a multiline value or specifying binary data, be sure to follow yaml syntax and use the `|` and `!binary` yaml format specifiers when appropriate

encoding: (`raw/base64/b64/gzip/gz`) Used to decode `data` provided. Allowed values are `raw`, `base64`, `b64`, `gzip`, or `gz`. Default: `raw`

command: (array of string) Execute this command to seed random. The command will have RANDOM_SEED_FILE in its environment set to the value of `file` above.

command_required: (boolean) If true, and `command` is not available to be run then an exception is raised and cloud-init will record failure. Otherwise, only debug error is mentioned. Default: `false`

Examples:

```
random_seed:
  file: /dev/urandom
  data: my random string
  encoding: raw
  command: ['sh', '-c', 'dd if=/dev/urandom of=$RANDOM_SEED_FILE']
  command_required: true

# --- Example2 ---
# 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.polinate.server"]
  command_required: true
```

Set Hostname

Summary: Set hostname and FQDN

This module handles setting the system hostname and fully qualified domain name (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, the `prefer_fqdn_over_hostname` will force the use of FQDN in all distros when true, and when false it will force the short hostname. Otherwise, the hostname to use is distro-dependent.

Note: cloud-init performs no hostname input validation before sending the hostname to distro-specific tools, and most tools will not accept a trailing dot on the FQDN.

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: once-per-instance

Supported distros: all

Config schema:

preserve_hostname: (boolean) If true, the hostname will not be changed. Default: `false`

hostname: (string) The hostname to set

fqdn: (string) The fully qualified domain name to set

prefer_fqdn_over_hostname: (boolean) If true, the fqdn will be used if it is set. If false, the hostname will be used. If unset, the result is distro-dependent

Examples:

```
preserve_hostname: true
# --- Example2 ---
hostname: myhost
fqdn: myhost.example.com
prefer_fqdn_over_hostname: true
```

Set Passwords

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

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.

The `chpasswd` config key accepts a dictionary containing either or both of `users` and `expire`. The `users` key is used to assign a password to a corresponding pre-existing user. The `expire` key is used to set whether to expire all user passwords specified by this module, such that a password will need to be reset on the user's next login.

Note: Prior to cloud-init 22.3, the `expire` key only applies to plain text (including `RANDOM`) passwords. Post 22.3, the `expire` key applies to both plain text and hashed passwords.

`password` config key is used to set the default user's password. It is ignored if the `chpasswd` `users` is used. Note: the `list` keyword is deprecated in favor of `users`.

Internal name: `cc_set_passwords`

Module frequency: once-per-instance

Supported distros: all

Config schema:

ssh_pwauth: (boolean/string) Sets whether or not to accept password authentication. `true` will enable password auth. `false` will disable. Default is to leave the value unchanged. In order for this config to be applied, SSH may need to be restarted. On systemd systems, this restart will only happen if the SSH service has already been started. On non-systemd systems, a restart will be attempted regardless of the service state..

Changed in version 22.3. Use of non-boolean values for this field is deprecated.

chpasswd: (object)

expire: (boolean) Whether to expire all user passwords such that a password will need to be reset on the user's next login. Default: `true`

users: (array of object) This key represents a list of existing users to set passwords for. Each item under users contains the following required keys: `name` and `password` or in the case of a randomly generated password, `name` and `type`. The `type` key has a default value of `hash`, and may alternatively be set to `text` or `RANDOM`.

list: (string/array) List of `username:password` pairs. 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.

Deprecated in version 22.2. Use ``users`` instead.

password: (string) Set the default user's password. Ignored if `chpasswd` `list` is used

Examples:

```
# Set a default password that would need to be changed
# at first login
ssh_pwauth: true
password: password1

# --- Example2 ---
# Disable ssh password authentication
# Don't require users to change their passwords on next login
# Set the password for user1 to be 'password1' (OS does hashing)
# Set the password for user2 to a pre-hashed password
# Set the password for user3 to be a randomly generated password,
#   which will be written to the system console
ssh_pwauth: false
chpasswd:
  expire: false
  users:
    - name: user1
```

(continues on next page)

(continued from previous page)

```
    password: password1
    type: text
  - name: user2
    password: $6$rounds=4096$5DJ8a9WMTEzIo5J4
↪$Yms6imfeBvf3Yfu84mQBerh18l7OR1Wm1BJXZqFSpJ6BVas0AYJqIjP7czkOaAZHZi1kxQ5Y1IhgWN8K9NgxR1
  - name: user3
    type: RANDOM
```

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.

Internal name: `cc_snap`

Module frequency: once-per-instance

Supported distros: ubuntu

Activate only on keys: `snap`

Config schema:

snap: (object)

assertions: (object/array of string) Properly-signed snap assertions which will run before and snap `commands`.

commands: (object/array of (string/array of string)) Snap commands to run on the target system

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 ---
# 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'

# --- Example3 ---
# You can use a list of commands
snap:
  commands:
    - ['install', 'vlc']
    - ['snap', 'install', 'vlc']
    - snap install vlc
    - 'snap install vlc'

# --- Example4 ---
# You can use a list of assertions
snap:
  assertions:
    - signed_assertion_blob_here
    - |
      signed_assertion_blob_here
```

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: once-per-instance

Supported distros: rhel, fedora

Activate only on keys: `spacewalk`

Config schema:

spacewalk: (object)

server: (string) The Spacewalk server to use

proxy: (string) The proxy to use when connecting to Spacewalk

activation_key: (string) The activation key to use when registering with Spacewalk

Examples:

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

SSH

Summary: Configure SSH and SSH keys

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`.

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 sigonly 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`.

Host keys can be added using the `ssh_keys` configuration key.

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.

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

---

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:

- dsa
- ecdsa
- ed25519
- rsa

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

- ecdsa-sk
- ed25519-sk

Internal name: `cc_ssh`

Module frequency: once-per-instance

Supported distros: all

Config schema:

ssh_keys: (object) A dictionary entries for the public and private host 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>`. Not all key types have to be specified, ones left unspecified will not be used. If this config option is used, then separate keys will not be automatically generated. In order to specify multiline private host keys and certificates, use yaml multiline syntax.

<key_type>: (string)

ssh_authorized_keys: (array of string) The SSH public keys to add `.ssh/authorized_keys` in the default user's home directory

ssh_deletekeys: (boolean) Remove host SSH keys. This prevents re-use of a private host key from an image with default host SSH keys. Default: `true`

ssh_genkeytypes: (array of string) The SSH key types to generate. Default: `[rsa, dsa, ecdsa, ed25519]`

disable_root: (boolean) Disable root login. Default: `true`

disable_root_opts: (string) Disable root login options. If `disable_root_opts` is specified and contains the string `$USER`, it will be replaced with the username of the default user. Default: `no-port-forwarding,no-agent-forwarding,no-X11-forwarding,command="echo 'Please login as the user \"$USER\" rather than the user \"$DISABLE_USER\".';echo;sleep 10;exit 142"`

allow_public_ssh_keys: (boolean) If `true`, will import the public SSH keys from the datasource's metadata to the user's `.ssh/authorized_keys` file. Default: `true`

ssh_quiet_keygen: (boolean) If `true`, will suppress the output of key generation to the console. Default: `false`

ssh_publish_hostkeys: (object)

enabled: (boolean) If true, will read host keys from `/etc/ssh/*.pub` and publish them to the datasource (if supported). Default: `true`

blacklist: (array of string) The SSH key types to ignore when publishing. Default: `[dsa]`

Examples:

```
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 AAAAIHNzaC1lZDI1NTE5LWNlcnQt ...
  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 AAAAIHNzaC1lZDI1NTE5LWNlcnQt ...
ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUU ...
  - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA3I7VUf2l5gSn5uavROsc5HRDpZ ...
```

(continues on next page)

(continued from previous page)

```
ssh_deletekeys: true
ssh_genkeytypes: [rsa, dsa, ecdsa, ed25519]
disable_root: true
disable_root_opts: no-port-forwarding,no-agent-forwarding,no-X11-forwarding
allow_public_ssh_keys: true
ssh_quiet_keygen: true
ssh_publish_hostkeys:
  enabled: true
  blacklist: [dsa]
```

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: once-per-instance

Supported distros: all

Config schema:

no_ssh_fingerprints: (boolean) If true, SSH fingerprints will not be written. Default: `false`

authkey_hash: (string) The hash type to use when generating SSH fingerprints. Default: `sha256`

Examples:

```
no_ssh_fingerprints: true
# --- Example2 ---
authkey_hash: sha512
```

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: once-per-instance

Supported distros: ubuntu, debian, cos

Config schema:

ssh_import_id: (array of string)

Examples:

```
ssh_import_id:
 - user
```

(continues on next page)

(continued from previous page)

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

Timezone

Summary: Set the system timezone

Sets the system timezone based on the value provided.

Internal name: `cc_timezone`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `timezone`

Config schema:
timezone: (string) The timezone to use as represented in /usr/share/zoneinfo

Examples:

```
timezone: US/Eastern
```

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, only named services will be activated. Whereas if the 'enable' list is not present, the contract's default services will be enabled.

On Pro instances, when `ubuntu_advantage` config is provided to cloud-init, Pro's auto-attach feature will be disabled and cloud-init will perform the Pro auto-attach ignoring the `token` key. The `enable` and `enable_beta` values will strictly determine what services will be enabled, ignoring contract defaults.

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

Activate only on keys: `ubuntu_advantage`, `ubuntu-advantage`

Config schema:
ubuntu_advantage: (object)

> **enable:** (array of string) Optional list of ubuntu-advantage services to enable. Any of: cc-eal, cis, esm-infra, fips, fips-updates, livepatch. By default, a given contract token will automatically enable a number of services, use this list to supplement which services should additionally be enabled. Any service unavailable on a given Ubuntu release or unentitled in a given contract will remain disabled. In Ubuntu Pro instances, if this list is given, then only those services will be enabled, ignoring contract defaults. Passing beta services here will cause an error.

enable_beta: (array of string) Optional list of ubuntu-advantage beta services to enable. By default, a given contract token will automatically enable a number of services, use this list to supplement which services should additionally be enabled. Any service unavailable on a given Ubuntu release or unentitled in a given contract will remain disabled. In Ubuntu Pro instances, if this list is given, then only those services will be enabled, ignoring contract defaults.

token: (string) Contract token obtained from https://ubuntu.com/advantage to attach. Required for non-Pro instances.

features: (object) Ubuntu Advantage features.

disable_auto_attach: (boolean) Optional boolean for controlling if ua-auto-attach.service (in Ubuntu Pro instances) will be attempted each boot. Default: `false`

config: (object) Configuration settings or override Ubuntu Advantage config.

http_proxy: (string/null) Ubuntu Advantage HTTP Proxy URL or null to unset.

https_proxy: (string/null) Ubuntu Advantage HTTPS Proxy URL or null to unset.

global_apt_http_proxy: (string/null) HTTP Proxy URL used for all APT repositories on a system or null to unset. Stored at `/etc/apt/apt.conf.d/90ubuntu-advantage-aptproxy`

global_apt_https_proxy: (string/null) HTTPS Proxy URL used for all APT repositories on a system or null to unset. Stored at `/etc/apt/apt.conf.d/90ubuntu-advantage-aptproxy`

ua_apt_http_proxy: (string/null) HTTP Proxy URL used only for Ubuntu Advantage APT repositories or null to unset. Stored at `/etc/apt/apt.conf.d/90ubuntu-advantage-aptproxy`

ua_apt_https_proxy: (string/null) HTTPS Proxy URL used only for Ubuntu Advantage APT repositories or null to unset. Stored at `/etc/apt/apt.conf.d/90ubuntu-advantage-aptproxy`

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:
```

(continues on next page)

(continued from previous page)

```
  mode: reboot
ubuntu_advantage:
  token: <ua_contract_token>
  enable:
  - fips

# --- Example4 ---
# Set a http(s) proxy before attaching the machine to an
# Ubuntu Advantage support contract and enabling the FIPS service.
ubuntu_advantage:
  token: <ua_contract_token>
  config:
    http_proxy: 'http://some-proxy:8088'
    https_proxy: 'https://some-proxy:8088'
    global_apt_https_proxy: 'https://some-global-apt-proxy:8088/'
    global_apt_http_proxy: 'http://some-global-apt-proxy:8088/'
    ua_apt_http_proxy: 'http://10.0.10.10:3128'
    ua_apt_https_proxy: 'https://10.0.10.10:3128'
  enable:
  - fips

# --- Example5 ---
# On Ubuntu PRO instances, auto-attach but enable no PRO services.
ubuntu_advantage:
  enable: []
  enable_beta: []

# --- Example6 ---
# Enable esm and beta realtime-kernel services in Ubuntu Pro instances.
ubuntu_advantage:
  enable:
  - esm
  enable_beta:
  - realtime-kernel

# --- Example7 ---
# Disable auto-attach in Ubuntu Pro instances.
ubuntu_advantage:
  features:
    disable_auto_attach: True
```

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

Activate only on keys: `drivers`

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
```

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`, 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`, 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

Activate only on keys: `manage_etc_hosts`

Config schema:

manage_etc_hosts: (`true/false/localhost/template`) Whether to manage `/etc/hosts` on the system. If `true`, render the hosts file using `/etc/cloud/templates/hosts.tmpl` replacing `$hostname` and `$fdqn`. If `localhost`, append a `127.0.1.1` entry that resolves from FQDN and hostname every boot. Default: `false`..

Changed in version 22.3. Use of ``template`` is deprecated, use ``true`` instead.

fqdn: (string) Optional fully qualified domain name to use when updating `/etc/hosts`. Preferred over `hostname` if both are provided. In absence of `hostname` and `fqdn` in cloud-config, the `local-hostname` value will be used from datasource metadata.

hostname: (string) Hostname to set when rendering `/etc/hosts`. If `fqdn` is set, the hostname extracted from `fqdn` overrides `hostname`.

Examples:

```
# Do not update or manage /etc/hosts at all. This is the default behavior.
#
# Whatever is present at instance boot time will be present after boot.
# User changes will not be overwritten.
manage_etc_hosts: false

# --- Example2 ---
# Manage /etc/hosts with cloud-init.
# On every boot, /etc/hosts will be re-written from
# ``/etc/cloud/templates/hosts.tmpl``.
#
# The strings '$hostname' and '$fqdn' are replaced in the template
# with the appropriate values either from the config-config ``fqdn`` or
# ``hostname`` if provided. When absent, the cloud metadata will be
# checked for ``local-hostname` which can be split into <hostname>.<fqdn>.
#
# To make modifications persistent across a reboot, you must modify
# ``/etc/cloud/templates/hosts.tmpl``.
manage_etc_hosts: true

# --- Example3 ---
# Update /etc/hosts every boot providing a "localhost" 127.0.1.1 entry
# with the latest hostname and fqdn as provided by either IMDS or
# cloud-config.
# All other entries will be left as is.
# 'ping `hostname`' will ping 127.0.1.1
manage_etc_hosts: localhost
```

Update Hostname

Summary: Update hostname and fqdn

This module will update the system hostname and fqdn. If `preserve_hostname` is set `true`, 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 schema:

preserve_hostname: (boolean) Do not update system hostname when `true`. Default: `false`.

prefer_fqdn_over_hostname: (boolean) By default, it is distro-dependent whether cloud-init uses the short hostname or fully qualified domain name when both `local-hostname` and `fqdn` are both present in instance metadata. When set `true`, use fully qualified domain name if present as hostname instead of short hostname. When set `false`, use `hostname` config value if present, otherwise fallback to `fqdn`.

Examples:

```
# By default: when ``preserve_hostname`` is not specified cloud-init
# updates ``/etc/hostname`` per-boot based on the cloud provided
# ``local-hostname`` setting. If you manually change ``/etc/hostname``
# after boot cloud-init will no longer modify it.
#
# This default cloud-init behavior is equivalent to this cloud-config:
preserve_hostname: false

# --- Example2 ---
# Prevent cloud-init from updating the system hostname.
preserve_hostname: true

# --- Example3 ---
# Prevent cloud-init from updating ``/etc/hostname``
preserve_hostname: true

# --- Example4 ---
# Set hostname to "external.fqdn.me" instead of "myhost"
fqdn: external.fqdn.me
hostname: myhost
prefer_fqdn_over_hostname: true

# --- Example5 ---
# Set hostname to "external" instead of "external.fqdn.me" when
# cloud metadata provides the ``local-hostname``: "external.fqdn.me".
prefer_fqdn_over_hostname: false
```

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 under the `groups` key as a string of comma-separated groups to create, or a list. Each item in the list should either contain a string of a single group to create, or a dictionary with the group name as the key and string of a single user as a member of that group or a list of users who should be members of the group.

Note: Groups are added before users, so any users in a group list must already exist on the system.

Users to add can be specified as a string or list under the `users` key. Each entry in the list should either be a string or a dictionary. If a string is specified, that string can be comma-separated usernames to create or the reserved string `default` which represents the primary admin user used to access the system. The `default` user varies per distribution and is generally configured in `/etc/cloud/cloud.cfg` by the `default_user` key.

Each `users` dictionary item must contain either a `name` or `snapuser` key, otherwise it will be ignored. Omission of `default` as the first item in the `users` list skips creation the default user. If no `users` key is provided the default behavior is to create the default user via this config:

```
users:
- default
```

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`.

The `user` key can be used to override the `default_user` configuration defined in `/etc/cloud/cloud.cfg`. The `user` value should be a dictionary which supports the same config keys as the `users` dictionary items.

Internal name: `cc_users_groups`

Module frequency: once-per-instance

Supported distros: all

Config schema:

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

Each object in **groups** list supports the following keys:

<group_name>: (string/array of string) Optional string of single username or a list of usernames to add to the group

user: (string/object) The `user` dictionary values override the `default_user` configuration from `/etc/cloud/cloud.cfg`. The *user* dictionary keys supported for the default_user are the same as the `users` schema.

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

Each object in **users** list supports the following keys:

name: (string) The user's login name. Required otherwise user creation will be skipped for this user.

expiredate: (string) Optional. Date on which the user's account will be disabled. Default: `null`

gecos: (string) Optional comment about the user, usually a comma-separated string of real name and contact information

groups: (string/object/array) Optional comma-separated string of groups to add the user to.

homedir: (string) Optional home dir for user. Default: `/home/<username>`

inactive: (string) Optional string representing the number of days until the user is disabled.

lock-passwd: (boolean) Default: `true`

Deprecated in version 22.3. Use ``lock_passwd`` instead.

lock_passwd: (boolean) Disable password login. Default: `true`

no_create_home: (boolean) Do not create home directory. Default: `false`

no_log_init: (boolean) Do not initialize lastlog and faillog for user. Default: `false`

no_user_group: (boolean) Do not create group named after user. Default: `false`

passwd: (string) Hash of user password applied when user does not exist. This will NOT be applied if the user already exists. To generate this hash, run: mkpasswd –method=SHA-512 –rounds=4096. **Note:** While hashed password is better than plain text, using `passwd` in user-data represents a security risk as user-data could be accessible by third-parties depending on your cloud platform.

hashed_passwd: (string) Hash of user password to be applied. This will be applied even if the user is pre-existing. To generate this hash, run: mkpasswd –method=SHA-512 –rounds=4096. **Note:** While `hashed_password` is better than `plain_text_passwd`, using `passwd` in user-data represents a security risk as user-data could be accessible by third-parties depending on your cloud platform.

plain_text_passwd: (string) Clear text of user password to be applied. This will be applied even if the user is pre-existing. There are many more secure options than using plain text passwords, such as `ssh_import_id` or `hashed_passwd`. Do not use this in production as user-data and your password can be exposed.

create_groups: (boolean) Boolean set `false` to disable creation of specified user `groups`. Default: `true`.

primary_group: (string) Primary group for user. Default: <username>

selinux_user: (string) SELinux user for user's login. Default to default SELinux user.

shell: (string) Path to the user's login shell. The default is to set no shell, which results in a system-specific default being used.

snapuser: (string) 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.

ssh_authorized_keys: (array of string) List of SSH keys to add to user's authkeys file. Can not be combined with `ssh_redirect_user`

ssh_import_id: (array of string) List of SSH IDs to import for user. Can not be combined with `ssh_redirect_user`.

ssh_redirect_user: (boolean) 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 `default_username` for this instance. Default: `false`. This key can not be combined with `ssh_import_id` or `ssh_authorized_keys`.

system: (boolean) Optional. Create user as system user with no home directory. Default: `false`.

sudo: (string/null/boolean) Sudo rule to use or false. Absence of a sudo value or `null` will result in no sudo rules added for this user..

Changed in version 22.2. The value ``false`` is deprecated for this key, use ``null`` instead.

uid: (integer/string) The user's ID. Default is next available value..

Changed in version 22.3. The use of ``string`` type is deprecated. Use an ``integer`` instead.

Examples:

```
# Add the ``default_user`` from /etc/cloud/cloud.cfg.
# This is also the default behavior of cloud-init when no `users` key
# is provided.
users:
- default

# --- Example2 ---
# Add the 'admingroup' with members 'root' and 'sys' and an empty
# group cloud-users.
groups:
- admingroup: [root,sys]
- cloud-users

# --- Example3 ---
# Skip creation of the <default> user and only create newsuper.
# Password-based login is rejected, but the github user TheRealFalcon
# and the launchpad user falcojr can SSH as newsuper. The default
# shell for newsuper is bash instead of system default.
users:
- name: newsuper
  gecos: Big Stuff
  groups: users, admin
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash
  lock_passwd: true
  ssh_import_id:
    - lp:falcojr
    - gh:TheRealFalcon

# --- Example4 ---
# On a system with SELinux enabled, add youruser and set the
# SELinux user to 'staff_u'. When omitted on SELinux, the system will
# select the configured default SELinux user.
users:
- default
- name: youruser
  selinux_user: staff_u

# --- Example5 ---
# To redirect a legacy username to the <default> user for a
# distribution, ssh_redirect_user will accept an SSH connection and
# emit a message telling the client to ssh as the <default> user.
# SSH clients will get the message:
users:
- default
- name: nosshlogins
  ssh_redirect_user: true

# --- Example6 ---
# Override any ``default_user`` config in /etc/cloud/cloud.cfg with
# supplemental config options.
# This config will make the default user to mynewdefault and change
# the user to not have sudo rights.
```

(continues on next page)

(continued from previous page)

```
ssh_import_id: [chad.smith]
user:
  name: mynewdefault
  sudo: null
```

Wireguard

Summary: Module to configure Wireguard tunnel

Wireguard module provides a dynamic interface for configuring Wireguard (as a peer or server) in an easy way.

This module takes care of:

- writing interface configuration files
- enabling and starting interfaces
- installing wireguard-tools package
- loading wireguard kernel module
- executing readiness probes

What's a readiness probe?

The idea behind readiness probes is to ensure Wireguard connectivity before continuing the cloud-init process. This could be useful if you need access to specific services like an internal APT Repository Server (e.g Landscape) to install/update packages.

Example:

An edge device can't access the internet but uses cloud-init modules which will install packages (e.g landscape, packages, ubuntu_advantage). Those modules will fail due to missing internet connection. The "wireguard" module fixes that problem as it waits until all readinessprobes (which can be arbitrary commands - e.g. checking if a proxy server is reachable over Wireguard network) are finished before continuing the cloud-init "config" stage.

Note: In order to use DNS with Wireguard you have to install `resolvconf` package or symlink it to systemd's `resolvectl`, otherwise `wg-quick` commands will throw an error message that executable `resolvconf` is missing which leads wireguard module to fail.

Internal name: `cc_wireguard`

Module frequency: once-per-instance

Supported distros: ubuntu

Activate only on keys: `wireguard`

Config schema:

wireguard: (null/object)

interfaces: (array of object)

Each object in **interfaces** list supports the following keys:

name: (string) Name of the interface. Typically wgx (example: wg0)

config_path: (string) Path to configuration file of Wireguard interface

content: (string) Wireguard interface configuration. Contains key, peer, ...

readinessprobe: (array of string) List of shell commands to be executed as probes.

Examples:

```
# Configure one or more WG interfaces and provide optional readinessprobes
wireguard:
  interfaces:
    - name: wg0
      config_path: /etc/wireguard/wg0.conf
      content: |
        [Interface]
        PrivateKey = <private_key>
        Address = <address>
        [Peer]
        PublicKey = <public_key>
        Endpoint = <endpoint_ip>:<endpoint_ip_port>
        AllowedIPs = <allowedip1>, <allowedip2>, ...
    - name: wg1
      config_path: /etc/wireguard/wg1.conf
      content: |
        [Interface]
        PrivateKey = <private_key>
        Address = <address>
        [Peer]
        PublicKey = <public_key>
        Endpoint = <endpoint_ip>:<endpoint_ip_port>
        AllowedIPs = <allowedip1>
  readinessprobe:
    - 'systemctl restart service'
    - 'curl https://webhook.endpoint/example'
    - 'nc -zv some-service-fqdn 443'
```

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.

Warning: Existing files will be overridden.

Internal name: `cc_write_files`

Module frequency: once-per-instance

Supported distros: all

Activate only on keys: `write_files`

Config schema:

write_files: (array of object)

Each object 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 and new directories. Default: `root:root`

permissions: (string) Optional file permissions to set on `path` represented as an octal string '0###'. Default: `0o644`

encoding: (`gz`/`gzip`/`gz+base64`/`gzip+base64`/`gz+b64`/`gzip+b64`/`b64`/`base64`/`text/plain`) 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: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHRoZSBzdGF0ZSBvZiBTRUxpbnV4...
  owner: root:root
  path: /etc/sysconfig/selinux
  permissions: '0644'

# --- Example2 ---
# Appending content to an existing file
write_files:
- content: |
    15 * * * * root ship_logs
  path: /etc/crontab
  append: true

# --- Example3 ---
# Provide gziped binary content
write_files:
- encoding: gzip
  content: !!binary |
      H4sIAIDb/U8C/1NW1E/KzNMvzuBKTc7IV8hIzcnJVyjPL8pJ4QIA6N+MVxsAAAA=
  path: /usr/bin/hello
```

(continues on next page)

(continued from previous page)

```
  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
```

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 opaque 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: once-per-instance

Supported distros: almalinux, centos, cloudlinux, eurolinux, fedora, mariner, openEuler, OpenCloudOS, openmandriva, photon, rhel, rocky, TencentOS, virtuozzo

Activate only on keys: `yum_repos`

Config schema:

yum_repo_dir: (string) The repo parts directory where individual yum repo config files will be written. Default: `/etc/yum.repos.d`

yum_repos: (object)

<repo_name>: (object) Object keyed on unique yum repo IDs. The key used will be used to write yum repo config files in `yum_repo_dir`/<repo_key_id>.repo.

baseurl: (string) URL to the directory where the yum repository's 'repodata' directory lives

name: (string) Optional human-readable name of the yum repo.

enabled: (boolean) Whether to enable the repo. Default: `true`.

<yum_config_option>: (integer/boolean/string) Any supported yum repository configuration options will be written to the yum repo config file. See: man yum.conf

Examples:

```
yum_repos:
  my_repo:
    baseurl: http://blah.org/pub/epel/testing/5/$basearch/
yum_repo_dir: /store/custom/yum.repos.d

# --- Example2 ---
# Enable cloud-init upstream's daily testing repo for EPEL 8 to
# install latest cloud-init from tip of `main` for testing.
yum_repos:
  cloud-init-daily:
    name: Copr repo for cloud-init-dev owned by @cloud-init
    baseurl: https://download.copr.fedorainfracloud.org/results/@cloud-init/cloud-init-
↪dev/epel-8-$basearch/
    type: rpm-md
    skip_if_unavailable: true
    gpgcheck: true
    gpgkey: https://download.copr.fedorainfracloud.org/results/@cloud-init/cloud-init-
↪dev/pubkey.gpg
    enabled_metadata: 1

# --- Example3 ---
# Add the file /etc/yum.repos.d/epel_testing.repo which can then
# subsequently be used by yum for later operations.
yum_repos:
# The name of the repository
 epel-testing:
   baseurl: https://download.copr.fedorainfracloud.org/results/@cloud-init/cloud-init-
↪dev/pubkey.gpg
   enabled: false
   failovermethod: priority
   gpgcheck: true
   gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL
   name: Extra Packages for Enterprise Linux 5 - Testing

# --- Example4 ---
# Any yum repo configuration can be passed directly into
# the repository file created. See: man yum.conf for supported
# config keys.
#
# Write /etc/yum.conf.d/my-package-stream.repo with gpgkey checks
# on the repo data of the repository enabled.
yum_repos:
  my package stream:
    baseurl: http://blah.org/pub/epel/testing/5/$basearch/
    mirrorlist: http://some-url-to-list-of-baseurls
    repo_gpgcheck: 1
    enable_gpgcheck: true
    gpgkey: https://url.to.ascii-armored-gpg-key
```

Zypper Add Repo

Summary: Configure zypper behavior and add zypper repositories

Zypper behavior can be configured using the `config` key, which will modify `/etc/zypp/zypp.conf`. The configuration writer will only append the provided configuration options to the configuration file. Any duplicate options will be resolved by the way the zypp.conf INI file is parsed.

Note: Setting `configdir` is not supported and will be skipped.

The `repos` key may be used to add repositories to the system. Beyond the required `id` and `baseurl` attributions, no validation is performed on the `repos` entries. It is assumed the user is familiar with the zypper repository file format. This configuration is also applicable for systems with transactional-updates.

Internal name: `cc_zypper_add_repo`

Module frequency: always

Supported distros: opensuse, opensuse-microos, opensuse-tumbleweed, opensuse-leap, sle_hpc, sle-micro, sles

Activate only on keys: `zypper`

Config schema:

zypper: (object)

repos: (array of object)

Each object in **repos** list supports the following keys:

id: (string) The unique id of the repo, used when writing /etc/zypp/repos.d/<id>.repo.

baseurl: (string) The base repositoy URL

config: (object) Any supported zypo.conf key is written to `/etc/zypp/zypp.conf`

Examples:

```
zypper:
  repos:
    - id: opensuse-oss
      name: os-oss
      baseurl: http://dl.opensuse.org/dist/leap/v/repo/oss/
      enabled: 1
      autorefresh: 1
    - id: opensuse-oss-update
      name: os-oss-up
      baseurl: http://dl.opensuse.org/dist/leap/v/update
      # any setting per
      # https://en.opensuse.org/openSUSE:Standards_RepoInfo
      # enable and autorefresh are on by default
  config:
    reposdir: /etc/zypp/repos.dir
    servicesdir: /etc/zypp/services.d
    download.use_deltarpm: true
    # any setting in /etc/zypp/zypp.conf
```

2.4.2 Cloud config examples

Including users and groups

```
#cloud-config
# Add groups to the system
# The following example adds the 'admingroup' group with members 'root' and 'sys'
# and the empty group cloud-users.
groups:
  - admingroup: [root,sys]
  - cloud-users

# Add users to the system. Users are added after groups are added.
# Note: Most of these configuration options will not be honored if the user
#       already exists. Following options are the exceptions and they are
#       applicable on already-existing users:
#       - 'plain_text_passwd', 'hashed_passwd', 'lock_passwd', 'sudo',
#         'ssh_authorized_keys', 'ssh_redirect_user'.
users:
  - default
  - name: foobar
    gecos: Foo B. Bar
    primary_group: foobar
    groups: users
    selinux_user: staff_u
    expiredate: '2032-09-01'
    ssh_import_id:
      - lp:falcojr
      - gh:TheRealFalcon
    lock_passwd: false
    passwd: $6$j212wezy$7H/1LT4f9/N3wpgNunhsIqtMj62OKiS3nyNwuizouQc3u7MbYCarYeAHWYPYb2FT.
↪lbioDm2RrkJPb9BZMN1O/
  - name: barfoo
    gecos: Bar B. Foo
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: users, admin
    ssh_import_id:
      - lp:falcojr
      - gh:TheRealFalcon
    lock_passwd: true
    ssh_authorized_keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSL7uWGj8cgWyIOaspgKdVy0cKJ+UTjfv7jBOjG2H/
↪GN8bJVXy72XAvnhM0dUM+CCs8FOf0YlPX+Frvz2hKInrmRhZVwRSL129PasD12MlI3l44u6IwS1o/
↪W86Q+tkQYEljtqDOo0a+cOsaZkvUNzUyEXUwz/
↪lmYa6G4hMKZH4NBj7nbAAF96wsMCoyNwbWryBnDYUr6wMbjRR1J9Pw7Xh7WRC73wy4Va2YuOgbD3V/
↪5ZrFPLbWZW/7TFXVrql04QVbyei4aiFR5n//GvoqwQDNe58LmbzX/xvxyKJYdny2zXmdAhMxbrpFQsfpkJ9E/
↪H5w0yOdSvnWbUoG5xNGoOB csmith@fringe
  - name: cloudy
    gecos: Magic Cloud App Daemon User
    inactive: '5'
    system: true
  - name: fizzbuzz
    sudo: false
```

(continues on next page)

(continued from previous page)

```
    shell: /bin/bash
    ssh_authorized_keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSL7uWGj8cgWyIOaspgKdVy0cKJ+UTjfv7jBOjG2H/
↪GN8bJVXy72XAvnhM0dUM+CCs8FOf0YlPX+Frvz2hKInrmRhZVwRSL129PasD12MlI3l44u6IwS1o/
↪W86Q+tkQYEljtqDOo0a+cOsaZkvUNzUyEXUwz/
↪lmYa6G4hMKZH4NBj7nbAAF96wsMCoyNwbWryBnDYUr6wMbjRR1J9Pw7Xh7WRC73wy4Va2YuOgbD3V/
↪5ZrFPLbWZW/7TFXVrql04QVbyei4aiFR5n//GvoqwQDNe58LmbzX/xvxyKJYdny2zXmdAhMxbrpFQsfpkJ9E/
↪H5w0yOdSvnWbUoG5xNGoOB csmith@fringe
  - snapuser: joe@joeuser.io
  - name: nosshlogins
    ssh_redirect_user: true

# Valid Values:
#   name: The user's login name
#   expiredate: Date on which the user's account will be disabled.
#   gecos: The user name's real name, i.e. "Bob B. Smith"
#   homedir: Optional. Set to the local path you want to use. Defaults to
#            /home/<username>
#   primary_group: define the primary group. Defaults to a new group created
#            named after the user.
#   groups:  Optional. Additional groups to add the user to. Defaults to none
#   selinux_user:  Optional. The SELinux user for the user's login, such as
#            "staff_u". When this is omitted the system will select the default
#            SELinux user.
#   lock_passwd: Defaults to true. Lock the password to disable password login
#   inactive: Number of days after password expires until account is disabled
#   passwd: The hash -- not the password itself -- of the password you want
#            to use for this user. You can generate a hash via:
#                mkpasswd --method=SHA-512 --rounds=4096
#            (the above command would create from stdin an SHA-512 password hash
#            with 4096 salt rounds)
#
#            Please note: while the use of a hashed password is better than
#                plain text, the use of this feature is not ideal. Also,
#                using a high number of salting rounds will help, but it should
#                not be relied upon.
#
#                To highlight this risk, running John the Ripper against the
#                example hash above, with a readily available wordlist, revealed
#                the true password in 12 seconds on a i7-2620QM.
#
#                In other words, this feature is a potential security risk and is
#                provided for your convenience only. If you do not fully trust the
#                medium over which your cloud-config will be transmitted, then you
#                should not use this feature.
#
#   no_create_home: When set to true, do not create home directory.
#   no_user_group: When set to true, do not create a group named after the user.
#   no_log_init: When set to true, do not initialize lastlog and faillog database.
#   ssh_import_id: Optional. Import SSH ids
#   ssh_authorized_keys: Optional. [list] Add keys to user's authorized keys file
#                        An error will be raised if no_create_home or system is
```

(continues on next page)

(continued from previous page)

```
#                         also set.
#   ssh_redirect_user: Optional. [bool] Set true to block ssh logins for cloud
#       ssh public keys and emit a message redirecting logins to
#       use <default_username> instead. This option only disables cloud
#       provided public-keys. An error will be raised if ssh_authorized_keys
#       or ssh_import_id is provided for the same user.
#
#   sudo: Defaults to none. Accepts a sudo rule string, a list of sudo rule
#         strings or False to explicitly deny sudo usage. Examples:
#
#         Allow a user unrestricted sudo access.
#             sudo:  ALL=(ALL) NOPASSWD:ALL
#
#         Adding multiple sudo rule strings.
#             sudo:
#               - ALL=(ALL) NOPASSWD:/bin/mysql
#               - ALL=(ALL) ALL
#
#         Prevent sudo access for a user.
#             sudo: False
#
#         Note: Please double check your syntax and make sure it is valid.
#               cloud-init does not parse/check the syntax of the sudo
#               directive.
#   system: Create the user as a system user. This means no home directory.
#   snapuser: Create a Snappy (Ubuntu-Core) user via the snap create-user
#             command available on Ubuntu systems.  If the user has an account
#             on the Ubuntu SSO, specifying the email will allow snap to
#             request a username and any public ssh keys and will import
#             these into the system with username specified by SSO account.
#             If 'username' is not set in SSO, then username will be the
#             shortname before the email domain.
#

# Default user creation:
#
# Unless you define users, you will get a 'ubuntu' user on Ubuntu systems with the
# legacy permission (no password sudo, locked user, etc). If however, you want
# to have the 'ubuntu' user in addition to other users, you need to instruct
# cloud-init that you also want the default user. To do this use the following
# syntax:
#   users:
#     - default
#     - bob
#     - ....
#  foobar: ...
#
# users[0] (the first user in users) overrides the user directive.
#
# The 'default' user above references the distro's config:
# system_info:
#   default_user:
```

(continues on next page)

(continued from previous page)

```
#     name: Ubuntu
#     plain_text_passwd: 'ubuntu'
#     home: /home/ubuntu
#     shell: /bin/bash
#     lock_passwd: True
#     gecos: Ubuntu
#     groups: [adm, audio, cdrom, dialout, floppy, video, plugdev, dip, netdev]
```

Writing out arbitrary files

```
#cloud-config
# vim: syntax=yaml
#
# This is the configuration syntax that the write_files module
# will know how to understand. Encoding can be given b64 or gzip or (gz+b64).
# The content will be decoded accordingly and then written to the path that is
# provided.
#
# Note: Content strings here are truncated for example purposes.
write_files:
- encoding: b64
  content: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHRoZSBzdGF0ZSBvZiBTRUxpbnV4...
  owner: root:root
  path: /etc/sysconfig/selinux
  permissions: '0644'
- content: |
    # My new /etc/sysconfig/samba file

    SMBDOPTIONS="-D"
  path: /etc/sysconfig/samba
- content: !!binary |
    f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAwARAAAAAAABAAAAAAAAAAJAVAAAAAAAAAAAAAEAAOAAI
    AEAAHgAdAAYAAAAFAAAAQAAAAAAAAABAAEAAAAAAAEAAQAAAAAAAwAEAAAAAAADAAQAAAAAAAAgA
    AAAAAAAAAwAAAAQAAAAAgAAAAAAAAACQAAAAAAAAAJAAAAAAAAAcAAAAAAAAABwAAAAAAAAAQAA
    ....
  path: /bin/arch
  permissions: '0555'
- encoding: gzip
  content: !!binary |
    H4sIAIDb/U8C/1NW1E/KzNMvzuBKTc7IV8hIzcnJVyjPL8pJ4QIA6N+MVxsAAAA=
  path: /usr/bin/hello
  permissions: '0755'
```

Adding a yum repository

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

Configure an instance's trusted CA certificates

```
#cloud-config
#
# This is an example file to configure an instance's trusted CA certificates
# system-wide for SSL/TLS trust establishment when the instance boots for the
# first time.
#
# Make sure that this file is valid yaml before starting instances.
# It should be passed as user-data when starting the instance.

ca_certs:
  # If present and set to True, the 'remove_defaults' parameter will either
  # disable all the trusted CA certifications normally shipped with
  # Alpine, Debian or Ubuntu. On RedHat, this action will delete those
  # certificates.
  # This is mainly for very security-sensitive use cases - most users will not
  # need this functionality.
  remove_defaults: true

  # If present, the 'trusted' parameter should contain a certificate (or list
  # of certificates) to add to the system as trusted CA certificates.
  # Pay close attention to the YAML multiline list syntax.  The example shown
  # here is for a list of multiline certificates.
  trusted:
  - |
    -----BEGIN CERTIFICATE-----
    YOUR-ORGS-TRUSTED-CA-CERT-HERE
```

(continues on next page)

(continued from previous page)

```
    -----END CERTIFICATE-----
  - |
    -----BEGIN CERTIFICATE-----
    YOUR-ORGS-TRUSTED-CA-CERT-HERE
    -----END CERTIFICATE-----
```

Install and run chef recipes

```
#cloud-config
#
# This is an example file to automatically install chef-client and run a
# list of recipes when the instance boots for the first time.
# Make sure that this file is valid yaml before starting instances.
# It should be passed as user-data when starting the instance.

# The default is to install from packages.

# Key from https://packages.chef.io/chef.asc
apt:
  sources:
    source1:
      source: "deb http://packages.chef.io/repos/apt/stable $RELEASE main"
      key: |
        -----BEGIN PGP PUBLIC KEY BLOCK-----
        Version: GnuPG v1.4.12 (Darwin)
        Comment: GPGTools - http://gpgtools.org

        mQGiBEppC7QRBADfsOkZU6KZK+YmKw4wev5mjKJEkVGlus+NxW8wItX5sGa6kdUu
        twAyj7Yr92rF+ICFEP3gGU6+lGo0Nve7KxkN/1W7/m3G4zuk+ccIKmjp8KS3qn99
        dxy64vcji9jIllVa+XXOGIp0G8GEaj7mbkixL/bMeGfdMlv8Gf2XPpp9vwCgn/GC
        JKacfnw7MpLKUHOYSlb//JsEAJqao3ViNfav83jJKEkD8cf59Y8xKia5OpZqTK5W
        ShVnNWS3U5IVQk10ZDH97Qn/YrK387H4CyhLE9mxPXs/ul18ioiaars/q2MEKU2I
        XKfV21eMLO9LYd6Ny/Kqj8o5WQK2J6+NAhSwvthZcIEphcFignIuobP+B5wNFQpe
        DbKfA/0WvN2OwFeWRcmmd3Hz7nHTpcnSF+4QX6yHRF/5BgxkG6IqBIACQbzPn6Hm
        sMtm/SVf11izmDqSsQptCrOZILfLX/mE+YOl+CwWSHhl+YsFts1WOuh1EhQD26aO
        Z84HuHV5HFRWjDLw9LriltBVQcXbpfSrRP5bdr7Wh8vhqJTPjrQnT3BzY29kZSBQ
        YWNrYWdlcyA8cGFja2FnZXNAb3BzY29kZS5jb20+iGAEExECACAFAkppC7QCGwMG
        CwkIBwMCBBUCCAMEFgIDAQIeAQIXgAAKCRApQKupg++Caj8sAKCOXmdG36gWji/K
        +o+XtBfvdMnFYQCfTCEWxRy2BnzLoBBFCjDSK6sJqCu0IENIRUYgUGFja2FnZXMg
        PHBhY2thZ2VzQGNoZWYuaW8+iGIEExECACIFAlQwYFECGwMGCwkIBwMCBhUIAgkK
        CwQWAgMBAh4BAheAAAoJECClAq6mD74JqX94An26z99XOHWpLN8ahzm7cp13t4Xid
        AJ9wVcgoUBzvgg91lKfv/34cmemZn7kCDQRKaQu0EAgAg7ZLCVGVTmLqBM6njZEd
        Zbv+mZbvwLBSomdiqddE6u3eH0X3GuwaQfQWHUVG2yedyDMiG+EMtCdEeeRebTCz
        SNXQ8Xvi22hRPoEsBSwWLZI8/XNg0n0f1+GEr+mOKO0BxDB2DG7DA0nnEISxwFkK
        OFJFebR3fRsrWjj0KjDxkhse2ddU/jVz1BY7Nf8toZmwpBmdozETMOTx3LJy1HZ/
        Te9FJXJMUaB2lRyluv15MVWCKQJro4MQG/7QGcIfrIZNfAGJ32DDSjV7/YO+IpRY
        IL4CUBQ65suY4gYUG4jhRH6u7H1p99sdwsg5OIpBe/v2Vbc/tbwAB+eJJAp89Zeu
        twADBQf/ZcGoPhTGFuzbkcNRSIz+boaeWPoSxK2DyfScyCAuG41CY9+g0HIw9Sq8
        DuxQvJ+vrEJjNvNE3EAEdKl/zkXMZDb1EXjGwDi845TxEMhhD1dDw2qpHqnJ2mtE
        WpZ7juGwA3sGhi6FapO04tIGacCfNNHmlRGipyq5ZiKIRq9mLEndlECr8cwaKgkS
```

(continues on next page)

(continued from previous page)

```
        0wWu+xmMZe7N5/t/TK19HXNh4tVacv0F3fYK54GUjt2FjCQV75USnmNY4KPTYLXA
        dzC364hEMlXpN21siIFgB04w+TXn5UF3B4FfAy5hevvr4DtV4MvMiGLu0oWjpaLC
        MpmrR3Ny2wkmO0h+vgri9uIP06ODWIhJBBgRAgAJBQJKaQu0AhsMAAoJECIAq6mD
        74Jq4hIAoJ5KrYS8kCwj26SAGzglwggpvt3CAJ0bekyky56vNqoegB+y4PQVDv4K
        zA==
        =IxPr
        -----END PGP PUBLIC KEY BLOCK-----

chef:

  # Valid values are 'accept' and 'accept-no-persist'
  chef_license: "accept"

  # Valid values are 'gems' and 'packages' and 'omnibus'
  install_type: "packages"

  # Boolean: run 'install_type' code even if chef-client
  #          appears already installed.
  force_install: false

  # Chef settings
  server_url: "https://chef.yourorg.com"

  # Node Name
  # Defaults to the instance-id if not present
  node_name: "your-node-name"

  # Environment
  # Defaults to '_default' if not present
  environment: "production"

  # Default validation name is chef-validator
  validation_name: "yourorg-validator"
  # if validation_cert's value is "system" then it is expected
  # that the file already exists on the system.
  validation_cert: |
    -----BEGIN RSA PRIVATE KEY-----
    YOUR-ORGS-VALIDATION-KEY-HERE
    -----END RSA PRIVATE KEY-----

  # A run list for a first boot json, an example (not required)
  run_list:
    - "recipe[apache2]"
    - "role[db]"

  # Specify a list of initial attributes used by the cookbooks
  initial_attributes:
    apache:
      prefork:
        maxclients: 100
      keepalive: "off"

```

(continues on next page)

(continued from previous page)

```
  # if install_type is 'omnibus', change the url to download
  omnibus_url: "https://www.chef.io/chef/install.sh"

  # if install_type is 'omnibus', pass pinned version string
  # to the install script
  omnibus_version: "12.3.0"

  # If encrypted data bags are used, the client needs to have a secrets file
  # configured to decrypt them
  encrypted_data_bag_secret: "/etc/chef/encrypted_data_bag_secret"

# Capture all subprocess output into a logfile
# Useful for troubleshooting cloud-init issues
output: {all: '| tee -a /var/log/cloud-init-output.log'}
```

Install and run *ansible-pull*

```
#cloud-config
package_update: true
package_upgrade: true

# if you're already installing other packages, you may
# wish to manually install ansible to avoid multiple calls
# to your package manager
packages:
  - git
ansible:
  install_method: pip
  pull:
    url: "https://github.com/holmanb/vmboot.git"
    playbook_name: ubuntu.yml
```

Configure instance to be managed by Ansible

```
#cloud-config
#
# A common use-case for cloud-init is to bootstrap user and ssh
# settings to be managed by a remote configuration management tool,
# such as ansible.
#
# This example assumes a default Ubuntu cloud image, which should contain
# the required software to be managed remotely by Ansible.
#
ssh_pwauth: false

users:
- name: ansible
  gecos: Ansible User
  groups: users,admin,wheel
```

(continues on next page)

(continued from previous page)

```
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash
  lock_passwd: true
  ssh_authorized_keys:
    - "ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABgQDRCJCQ1UD9QslWDSw5Pwsvba0Wsf1pO4how5BtNaZn0xLZpTq2nqFEJshUkd/
↪zCWF7DWyhmNphQ8c+U+wcmdNVcg2pI1kPxq0VZzBfZ7cDwhjgeLsIvTXvU+HVRtsXh4c5FlUXpRjf/
↪x+a3vqFRvNsRd1DE+5ZqQHbOVbnsStk3PZppaByMg+AZZMx56OUk2pZCgvpCwj6LIixqwuxNKPxmJf45RyOsPUXwCwkq9UD4me5jk
↪141moWsGxSnd5NxCbPUWGoRfYcHc865E70nN4WrZkM7RFI/s5mvQtuj8dRL67JUEwvdvEDO0EBz21FV/
↪iOracXd2omlTUSK+wYrWGtiwQwEgr4r5bimxDKy9L8UlaJZ+ONhLTP8ecTHYkaU1C75sLX9ZYd5YtqjiNGsNF+wdW6WrXrQiWeyrGI
↪"


# use the following passwordless demonstration key for testing or
# replace with your own key pair
#
# -----BEGIN OPENSSH PRIVATE KEY-----
# b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
# NhAAAAAwEAAQAAAYEA0QiQkNVA/ULJVg0sOT8LL22tFrH9aTuIaMOQbTWmZ9MS2aU6tp6h
# RCbIVJHf8wlhew1soZjaYUPHPlPsHJnTVXINqSNZD8atFWcwX2e3A8IY4Hi7CL0171Ph1U
# bbF4eHORZVF6UY3/8fmt76hUbzbEXdQxPuWakB2zlW57ErZNz2aaWgcjIPgGWTMeejlJNq
# WQoL6QsI+iyIsasLsTSj8ZiX+OUcjrD1F8AsJKvVA+JnuY5LEzz5Ld6HlFsNWUkhfBf9eN
# ZqFrBsUp3eTcQmz1FhqEX2HB3POuRO9JzeFq2ZDO0RSP7OZr0Lbo/HUS+uyVBML3bxAztB
# Ac9tRVf4jq2nF3dqJpU1EivsGK1hrYsEMBIK+K+W4psQysvS/FJWiWfjjYS0z/HnEx2JGl
# NQu+bC1/WWHeWLao4jRrDRfsHVulq160Ilnsqxiu2cGwO5WoEsSGu8nqpyg43ZHCb0FwmB
# izPQDASlniWjqcKmfnTrpzAy3eVWawwlNpaQkidTAAAFgGKSj8diko/HAAAAB3NzaC1yc2
# EAAAGBANEIkJDVQP1CyVYNLDk/Cy9trRax/Wk7iGjDkG01pmfTEtmlOraeoUQmyFSR3/MJ
# YXsNbKGY2mFDxz5T7ByZ01VyDakjWQ/GrRVnMF9ntwPCGOB4uwi9Ne9T4dVG2xeHhzkWVR
# elGN//H5re+oVG82xF3UMT7lmpAds5VuexK2Tc9mmloHIyD4BlkzHno5STalkKC+kLCPos
# iLGrC7E0o/GYl/jlHI6w9RfALCSr1QPiZ7mOSxM8+S3eh5RbDVlJIXwX/XjWahawbFKd3k
# 3EJs9RYahF9hwdzzrkTvSc3hatmQztEUj+zma9C26Px1EvrslQTC928QM7QQHPbUVX+I6t
# pxd3aiaVNRIr7BitYa2LBDASCvivluKbEMrL0vxSVoln442EtM/x5xMdiRpTULvmwtf1lh
# 3li2qOI0aw0X7B1bpatetCJZ7KsYrtnBsDuVqBLEhrvJ6qcoON2Rwm9BcJgYsz0AwEpZ4l
# o6nCpn5066cwMt3lVmsMJTaWkJInUwAAAAMBAAEAAAGAEuz77Hu9EEZyujLOdTnAW9afRv
# XDOZA6pS7yWEufjw5CSlMLwisR83yww09t1QWyvhRqEyYmvOBecsXgaSUtnYfftWz44apy
# /gQYvMVELGKaJAC/q7vjMpGyrxUPkyLMhckALU2KYgV+/rj/j6pBMeVlchmk3pikYrffUX
# JDY990WVO194Dm0buLRzJvfMKYF2BcfF4TvarjOXWAxSuR8www050oJ8HdKahW7Cm5S0po
# FRnNXFGMnLA62vN00vJW8V7j7vui9ukBbhjRWaJuY5rdG/UYmzAe4wvdIEnpk9xIn6JGCp
# FRYTRn7lTh5+/QlQ6FXRP8Ir1vXZFnhKzl0K8Vqh2sf4M79MsIUGAqGxg9xdhjIa5dmgp8
# N18IEDoNEVKUbKuKe/Z5yf8Z9tmexfH1YttjmXMOojBvUHIjRS5hdI9NxnPGRLY2kjAzcm
# gV9Rv3vtdF/+zalk3fAVLeK8hXK+di/7XTvYpfJ2EZBWiNrTeagfNNGiYydsQy3zjZAAAA
# wBNRak7UrqnIHMZn7pkCTgceb1MfByaFtlNzd+Obah54HYIQj5WdZTBAITReMZNt9S5NAR
# M8sQB8UoZPaVSC3ppILIOfLhs6KYj6RrGdiYwyIhMPJ5kRWF8xGCLUX5CjwH2EOq7XhIWt
# MwEFtd/gF2Du7HUNFPsZGnzJ3e7pDKDnE7w2khZ8CIpTFgD769uBYGAtk45QYTDo5JroVM
# ZPDq08Gb/RhIgJLmIpMwyreVpLLLe8SwoMJJ+rihmnJZxO8gAAAMEA0lhiKezeTshht4xu
# rWc0NxxD84a29gSGfTphDPOrlKSEYbkSXhjqCsAZHd8S8kMr3iF6poOk3IWSvFJ6mbd3ie
# qdRTgXH9Thwk4KgpjUhNsQuYRHBbI59Mo+BxSI1B1qzmJSGdmCBL54wwzZmFKDQPQKPxiL
# n0Mlc7GooiDMjT1tbuW/O1EL5EqTRqwgWPTKhBA6r4PnGF150hZRIMooZkD2zX6b1sGojk
# QpvKkEykTwnKCzF5TXO8+wJ3qbcEo9AAAAwQD+Z0r68c2YMNpsmyj3ZKtZNPSvJNcLmyD/
# lWoNJq3djJN4s2JbK8l5ARUdW3xSFEDI9yx/wpfsXoaqWnygP3PoFw2CM4i0EiJiyvrLFU
# r3JLfDUFRy3EJ24RsqbigmEsgQOzTl3xfzeFPfxFoOhokSvTG88PQji1AYHz5kA7p6Zfaz
# Ok11rJYIe7+e9B0lhku0AFwGyqlWQmS/MhIpnjHIk5tP4heHGSmzKQWJDbTskNWd6aq1G7
```

(continues on next page)

(continued from previous page)

```
# 6HWfDpX4HgoM8AAAALaG9sbWFuYkBhcmM=
# -----END OPENSSH PRIVATE KEY-----
#
```

Configure instance to be an Ansible controller

```
#cloud-config
#
# Demonstrate setting up an ansible controller host on boot.
# This example installs a playbook repository from a remote private repository
# and then runs two of the plays.

package_update: true
package_upgrade: true
packages:
  - git
  - python3-pip

# Set up an ansible user
# ----------------------
# In this case I give the local ansible user passwordless sudo so that ansible
# may write to a local root-only file.
users:
- name: ansible
  gecos: Ansible User
  shell: /bin/bash
  groups: users,admin,wheel,lxd
  sudo: ALL=(ALL) NOPASSWD:ALL

# Initialize lxd using cloud-init.
# --------------------------------
# In this example, a lxd container is
# started using ansible on boot, so having lxd initialized is required.
lxd:
  init:
    storage_backend: dir

# Configure and run ansible on boot
# ---------------------------------
# Install ansible using pip, ensure that community.general collection is
# installed [1].
# Use a deploy key to clone a remote private repository then run two playbooks.
# The first playbook starts a lxd container and creates a new inventory file.
# The second playbook connects to and configures the container using ansible.
# The public version of the playbooks can be inspected here [2]
#
# [1] community.general is likely already installed by pip
# [2] https://github.com/holmanb/ansible-lxd-public
#
ansible:
```

(continues on next page)

(continued from previous page)

```
  install_method: pip
  package_name: ansible
  run_user: ansible
  galaxy:
    actions:
      - ["ansible-galaxy", "collection", "install", "community.general"]

  setup_controller:
    repositories:
      - path: /home/ansible/my-repo/
        source: git@github.com:holmanb/ansible-lxd-private.git
    run_ansible:
      - playbook_dir: /home/ansible/my-repo
        playbook_name: start-lxd.yml
        timeout: 120
        forks: 1
        private_key: /home/ansible/.ssh/id_rsa
      - playbook_dir: /home/ansible/my-repo
        playbook_name: configure-lxd.yml
        become_user: ansible
        timeout: 120
        forks: 1
        private_key: /home/ansible/.ssh/id_rsa
        inventory: new_ansible_hosts

# Write a deploy key to the filesystem for ansible.
# ------------------------------------------------
# This deploy key is tied to a private github repository [1]
# This key exists to demonstrate deploy key usage in ansible
# a duplicate public copy of the repository exists here[2]
#
# [1] https://github.com/holmanb/ansible-lxd-private
# [2] https://github.com/holmanb/ansible-lxd-public
#
write_files:
  - path: /home/ansible/.ssh/known_hosts
    owner: ansible:ansible
    permissions: 0o600
    defer: true
    content: |
      |1|YJEFAk6JjnXpUjUSLFiBQS55W9E=|OLNePOn3eBa1PWhBBmt5kXsbGM4= ssh-ed25519␣
↪AAAAC3NzaC1lZDI1NTE5AAAAIOMqqnkVzrm0SdG6UOoqKLsabgH5C9okWi0dh2l9GKJl
      |1|PGGnpCpqi0aakERS4BWnYxMkMwM=|Td0piZoS4ZVC0OzeuRwKcH1MusM= ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAABIwAAAQEAq2A7hRGmdnm9tUDbO9IDSwBK6TbQa+PXYPCPy6rbTrTtw7PHkccKrpp0yVhp5HdEIcKr6pLlVD
↪yMf+Se8xhHTvKSCZIFImWwoG6mbUoWf9nzpIoaSjB+weqqUUmpaaasXVal72J+UX2B+2RPW3RcT0eOzQgqlJL3RKrTJvdsjE3JEAvG
↪w4yCE6gbODqnTWlg7+wC604ydGXA8VJiS5ap43JXiUFFAaQ==
      |1|OJ89KrsNcFTOvoCP/fPGKpyUYFo=|cu7mNzF+QB/5kR0spiYmUJL7DAI= ecdsa-sha2-nistp256␣
↪AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBEmKSENjQEezOmxkZMy7opKgwFB9nkt5YRrYMjNuG5N87uRgg6
↪y6v0mKV0U2w0WZ2YB/++Tpockg=

  - path: /home/ansible/.ssh/id_rsa
    owner: ansible:ansible
```

(continues on next page)

(continued from previous page)

```
    permissions: 0o600
    defer: true
    encoding: base64
    content: |
      LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdGRqRUFB
      QUFBQkc1dmJtVUFBQUFFYm05dVpRQUFBQUFBQUFBQkFBQUJsd0FBQUFkemMyZ3RjbgpOaEFBQUFB
      d0VBQVFBQUFZRUEwUWlRa05WQS9VTEpWZzBzT1Q4TEwyMnRGckg5YVR1SWFNT1FiVFdtWjlNUzJh
      VTZ0cDZoClJDYklWSkhmOHdsaGV3MXNvWmphWVVQSFBsUHNISm5UVlhJTnFTTlpEOGF0Rldjd1gy
      ZTNBOElZNEhpN0NMMDE3MVBoMVUKYmJGNGVIT1JaVkY2VVkzLzhmbXQ3NmhVYnpiRVhkUXhQdVdh
      a0IyemxXNTdFclpOejJhYVdnY2pJUGdHV1RNZWVqbEpOcQpXUW9MNlFzSStpeUlzYXNMc1RTajha
      aVgrT1VjanJEMUY4QXNKS3ZWQStKbnVZNUxFeno1TGQ2SGxGc05XVWtoZkJmOWVOClpxRnJCc1Vw
      M2VUY1FtejFGaHFFWDJIQjNQT3VSTzlKemVGcTJaRE8wUlNQN09acjBMYm8vSFVTK3V5VkJNTDNi
      eEF6dEIKQWM5dFJWZjRqcTJuRjNkcUpwVTFFaXZzR0sxaHJZc0VNQklLK0srVzRwc1F5c3ZTL0ZK
      V2lXZmpqWVMwei9IbkV4MkpHbApOUXUrYkMxL1dXSGVXTGFvNGpSckRSZnNIVnVscTE2MElsbnNx
      eGl1MmNHd081V29Fc1NHdThucXB5ZzQzWkhDYjBGd21CCml6UFFEQVNsbmlXanFjS21mblRycHpB
      eTNlVldhd3dsTnBhUWtpZFRBQUFGZ0dLU2o4ZGlrby9IQUFBQUIzTnphQzF5YzIKRUFBQUdCQU5F
      SWtKRFZRUDFDeVZZTkxEay9DeTl0clJheC9XazdpR2pEa0cwMXBtZlRFdG1sT3JhZW9VUW15RlNS
      My9NSgpZWHNOYktHWTJtRkR4ejVUN0J5WjAxVnlEYWtqV1EvR3JSVm5NRjludHdQQ0dPQjR1d2k5
      TmU5VDRkVkcyeGVIaHprV1ZSCmVsR04vL0g1cmUrb1ZHODJ4RjNVTVQ3bG1wQWRzNVZ1ZXhLMlRj
      OW1tbG9ISXlENEJsa3pIbm81U1RhbGtLQytrTENQb3MKaUxHckM3RTBvL0dZbC9qbEhJNnc5UmZB
      TENTcjFRUGlaN21PU3hNOCtTM2VoNVJiRFZsSklYd1gvWGpXYWhhd2JGS2QzawozRUpzOVJZYWhG
      OWh3ZHp6cmtUdlNjM2hhdG1RenRFVWorem1hOUMyNlB4MUV2cnNsUVRDOTI4UU03UVFIUGJVVlgr
      STZ0CnB4ZDNhaWFWTlJJcjdCaXRZYTJMQkRBU0N2aXZsdUtiRU1yTDB2eFNWb2xuNDQyRXRNL3g1
      eE1kaVJwVFVMdm13dGYxbGgKM2xpMnFPSTBhdzBYN0IxYnBhdGV0Q0paN0tzWXJ0bkJzRHVWcUJM
      RWhydko2cWNvT04yUndtOUJjSmdZc3owQXdFcFo0bApvNm5DcG41MDY2Y3dNdDNsVm1zTUpUYVdr
      SkluVXdBQUFBTUJBQUVBQUFHQUV1ejc3SHU5RUVaeXVqTE9kVG5BVzlhZlJ2ClhET1pBNnBTN3lX
      RXVmanc1Q1NsTUx3aXNSODN5d3cwOXQxUVd5dmhScUV5WW12T0JlY3NYZ2FTVXRuWWZmdFd6NDRh
      cHkKL2dRWXZNVkVMR0thSkFDL3E3dmpNcEd5cnhVUGt5TE1oY2tBTFUyS1lnVisvcmovajZwQk1l
      VmxjaG1rM3Bpa1lyZmZVWApKRFk5OTBXVk8xOTREbTBidUxSekp2Zk1LWUYyQmNmRjRUdmFyak9Y
      V0F4U3VSOHd3dzA1MG9KOEhkS2FoVzdDbTVTMHBvCkZSbk5YRkdNbkxBNjJ2TjAwdkpXOFY3ajd2
      dWk5dWtCYmhqUldhSnVZNXJkRy9VWW16QWU0d3ZkSUVucGs5eEluNkpHQ3AKRlJZVFJuN2xUaDUr
      L1FsUTZGWFJQOElyMXZYWkZuaEt6bDBLOFZxaDJzZjRNNzlNc0lVR0FxR3hnOXhkaGpJYTVkbWdw
      OApOMThJRURvTkVWS1ViS3VLZS9aNXlmOFo5dG1leGZIMVl0dGptWE1Pb2pCdlVISWpSUzVoZEk5
      TnhuUEdSTFkya2pBemNtCmdWOVJ2M3Z0ZEYvK3phbGszZkFWTGVLOGhYSytkaS83WFR2WXBmSjJF
      WkJXaU5yVGVhZ2ZOTkdpWXlkc1F5M3pqWkFBQUEKd0JOUmFrN1VycW5JSE1abjdwa0NUZ2NlYjFN
      ZkJ5YUZ0bE56ZCtPYmFoNTRIWUlRajVXZFpUQkFJVFJlTVpOdDlTNU5BUgpNOHNRQjhVb1pQYVZT
      QzNwcElMSU9mTGhzNktZajZSckdkaVl3eUloTVBKNWtSV0Y4eEdDTFVYNUNqd0gyRU9xN1hoSVd0
      Ck13RUZ0ZC9nRjJEdTdIVU5GUHNaR256SjNlN3BES0RuRTd3MmtoWjhDSXBURmdENzY5dUJZR0F0
      azQ1UVlURG81SnJvVk0KWlBEcTA4R2IvUmhJZ0pMbUlwTXd5cmVWcExMTGU4U3dvTUpKK3JpaG1u
      Slp4TzhnQUFBTUVBMGxoaUtlemVUc2hodDR4dQpyV2MwTnh4RDg0YTI5Z1NHZlRwaERQT3JsS1NF
      WWJrU1hoanFDc0FaSGQ4UzhrTXIzaUY2cG9PazNJV1N2Rko2bWJkM2llCnFkUlRnWEg5VGh3azRL
      Z3BqVWhOc1F1WVJIQmJJNTlNbytCeFNJMUIxcXptSlNHZG1DQkw1NHd3elptRktEUVBRS1B4aUwK
      bjBNbGM3R29vaURNalQxdGJ1Vy9PMUVMNUVxVFJxd2dXUFRLaEJBNnI0UG5HRjE1MGhaUklNb29a
      a0Qyelg2YjFzR29qawpRcHZLa0V5a1R3bktDekY1VFhPOCt3SjNxYmNFbzlBQUFBd1FEK1owcjY4
      YzJZTU5wc215ajNaS3RaTlBTdkpOY0xteUQvCmxXb05KcTNkakpONHMySmJLOGw1QVJVZFczeFNG
      RURJOXl4L3dwZnNYb2FxV255Z1AzUG9GdzJDTTRpMEVpSml5dnJMRlUKcjNKTGZEVUZSeTNFSjI0
      UnNxYmlnbUVzZ1FPelRsM3hmemVGUGZ4Rm9PaG9rU3ZURzg4UFFqaTFBWUh6NWtBN3A2WmZhegpP
      azExckpZSWU3K2U5QjBsaGt1MEFGd0d5cWxXUW1TL01oSXBuakhJazV0UDRoZUhHU216S1FXSkRi
      VHNrTldkNmFxMUc3CjZIV2ZEcFg0SGdvTThBQUFBTGFHOXNiV0Z1WWtCaGNtTT0KLS0tLS1FTkQg
      T1BFTlNTSCBQUklWQVRFIEtFWS0tLS0tCg==
```

Add primary apt repositories

```
#cloud-config

# Add primary apt repositories
#
# To add 3rd party repositories, see cloud-config-apt.txt or the
# Additional apt configuration and repositories section.
#
#
# Default: auto select based on cloud metadata
#  in ec2, the default is <region>.archive.ubuntu.com
# apt:
#   primary:
#     - arches [default]
#       uri:
#     use the provided mirror
#       search:
#     search the list for the first mirror.
#     this is currently very limited, only verifying that
#     the mirror is dns resolvable or an IP address
#
# if neither mirror is set (the default)
# then use the mirror provided by the DataSource found.
# In EC2, that means using <region>.ec2.archive.ubuntu.com
#
# if no mirror is provided by the DataSource, but 'search_dns' is
# true, then search for dns names '<distro>-mirror' in each of
# - fqdn of this host per cloud metadata
# - localdomain
# - no domain (which would search 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>
#
# That gives the cloud provider the opportunity to set mirrors of a distro
# up and expose them only by creating dns entries.
#
# if none of that is found, then the default distro mirror is used
apt:
  primary:
    - arches: [default]
      uri: http://us.archive.ubuntu.com/ubuntu/
# or
apt:
  primary:
    - arches: [default]
      search:
        - http://local-mirror.mydomain
        - http://archive.ubuntu.com
# or
apt:
  primary:
    - arches: [default]
```

(continues on next page)

(continued from previous page)

```
52      search_dns: True
```

Run commands on first boot

```
#cloud-config

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

```
#cloud-config

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

Install arbitrary packages

```
#cloud-config

# Install additional packages on first boot
#
# Default: none
#
# if packages are specified, then package_update will be set to true
#
# packages may be supplied as a single package name or as a list
# with the format [<package>, <version>] wherein the specific
# package version will be installed.
packages:
 - pwgen
 - pastebinit
 - [libpython2.7, 2.7.3-0ubuntu3.1]
```

Update apt database on first boot

```
#cloud-config
# Update apt database on first boot (run 'apt-get update').
# Note, if packages are given, or package_upgrade is true, then
# update will be done independent of this setting.
#
# Default: false
package_update: true
```

Run apt or yum upgrade

```
#cloud-config

# Upgrade the instance on first boot
#
# Default: false
package_upgrade: true
```

Adjust mount points mounted

```
#cloud-config

# set up mount points
# 'mounts' contains a list of lists
#  the inner list are entries for an /etc/fstab line
#  ie : [ fs_spec, fs_file, fs_vfstype, fs_mntops, fs-freq, fs_passno ]
#
# default:
# mounts:
```

(continues on next page)

(continued from previous page)

```
#  - [ ephemeral0, /mnt ]
#  - [ swap, none, swap, sw, 0, 0 ]
#
# in order to remove a previously listed mount (ie, one from defaults)
# list only the fs_spec.  For example, to override the default, of
# mounting swap:
# - [ swap ]
# or
# - [ swap, null ]
#
# - if a device does not exist at the time, an entry will still be
#   written to /etc/fstab.
# - '/dev' can be omitted for device names that begin with: xvd, sd, hd, vd
# - if an entry does not have all 6 fields, they will be filled in
#   with values from 'mount_default_fields' below.
#
# Note, that you should set 'nofail' (see man fstab) for volumes that may not
# be attached at instance boot (or reboot).
#
mounts:
 - [ ephemeral0, /mnt, auto, "defaults,noexec" ]
 - [ sdc, /opt/data ]
 - [ xvdh, /opt/data, "auto", "defaults,nofail", "0", "0" ]
 - [ dd, /dev/zero ]

# mount_default_fields
# These values are used to fill in any entries in 'mounts' that are not
# complete.  This must be an array, and must have 6 fields.
mount_default_fields: [ None, None, "auto", "defaults,nofail", "0", "2" ]


# swap can also be set up by the 'mounts' module
# default is to not create any swap files, because 'size' is set to 0
swap:
  filename: /swap.img
  size: "auto" # or size in bytes
  maxsize: 10485760   # size in bytes
```

Configure instance's SSH keys

```
#cloud-config

# add each entry to ~/.ssh/authorized_keys for the configured user or the
# first user defined in the user definition directive.
ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUUk8EEAnnkhXlukKoUPND/
↪RRClWz2s5TCzIkd3Ou5+Cyz71X0XmazM3l5WgeErvtIwQMyT1KjNoMhoJMrJnWqQPOt5Q8zWd9qG7PBl9+eiH5qV7NZ␣
↪mykey@host
  - ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAABIwAAAQEA3I7VUf2l5gSn5uavROsc5HRDpZdQueUq5ozemNSj8T7enqKHOEaFoU2VoPgGEWC9RyzSQVeyD
```

(continues on next page)

(continued from previous page)

```
↪+i1D+ey3ONkZLN+LQ714cgj8fRS4Hj29SCmXp5Kt5/82cD/VN3NtHw== smoser@brickies

# Send pre-generated SSH private keys to the server
# If these are present, they will be written to /etc/ssh and
# new random keys will not be generated
#  in addition to 'rsa' and 'dsa' as shown below, 'ecdsa' is also supported
ssh_keys:
  rsa_private: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIBxwIBAAJhAKD0YSHy73nUgysO13XsJmd4fHiFyQ+00R7VVu2iV9Qcon2LZS/x
    1cydPZ4pQpfjEha6WxZ6o8ci/Ea/w0n+0HGPwaxlEG2Z9inNtj3pgFrYcRztfECb
    1j6HCibZbAzYtwIBIwJgO8h72WjcmvcpZ8OvHSvTwAguO2TkR6mPgHsgSaKy6GJo
    PUJnaZRWuba/HX0KGyhz19nPzLpzG5f0fYahlMJAyc13FV7K6kMBPXTRR6FxgHEg
    L0MPC7cdqAwOVNcPY6A7AjEA1bNaIjOzFN2sfZX0j7OMhQuc4zP7r80zaGc5oy6W
    p58hRAncFKEvnEq2CeL3vtuZAjEAwNBHpbNsBYTRPCHM7rZuG/iBtwp8Rxhc9I5w
    ixvzMgi+HpGLWzUIBS+P/XhekIjPAjA285rVmEP+DR255Ls65QbgYhJmTzIXQ2T9
    luLvcmFBC6l35Uc4gTgg4ALsmXLn71MCMGMpSWspEvuGInayTCL+vEjmNBT+FAdO
    W7D4zCpI43jRS9U06JVOeSc9CDk2lwiA3wIwCTB/6uc8Cq85D9YqpM10FuHjKpnP
    REPPOyrAspdeOAV+6VKRavstea7+2DZmSUgE
    -----END RSA PRIVATE KEY-----

  rsa_public: ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAABIwAAAGEAoPRhIfLvedSDKw7XdewmZ3h8eIXJD7TRHtVW7aJX1ByifYtlL/
↪HVzJ09nilCl+MSFrpbFnqjxyL8Rr/DSf7QcY/BrGUQbZn2Kc22PemAWthxHO18QJvWPocKJtlsDNi3␣
↪smoser@localhost

  dsa_private: |
    -----BEGIN DSA PRIVATE KEY-----
    MIIBuwIBAAKBgQDP2HLu7pTExL89USyM0264RCyWX/CMLmukxX0Jdbm29ax8FBJT
    pLrO8TIXVY5rPAJm1dTHnpuyJhOvU9G7M8tPUABtzSJh4GVSHlwaCfycwcpLv9TX
    DgWIpSj+6EiHCyaRlB1/CBp9RiaB+10QcFbm+lapuET+/Au6vSDp9IRtlQIVAIMR
    8KucvUYbOEI+yv+5LW9u3z/BAoGBAI0q6JP+JvJmwZFaeCMMVxXUbqiSko/P1lsa
    LNNBHZ5/8MOUIm8rB2FC6ziidfueJpqTMqeQmSAlEBCwnwreUnGfRrKoJpyPNENY
    d15MG6N5J+z81sEcHFeprryZ+D3Ge9VjPq3Tf3NhKKwCDQ0240aPezbnjPeFm4mH
    bYxxcZ9GAoGAXmLIFSQgiAPu459rCKxT46tHJtM0QfnNiEnQLbFluefZ/yiI4DI3
    8UzTCOXLhUA7ybmZha+D/csj15Y9/BNFuO7unzVhikCQV9DTeXX46pG4s1o23JKC
    /QaYWNMZ7kTRv+wWow9MhGiVdML4ZN4XnifuO5krqAybngIy66PMEoQCFEIsKKWv
    99iziAH0KBMVbxy03Trz
    -----END DSA PRIVATE KEY-----

  dsa_public: ssh-dss AAAAB3NzaC1kc3MAAACBAM/
↪Ycu7ulMTEvz1RLIzTbrhELJZf8Iwua6TFfQl1ubb1rHwUElOkus7xMhdVjms8AmbV1Meem7ImE69T0bszy09QAG3NImHgZVIeXBoJ/
↪JzByku/
↪1NcOBYilKP7oSIcLJpGUHX8IGn1GJoH7XRBwVub6Vqm4RP78C7q9IOn0hG2VAAAAFQCDEfCrnL1GGzhCPsr/
↪uS1vbt8/wQAAAIEAjSrok/4m8mbBkVp4IwxXFdRuqJKSj8/WWxos00Ednn/
↪ww5QibysHYULrOKJ1+54mmpMyp5CZICUQELCfCt5ScZ9GsqgmnI80Q1h3Xkwbo3kn7PzWwRwcV6muvJn4PcZ71WM+rdN/
↪c2EorAINDTbjRo97NueM94WbiYdtjHFxn0YAAACAXmLIFSQgiAPu459rCKxT46tHJtM0QfnNiEnQLbFluefZ/
↪yiI4DI38UzTCOXLhUA7ybmZha+D/csj15Y9/BNFuO7unzVhikCQV9DTeXX46pG4s1o23JKC/
↪QaYWNMZ7kTRv+wWow9MhGiVdML4ZN4XnifuO5krqAybngIy66PMEoQ= smoser@localhost

# By default, the fingerprints of the authorized keys for the users
# cloud-init adds are printed to the console. Setting
```

(continues on next page)

(continued from previous page)

```
# no_ssh_fingerprints to true suppresses this output.
no_ssh_fingerprints: false

# By default, (most) ssh host keys are printed to the console. Setting
# emit_keys_to_console to false suppresses this output.
ssh:
  emit_keys_to_console: false
```

Additional apt configuration and repositories

```
#cloud-config
# apt_pipelining (configure Acquire::http::Pipeline-Depth)
# Default: disables HTTP pipelining. Certain web servers, such
# as S3 do not pipeline properly (LP: #948461).
# Valid options:
#   False/default: Disables pipelining for APT
#   None/Unchanged: Use OS default
#   Number: Set pipelining to some number (not recommended)
apt_pipelining: False

## apt config via system_info:
# under the 'system_info', you can customize cloud-init's interaction
# with apt.
#  system_info:
#    apt_get_command: [command, argument, argument]
#    apt_get_upgrade_subcommand: dist-upgrade
#
# apt_get_command:
#  To specify a different 'apt-get' command, set 'apt_get_command'.
#  This must be a list, and the subcommand (update, upgrade) is appended to it.
#  default is:
#    ['apt-get', '--option=Dpkg::Options::=--force-confold',
#     '--option=Dpkg::options::=--force-unsafe-io', '--assume-yes', '--quiet']
#
# apt_get_upgrade_subcommand: "dist-upgrade"
#  Specify a different subcommand for 'upgrade. The default is 'dist-upgrade'.
#  This is the subcommand that is invoked for package_upgrade.
#
# apt_get_wrapper:
#   command: eatmydata
#   enabled: [True, False, "auto"]
#

# Install additional packages on first boot
#
# Default: none
#
# if packages are specified, then package_update will be set to true

packages: ['pastebinit']
```

(continues on next page)

(continued from previous page)

```
41
42 apt:
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
```

(continues on next page)

(continued from previous page)

```
  # 1.3 primary/security archives
  #
  # Default: none - instead it is auto select based on cloud metadata
  # so if neither "uri" nor "search", nor "search_dns" is set (the default)
  # then use the mirror provided by the DataSource found.
  # In EC2, that means using <region>.ec2.archive.ubuntu.com
  #
  # define a custom (e.g. localized) mirror that will be used in sources.list
  # and any custom sources entries for deb / deb-src lines.
  #
  # One can set primary and security mirror to different uri's
  # the child elements to the keys primary and secondary are equivalent
  primary:
    # arches is list of architectures the following config applies to
    # the special keyword "default" applies to any architecture not explicitly
    # listed.
    - arches: [amd64, i386, default]
      # uri is just defining the target as-is
      uri: http://us.archive.ubuntu.com/ubuntu
      #
      # via search one can define lists that are tried one by one.
      # The first with a working DNS resolution (or if it is an IP) will be
      # picked. That way one can keep one configuration for multiple
      # subenvironments that select the working one.
      search:
        - http://cool.but-sometimes-unreachable.com/ubuntu
        - http://us.archive.ubuntu.com/ubuntu
      # if no mirror is provided by uri or search but 'search_dns' is
      # true, then search for dns names '<distro>-mirror' in each of
      # - fqdn of this host per cloud metadata
      # - localdomain
      # - no domain (which would search 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>
      #
      # That gives the cloud provider the opportunity to set mirrors of a distro
      # up and expose them only by creating dns entries.
      #
      # if none of that is found, then the default distro mirror is used
      search_dns: true
      #
      # If multiple of a category are given
      #   1. uri
      #   2. search
      #   3. search_dns
      # the first defining a valid mirror wins (in the order as defined here,
      # not the order as listed in the config).
      #
      # Additionally, if the repository requires a custom signing key, it can be
      # specified via the same fields as for custom sources:
      #   'keyid': providing a key to import via shortid or fingerprint
      #   'key': providing a raw PGP key
```

(continues on next page)

(continued from previous page)

```
    #   'keyserver': specify an alternate keyserver to pull keys from that
    #                were specified by keyid
  - arches: [s390x, arm64]
    # as above, allowing to have one config for different per arch mirrors
# security is optional, if not defined it is set to the same value as primary
security:
  - uri: http://security.ubuntu.com/ubuntu
    arches: [default]
# If search_dns is set for security the searched pattern is:
#   <distro>-security-mirror

# if no mirrors are specified at all, or all lookups fail it will try
# to get them from the cloud datasource and if those neither provide one fall
# back to:
#   primary: http://archive.ubuntu.com/ubuntu
#   security: http://security.ubuntu.com/ubuntu

# 1.4 sources_list
#
# Provide a custom template for rendering sources.list
# without one provided cloud-init uses builtin templates for
# ubuntu and debian.
# Within these sources.list templates you can use the following replacement
# variables (all have sane Ubuntu defaults, but mirrors can be overwritten
# as needed (see above)):
# => $RELEASE, $MIRROR, $PRIMARY, $SECURITY
sources_list: | # written by cloud-init custom template
  deb $MIRROR $RELEASE main restricted
  deb-src $MIRROR $RELEASE main restricted
  deb $PRIMARY $RELEASE universe restricted
  deb $SECURITY $RELEASE-security multiverse

# 1.5 conf
#
# Any apt config string that will be made available to apt
# see the APT.CONF(5) man page for details what can be specified
conf: | # APT config
  APT {
    Get {
      Assume-Yes "true";
      Fix-Broken "true";
    };
  };

# 1.6 (http_|ftp_|https_)proxy
#
# Proxies are the most common apt.conf option, so that for simplified use
# there is a shortcut for those. Those get automatically translated into the
# correct Acquire::*::Proxy statements.
#
# note: proxy actually being a short synonym to http_proxy
proxy: http://[[user][:pass]@]host[:port]/
```

(continues on next page)

(continued from previous page)

```
http_proxy: http://[[user][:pass]@]host[:port]/
ftp_proxy: ftp://[[user][:pass]@]host[:port]/
https_proxy: https://[[user][:pass]@]host[:port]/

# 1.7 add_apt_repo_match
#
# 'source' entries in apt-sources that match this python regex
# expression will be passed to add-apt-repository
# The following example is also the builtin default if nothing is specified
add_apt_repo_match: '^[\w-]+:\w'


#############################################################################
# Section 2: source list entries
#
# This is a dictionary (unlike most block/net which are lists)
#
# The key of each source entry is the filename and will be prepended by
# /etc/apt/sources.list.d/ if it doesn't start with a '/'.
# If it doesn't end with .list it will be appended so that apt picks up its
# configuration.
#
# Whenever there is no content to be written into such a file, the key is
# not used as filename - yet it can still be used as index for merging
# configuration.
#
# The values inside the entries consist of the following optional entries:
#   'source': a sources.list entry (some variable replacements apply)
#   'keyid': providing a key to import via shortid or fingerprint
#   'key': providing a raw PGP key
#   'keyserver': specify an alternate keyserver to pull keys from that
#                were specified by keyid

# This allows merging between multiple input files than a list like:
# cloud-config1
# sources:
#   s1: {'key': 'key1', 'source': 'source1'}
# cloud-config2
# sources:
#   s2: {'key': 'key2'}
#   s1: {'keyserver': 'foo'}
# This would be merged to
# sources:
#   s1:
#     keyserver: foo
#     key: key1
#     source: source1
#   s2:
#     key: key2
#
# The following examples number the subfeatures per sources entry to ease
# identification in discussions.
```

(continues on next page)

(continued from previous page)

```
sources:
  curtin-dev-ppa.list:
    # 2.1 source
    #
    # Creates a file in /etc/apt/sources.list.d/ for the sources list entry
    # based on the key: "/etc/apt/sources.list.d/curtin-dev-ppa.list"
    source: "deb http://ppa.launchpad.net/curtin-dev/test-archive/ubuntu bionic main"

    # 2.2 keyid
    #
    # Importing a gpg key for a given key id. Used keyserver defaults to
    # keyserver.ubuntu.com
    keyid: F430BBA5 # GPG key ID published on a key server

  ignored1:
    # 2.3 PPA shortcut
    #
    # Setup correct apt sources.list line and Auto-Import the signing key
    # from LP
    #
    # See https://help.launchpad.net/Packaging/PPA for more information
    # this requires 'add-apt-repository'. This will create a file in
    # /etc/apt/sources.list.d automatically, therefore the key here is
    # ignored as filename in those cases.
    source: "ppa:curtin-dev/test-archive"    # Quote the string

  my-repo2.list:
    # 2.4 replacement variables
    #
    # sources can use $MIRROR, $PRIMARY, $SECURITY, $RELEASE and $KEY_FILE
    # replacement variables.
    # They will be replaced with the default or specified mirrors and the
    # running release.
    # The entry below would be possibly turned into:
    #   source: deb http://archive.ubuntu.com/ubuntu bionic multiverse
    source: deb [signed-by=$KEY_FILE] $MIRROR $RELEASE multiverse
    keyid: F430BBA5

  my-repo3.list:
    # this would have the same end effect as 'ppa:curtin-dev/test-archive'
    source: "deb http://ppa.launchpad.net/curtin-dev/test-archive/ubuntu bionic main"
    keyid: F430BBA5 # GPG key ID published on the key server
    filename: curtin-dev-ppa.list

  ignored2:
    # 2.5 key only
    #
    # this would only import the key without adding a ppa or other source spec
    # since this doesn't generate a source.list file the filename key is ignored
    keyid: F430BBA5 # GPG key ID published on a key server
```

(continues on next page)

(continued from previous page)

```
  ignored3:
    # 2.6 key id alternatives
    #
    # Keyid's can also be specified via their long fingerprints
    keyid: B59D 5F15 97A5 04B7 E230  6DCA 0620 BBCF 0368 3F77

  ignored4:
    # 2.7 alternative keyservers
    #
    # One can also specify alternative keyservers to fetch keys from.
    keyid: B59D 5F15 97A5 04B7 E230  6DCA 0620 BBCF 0368 3F77
    keyserver: pgp.mit.edu

  ignored5:
    # 2.8 signed-by
    #
    # One can specify [signed-by=$KEY_FILE] in the source definition, which
    # will make the key be installed in the directory /etc/cloud-init.gpg.d/
    # and the $KEY_FILE replacement variable will be replaced with the path
    # to the specified key. If $KEY_FILE is used, but no key is specified,
    # apt update will (rightfully) fail due to an invalid value.
    source: deb [signed-by=$KEY_FILE] $MIRROR $RELEASE multiverse
    keyid: B59D 5F15 97A5 04B7 E230  6DCA 0620 BBCF 0368 3F77

  my-repo4.list:
    # 2.9 raw key
    #
    # The apt signing key can also be specified by providing a pgp public key
    # block. Providing the PGP key this way is the most robust method for
    # specifying a key, as it removes dependency on a remote key server.
    #
    # As with keyid's this can be specified with or without some actual source
    # content.
    key: | # The value needs to start with -----BEGIN PGP PUBLIC KEY BLOCK-----
      -----BEGIN PGP PUBLIC KEY BLOCK-----
      Version: SKS 1.0.10

      mI0ESpA3UQEEALdZKVIMq0j6qWAXAyxSlF63SvPVIgxHPb9Nk0DZUixn+akqytxG4zKCONz6
      qLjoBBfHnynyVLfT4ihg9an1PqxRnTO+JKQxl8NgKGz6Pon569GtAOdWNKw15XKinJTDLjnj
      9y96ljJqRcpV9t/WsIcdJPcKFR5voHTEoABE2aEXABEBAAG0GUxhdW5jaHBhZCBQUEEgZm9y
      IEFsZXN0aWOItgQTAQIAIAUCSpA3UQIbAwYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAAoJEA7H
      5Qi+CcVxWZ8D/1MyYvfj3FJPZUm2Yo1zZsQ657vHI9+pPouqflWOayRR9jbiyUFIn0VdQBrP
      t0FwvnOFArUovUWoKAEdqR8hPy3M3APUZjl5K4cMZR/xaMQeQRZ5CHpS4DBKURKAHC0ltS5o
      uBJKQOZm5iltJp15cgyIkBkGe8Mx18VFyVglAZey
      =Y2oI
      -----END PGP PUBLIC KEY BLOCK-----
```

Disk setup

```
#cloud-config
# Cloud-init supports the creation of simple partition tables and filesystems
# on devices.

# Default disk definitions for AWS
# --------------------------------
# (Not implemented yet, but provided for future documentation)

disk_setup:
  ephemeral0:
    table_type: 'mbr'
    layout: True
    overwrite: False

fs_setup:
  - label: None,
    filesystem: ext3
    device: ephemeral0
    partition: auto

# Default disk definitions for Microsoft Azure
# ------------------------------------------

device_aliases: {'ephemeral0': '/dev/sdb'}
disk_setup:
  ephemeral0:
    table_type: mbr
    layout: True
    overwrite: False

fs_setup:
  - label: ephemeral0
    filesystem: ext4
    device: ephemeral0.1
    replace_fs: ntfs


# Data disks definitions for Microsoft Azure
# ------------------------------------------

disk_setup:
  /dev/disk/azure/scsi1/lun0:
    table_type: gpt
    layout: True
    overwrite: True

fs_setup:
  - device: /dev/disk/azure/scsi1/lun0
    partition: 1
    filesystem: ext4

```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
#                values which are specific to the cloud. For these devices
#                Cloud-init will look up what the real devices is and then
#                use it.
#
#                For other devices, the kernel device name is used. At this
#                time only simply kernel devices are supported, meaning
#                that device mapper and other targets may not work.
#
#                Note: At this time, there is no handling or setup of
#                device mapper targets.
#
#   table_type=<TYPE>: Currently the following are supported:
#                    'mbr': default and setups a MS-DOS partition table
#                    'gpt': setups a GPT partition table
#
#                Note: At this time only 'mbr' and 'gpt' partition tables
#                    are allowed. It is anticipated in the future that
#                    we'll also have "RAID" to create a mdadm RAID.
#
#   layout={...}: The device layout. This is a list of values, with the
#                percentage of disk that partition will take.
#                Valid options are:
#                    [<SIZE>, [<SIZE>, <PART_TYPE]]
#
#                Where <SIZE> is the _percentage_ of the disk to use, while
#                <PART_TYPE> is the numerical value of the partition type.
#
#                The following setups two partitions, with the first
#                partition having a swap label, taking 1/3 of the disk space
#                and the remainder being used as the second partition.
#                  /dev/xvdh':
#                    table_type: 'mbr'
#                    layout:
#                      - [33,82]
#                      - 66
#                    overwrite: True
#
#                When layout is "true" it means single partition the entire
#                device.
#
#                When layout is "false" it means don't partition or ignore
#                existing partitioning.
#
#                If layout is set to "true" and overwrite is set to "false",
#                it will skip partitioning the device without a failure.
#
#   overwrite=<BOOL>: This describes whether to ride with safetys on and
#                everything holstered.
#
#                'false' is the default, which means that:
#                    1. The device will be checked for a partition table
#                    2. The device will be checked for a filesystem
```

(continues on next page)

(continued from previous page)

```
#                   3. If either a partition of filesystem is found, then
#                       the operation will be _skipped_.
#
#               'true' is cowboy mode. There are no checks and things are
#                   done blindly. USE with caution, you can do things you
#                   really, really don't want to do.
#
#
# fs_setup: Setup the filesystem
# ------------------------------
#
# fs_setup describes the how the filesystems are supposed to look.

fs_setup:
  - label: ephemeral0
    filesystem: 'ext3'
    device: 'ephemeral0'
    partition: 'auto'
  - label: mylabl2
    filesystem: 'ext4'
    device: '/dev/xvda1'
  - cmd: mkfs -t %(filesystem)s -L %(label)s %(device)s
    label: mylabl3
    filesystem: 'btrfs'
    device: '/dev/xvdh'

# The general format is:
#   fs_setup:
#     - label: <LABEL>
#       filesystem: <FS_TYPE>
#       device: <DEVICE>
#       partition: <PART_VALUE>
#       overwrite: <OVERWRITE>
#       replace_fs: <FS_TYPE>
#
# Where:
#   <LABEL>: The filesystem label to be used. If set to None, no label is
#     used.
#
#   <FS_TYPE>: The filesystem type. It is assumed that the there
#     will be a "mkfs.<FS_TYPE>" that behaves likes "mkfs". On a standard
#     Ubuntu Cloud Image, this means that you have the option of ext{2,3,4},
#     and vfat by default.
#
#   <DEVICE>: The device name. Special names of 'ephemeralX' or 'swap'
#     are allowed and the actual device is acquired from the cloud datasource.
#     When using 'ephemeralX' (i.e. ephemeral0), make sure to leave the
#     label as 'ephemeralX' otherwise there may be issues with the mounting
#     of the ephemeral storage layer.
#
#     If you define the device as 'ephemeralX.Y' then Y will be interpetted
#     as a partition value. However, ephermalX.0 is the _same_ as ephemeralX.
```

(continues on next page)

(continued from previous page)

```
#
#   <PART_VALUE>:
#     Partition definitions are overwritten if you use the '<DEVICE>.Y' notation.
#
#     The valid options are:
#     "auto|any": tell cloud-init not to care whether there is a partition
#       or not. Auto will use the first partition that does not contain a
#       filesystem already. In the absence of a partition table, it will
#       put it directly on the disk.
#
#       "auto": If a filesystem that matches the specification in terms of
#       label, filesystem and device, then cloud-init will skip the creation
#       of the filesystem.
#
#       "any": If a filesystem that matches the filesystem type and device,
#       then cloud-init will skip the creation of the filesystem.
#
#       Devices are selected based on first-detected, starting with partitions
#       and then the raw disk. Consider the following:
#           NAME     FSTYPE LABEL
#           xvdb
#           |-xvdb1  ext4
#           |-xvdb2
#           |-xvdb3  btrfs  test
#           \-xvdb4  ext4   test
#
#         If you ask for 'auto', label of 'test, and filesystem of 'ext4'
#         then cloud-init will select the 2nd partition, even though there
#         is a partition match at the 4th partition.
#
#         If you ask for 'any' and a label of 'test', then cloud-init will
#         select the 1st partition.
#
#         If you ask for 'auto' and don't define label, then cloud-init will
#         select the 1st partition.
#
#         In general, if you have a specific partition configuration in mind,
#         you should define either the device or the partition number. 'auto'
#         and 'any' are specifically intended for formatting ephemeral storage
#         or for simple schemes.
#
#       "none": Put the filesystem directly on the device.
#
#       <NUM>: where NUM is the actual partition number.
#
#   <OVERWRITE>: Defines whether or not to overwrite any existing
#     filesystem.
#
#     "true": Indiscriminately destroy any pre-existing filesystem. Use at
#         your own peril.
#
#     "false": If an existing filesystem exists, skip the creation.
```

(continues on next page)

(continued from previous page)

```
#
#   <REPLACE_FS>: This is a special directive, used for Microsoft Azure that
#     instructs cloud-init to replace a filesystem of <FS_TYPE>. NOTE:
#     unless you define a label, this requires the use of the 'any' partition
#     directive.
#
# Behavior Caveat: The default behavior is to _check_ if the filesystem exists.
#   If a filesystem matches the specification, then the operation is a no-op.
```

Configure data sources

```
#cloud-config

# Documentation on data sources configuration options
datasource:
  # Ec2
  Ec2:
    # timeout: the timeout value for a request at metadata service
    timeout : 50
    # The length in seconds to wait before giving up on the metadata
    # service.  The actual total wait could be up to
    #   len(resolvable_metadata_urls)*timeout
    max_wait : 120

    #metadata_url: a list of URLs to check for metadata services
    metadata_urls:
     - http://169.254.169.254:80
     - http://instance-data:8773

  MAAS:
    timeout : 50
    max_wait : 120

    # there are no default values for metadata_url or oauth credentials
    # If no credentials are present, non-authed attempts will be made.
    metadata_url: http://mass-host.localdomain/source
    consumer_key: Xh234sdkljf
    token_key: kjfhgb3n
    token_secret: 24uysdfx1w4

  NoCloud:
    # default seedfrom is None
    # if found, then it should contain a url with:
    #    <url>/user-data and <url>/meta-data
    # seedfrom: http://my.example.com/i-abcde/
    seedfrom: None

    # fs_label: the label on filesystems to be searched for NoCloud source
    fs_label: cidata

```

(continues on next page)

(continued from previous page)

```
    # these are optional, but allow you to basically provide a datasource
    # right here
    user-data: |
      # This is the user-data verbatim
    meta-data:
      instance-id: i-87018aed
      local-hostname: myhost.internal

  SmartOS:
    # For KVM guests:
    # Smart OS datasource works over a serial console interacting with
    # a server on the other end. By default, the second serial console is the
    # device. SmartOS also uses a serial timeout of 60 seconds.
    serial_device: /dev/ttyS1
    serial_timeout: 60

    # For LX-Brand Zones guests:
    # Smart OS datasource works over a socket interacting with
    # the host on the other end. By default, the socket file is in
    # the native .zoncontrol directory.
    metadata_sockfile: /native/.zonecontrol/metadata.sock

    # a list of keys that will not be base64 decoded even if base64_all
    no_base64_decode: ['root_authorized_keys', 'motd_sys_info',
                       'iptables_disable']
    # a plaintext, comma delimited list of keys whose values are b64 encoded
    base64_keys: []
    # a boolean indicating that all keys not in 'no_base64_decode' are encoded
    base64_all: False
```

Create partitions and filesystems

```
#cloud-config
# Cloud-init supports the creation of simple partition tables and filesystems
# on devices.

# Default disk definitions for AWS
# --------------------------------
# (Not implemented yet, but provided for future documentation)

disk_setup:
  ephemeral0:
    table_type: 'mbr'
    layout: True
    overwrite: False

fs_setup:
  - label: None,
    filesystem: ext3
    device: ephemeral0
```

(continues on next page)

(continued from previous page)

```
    partition: auto

# Default disk definitions for Microsoft Azure
# ------------------------------------------

device_aliases: {'ephemeral0': '/dev/sdb'}
disk_setup:
  ephemeral0:
    table_type: mbr
    layout: True
    overwrite: False

fs_setup:
  - label: ephemeral0
    filesystem: ext4
    device: ephemeral0.1
    replace_fs: ntfs


# Data disks definitions for Microsoft Azure
# ------------------------------------------

disk_setup:
  /dev/disk/azure/scsi1/lun0:
    table_type: gpt
    layout: True
    overwrite: True

fs_setup:
  - device: /dev/disk/azure/scsi1/lun0
    partition: 1
    filesystem: ext4


# Default disk definitions for SmartOS
# ------------------------------------

device_aliases: {'ephemeral0': '/dev/vdb'}
disk_setup:
  ephemeral0:
    table_type: mbr
    layout: False
    overwrite: False

fs_setup:
  - label: ephemeral0
    filesystem: ext4
    device: ephemeral0.0

# Caveat for SmartOS: if ephemeral disk is not defined, then the disk will
#    not be automatically added to the mounts.

```

(continues on next page)

(continued from previous page)

```
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
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 #
```

(continues on next page)

(continued from previous page)

```
#   layout={...}: The device layout. This is a list of values, with the
#               percentage of disk that partition will take.
#               Valid options are:
#                   [<SIZE>, [<SIZE>, <PART_TYPE]]
#
#               Where <SIZE> is the _percentage_ of the disk to use, while
#               <PART_TYPE> is the numerical value of the partition type.
#
#               The following setups two partitions, with the first
#               partition having a swap label, taking 1/3 of the disk space
#               and the remainder being used as the second partition.
#                 /dev/xvdh':
#                   table_type: 'mbr'
#                   layout:
#                     - [33,82]
#                     - 66
#                   overwrite: True
#
#               When layout is "true" it means single partition the entire
#               device.
#
#               When layout is "false" it means don't partition or ignore
#               existing partitioning.
#
#               If layout is set to "true" and overwrite is set to "false",
#               it will skip partitioning the device without a failure.
#
#   overwrite=<BOOL>: This describes whether to ride with safetys on and
#               everything holstered.
#
#               'false' is the default, which means that:
#                   1. The device will be checked for a partition table
#                   2. The device will be checked for a filesystem
#                   3. If either a partition of filesystem is found, then
#                       the operation will be _skipped_.
#
#               'true' is cowboy mode. There are no checks and things are
#                   done blindly. USE with caution, you can do things you
#                   really, really don't want to do.
#
#
# fs_setup: Setup the filesystem
# ------------------------------
#
# fs_setup describes the how the filesystems are supposed to look.

fs_setup:
  - label: ephemeral0
    filesystem: 'ext3'
    device: 'ephemeral0'
    partition: 'auto'
  - label: mylabl2
```

(continues on next page)

(continued from previous page)

```
    filesystem: 'ext4'
    device: '/dev/xvda1'
  - cmd: mkfs -t %(filesystem)s -L %(label)s %(device)s
    label: mylabl3
    filesystem: 'btrfs'
    device: '/dev/xvdh'

# The general format is:
#   fs_setup:
#     - label: <LABEL>
#       filesystem: <FS_TYPE>
#       device: <DEVICE>
#       partition: <PART_VALUE>
#       overwrite: <OVERWRITE>
#       replace_fs: <FS_TYPE>
#
# Where:
#   <LABEL>: The filesystem label to be used. If set to None, no label is
#     used.
#
#   <FS_TYPE>: The filesystem type. It is assumed that the there
#     will be a "mkfs.<FS_TYPE>" that behaves likes "mkfs". On a standard
#     Ubuntu Cloud Image, this means that you have the option of ext{2,3,4},
#     and vfat by default.
#
#   <DEVICE>: The device name. Special names of 'ephemeralX' or 'swap'
#     are allowed and the actual device is acquired from the cloud datasource.
#     When using 'ephemeralX' (i.e. ephemeral0), make sure to leave the
#     label as 'ephemeralX' otherwise there may be issues with the mounting
#     of the ephemeral storage layer.
#
#     If you define the device as 'ephemeralX.Y' then Y will be interpetted
#     as a partition value. However, ephermalX.0 is the _same_ as ephemeralX.
#
#   <PART_VALUE>:
#     Partition definitions are overwritten if you use the '<DEVICE>.Y' notation.
#
#     The valid options are:
#     "auto|any": tell cloud-init not to care whether there is a partition
#       or not. Auto will use the first partition that does not contain a
#       filesystem already. In the absence of a partition table, it will
#       put it directly on the disk.
#
#       "auto": If a filesystem that matches the specification in terms of
#       label, filesystem and device, then cloud-init will skip the creation
#       of the filesystem.
#
#       "any": If a filesystem that matches the filesystem type and device,
#       then cloud-init will skip the creation of the filesystem.
#
#       Devices are selected based on first-detected, starting with partitions
#       and then the raw disk. Consider the following:
```

(continues on next page)

(continued from previous page)

```
#           NAME     FSTYPE LABEL
#           xvdb
#           |-xvdb1  ext4
#           |-xvdb2
#           |-xvdb3  btrfs  test
#           \-xvdb4  ext4   test
#
#         If you ask for 'auto', label of 'test, and filesystem of 'ext4'
#         then cloud-init will select the 2nd partition, even though there
#         is a partition match at the 4th partition.
#
#         If you ask for 'any' and a label of 'test', then cloud-init will
#         select the 1st partition.
#
#         If you ask for 'auto' and don't define label, then cloud-init will
#         select the 1st partition.
#
#         In general, if you have a specific partition configuration in mind,
#         you should define either the device or the partition number. 'auto'
#         and 'any' are specifically intended for formatting ephemeral storage
#         or for simple schemes.
#
#       "none": Put the filesystem directly on the device.
#
#       <NUM>: where NUM is the actual partition number.
#
#   <OVERWRITE>: Defines whether or not to overwrite any existing
#     filesystem.
#
#     "true": Indiscriminately destroy any pre-existing filesystem. Use at
#         your own peril.
#
#     "false": If an existing filesystem exists, skip the creation.
#
#   <REPLACE_FS>: This is a special directive, used for Microsoft Azure that
#     instructs cloud-init to replace a filesystem of <FS_TYPE>. NOTE:
#     unless you define a label, this requires the use of the 'any' partition
#     directive.
#
# Behavior Caveat: The default behavior is to _check_ if the filesystem exists.
#   If a filesystem matches the specification, then the operation is a no-op.
```

2.4.3 CLI commands

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

```
$ cloud-init --help
```

Example output:

```
usage: cloud-init [-h] [--version] [--file FILES] [--debug] [--force]
                                                          {init,modules,single,query,
↪dhclient-hook,features,analyze,devel,collect-logs,clean,status,schema} ...

 options:
   -h, --help            show this help message and exit
   --version, -v         Show program's version number and exit.
   --file FILES, -f FILES
                         Use additional yaml configuration files.
   --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,schema}
     init                Initialize cloud-init and perform initial modules.
     modules             Activate 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.
     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.
     schema              Validate cloud-config files using jsonschema.
```

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

analyze

Get detailed reports of where `cloud-init` spends its time during the boot process. For more complete reference see *Performance*.

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 initialisation, kernel finish initialisation, and `cloud-init` start.

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.
- **`--machine-id`**: Set `/etc/machine-id` to `uninitialized\n` on this image for systemd environments. On distributions without systemd, remove the file. Best practice when cloning a golden image, to ensure the next boot of that image auto-generates a unique machine ID. More details on machine-id.

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 using **`ubuntu-bug cloud-init`** to automatically attach these logs to a bug report.

---

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.

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.

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
```

Example output:

```
NETWORK_CONFIG_V1
NETWORK_CONFIG_V2
```

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*.

modules

Generally run by OS init systems to execute `modules:config` and `modules:final` boot stages. This executes cloud config *Module reference* 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.

query

Query standardised 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 standardised aliases:

```
$ cloud-init query --list-keys
```

Example output:

```
_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
```

Here are a few examples of how to query standardised metadata from clouds:

```
$ cloud-init query v1.cloud_name
```

Example output:

```
aws  # or openstack, azure, gce etc.
```

Any standardised `instance-data` under a <v#> key is aliased as a top-level key for convenience:

```
$ cloud-init query cloud_name
```

Example output:

```
aws  # or openstack, azure, gce etc.
```

One can also query datasource-specific metadata on EC2, e.g.:

```
$ cloud-init query ds.meta_data.public_ipv4
```

Note: The standardised instance data keys under **v#** are guaranteed not to change behaviour or format. If using top-level convenience aliases for any standardised 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. For example, we can 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
```

schema

Validate cloud-config files using jsonschema.

- **-h, --help**: Show this help message and exit.
- **-c CONFIG_FILE, --config-file CONFIG_FILE**: Path of the cloud-config YAML file to validate.
- **--system**: Validate the system cloud-config user data.
- **-d DOCS [cc_module ...], --docs DOCS [cc_module ...]**: Print schema module docs. Choices are: "all" or "space-delimited" `cc_names`.
- **--annotate**: Annotate existing cloud-config file with errors.

The following example checks a config file and annotates the config file with errors on `stdout`.

```
$ cloud-init schema -c ./config.yml --annotate
```

single

Attempt to run a single, named, cloud config module.

- **--name**: The cloud-config module name to run.
- **--frequency**: Module frequency for this run. One of (`always``|``once-per-instance``|``once`).
- **--report**: Enable reporting.

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.

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.
- **`--format [yaml|json|tabular]`**: Machine-readable JSON or YAML detailed output.

The **`status`** command can be used simply as follows:

```
$ cloud-init status
```

Which shows whether `cloud-init` is currently running, done, disabled, or in error, as in this example output:

```
status: running
```

The **`--long`** option, shown below, provides a more verbose output.

```
$ cloud-init status --long
```

Example output when `cloud-init` is running:

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

Example output when `cloud-init` is done:

```
status: done
boot_status_code: enabled-by-generator
last_update: Tue, 16 Aug 2022 19:12:58 +0000
detail:
DataSourceNoCloud [seed=/var/lib/cloud/seed/nocloud-net][dsmode=net]
```

The detailed output can be shown in machine-readable JSON or YAML with the **`format`** option, for example:

```
$ cloud-init status --format=json
```

Which would produce the following example output:

```
{
 "boot_status_code": "enabled-by-generator",
 "datasource": "nocloud",
 "detail": "DataSourceNoCloud [seed=/var/lib/cloud/seed/nocloud-net][dsmode=net]",
 "errors": [],
 "last_update": "Tue, 16 Aug 2022 19:12:58 +0000",
 "status": "done"
}
```

2.4.4 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`!

Distributions

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

- Alpine Linux
- Arch Linux
- Debian
- DragonFlyBSD
- Fedora
- FreeBSD
- Gentoo Linux
- NetBSD
- OpenBSD
- Photon OS
- RHEL/CentOS/AlmaLinux/Rocky Linux/EuroLinux
- SLES/openSUSE
- Ubuntu

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
- 3DS Outscale

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

- Bare metal installs
- OpenStack
- LXD
- KVM
- Metal-as-a-Service (MAAS)
- VMware

2.4.5 FAQ

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 our documentation
- Ask questions 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 to find out how to report one

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` did 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 information present from more than one boot.

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

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 initialisation, the datasources to evaluate on boot, as well as other settings.

See the *configuration sources explanation* and *configuration reference* pages for more details.

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 which datasource was identified and used to set up the system.

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

`data`

The `/var/lib/cloud/data` directory contain 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 that will show both the datasource used to set up the instance, and whether any errors occurred.
- `status.json`: JSON file showing the datasource used, a breakdown of all four modules, whether any errors occurred, and the start and stop times.

What datasource am I using?

To correctly set up an instance, `cloud-init` must correctly identify the cloud that it is on. Therefore, knowing which datasource is used on an instance launch can aid in debugging.

To find out which datasource is being used run the **`cloud-id`** command:

```
$ cloud-id
```

This will tell you which datasource is being used, for example:

```
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 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`.

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.**

How can I debug my user data?

Two of the most common issues with cloud config user data are:

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

Static user data validation

To verify your cloud config is valid YAML you may use validate-yaml.py.

To ensure that the keys and values in your user data are correct, you may run:

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

or to test YAML in a file:

```
$ cloud-init schema -c test.yml --annotate
```

Log analysis

If you can log into your system, the best way to debug your system is to check the contents of the log files `/var/log/cloud-init.log` and `/var/log/cloud-init-output.log` for warnings, errors, and tracebacks. Tracebacks are always reportable bugs.

Why did `cloud-init` never complete?

To check if `cloud-init` is running still, run:

```
$ cloud-init status
```

To wait for `cloud-init` to complete, run:

```
$ cloud-init status --wait
```

There are a number of reasons that `cloud-init` might never complete. This list is not exhaustive, but attempts to enumerate potential causes:

External reasons

- Failed dependent services in the boot.
- Bugs in the kernel or drivers.
- Bugs in external userspace tools that are called by `cloud-init`.

Internal reasons

- A command in `bootcmd` or `runcmd` that never completes (e.g., running **`cloud-init status --wait`** will wait forever on itself and never complete).
- Non-standard configurations that disable timeouts or set extremely high values ("never" is used in a loose sense here).

Failing to complete on `systemd`

`Cloud-init` consists of multiple services on `systemd`. If a service that `cloud-init` depends on stalls, `cloud-init` will not continue. If reporting a bug related to `cloud-init` failing to complete on `systemd`, please make sure to include the following logs.

```
$ systemd-analyze critical-chain cloud-init.target
$ journalctl --boot=-1
$ systemctl --failed
```

`autoinstall`, `preruncmd`, `postruncmd`

Since `cloud-init` ignores top level user data `cloud-config` keys, other projects such as Juju and Subiquity autoinstaller use a YAML-formatted config that combines `cloud-init`'s user data cloud-config YAML format with their custom YAML keys. Since `cloud-init` ignores unused top level keys, these combined YAML configurations may be valid `cloud-config` files, however keys such as `autoinstall`, `preruncmd`, and `postruncmd` are not used by `cloud-init` to configure anything.

Please direct bugs and questions about other projects that use `cloud-init` to their respective support channels. For Subiquity autoinstaller that is via IRC (`#ubuntu-server` on Libera) or Discourse. For Juju support see their discourse page.

Where can I learn more?

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

Videos:

- cloud-init - The Good Parts
- Perfect Proxmox Template with Cloud Image and Cloud Init [proxmox, cloud-init, template]
- cloud-init - Building clouds one Linux box at a time (Video)
- Metadata and cloud-init
- Introduction to cloud-init

Blog Posts:

- cloud-init - The cross-cloud Magic Sauce (PDF)
- cloud-init - Building clouds one Linux box at a time (PDF)
- The beauty of cloud-init
- Cloud-init Getting Started [fedora, libvirt, cloud-init]
- Build Azure Devops Agents With Linux cloud-init for Dotnet Development [terraform, azure, devops, docker, dotnet, cloud-init]
- Cloud-init Getting Started [fedora, libvirt, cloud-init]
- Setup Neovim cloud-init Completion [neovim, yaml, Language Server Protocol, jsonschema, cloud-init]

Events:

- cloud-init Summit 2019
- cloud-init Summit 2018

- cloud-init Summit 2017

Whitepapers:

- Utilising cloud-init on Microsoft Azure (Whitepaper)
- Cloud Instance Initialization with cloud-init (Whitepaper)

2.4.6 Merging user data sections

The ability to merge user data sections is a feature that was implemented by popular request. It was identified that there should be a way to specify how cloud-config YAML "dictionaries" provided as user data are handled when there are multiple YAML files to be merged together (e.g., when performing an #include).

The previous merging algorithm was very simple and would only overwrite (and not append). So, it was decided to create a new and improved way to merge dictionaries (and their contained objects) together in a customisable way, thus allowing 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 prefer:

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

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

Built-in mergers

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

- **`Dict`**
- **`List`**
- **`String`**

`Dict`

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.

`List`

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.

String

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

These are the 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.

Customisation

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 customised merging was introduced through *merge classes*.

A *merge class* is a class definition providing 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:
    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.
    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 an `_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 analysing the object types to merge, and attempting to find a known object that will merge that type. An example of this can be found in the `mergers/__init__.py` file (see `LookupMerger` and `UnknownMerger`).

So, following the typical `cloud-init` approach 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 attempting to merge, or a string merger can be told to append strings instead of discarding other strings to merge with.

How to activate

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

1. The first way involves the usage of MIME messages in `cloud-init` to specify multi-part documents (this is one way in which multiple cloud-config can be 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 to specify the *merge type* in the body of the cloud-config dictionary. There are two 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 following string format is expected:

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

The `class name` 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 upon construction of that class.

The following example shows the default string that gets used when none is otherwise provided:

```
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 dictionary equivalent of the default string format.

Specifying multiple types, and what this does

Now you may be asking yourself: "What exactly happens if I specify a `merge-type` header or dictionary for every cloud-config I provide?"

The answer is that when merging, a stack of `'merging classes'` is kept. The first one in the stack is the default merging class. 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. In this way a cloud-config can decide how it will merge with a cloud-config dictionary coming after it.

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, to take advantage of the full benefits (and customisation) here 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* configuration types. As was the case before merging was implemented, user data will overwrite `'conf.d'` configuration without merging.

Example cloud-config

A common request is to include multiple `runcmd` 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]

runcmd:
  - bash1
  - bash2
```

The second config:

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

runcmd:
  - bash3
  - bash4
```

2.4.7 Datasources

Datasources are sources of configuration data for `cloud-init` that typically come from the user (i.e., user data) or come from the cloud that created the configuration drive (i.e., metadata). Typical user data includes files, YAML, and shell scripts whereas typical metadata includes 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), a datasource abstract class was internally created to allow for a single way to access the different cloud systems methods, providing 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.

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/
```

Example output:

```
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
```

Example output:

```
dns-conf/
eipv4
hostname
image-id
instance-id
instance/
mac
network-type
network/
ntp-conf/
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
```

Example output:

```
#!/bin/sh
echo "Hello World."
```

AltCloud

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

RHEVm

For RHEVm v3.0 the user data 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
IyEvYmluL2Jhc2gKZWNobyAiSGVsbG8gSm9lISIgPj4gL3RtcC9KSlZfSm9lX291dC50eHQK
```

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:IyEvYmluL2Jhc2gKZWNobyAiSGVsbG8gSm9lISIgPj4gL3RtcC9KSlZfSm9lX291dC50eHQK
```

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.

vSphere

For VMWare's vSphere the user data is injected into the VM as an ISO via the CD-ROM. 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 CD-ROM.

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

Amazon EC2

The EC2 datasource is the oldest and most widely used datasource that `cloud-init` supports. This datasource interacts with a *magic* IP provided to the instance by the cloud provider (typically this IP is `169.254.169.254`). At this IP a http server is provided to the instance so that the instance can make calls to get instance user data 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/
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

User data 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 EC2 Metadata versions of this data provided to instances. `Cloud-init` attempts to use the most recent API version it supports in order to get the latest API features and instance-data. If a given API version is not exposed to the instance, those API features will be unavailable to the instance.

EC2 version	supported instance-data/feature
2021-03-23	Required for Instance tag support. This feature must be enabled individually on each instance. See the EC2 tags user guide.
2016-09-02	Required for secondary IP address support.
2009-04-04	Minimum supports EC2 API version for metadata and user data.

To see which versions are supported by 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 settings

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://{[}fd00:ec2::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 a 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 the 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: Virtual Private Cloud (VPC) ones 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 on 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 have 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. This CD contains a `ovf-env.xml` file that provides some information. Additional information is obtained via interaction with the "endpoint".

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 pre-provisioning gate which blocks instance configuration until Azure fabric is ready to provision.
- Retrieving SSH public keys. `Cloud-init` will first try to utilise SSH keys returned from IMDS, and if they are not provided from IMDS then it will fall back 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.

- **data_dir**

  Path used to read metadata files and write crawled data.

- **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
    disk_aliases:
      ephemeral0: /dev/disk/cloud/azure_resource
```

User data

User data is provided to `cloud-init` inside the `ovf-env.xml` file. `Cloud-init` expects that user data will be provided as a base64 encoded value inside the text child of an element named `UserData` or `CustomData`, which is a direct child of the `LinuxProvisioningConfigurationSet` (a sibling to `UserName`).

If both `UserData` and `CustomData` are provided, the behaviour 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>dGhpcyBpcyBteSB1c2VyZGF0YQ===</CustomData>
  <dscfg>eyJhZ2VudF9jb21tYW5kIjogWyJzdGFydCIsICJ3YWxpbnV4YWdlbnQiXX0=</dscfg>
  <DisableSshPasswordAuthentication>true</DisableSshPasswordAuthentication>
```

(continues on next page)

(continued from previous page)

```
  <SSH>
   <PublicKeys>
    <PublicKey>
     <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 behaviour 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 a network adapter is **NOT** a requirement. See server context in their 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 can 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 having 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 the value. If this field does not exist, the default value is "net".

CloudStack

Apache CloudStack exposes user data, metadata, user password, and account SSH key through the `virtual router`. The datasource obtains the `virtual router` address via DHCP lease information given to the instance. For more details on metadata and user data, refer to the CloudStack Administrator Guide.

The following URLs provide to access user data and metadata 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

Example

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.

By default, `cloud-init` *always* considers this source to be a fully-fledged datasource. Instead, the typical behavior is to assume it is really only present to provide networking information. `Cloud-init` will copy 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).

Note: See the config drive extension and metadata introduction in the public documentation for more information.

Version 1 (deprecated)

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 use 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 ssk 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 use a config drive:

1. Must be formatted with vfat or iso9660 filesystem, or have a *filesystem* label of `config-2` or `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)
```

(continues on next page)

(continued from previous page)

```
  - content/
    - 0000 (referenced content files)
    - 0001
    - ....
ec2
  - latest/
    - meta-data.json (not mandatory)
```

Keys and values

`Cloud-init`'s behaviour can be modified by keys found in the `meta.js` (version 1 only) file in the following ways.

ds-mode

```
dsmode:
  values: local, net, pass
  default: pass
```

This is what indicates if config drive is a final datasource 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 to search for other datasources 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

```
instance-id:
  default: iid-dsconfigdrive
```

This is utilised 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

```
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

```
user-data:
  default: None
```

This provides `cloud-init` user data. See *examples* for details of what needs to 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 in the DigitalOcean metadata documentation.

Configuration

DigitalOcean's datasource can be configured as follows:

```
datasource:
  DigitalOcean:
    retries: 3
    timeout: 2
```

- `retries`

  Specifies the number of times to attempt connection to the metadata service.

- `timeout`

  Specifies 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 base 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 a http request.

  Defaults to `6`.

Example

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/no datasource

This is the fallback datasource when no other datasource can be selected. It is the equivalent of an empty datasource, in that it provides an empty string as user data, and an empty dictionary as metadata.

It is useful for testing, as well as for occasions when you do not need an actual datasource to meet your instance requirements (i.e. 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 metadata 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 attempted for a http request. This value is used only after `metadata_url` is selected.

  Default: 5

- `sec_between_retries`

  The amount of wait time between retries when crawling the metadata service.

  Default: 1

Example

An example configuration with the default values is provided below:

```
datasource:
  GCE:
    retries: 5
    sec_between_retries: 1
```

LXD

The LXD datasource allows the user to provide custom user data, vendor data, metadata 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 the `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 is detected as viable by `ds-identify` during `systemd` generator time when either `/dev/lxd/sock` exists, or `/sys/class/dmi/id/board_name` matches "LXD".

The LXD datasource provides `cloud-init` with the ability to react to metadata, vendor data, user data and network-config changes, and to render the updated configuration across a system reboot.

To modify which metadata, vendor data or user data are provided to the launched container, use either LXD profiles or `lxc launch ... -c <key>="<value>"` at initial container launch, by setting one of the following keys:

- `user.meta-data`: YAML metadata which will be appended to base metadata.
- `user.vendor-data`: YAML which overrides any metadata values.
- `user.network-config`: YAML representing either *Networking config Version 1* or *Networking config Version 2* format.
- `user.user-data`: YAML which takes precedence and overrides both metadata and vendor data values.
- `user.any-key`: Custom user configuration key and value pairs, which 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.

Note: LXD version 4.22 introduced a new scope of config keys prefaced by `cloud-init.`, which are preferred above the related `user.*` keys:

- `cloud-init.meta-data`
- `cloud-init.vendor-data`
- `cloud-init.network-config`
- `cloud-init.user-data`

Configuration

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.

Hotplug

Network hotplug functionality is supported for the LXD datasource as described in the *Events and updates* documentation. As hotplug functionality relies on the cloud-provided network metadata, the LXD datasource will only meaningfully react to a hotplug event if it has the configuration necessary to respond to the change. Practically, this means that even with hotplug enabled, **the default behavior for adding a new virtual NIC will result in no change**.

To update the configuration to be used by hotplug, first pass the network configuration via the `cloud-init.network-config` (or `user.network-config` on older versions).

Example

Given an LXD instance named `my-lxd` with hotplug enabled and an LXD bridge named `my-bridge`, the following will allow for additional DHCP configuration of `eth1`:

```
$ cat /tmp/cloud-network-config.yaml
version: 2
ethernets:
    eth0:
        dhcp4: true
    eth1:
        dhcp4: true

$ lxc config set my-lxd cloud-init.network-config="$(cat /tmp/cloud-network-config.yaml)"
$ lxc config device add my-lxd eth1 nic name=eth1 nictype=bridged parent=my-bridge
Device eth1 added to my-lxd
```

MAAS

For now see: https://maas.io/docs

NoCloud

The data source `NoCloud` allows the user to provide user data and metadata to the instance without running a network service (or even without having a network at all).

You can provide metadata 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 metadata via the 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]
```

Permitted keys

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://` and end with a trailing `/`.

Cloud-init performs variable expansion of the `seedfrom` URL for any DMI kernel variables present in `/sys/class/dmi/id` (kenv on FreeBSD). Your `seedfrom` URL can contain variable names of the format `__dmi.varname__` to indicate to the `cloud-init` NoCloud datasource that `dmi.varname` should be expanded to the value of the DMI system attribute wanted.

Table 1: Available DMI variables for expansion in `seedfrom` URL

`dmi.baseboard-asset-tag`	`dmi.baseboard-manufacturer`	`dmi.baseboard-version`
`dmi.bios-release-date`	`dmi.bios-vendor`	`dmi.bios-version`
`dmi.chassis-asset-tag`	`dmi.chassis-manufacturer`	`dmi.chassis-serial-number`
`dmi.chassis-version`	`dmi.system-manufacturer`	`dmi.system-product-name`
`dmi.system-serial-number`	`dmi.system-uuid`	`dmi.system-version`

For example, you can pass this option to QEMU:

```
-smbios type=1,serial=ds=nocloud-net;s=http://10.10.0.1:8000/__dmi.chassis-serial-number_
↪_/
```

This will cause NoCloud to fetch the full metadata from a URL based on YOUR_SERIAL_NUMBER as seen in `/sys/class/dmi/id/chassis_serial_number` (kenv on FreeBSD) from http://10.10.0.1:8000/YOUR_SERIAL_NUMBER/meta-data after the network initialisation is complete.

File formats

These user data and metadata files are required as separate files at the same base URL:

```
/user-data
/meta-data
```

Both files must 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 adhering to *user data formats* at the same base URL:

```
/vendor-data
```

Creating a disk

Given a disk Ubuntu cloud image in `disk.img`, you can create a sufficient disk by following the following example.

1. Create the `user-data` and `meta-data` files that will be used to modify the image on first boot.

```
$ echo -e "instance-id: iid-local01\nlocal-hostname: cloudimg" > meta-data
$ echo -e "#cloud-config\npassword: passw0rd\nchpasswd: { expire: False }\nssh_pwauth:␣
↪True\n" > user-data
```

2. At this stage you have three options:

   a. Create a disk to attach with some user data and metadata:

   ```
   $ genisoimage  -output seed.iso -volid cidata -joliet -rock user-data meta-data
   ```

   b. Alternatively, create a `vfat` filesystem with the same files:

   ```
   $ truncate --size 2M seed.iso
   $ mkfs.vfat -n cidata seed.iso
   ```

   - 2b) Option 1: mount and copy files:

   ```
   $ sudo mount -t vfat seed.iso /mnt
   $ sudo cp user-data meta-data /mnt
   $ sudo umount /mnt
   ```

   - 2b) Option 2: the `mtools` package provides `mcopy`, which can access `vfat` filesystems without mounting them:

   ```
   $ mcopy -oi seed.iso user-data meta-data
   ```

3. Create a new qcow image to boot, backed by your original image:

```
$ qemu-img create -f qcow2 -b disk.img -F qcow2 boot-disk.img
```

4. Boot the image and log in as "Ubuntu" with password "passw0rd":

```
$ kvm -m 256 \
   -net nic -net user,hostfwd=tcp::2222-:22 \
   -drive file=boot-disk.img,if=virtio \
   -drive driver=raw,file=seed.iso,if=virtio
```

Note: Note that "passw0rd" was set as password through the user data above. There is no password set on these images.

Note: 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 the `instance-id`, 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 `meta-data`.

Example `meta-data`

```
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.

Example config

```
version: 1
config:
   - type: physical
     name: interface0
     mac_address: "52:54:00:12:34:00"
     subnets:
        - type: static
          address: 192.168.1.10
```

(continues on next page)

(continued from previous page)

```
          netmask: 255.255.255.0
          gateway: 192.168.1.254
```

```
version: 2
ethernets:
  interface0:
    match:
      macaddress: "52:54:00:12:34:00"
    set-name: interface0
    addresses:
      - 192.168.1.10/255.255.255.0
    gateway4: 192.168.1.254
```

NWCS

The NWCS datasource retrieves basic configuration values from the locally accessible metadata service. All data is served over HTTP from the address `169.254.169.254`.

Configuration

The NWCS datasource can be configured as follows:

```
datasource:
  NWCS:
    url: 'http://169.254.169.254'
    retries: 3
    timeout: 2
    wait: 2
```

- `url`: The URL used to acquire the metadata configuration.
- `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.

OpenNebula

The OpenNebula (ON) datasource supports the contextualisation disk.

OpenNebula's virtual machines are contextualised (parametrised) by CD-ROM image, which contains a shell script `context.sh`, with custom variables defined on virtual machine start. There are no fixed contextualisation variables, but the datasource accepts many used and recommended across the documentation.

Datasource configuration

Datasource accepts the following configuration options:

```
dsmode:
  values: local, net, disabled
  default: net
```

These specify whether the datasource will be processed in `local` (pre-networking) stage, `net` (post-networking) stage or be `disabled`.

```
parseuser:
  default: nobody
```

Unprivileged system user used for contextualisation script processing.

Contextualisation 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 the file `context.sh` with contextualisation variables. The file is generated by OpenNebula and has a `KEY='VALUE'` format that can be easily read by bash.

Contextualisation variables

There are no fixed or standard contextualisation variables in OpenNebula. The following variables were found in various places and in revisions of the OpenNebula documentation. Where multiple similar variables are specified, only the one found first is taken.

- `DSMODE`: Datasource mode configuration override. Values are: `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 a hostname from the instance's IP address in `local` dsmode. In `net` dsmode, `cloud-init` tries to resolve one of its IP addresses to get the 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:

- May be 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 `HUAWEICLOUD`, `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 as follows:

`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 attempted 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 instance.

Default: True

Example configuration

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 our *explanation topic*.

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 the 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/`).

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).

Example configuration

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 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`.

Rbx Cloud

The Rbx datasource consumes the metadata drive available on the HyperOne and Rootbox platforms.

This datasource supports network configurations, hostname, user accounts and user metadata.

Metadata drive

Drive metadata is a FAT-formatted partition with the `CLOUDMD` or `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 virtualisation platform (i.e., Joyent).

Please see http://smartos.org/ for information about SmartOS.

SmartOS platform

The SmartOS virtualisation platform uses metadata from 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.

Metadata channels

`Cloud-init` supports three modes of delivering user data and metadata via the flexible channels of SmartOS.

1. User data is written to `/var/db/user-data`:
   - As per the spec, user data is for consumption by the end user, not provisioning tools.
   - `Cloud-init` 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 metadata is maintained as `/var/db/user-data.<timestamp>`. `<timestamp>` is the epoch time when `cloud-init` ran.
2. `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 `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` metadata key goes missing, `user-script` is removed from the file system, although a backup is maintained.
- If the script does not start with a shebang (i.e., it starts with #!<executable>), or it is not an executable, `cloud-init` will add a shebang of "#!/bin/bash".

3. `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., they must have a shebang).

`Cloud-init` supports reading the traditional metadata 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
 - 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.

The cloud-config needs to be delivered over the `cloud-init:user-data` channel in order for `cloud-init` to ingest it.

Note: Unless you have an explicit use-case, it is recommended that you do not disable the per-boot script execution, especially if you are using any of the life-cycle management features of SmartOS.

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 the *datasource base configuration* variable `no_base64_decode`.

This means that `user-script`, `user-data` and other values can be Base64 encoded. Since `cloud-init` can only guess whether or not something is truly Base64 encoded, the following metadata 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 an entry exists in the metadata for `'b64-<key>'`, then `'b64-<key>'` is expected to be a plain-text 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` in 2 ways:

1. Through the datasource config, you can change the 'alias' of `ephermeral0` to reference another device. The default is:

   ```
   'disk_aliases': {'ephemeral0': '/dev/vdb'}
   ```

   This means that 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. 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.

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 the 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:

- Guest OS Customization
- GuestInfo keys

The configuration method is dependent upon the transport.

Guest OS customisation

The following configuration can be set for this datasource in cloud-init configuration (in /etc/cloud/cloud.cfg or /etc/cloud/cloud.cfg.d/).

System configuration

- disable_vmware_customization: true (disable) or false (enable) the VMware traditional Linux guest customisation. Traditional Linux guest customisation is customising a Linux virtual machine with a traditional Linux customisation specification. Setting this configuration to false is required to make sure this datasource is found in ds-identify when using Guest OS customisation transport.

  Default: true

Datasource configuration

- allow_raw_data: true (enable) or false (disable) the VMware customisation using cloud-init metadata and user data directly. Since vSphere 7.0 Update 3 version, users can create a Linux customisation specification with minimal cloud-init metadata and user data, and apply this specification to a virtual machine. This datasource will parse the metadata and user data and configure the virtual machine with them. See Guest customisation using cloud-init for more information.

  Default: true

- vmware_cust_file_max_wait: The maximum amount of clock time (in seconds) that should be spent waiting for VMware customisation files.

  Default: 15

Configuration examples

1. Create the `/etc/cloud/cloud.cfg.d/99-vmware-guest-customization.cfg` file with the following content, which will enable VMware customisation and set the maximum waiting time for the VMware customisation file to 10 seconds:

```
disable_vmware_customization: false
datasource:
  VMware:
    vmware_cust_file_max_wait: 10
```

2. Create the `/etc/cloud/cloud.cfg.d/99-vmware-guest-customization.cfg` file with the following content, which will enable VMware customisation but will only try to apply a traditional Linux Guest Customisation configuration, and set the maximum waiting time for the VMware customisation file to 10 seconds:

```
disable_vmware_customization: false
datasource:
  VMware:
    allow_raw_data: false
    vmware_cust_file_max_wait: 10
```

VMware Tools configuration

VMware Tools is required for this datasource's configuration settings, as well as vCloud and vSphere admin configuration. Users can change the VMware Tools configuration options with the following command:

```
vmware-toolbox-cmd config set <section> <key> <value>
```

The following VMware Tools configuration option affects this datasource's behaviour when applying customisation configuration with custom scripts:

- `[deploypkg] enable-custom-scripts`: If this option is absent in VMware Tools configuration, the custom script is disabled by default for security reasons. Some VMware products could change this default behaviour (for example: enabled by default) via customisation of the specification settings.

  VMware admins can refer to customization configuration and set the customisation specification settings.

For more information, see VMware vSphere Product Documentation and specific VMware Tools configuration options.

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.

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": {
                "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": {
```

(continues on next page)

(continued from previous page)

```
                    "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 (GuestInfo keys transport only)

Sometimes the `cloud-init` user data 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 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 a `/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 of GuestInfo keys transport

The following series of steps is a demonstration of how to configure a VM with this datasource using the GuestInfo keys transport:

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:
   - default
   - name: akutz
   ```

(continues on next page)

(continued from previous page)

```
    primary_group: akutz
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: sudo, wheel
    lock_passwd: true
    ssh_authorized_keys:
    - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDE0c5FczvcGSh/tG4iw+Fhfi/O5/EvUM/
↪96js65tly4++YTXK1d9jcznPS5ruDlbIZ30oveCBd3kT8LLVFwzh6hepYTf0YmCTpF4eDunyqmpCXDvVscQYRXyasEm5olGmV
↪mt2PcPjooeX00vAj81jjU2f3XKrjjz2u2+KIt9eba+vOQ6HiC8c2IzRkUAJ5i1atLy8RIbejo23+0P4N2jjk17QySFOVHwPBI
↪0M/4ideeU74EN/
↪CgVsvO6JrLsPBR4dojkV5qNbMNxIVv5cUwIy2ThlLgqpNCeFIDLCWNZEFKlEuNeSQ2mPtIO7ETxEL2Cz5y/
↪7AIuildzYMc6wi2bofRC8HmQ7rMXRWdwLKWsR0L7SKjHblIwarxOGqLnUI+k2E71YoP7SZSlxaKi17pqkr0OMCF+kKqvcvHAC
↪tZKSpjYtjMb5+RonfhaFRNzvj7cCE1f3Kp8UVqAdcGBTtReoE8eRUT63qIxjw03a7VwAyB2w+9cu1R9/
↪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 --logs --machine-id
```

   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 user data:

```
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 user data 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"
```

Note: 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 -on "${VM}"
```

If all went according to plan, the CentOS box is:

- Locked down, allowing SSH access only for the user in the user data.
- Configured for a dynamic IP address via DHCP.
- Has a hostname of `cloud-vm`.

Examples of 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 *Version 1* or *Version 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 the metadata service documentation.

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.
- `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 an AWS EC2 metadata service, but with different datasource identity. More information about ZStack can be found at ZStack.

Discovery

To determine whether a VM is running on the ZStack platform, `cloud-init` checks DMI information via `dmidecode -s chassis-asset-tag`. If the output ends with `.zstack.io`, it's running on the ZStack platform.

Metadata

The same way as with EC2, instance metadata can be queried at:

```
GET http://169.254.169.254/2009-04-04/meta-data/
instance-id
local-hostname
```

User data

The same way as with EC2, instance user data can be queried at:

```
GET http://169.254.169.254/2009-04-04/user-data/
meta_data.json
user_data
password
```

Datasource 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 built-in 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`, which 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`.

2.4.8 Network configuration

Default behaviour

`Cloud-init` 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.

Disabling network configuration

Users may disable `cloud-init`'s network configuration capability and rely on other methods, such as embedded configuration or other customisations.

`cloud-init` supports the following methods for disabling `cloud-init`.

Kernel command line

`Cloud-init` will 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 initialised until after the network has been brought up. In this case, `cloud-init` will attempt to bring up the interfaces specified by the datasource metadata using a network activator discovered by cloudinit.net.activators.select_activators.

This behaviour 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
```

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 as 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.

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
- *LXD*
  - LXD
- *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, `cloud-init` will prefer to use the other, newer 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 customise 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 *Base configuration*.

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`.

- `subnets`: Subnet/IP

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: It is best practice to "quote" all MAC addresses, since an unquoted MAC address might be incorrectly interpreted as an integer in YAML.

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, which is 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 are not available at configuration time. It's possible to specify a value too large or to 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's name must be less than 15 characters. Names exceeding the maximum will be truncated. This is a limitation of the Linux kernel network-device structure.

`mac_address: <MAC Address>`

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

Note: It is best practice to "quote" all MAC addresses, since an unquoted MAC address might be incorrectly interpreted as an integer in YAML.

`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 are not available at configuration time. It's possible to specify a value too large or to 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:
       - 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
       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 are not available at configuration time. It's possible to specify a value too large or to 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
```

(continues on next page)

(continued from previous page)

```
      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 aliases, 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 State-Less 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'
      subnets:
        - type: static
          address: 192.168.23.14/27
          gateway: 192.168.23.1
          dns_nameservers:
            - 192.168.23.2
            - 8.8.8.8
```

(continues on next page)

(continued from previous page)

```
        dns_search:
          - exemplary.maas
```

Multiple subnet example

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

```
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
```

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
```

(continues on next page)

(continued from previous page)

```
- id: eth1
  mac_address: 'd4:be:d9:a8:49:15'
  mtu: 1500
  name: eth1
  subnets:
  - address: 10.245.188.2/24
    dns_nameservers: []
    type: static
  type: physical
- id: eth1.2667
  mtu: 1500
  name: eth1.2667
  subnets:
  - address: 10.245.184.2/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2667
  vlan_link: eth1
- id: eth1.2668
  mtu: 1500
  name: eth1.2668
  subnets:
  - address: 10.245.185.1/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2668
  vlan_link: eth1
- id: eth1.2669
  mtu: 1500
  name: eth1.2669
  subnets:
  - address: 10.245.186.1/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2669
  vlan_link: eth1
- id: eth1.2670
  mtu: 1500
  name: eth1.2670
  subnets:
  - address: 10.245.187.2/24
    dns_nameservers: []
    type: static
  type: vlan
  vlan_id: 2670
  vlan_link: eth1
- address: 10.245.168.2
  search:
  - dellstack
```

(continues on next page)

(continued from previous page)

```
    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. Writing 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 *Base configuration*.

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 *Network configuration* for other places.

Supported device `types` values are as follows:

- `ethernets`: Ethernets
- `bonds`: Bonds
- `bridges`: Bridges
- `vlans`: VLANs

Each `type` block contains device definitions as a map (where the keys are 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 (e.g., 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 (e.g., 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.

Example:

```
# all cards on second PCI bus
match:
  name: enp2*
```

macaddress: *<(scalar)>*

Device's MAC address in the form xx:xx:xx:xx:xx:xx. Globs are not allowed. Letters must be lowercase.

Example:

```
# fixed MAC address
match:
  macaddress: "11:22:33:aa:bb:ff"
```

Note: It is best practice to "quote" all MAC addresses, since an unquoted MAC address might be incorrectly interpreted as an integer in YAML.

driver: *<(scalar)>*

Kernel driver name, corresponding to the `DRIVER` udev property. Globs are supported. Matching on driver is *only* supported with `networkd`.

Example:

```
# 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 (e.g., in `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.

dhcp4-overrides and dhcp6-overrides: <(mapping)>

DHCP behaviour overrides. Overrides will only have an effect if the corresponding DHCP type is enabled. Refer to Netplan#dhcp-overrides for more documentation.

Note: These properties are only consumed on `netplan` and `networkd` renderers.

The `netplan` renderer *passes through* everything and the `networkd` renderer consumes the following sub-properties:

- `hostname` *
- `route-metric` *
- `send-hostname` *
- `use-dns`
- `use-domains`
- `use-hostname`
- `use-mtu` *
- `use-ntp`
- `use-routes` *

Note: Sub-properties marked with a * are unsupported for `dhcp6-overrides` when used with the `networkd` renderer.

Example:

```
dhcp4-overrides:
  hostname: hal
  route-metric: 1100
  send-hostname: false
  use-dns: false
  use-domains: false
  use-hostname: false
  use-mtu: false
  use-ntp: false
  use-routes: false
```

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 recognised 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)>

Deprecated, see Netplan#default-routes. Set default gateway for IPv4/6, for manual address configuration. This requires setting `addresses` too. Gateway IPs must be in a form recognised 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)>

Customisation 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 LACPDUs 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 behaviour 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`.

`gratuitous-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)>

Customisation parameters for special bridging options. Time values are specified in seconds unless otherwise stated.

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 a 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: {...}
vlans:
  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:
```

(continues on next page)

(continued from previous page)

```
  # 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
  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: [wlp1s0, switchports]
    dhcp4: true
vlans:
  en-intra:
    id: 1
    link: id0
    dhcp4: yes
```

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`:

NetworkManager

NetworkManager is the standard Linux network configuration tool suite. It supports a wide range of networking setups. Configuration is typically stored in `/etc/NetworkManager`.

It is the default for a number of Linux distributions; notably Fedora, CentOS/RHEL, and their derivatives.

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.

NetBSD, OpenBSD, FreeBSD

Network renders supporting BSD releases, which typically write configuration to `/etc/rc.conf`. Unique to BSD renderers is that each renderer also calls something akin to FreeBSD.start_services which will invoke applicable network services to setup the network, making network activators unneeded for BSD flavors at the moment.

Network output policy

The default policy for selecting a network `renderer` (in order of preference) is as follows:

- ENI
- Sysconfig
- Netplan
- NetworkManager
- FreeBSD
- NetBSD
- OpenBSD
- Networkd

The default policy for selecting a network `activator` (in order of preference) is as follows:

- **ENI**: using `ifup`, `ifdown` to manage device setup/teardown
- **Netplan**: using `netplan apply` to manage device setup/teardown
- **NetworkManager**: using `nmcli` to manage device setup/teardown
- **Networkd**: using `ip` to manage device setup/teardown

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', 'network-manager', 'eni', 'sysconfig', 'freebsd', 'netbsd',
↪'openbsd']
    activators: ['eni', 'netplan', 'network-manager', 'networkd']
```

Network configuration tools

`Cloud-init` contains a command used to test input/output conversion between formats. The `tools/net-convert.py` in the `cloud-init` source repository is helpful in examining expected output for a given input format. If running these commands from the cloud-init source directory, make sure to set the correct path `PYTHON_PATH=.`

CLI Interface:

```
$ cloud-init devel net-convert --help
```

Example output:

```
usage: /usr/bin/cloud-init devel net-convert [-h] -p PATH -k {eni,network_data.json,yaml,
↪azure-imds,vmware-imc} -d PATH -D
                                             {alpine,arch,debian,ubuntu,freebsd,
↪dragonfly,gentoo,cos,netbsd,openbsd,almalinux,amazon,centos,cloudlinux,eurolinux,
↪fedora,mariner,miraclelinux,openmandriva,photon,rhel,rocky,virtuozzo,opensuse,sles,
↪openEuler}
                                             [-m name,mac] [--debug] -O {eni,netplan,
↪networkd,sysconfig,network-manager}

options:
  -h, --help            show this help message and exit
  -p PATH, --network-data PATH
                        The network configuration to read
  -k {eni,network_data.json,yaml,azure-imds,vmware-imc}, --kind {eni,network_data.json,
↪yaml,azure-imds,vmware-imc}
                        The format of the given network config
  -d PATH, --directory PATH
                        directory to place output in
  -D {alpine,arch,debian,ubuntu,freebsd,dragonfly,gentoo,cos,netbsd,openbsd,almalinux,
↪amazon,centos,cloudlinux,eurolinux,fedora,mariner,miraclelinux,openmandriva,photon,
↪rhel,rocky,virtuozzo,opensuse,sles,openEuler}, --distro {alpine,arch,debian,ubuntu,
↪freebsd,dragonfly,gentoo,cos,netbsd,openbsd,almalinux,amazon,centos,cloudlinux,
↪eurolinux,fedora,mariner,miraclelinux,openmandriva,photon,rhel,rocky,virtuozzo,
↪opensuse,sles,openEuler}
```

(continues on next page)

(continued from previous page)

```
  -m name,mac, --mac name,mac
                        interface name to mac mapping
  --debug               enable debug logging to stderr.
  -O {eni,netplan,networkd,sysconfig,network-manager}, --output-kind {eni,netplan,
↪networkd,sysconfig,network-manager}
                        The network config format to emit
```

Example of converting V2 to sysconfig:

```
$ cloud-init devel net-convert --network-data v2.yaml --kind yaml \
   --output-kind sysconfig -d target
$ cat target/etc/sysconfig/network-scripts/ifcfg-eth*
```

Example output:

```
# 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
```

2.4.9 Base configuration

Warning: This documentation is intended for custom image creators, such as distros and cloud providers, not end users. Modifying the base configuration should not be necessary for end users and can result in a system that may be unreachable or may no longer boot.

`Cloud-init` base config is primarily defined in two places:

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

See the *configuration sources explanation* for more information on how these files get sourced and combined with other configuration.

Generation

`cloud.cfg` isn't present in any of `cloud-init`'s source files. The configuration is templated and customised for each distribution supported by `cloud-init`.

Base configuration keys

Module keys

Modules are grouped into the following keys:

- `cloud_init_modules`: Modules run during *network* timeframe.
- `cloud_config_modules`: Modules run during *config* timeframe.
- `cloud_final_modules`: Modules run during *final* timeframe.

Each `module` definition contains an array of strings, where each string is the name of the module. Each name is taken directly from the module filename, with the `cc_` prefix and `.py` suffix removed, and with `-` and `_` being interchangeable.

Alternatively, in place of the module name, an array of `<name>, <frequency>[, <args>]` args may be specified. See *the module creation guidelines* for more information on `frequency` and `args`.

Note: Most modules won't run at all if they're not triggered via a respective user data key, so removing modules or changing the run frequency is **not** a recommended way to reduce instance boot time.

Examples

To specify that only cc_final_message.py run during final timeframe:

```
cloud_final_modules:
- final_message
```

To change the frequency from the default of `ALWAYS` to `ONCE`:

```
cloud_final_modules:
- [final_message, once]
```

To include default arguments to the module (that may be overridden by user data):

```
cloud_final_modules:
- [final_message, once, "my final message"]
```

Datasource keys

Many datasources allow configuration of the datasource for use in querying the datasource for metadata using the `datasource` key. This configuration is datasource dependent and can be found under each datasource's respective *documentation*. It will generally take the form of:

```
datasource:
  <datasource_name>:
    ...
```

System info keys

These keys are used for setup of `cloud-init` itself, or the datasource or distro. Anything under `system_info` cannot be overridden by vendor data, user data, or any other handlers or transforms. In some cases there may be a `system_info` key used for the distro, while the same key is used outside of `system_info` for a user data module. Both keys will be processed independently.

- `system_info`: Top-level key.
  - `paths`: Definitions of common paths used by `cloud-init`.
    - `cloud_dir`: Defaults to `/var/lib/cloud`.
    - `templates_dir`: Defaults to `/etc/cloud/templates`.
  - `distro`: Name of distro being used.
  - `default_user`: Defines the default user for the system using the same user configuration as *Users and Groups*. Note that this CAN be overridden if a `users` configuration is specified without a `- default` entry.
  - `ntp_client`: The default NTP client for the distro. Takes the same form as `ntp_client` defined in *NTP*.
  - `package_mirrors`: Defines the package mirror info for apt.
  - `ssh_svcname`: The SSH service name. For most distros this will be either `ssh` or `sshd`.
  - `network`: Top-level key for distro-specific networking configuration.
    - `renderers`: Prioritised list of networking configurations to try on this system. The first valid entry found will be used. Options are:
      - `eni`: For `/etc/network/interfaces`.
      - `network-manager`
      - `netplan`
      - `networkd`: For `systemd-networkd`.
      - `freebsd`
      - `netbsd`
      - `openbsd`
    - `activators`: Prioritised list of networking tools to try to activate network on this system. The first valid entry found will be used. Options are:
      - `eni`: For `ifup/ifdown`.
      - `netplan`: For `netplan generate/netplan apply`.

- `network-manager`: For `nmcli connection load/ nmcli connection up`.
- `networkd`: For `ip link set up/ip link set down`.

Logging keys

See *the logging explanation* for a comprehensive logging explanation. Note that `cloud-init` has a default logging definition that shouldn't need to be altered. It is defined in this instance at `/etc/cloud/cloud.cfg.d/05_logging.cfg`.

The logging keys used in the base configuration are as follows:

`logcfg`

A standard python fileConfig formatted log configuration. This is the primary logging configuration key and will take precedence over `log_cfgs` or `log_basic` keys.

`log_cfgs`

A list of logging configs in fileConfig format to apply when running `cloud-init`. Note that `log_cfgs` is used in `/etc/cloud.cfg.d/05_logging.cfg`.

`log_basic`

Boolean value to determine if `cloud-init` should apply a basic default logging configuration if none has been provided. Defaults to `true` but only takes effect if `logcfg` or `log_cfgs` hasn't been defined.

`output`

If and how to redirect `stdout/stderr`. Defined in `/etc/cloud.cfg.d/05_logging.cfg` and explained in *the logging explanation*.

`syslog_fix_perms`

Takes a list of `<owner:group>` strings and will set the owner of `def_log_file` accordingly.

`def_log_file`

Only used in conjunction with `syslog_fix_perms`. Specifies the filename to be used for setting permissions. Defaults to `/var/log/cloud-init.log`.

Other keys

network

The *network configuration* to be applied to this instance.

datasource_pkg_list

Prioritised list of python packages to search when finding a datasource. Automatically includes `cloudinit.sources`.

datasource_list

Prioritised list of datasources that `cloud-init` will attempt to find on boot. By default, this will be defined in `/etc/cloud/cloud.cfg.d`. There are two primary use cases for modifying the `datasource_list`:

1. Remove known invalid datasources. This may avoid long timeouts when attempting to detect datasources on any system without a systemd-generator hook that invokes `ds-identify`.
2. Override default datasource ordering to discover a different datasource type than would typically be prioritised.

If `datasource_list` has only a single entry (or a single entry + `None`), *cloud-init's generator script* will automatically assume and use this datasource without attempting detection.

vendor_data/vendor_data2

Allows the user to disable `vendor_data` or `vendor_data2` along with providing a prefix for any executed scripts.

Format is a dict with `enabled` and `prefix` keys:

- `enabled`: A boolean indicating whether to enable or disable the `vendor_data`.
- `prefix`: A path to prepend to any `vendor_data`-provided script.

Example

On an Ubuntu system, `/etc/cloud/cloud.cfg` should look similar to:

```
# The top level settings are used as module and base configuration.
# A set of users which may be applied and/or used by various modules
# when a 'default' entry is found it will reference the 'default_user'
# from the distro configuration specified below
users:
- default


# If this is set, 'root' will not be able to ssh in and they
# will get a message to login instead as the default $user
disable_root: true

# This will cause the set+update hostname module to not operate (if true)
preserve_hostname: false
```

(continues on next page)

(continued from previous page)

```
# If you use datasource_list array, keep array items in a single line.
# If you use multi line array, ds-identify script won't read array items.
# Example datasource config
# datasource:
#    Ec2:
#      metadata_urls: [ 'blah.com' ]
#      timeout: 5 # (defaults to 50 seconds)
#      max_wait: 10 # (defaults to 120 seconds)

# The modules that run in the 'init' stage
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

# The modules that run in the 'config' stage
cloud_config_modules:
- snap
- ssh-import-id
- keyboard
- locale
- set-passwords
- grub-dpkg
- apt-pipelining
- apt-configure
- ubuntu-advantage
- ntp
- timezone
- disable-ec2-metadata
- runcmd
- byobu

# The modules that run in the 'final' stage
cloud_final_modules:
- package-update-upgrade-install
- fan
- landscape
- lxd
- ubuntu-drivers
- write-files-deferred
```

(continues on next page)

(continued from previous page)

```
- puppet
- chef
- mcollective
- salt-minion
- reset_rmc
- refresh_rmc_and_interface
- rightscale_userdata
- scripts-vendor
- scripts-per-once
- scripts-per-boot
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
- install-hotplug
- phone-home
- final-message
- power-state-change

# System and/or distro specific settings
# (not accessible to handlers/transforms)
system_info:
  # This will affect which distro class gets used
  distro: ubuntu
  # Default user name + that default users groups (if added/used)
  default_user:
    name: ubuntu
    lock_passwd: True
    gecos: Ubuntu
    groups: [adm, audio, cdrom, dialout, dip, floppy, lxd, netdev, plugdev, sudo, video]
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    shell: /bin/bash
  network:
    renderers: ['netplan', 'eni', 'sysconfig']
  # Automatically discover the best ntp_client
  ntp_client: auto
  # Other config here will be given to the distro class and/or path classes
  paths:
    cloud_dir: /var/lib/cloud/
    templates_dir: /etc/cloud/templates/
  package_mirrors:
    - arches: [i386, amd64]
    failsafe:
        primary: http://archive.ubuntu.com/ubuntu
        security: http://security.ubuntu.com/ubuntu
    search:
        primary:
        - http://%(ec2_region)s.ec2.archive.ubuntu.com/ubuntu/
        - http://%(availability_zone)s.clouds.archive.ubuntu.com/ubuntu/
        - http://%(region)s.clouds.archive.ubuntu.com/ubuntu/
        security: []
    - arches: [arm64, armel, armhf]
```

(continues on next page)

(continued from previous page)

```
    failsafe:
        primary: http://ports.ubuntu.com/ubuntu-ports
        security: http://ports.ubuntu.com/ubuntu-ports
    search:
        primary:
        - http://%(ec2_region)s.ec2.ports.ubuntu.com/ubuntu-ports/
        - http://%(availability_zone)s.clouds.ports.ubuntu.com/ubuntu-ports/
        - http://%(region)s.clouds.ports.ubuntu.com/ubuntu-ports/
        security: []
    - arches: [default]
    failsafe:
        primary: http://ports.ubuntu.com/ubuntu-ports
        security: http://ports.ubuntu.com/ubuntu-ports
  ssh_svcname: ssh
```

2.5 Development

`Cloud-init` is an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback. If you would like to contribute to `cloud-init`, this set of documentation will help orient you with our processes.

2.5.1 Contributing

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.

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 your GitHub username (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 testing* (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 in to your GitHub account.
- Fork the upstream repository on GitHub and click on the `Fork` button
- Create a new remote pointing to your personal GitHub repository.

```
git clone git@github.com:GH_USER/cloud-init.git
cd cloud-init
git remote add upstream git@github.com:canonical/cloud-init.git
git push origin 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 your GitHub username 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.
  - You may use `tools/check-cla-signers` to sort `.github-cla-signers` or check that it is sorted.
  - 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 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, 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.

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, and add more commits.):

```
git commit
```

- Apply `black` and `isort` formatting rules with tox:

```
tox -e do_format
```

- Run unit tests and lint/formatting checks with tox:

```
tox
```

- Push your changes to your personal GitHub repository:

```
git push -u origin 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.
```

(continues on next page)

(continued from previous page)

```
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 sqaushed 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.

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.)

`cloudinit.features.`**`EXPIRE_APPLIES_TO_HASHED_USERS = True`**

If `EXPIRE_APPLIES_TO_HASHED_USERS` is True, then when expire is set true in cc_set_passwords, hashed passwords will be expired. Previous to 22.3, only non-hashed passwords were expired.

(This flag can be removed after Jammy is no longer supported.)

cloudinit.features.**NETPLAN_CONFIG_ROOT_READ_ONLY = True**

If NETPLAN_CONFIG_ROOT_READ_ONLY is True, then netplan configuration will be written as a single root readon-only file /etc/netplan/50-cloud-init.yaml. This prevents wifi passwords in network v2 configuration from being world-readable. Prior to 23.1, netplan configuration is world-readable.

(This flag can be removed after Jammy is no longer supported.)

cloudinit.features.**NOCLOUD_SEED_URL_APPEND_FORWARD_SLASH = True**

Append a forward slash '/' if NoCloud seedurl does not end with either a querystring or forward slash. Prior to 23.1, nocloud seedurl would be used unaltered, appending meta-data, user-data and vendor-data to without URL path separators.

(This flag can be removed when Jammy is no longer supported.)

Module creation

Much of `cloud-init`'s functionality is provided by *modules*. All modules follow a similar layout in order to provide consistent execution and documentation. Use the example provided here to create a new module.

Example

```
# This file is part of cloud-init. See LICENSE file for license information.
"""Example Module: Shows how to create a module"""

from logging import Logger

from cloudinit.cloud import Cloud
from cloudinit.config import Config
from cloudinit.config.schema import MetaSchema, get_meta_doc
from cloudinit.distros import ALL_DISTROS
from cloudinit.settings import PER_INSTANCE

MODULE_DESCRIPTION = """\
Description that will be used in module documentation.

This will likely take multiple lines.
"""

meta: MetaSchema = {
    "id": "cc_example",
    "name": "Example Module",
    "title": "Shows how to create a module",
    "description": MODULE_DESCRIPTION,
    "distros": [ALL_DISTROS],
    "frequency": PER_INSTANCE,
    "activate_by_schema_keys": ["example_key, example_other_key"],
    "examples": [
        "example_key: example_value",
        "example_other_key: ['value', 2]",
    ],
}
```

(continues on next page)

(continued from previous page)

```
__doc__ = get_meta_doc(meta)


def handle(
    name: str, cfg: Config, cloud: Cloud, log: Logger, args: list
) -> None:
    log.debug(f"Hi from module {name}")
```

Guidelines

- Create a new module in the `cloudinit/config` directory with a `cc_` prefix.
- Your module must include a `handle` function. The arguments are:
  - `name`: The module name specified in the configuration.
  - `cfg`: A configuration object that is the result of the merging of cloud-config configuration with any datasource-provided configuration.
  - `cloud`: A cloud object that can be used to access various datasource and paths for the given distro and data provided by the various datasource instance types.
  - `log`: A logger object that can be used to log messages.
  - `args`: An argument list. This is usually empty and is only populated if the module is called independently from the command line or if the module definition in `/etc/cloud/cloud.cfg[.d]` has been modified to pass arguments to this module.
- If your module introduces any new cloud-config keys, you must provide a schema definition in cloud-init-schema.json.
- The `meta` variable must exist and be of type MetaSchema.
  - `id`: The module ID. In most cases this will be the filename without the `.py` extension.
  - `distros`: Defines the list of supported distros. It can contain any of the values (not keys) defined in the OSFAMILIES map or `[ALL_DISTROS]` if there is no distro restriction.
  - `frequency`: Defines how often module runs. It must be one of:
    - `PER_ALWAYS`: Runs on every boot.
    - `ONCE`: Runs only on first boot.
    - `PER_INSTANCE`: Runs once per instance. When exactly this happens is dependent on the datasource, but may triggered any time there would be a significant change to the instance metadata. An example could be an instance being moved to a different subnet.
  - `activate_by_schema_keys`: Optional list of cloud-config keys that will activate this module. When this list not empty, the config module will be skipped unless one of the `activate_by_schema_keys` are present in merged cloud-config instance-data.
  - `examples`: Lists examples of any cloud-config keys this module reacts to. These examples will be rendered in the module reference documentation and will automatically be tested against the defined schema during testing.
- `__doc__ = get_meta_doc(meta)` is necessary to provide proper module documentation.

Module execution

In order for a module to be run, it must be defined in a module run section in `/etc/cloud/cloud.cfg` or `/etc/cloud/cloud.cfg.d` on the launched instance. The three module sections are cloud_init_modules, cloud_config_modules, and cloud_final_modules, corresponding to the *Network*, *Config*, and *Final* boot stages respectively.

Add your module to cloud.cfg.tmpl under the appropriate module section. Each module gets run in the order listed, so ensure your module is defined in the correct location based on dependencies. If your module has no particular dependencies or is not necessary for a later boot stage, it should be placed in the `cloud_final_modules` section before the `final-message` module.

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. It 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.

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,
- send an email to the cloud-init mailing list:

```
cloud-init@lists.launchpad.net
```

These are listed in order of preference, but please use whichever of them you are most comfortable with.

Goals

This process has the following goals:

- To ensure code reviews occur in a timely fashion and provide actionable feedback if changes are desired.
- To ensure the minimisation of ancillary problems to increase the efficiency for those reviewing the submitted code.

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`).

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 has been received from a **Committer**, and
- no "Request changes" reviews from any **Committer** are outstanding.

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, and
- no "Request changes" reviews from any **Reviewer** are outstanding.

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.)

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 rather an acknowledgement 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.

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 a 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*.

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 this section, 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 to 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 straightforward *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.

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.

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.

`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 within 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 (the active instance 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 to 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/`

Contains seeded data files: `meta-data`, `network-config`, `user-data`, `vendor-data`.

`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 run "per-once", "per-instance", or "per-always". This folder contains semaphore `files` which are only supposed to run "per-once" (not tied to the instance ID).

2.5.2 Debugging and reporting

Logging

`Cloud-init` supports both local and remote logging configurable through multiple configurations:

- Python's built-in logging configuration
- `Cloud-init`'s event reporting system
- The `cloud-init` `rsyslog` module

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 setup. 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 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 to 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]
 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.

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"
```

Event reporting

`Cloud-init` contains an eventing system that allows events to be emitted to a variety of destinations.

Three configurations are available for reporting events:

- `webhook`: POST to a web server.
- `log`: Write to the `cloud-init` log at configurable log level.
- `stdout`: Print to `stdout`.

The default configuration is to emit events to the `cloud-init` log file at `DEBUG` level.

Event reporting can be configured using the `reporting` key in `cloud-config` user data.

Configuration

webhook

```
reporting:
  <user-defined name>:
    type: webhook
    endpoint: <url>
    timeout: <timeout in seconds>
    retries: <number of retries>
    consumer_key: <OAuth consumer key>
    token_key: <OAuth token key>
    token_secret: <OAuth token secret>
    consumer_secret: <OAuth consumer secret>
```

`endpoint` is the only additional required key when specifying `type: webhook`.

log

```
reporting:
  <user-defined name>:
    type: log
    level: <DEBUG|INFO|WARN|ERROR|FATAL>
```

`level` is optional and defaults to "DEBUG".

print

```
reporting:
  <user-defined name>:
    type: print
```

Example

The follow example shows configuration for all three sources:

```
#cloud-config
reporting:
  webserver:
    type: webhook
    endpoint: "http://10.0.0.1:55555/asdf"
    timeout: 5
    retries: 3
    consumer_key: <consumer_key>
    token_key: <token_key>
    token_secret: <token_secret>
    consumer_secret: <consumer_secret>
  info_log:
    type: log
    level: WARN
  stdout:
    type: print
```

`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 *our module reference page*.

Debugging `cloud-init`

Overview

This topic will discuss general approaches for testing and debugging `cloud-init` on deployed instances.

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 which operations took the longest during boot and setup.

The script `/usr/bin/cloud-init` has an analysis 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, and 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 organise `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
```

Example output:

```
-- 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
|`->config-snap_config ran successfully @05.47700s +00.00100s
|`->config-ssh-import-id ran successfully @05.47800s +00.00200s
```

(continues on next page)

(continued from previous page)

```
|`->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
```

Example output:

```
[
 {
  "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 the highest time cost for a quick assessment of areas of `cloud-init` that may need improvement.

```
$ cloud-init analyze blame -i my-cloud-init.log
```

Example output:

```
-- Boot Record 11 --
     00.01300s (modules-final/config-scripts-per-boot)
     00.00400s (modules-final/config-final-message)
     00.00100s (modules-final/config-rightscale_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
```

Example output:

```
-- 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
```

(continues on next page)

(continued from previous page)

```
   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 :command:`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 `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 -
```

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 user data 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 via the command line:

```
$ sudo cloud-init single --name cc_ssh --frequency always
```

Example output:

```
...
Generating public/private ed25519 key pair
...
```

Inspect `cloud-init.log` for output of what operations were performed as a result.

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" (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 that the new version works properly on the specific platform and Ubuntu series.
- Check for tracebacks or errors in behaviour.

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. Be sure to make a note of the datasource `cloud-init` detected in `--long` output. You will need this during step 5, where you will use it to confirm the same datasource is detected after the upgrade:

```
CI_VM_IP=<YOUR_VM_IP>
$ 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`. To do this, create the following bash script, which 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. First, we block until `cloud-init` completes, then verify from `--long` that the datasource is the same as the one picked up from step 1. Errors will show up in `--long`:

```
$ ssh ubuntu@$CI_VM_IP -- cloud-init status --wait --long
```

Make sure the hostname was set properly to *SRU-worked-<cloud name>*:

```
$ ssh ubuntu@$CI_VM_IP -- hostname
```

Then, check for any errors or warnings in `cloud-init` logs. If successful, this will produce no output:

```
$ 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

2.5.3 Testing

Testing

Integration testing

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`.

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.

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
```

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/
```

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]
```

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 identified image 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 RHEL8 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.

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" behaviour.
- Take a snapshot of the instance to be used as a new image from which new instances can be launched.

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 then torn down after.

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.

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"
```

`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.

Guidelines

The following guidelines should be followed.

Test layout

- For ease of organisation and greater accessibility for developers unfamiliar with `pytest`, all `cloud-init` unit tests must be contained within test classes. In other words, module-level test functions should not be used.
- Since 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`.
  - E.g., 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 available[1]:
  - `cache`
  - `capfd`
  - `capfdbinary`
  - `caplog`
  - `capsys`
  - `capsysbinary`
  - `doctest_namespace`
  - `monkeypatch`
  - `pytestconfig`
  - `record_xml_property`
  - `recwarn`
  - `tmpdir_factory`
  - `tmpdir`

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 a non-autospecced `Mock`, 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`

[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/ \[session scope\
↪]//g;s/.*/* ``\0``/g'
```

in an ubuntu lxd container with python3-pytest installed.

  - `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`

2.5.4 Documentation

Documentation

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, with some pages written in standard Markdown.

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 built, the HTML files will be viewable in *doc/rtd_html*. Use your web browser to open *index.html* to view and navigate the site.

Style guide

Language

Where possible, text should be written in UK English. However, discretion and common sense can both be applied. For example, where text refers to code elements that exist in US English, the spelling of these elements should not be changed to UK English.

Headings

In reStructuredText, headings are denoted using symbols to underline the text. The headings used across the documentation use the following hierarchy:

- `#####`: Top level header (reserved for the main index page)
- `*****`: Title header (used once at the top of a new page)
- `=====`: Section headers
- `-----`: Subsection headers
- `^^^^^`: Sub-subsection headers
- `"""""`: Paragraphs

The length of the underline must be at least as long as the title itself.

Ensure that you do not skip header levels when creating your document structure, i.e., that a section is followed by a subsection, and not a sub-subsection.

Line length

Please keep the line lengths to a maximum of **79** characters. This ensures that the pages and tables do not get so wide that side scrolling is required.

Anchor labels

Adding an anchor label 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:

FAQ
***
```

When the reference is used in a document, the displayed text will be that of the next heading immediately following the label (so, FAQ in this example), unless specifically overridden.

If you use labels within a page to refer, for example, to a subsection, use a label that follows the format: `[pagelabel]-[Section]` e.g., for this "Anchor labels" section, something like `_docs-Anchor:` or `_docs-Label:`. Using a consistent style will aid greatly when referencing from other pages.

Links

To aid in documentation maintenance and keeping links up-to-date, links should be presented in a single block at the end of the page.

Where possible, use contextual text in your links to aid users with screen readers and other accessibility tools. For example, "check out our *documentation style guide*" is preferable to "click *here* for more".

Code blocks

Our documentation uses the Sphinx extension "sphinx-copybutton", which creates a small button on the right-hand side of code blocks for users to copy the code snippets we provide.

The copied code will strip out the prompt symbol (`$`) so that users can paste commands directly into their terminal. For user convenience, please ensure that code output is presented in a separate code block to the commands.

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 hyphenated. Follows sentence case, so only capitalised at the start of a sentence (e.g., `Cloud-init`).
- `metadata`, `datasource`: One word.
- `user data`, `vendor data`: Two words, not to be combined or hyphenated.

Acronyms

Acronyms are always capitalised (e.g., JSON, YAML, QEMU, LXD) in text.

The first time an acronym is used on a page, it is best practice to introduce it by showing the expanded name followed by the acronym in parentheses. E.g., Quick EMUlator (QEMU). If the acronym is very common, or you provide a link to a documentation page that provides such details, you will not need to do this.

PYTHON MODULE INDEX

C

cloudinit.config.cc_ansible, 50
cloudinit.config.cc_apk_configure, 53
cloudinit.config.cc_apt_configure, 54
cloudinit.config.cc_apt_pipelining, 59
cloudinit.config.cc_bootcmd, 59
cloudinit.config.cc_byobu, 60
cloudinit.config.cc_ca_certs, 60
cloudinit.config.cc_chef, 61
cloudinit.config.cc_disable_ec2_metadata, 64
cloudinit.config.cc_disk_setup, 64
cloudinit.config.cc_fan, 66
cloudinit.config.cc_final_message, 67
cloudinit.config.cc_growpart, 68
cloudinit.config.cc_grub_dpkg, 69
cloudinit.config.cc_install_hotplug, 70
cloudinit.config.cc_keyboard, 70
cloudinit.config.cc_keys_to_console, 71
cloudinit.config.cc_landscape, 72
cloudinit.config.cc_locale, 73
cloudinit.config.cc_lxd, 74
cloudinit.config.cc_mcollective, 77
cloudinit.config.cc_migrator, 78
cloudinit.config.cc_mounts, 79
cloudinit.config.cc_ntp, 80
cloudinit.config.cc_package_update_upgrade_install, 82
cloudinit.config.cc_phone_home, 82
cloudinit.config.cc_power_state_change, 84
cloudinit.config.cc_puppet, 85
cloudinit.config.cc_resizefs, 87
cloudinit.config.cc_resolv_conf, 88
cloudinit.config.cc_rh_subscription, 89
cloudinit.config.cc_rightscale_userdata, 90
cloudinit.config.cc_rsyslog, 91
cloudinit.config.cc_runcmd, 92
cloudinit.config.cc_salt_minion, 93
cloudinit.config.cc_scripts_per_boot, 94
cloudinit.config.cc_scripts_per_instance, 94
cloudinit.config.cc_scripts_per_once, 94
cloudinit.config.cc_scripts_user, 94
cloudinit.config.cc_scripts_vendor, 95
cloudinit.config.cc_seed_random, 95
cloudinit.config.cc_set_hostname, 96
cloudinit.config.cc_set_passwords, 97
cloudinit.config.cc_snap, 99
cloudinit.config.cc_spacewalk, 100
cloudinit.config.cc_ssh, 101
cloudinit.config.cc_ssh_authkey_fingerprints, 104
cloudinit.config.cc_ssh_import_id, 104
cloudinit.config.cc_timezone, 105
cloudinit.config.cc_ubuntu_advantage, 105
cloudinit.config.cc_ubuntu_drivers, 107
cloudinit.config.cc_update_etc_hosts, 108
cloudinit.config.cc_update_hostname, 109
cloudinit.config.cc_users_groups, 110
cloudinit.config.cc_wireguard, 114
cloudinit.config.cc_write_files, 115
cloudinit.config.cc_yum_add_repo, 117
cloudinit.config.cc_zypper_add_repo, 118
cloudinit.features, 249

INDEX

A

ALLOW_EC2_MIRRORS_ON_NON_AWS_INSTANCE_TYPES
    (*in module cloudinit.features*), 249

C

cloudinit.config.cc_ansible
    module, 50
cloudinit.config.cc_apk_configure
    module, 53
cloudinit.config.cc_apt_configure
    module, 54
cloudinit.config.cc_apt_pipelining
    module, 59
cloudinit.config.cc_bootcmd
    module, 59
cloudinit.config.cc_byobu
    module, 60
cloudinit.config.cc_ca_certs
    module, 60
cloudinit.config.cc_chef
    module, 61
cloudinit.config.cc_disable_ec2_metadata
    module, 64
cloudinit.config.cc_disk_setup
    module, 64
cloudinit.config.cc_fan
    module, 66
cloudinit.config.cc_final_message
    module, 67
cloudinit.config.cc_growpart
    module, 68
cloudinit.config.cc_grub_dpkg
    module, 69
cloudinit.config.cc_install_hotplug
    module, 70
cloudinit.config.cc_keyboard
    module, 70
cloudinit.config.cc_keys_to_console
    module, 71
cloudinit.config.cc_landscape
    module, 72
cloudinit.config.cc_locale
    module, 73
cloudinit.config.cc_lxd
    module, 74
cloudinit.config.cc_mcollective
    module, 77
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, 82
cloudinit.config.cc_phone_home
    module, 82
cloudinit.config.cc_power_state_change
    module, 84
cloudinit.config.cc_puppet
    module, 85
cloudinit.config.cc_resizefs
    module, 87
cloudinit.config.cc_resolv_conf
    module, 88
cloudinit.config.cc_rh_subscription
    module, 89
cloudinit.config.cc_rightscale_userdata
    module, 90
cloudinit.config.cc_rsyslog
    module, 91
cloudinit.config.cc_runcmd
    module, 92
cloudinit.config.cc_salt_minion
    module, 93
cloudinit.config.cc_scripts_per_boot
    module, 94
cloudinit.config.cc_scripts_per_instance
    module, 94
cloudinit.config.cc_scripts_per_once
    module, 94
cloudinit.config.cc_scripts_user
    module, 94
cloudinit.config.cc_scripts_vendor

module, 95
cloudinit.config.cc_seed_random
module, 95
cloudinit.config.cc_set_hostname
module, 96
cloudinit.config.cc_set_passwords
module, 97
cloudinit.config.cc_snap
module, 99
cloudinit.config.cc_spacewalk
module, 100
cloudinit.config.cc_ssh
module, 101
cloudinit.config.cc_ssh_authkey_fingerprints
module, 104
cloudinit.config.cc_ssh_import_id
module, 104
cloudinit.config.cc_timezone
module, 105
cloudinit.config.cc_ubuntu_advantage
module, 105
cloudinit.config.cc_ubuntu_drivers
module, 107
cloudinit.config.cc_update_etc_hosts
module, 108
cloudinit.config.cc_update_hostname
module, 109
cloudinit.config.cc_users_groups
module, 110
cloudinit.config.cc_wireguard
module, 114
cloudinit.config.cc_write_files
module, 115
cloudinit.config.cc_yum_add_repo
module, 117
cloudinit.config.cc_zypper_add_repo
module, 118
cloudinit.features
module, 249

E

ERROR_ON_USER_DATA_FAILURE (*in module cloudinit.features*), 249
EXPIRE_APPLIES_TO_HASHED_USERS (*in module cloudinit.features*), 249

M

module
cloudinit.config.cc_ansible, 50
cloudinit.config.cc_apk_configure, 53
cloudinit.config.cc_apt_configure, 54
cloudinit.config.cc_apt_pipelining, 59
cloudinit.config.cc_bootcmd, 59
cloudinit.config.cc_byobu, 60
cloudinit.config.cc_ca_certs, 60
cloudinit.config.cc_chef, 61
cloudinit.config.cc_disable_ec2_metadata, 64
cloudinit.config.cc_disk_setup, 64
cloudinit.config.cc_fan, 66
cloudinit.config.cc_final_message, 67
cloudinit.config.cc_growpart, 68
cloudinit.config.cc_grub_dpkg, 69
cloudinit.config.cc_install_hotplug, 70
cloudinit.config.cc_keyboard, 70
cloudinit.config.cc_keys_to_console, 71
cloudinit.config.cc_landscape, 72
cloudinit.config.cc_locale, 73
cloudinit.config.cc_lxd, 74
cloudinit.config.cc_mcollective, 77
cloudinit.config.cc_migrator, 78
cloudinit.config.cc_mounts, 79
cloudinit.config.cc_ntp, 80
cloudinit.config.cc_package_update_upgrade_install, 82
cloudinit.config.cc_phone_home, 82
cloudinit.config.cc_power_state_change, 84
cloudinit.config.cc_puppet, 85
cloudinit.config.cc_resizefs, 87
cloudinit.config.cc_resolv_conf, 88
cloudinit.config.cc_rh_subscription, 89
cloudinit.config.cc_rightscale_userdata, 90
cloudinit.config.cc_rsyslog, 91
cloudinit.config.cc_runcmd, 92
cloudinit.config.cc_salt_minion, 93
cloudinit.config.cc_scripts_per_boot, 94
cloudinit.config.cc_scripts_per_instance, 94
cloudinit.config.cc_scripts_per_once, 94
cloudinit.config.cc_scripts_user, 94
cloudinit.config.cc_scripts_vendor, 95
cloudinit.config.cc_seed_random, 95
cloudinit.config.cc_set_hostname, 96
cloudinit.config.cc_set_passwords, 97
cloudinit.config.cc_snap, 99
cloudinit.config.cc_spacewalk, 100
cloudinit.config.cc_ssh, 101
cloudinit.config.cc_ssh_authkey_fingerprints, 104
cloudinit.config.cc_ssh_import_id, 104
cloudinit.config.cc_timezone, 105
cloudinit.config.cc_ubuntu_advantage, 105
cloudinit.config.cc_ubuntu_drivers, 107
cloudinit.config.cc_update_etc_hosts, 108
cloudinit.config.cc_update_hostname, 109
cloudinit.config.cc_users_groups, 110

cloudinit.config.cc_wireguard, 114
cloudinit.config.cc_write_files, 115
cloudinit.config.cc_yum_add_repo, 117
cloudinit.config.cc_zypper_add_repo, 118
cloudinit.features, 249

N

NETPLAN_CONFIG_ROOT_READ_ONLY (*in module cloudinit.features*), 249
NOCLOUD_SEED_URL_APPEND_FORWARD_SLASH (*in module cloudinit.features*), 250