Everything about cloud-init, a set of python scripts and utilities to make your cloud images be all they can be!
Cloud-init is the defacto multi-distribution package that handles early initialization of a cloud instance.

## 1.1 Capabilities

- Setting a default locale
- Setting an instance hostname
- Generating instance SSH private keys
- Adding SSH keys to a user’s `.ssh/authorized_keys` so they can log in
- Setting up ephemeral mount points
- Configuring network devices

### 1.1.1 User configurability

Cloud-init’s behavior can be configured via user-data.

User-data can be given by the user at instance launch time. See *User-Data Formats* for acceptable user-data content.

This is done via the `--user-data` or `--user-data-file` argument to `ec2-run-instances` for example.

- Check your local client’s documentation for how to provide a `user-data` string or `user-data` file to cloud-init on instance creation.
1.1.2 Feature detection

Newer versions of cloud-init may have a list of additional features that they support. This allows other applications to detect what features the installed cloud-init supports without having to parse its version number. If present, this list of features will be located at `cloudinit.version.FEATURES`.

Currently defined feature names include:

- `NETWORK_CONFIG_V1` support for v1 networking configuration, see *Networking Config Version 1* documentation for examples.

- `NETWORK_CONFIG_V2` support for v2 networking configuration, see *Networking Config Version 2* documentation for examples.

1.1.3 CLI Interface

The command line documentation is accessible on any cloud-init installed system:

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

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

Subcommands:
  {init,modules,single,query,dhclient-hook,features,analyze,devel,collect-logs,clean,
   status}
  init                    initializes cloud-init and performs initial modules
  modules                 activates modules using a given configuration key
  single                  run a single module
  query                   Query 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
```

1.1.4 CLI Subcommand details

cloud-init 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 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.

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

```
# Cloud-init running still short versus long options

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

cloud-init collect-logs

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

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

Logs collected are:

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

cloud-init query

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

- **--all**: Dump all available instance data as json which can be queried.
- **--instance-data**: Optional path to a different instance-data.json file to source for queries.
• -list-keys: List available query keys from cached instance data.

```
# List all top-level query keys available (includes standardized aliases)
% cloud-init query --list-keys
availability_zone
base64_encoded_keys
cloud_name
ds
instance_id
local_hostname
region
v1
```

• `<varname>`: A dot-delimited variable path into the instance-data.json object.

```
# Query cloud-init standardized metadata on any cloud
% cloud-init query v1.cloud_name
aws # or openstack, azure, gce etc.

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

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

• --format A string that will use jinja-template syntax to render a string replacing)

```
# 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
```

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

cloud-init analyze

Get detailed reports of where cloud-init spends most of its time. See Boot Time Analysis - cloud-init analyze for more info.

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

cloud-init devel

Collection of development tools under active development. These tools will likely be promoted to top-level subcommands when stable.
• `cloud-init devel schema`: A #cloud-config format and schema validator. It accepts a cloud-config yaml file and annotates potential schema errors locally without the need for deployment. Schema validation is work in progress and supports a subset of cloud-config modules.

• `cloud-init devel 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.

**cloud-init clean**

Remove cloud-init artifacts from /var/lib/cloud and optionally reboot the machine to so cloud-init re-runs all stages as it did on first boot.

  • `--logs`: Optionally remove /var/log/cloud-init*log files.
  • `--reboot`: Reboot the system after removing artifacts.

**cloud-init 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`.

**cloud-init modules**

Generally run by OS init systems to execute modules:config and modules:final boot stages. This executes cloud config Modules configured to run in the init, config and final stages. The modules are declared to run in various boot stages in the file /etc/cloud/cloud.cfg under keys cloud_init_modules, cloud_init_modules and cloud_init_modules. Can be run on the commandline, 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.

**cloud-init single**

Attempt to run a single named cloud config module. The following example re-runs the cc_set_hostname module ignoring the module default frequency of once-per-instance:

  • `--name`: The cloud-config module name to run
  • `--frequency`: Optionally override the declared module frequency with one of (alwaysonce-per-instanceonce)

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

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

### 1.2 Availability

It is currently installed in the Ubuntu Cloud Images and also in the official Ubuntu images available on EC2, Azure, GCE and many other clouds.
Versions for other systems can be (or have been) created for the following distributions:

- Ubuntu
- Fedora
- Debian
- RHEL
- CentOS
- and more…

So ask your distribution provider where you can obtain an image with it built-in if one is not already available

### 1.3 User-Data Formats

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

#### 1.3.1 Gzip Compressed Content

Content found to be gzip compressed will be uncompressed. The uncompressed data will then be used as if it were not compressed. This is typically useful because user-data is limited to ~16384\(^1\) bytes.

#### 1.3.2 Mime Multi Part Archive

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

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

Supported content-types:

- text/x-include-once-url
- text/x-include-url
- text/cloud-config-archive
- text/upstart-job
- text/cloud-config
- text/part-handler
- text/x-shellscript
- text/cloud-boothook

**Helper script to generate mime messages**

\(^{1}\) See your cloud provider for applicable user-data size limitations…
#!/usr/bin/python
import sys

from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

if len(sys.argv) == 1:
    print("%s input-file:type ..." % (sys.argv[0]))
    sys.exit(1)

combined_message = MIMEMultipart()
for i in sys.argv[1:]:
    (filename, format_type) = i.split(":", 1)
    with open(filename) as fh:
        contents = fh.read()
        sub_message = MIMEText(contents, format_type, sys.getdefaultencoding())
        sub_message.add_header('Content-Disposition', 'attachment; filename="%s"' % (filename))
        combined_message.attach(sub_message)

print(combined_message)

1.3.3 User-Data Script

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

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

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

Example

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

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

1.3.4 Include File

This content is a include file.

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

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

1.3. User-Data Formats
1.3.5 Cloud Config Data

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

These things include:

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

Note: This file must be valid yaml syntax.

See the Cloud config examples section for a commented set of examples of supported cloud config formats.

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

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

1.3.6 Upstart Job

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

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

1.3.7 Cloud Boothook

This content is boothook data. It is stored in a file under /var/lib/cloud and then executed immediately. This is the earliest hook available. Note, that there is no mechanism provided for running only once. The boothook must take care of this itself. It is provided with the instance id in the environment variable INSTANCE_ID. This could be made use of to provide a ‘once-per-instance’ type of functionality.

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

1.3.8 Part Handler

This is a part-handler: It contains custom code for either supporting new mime-types in multi-part user data, or overriding the existing handlers for supported mime-types. It will be written to a file in /var/lib/cloud/data based on its filename (which is generated). This must be python code that contains a list_types function and a handle_part function. Once the section is read the list_types method will be called. It must return a list of mime-types that this part-handler handles. Because mime parts are processed in order, a part-handler part must precede any parts with mime-types it is expected to handle in the same user data.

The handle_part function must be defined like:
def handle_part(data, ctype, filename, payload):
    # data = the cloudinit object
    # ctype = "__begin__", "__end__", or the mime-type of the part that is being handled.
    # filename = the filename of the part (or a generated filename if none is present in mime data)
    # payload = the parts' content

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

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

Example

```
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"
    print "==== received ctype=%s filename=%s ====
```

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

1.4 Instance Metadata

1.4.1 What is instance data?

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

- cloud-provided metadata services (aka metadata)
- custom config-drive attached to the instance
- cloud-config seed files in the booted cloud image or distribution
Cloud-init provides a cache of any crawled metadata as well as a versioned set of standardized instance data keys which it makes available on all platforms.

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

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

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

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

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

### 1.4.2 Format of instance-data.json

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

There are three basic top-level keys:

- **base64_encoded_keys**: A list of forward-slash delimited key paths into the instance-data.json object whose value is base64encoded for json compatibility. Values at these paths should be decoded to get the original value.

- **sensitive_keys**: A list of forward-slash delimited key paths into the instance-data.json object whose value is considered by the datasource as ‘security sensitive’. Only the keys listed here will be redacted from instance-data.json for non-root users.

- **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.

- **v1**: Standardized cloud-init metadata keys, these keys are guaranteed to exist on all cloud platforms. They will also retain their current behavior and format and will be carried forward even if cloud-init introduces a new version of standardized keys with v2.
The standardized keys present:

<table>
<thead>
<tr>
<th>Key</th>
<th>path</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1._beta_keys</td>
<td>standardized keys still in ‘beta’. The format, intent or presence of these keys can change. Do not consider them production-ready.</td>
<td>[subplatform]</td>
<td></td>
</tr>
<tr>
<td>v1.cloud_name</td>
<td>possible this will indicate the ‘name’ of the cloud this system is running on. This is specifically different than the ‘platform’ below. As an example, the name of Amazon Web Services is ‘aws’ while the platform is ‘ec2’. If no specific name is determinable or provided in meta-data, then this field may contain the same content as ‘platform’.</td>
<td>aws, openstack, azure, configdrive, nocloud, ovf, etc.</td>
<td></td>
</tr>
<tr>
<td>v1.instance_id</td>
<td>instance_id allocated by the cloud</td>
<td>i-&lt;somehash&gt;</td>
<td></td>
</tr>
<tr>
<td>v1.local_hostname</td>
<td>The internal or local hostname of the system</td>
<td>ip-10-41-41-70, &lt;user-provided-hostname&gt;</td>
<td></td>
</tr>
<tr>
<td>v1.platform</td>
<td>An attempt to identify the cloud platform instance that the system is running on.</td>
<td>ec2, openstack, lxd, gce nocloud, ovf</td>
<td></td>
</tr>
<tr>
<td>v1.subplatform</td>
<td>Additional platform details describing the specific source or type of meta-data used. The format of subplatform will be: &lt;subplatform_type&gt; (&lt;url_file_or_dev_path&gt;)</td>
<td>metadata (<a href="http://168.254.169.254">http://168.254.169.254</a>), seed-dir (/path/to/seed-dir), config-disk (/dev/cd0), configdrive (/dev/sr0)</td>
<td></td>
</tr>
<tr>
<td>v1.public_ssh_keys</td>
<td>keys provided to the instance by the datasource metadata.</td>
<td>['ssh-rsa AA...']...</td>
<td></td>
</tr>
<tr>
<td>v1.region</td>
<td>The physical region/datacenter in which the instance is deployed</td>
<td>us-east-2</td>
<td></td>
</tr>
<tr>
<td>v1.availability_zone</td>
<td>The physical availability zone in which the instance is deployed</td>
<td>us-east-2b, nova, null</td>
<td></td>
</tr>
</tbody>
</table>

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

```
{
    "base64_encoded_keys": [],
    "ds": {
        ",_doc": "EXPERIMENTAL: The structure and format of content scoped under the 'ds' key may change in subsequent releases of cloud-init.",
        "_metadata_api_version": "2016-09-02",
        "dynamic": {
            "instance-identity": {
                "document": {
                    "accountId": "437526006925",
                    "architecture": "x86_64",
                    "availabilityZone": "us-east-2b",
                    "billingProducts": null,
                    "devpayProductCodes": null,
                    "imageId": "ami-079638aae7046bdd2",
                    "instanceId": "i-075f088c72ad3271c",
                    "instanceType": "t2.micro",
                    "kernelId": null,
                    "marketplaceProductCodes": null,
                    "pendingTime": "2018-10-05T20:10:43Z",
                    "privateIp": "10.41.41.95",
                    "ramdiskId": null,
                    "region": "us-east-2",
                    "version": "2017-09-30"
                },
                "pkcs7": [
                    "MIAGCSqGSIb3DQEHAqCAMIACAQExCzAjbUrDgMCGgUAMIAGCSqGSIb3DQEHAaaCAJIAEggHbexog",
                ]
            }
        }
    }
```
Cloud-Init, Release 19.2

(continued from previous page)
"local-hostname": "ip-10-41-41-95.us-east-2.compute.internal",
"local-ipv4": "10.41.41.95",
"mac": "06:74:8f:39:cd:a6",
"metrics": {
  "vhostmd": "<?xml version="1.0" encoding="UTF-8"?>",
},
"network": {
  "interfaces": {
    "macs": {
      "06:74:8f:39:cd:a6": {
        "device-number": "0",
        "interface-id": "eni-052058bbd7831eaae",
        "ipv4-associations": {
          "18.218.221.122": "10.41.41.95"
        },
        "local-hostname": "ip-10-41-41-95.us-east-2.compute.internal",
        "local-ipv4": "10.41.41.95",
        "mac": "06:74:8f:39:cd:a6",
        "owner-id": "437526006925",
        "public-hostname": "ec2-18-218-221-122.us-east-2.compute.amazonaws.com",
        "public-ipv4": "18.218.221.122",
        "security-group-ids": "sg-828247e9",
        "security-groups": "Cloud-init integration test secgroup",
        "subnet-id": "subnet-282f3053",
        "subnet-ipv4-cidr-block": "10.41.41.0/24",
        "subnet-ipv6-cidr-blocks": "2600:1f16:b80:ad00::/64",
        "vpc-id": "vpc-252ef24d",
        "vpc-ipv4-cidr-block": "10.41.0.0/16",
        "vpc-ipv4-cidr-blocks": "10.41.0.0/16",
        "vpc-ipv6-cidr-blocks": "2600:1f16:80::/64"
      }
    }
  }
},
"placement": {
  "availability-zone": "us-east-2b"
},
"profile": "default-hvm",
"public-hostname": "ec2-18-218-221-122.us-east-2.compute.amazonaws.com",
"public-ipv4": "18.218.221.122",
"public-keys": {
  "cloud-init-integration": [
    "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSL7uWGj8cqWyIOaspqKdVY0cKJ+UTjfv7jBojG2H/
    GQ8bJVxY72XAvnhM0dUM+Ccs8FQfOY1PX+FrVz2hKInrmRHzZVwRSL129PasD12M13I344u6lW6o/
    W86Q+tKQYE1jtgDO0o+a+C0sa2KvUNzUyEXUwz/
    mmlYa6G4hMKZ4HNBj7nbaA9F96wmMCoyNwBbYnBnDyUr6wMbqjRR1J9pR7x7h7WRC73wy4Va2YuOgbD3V/
    5ZrfPflbWZw/7TFXVeql04QVbyei4aiFR5n/GvoqwQDiNe581mbzX/
    vxyKyJYdny2ZxmdAhMxbpFQspkJ9E/H5w0yOdSvnWbUoG5xNGoOB cloud-init-integration"
  ]
},
"reservation-id": "r-0594a20e31f6cfe46",
"security-groups": "Cloud-init integration test secgroup",
"services": {
  "domain": "amazonaws.com",
  "partition": "aws"
}
1.4.3 Using instance-data

As of cloud-init v. 18.4, any variables present in /run/cloud-init/instance-data.json can be used in:

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

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

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

Below are some examples of providing these types of user-data:

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

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

- Custom user-data script performing different operations based on region
```bash
#!/bin/bash
{% if v1.region == 'us-east-2' -%}
    echo 'Installing custom proxies for {{ v1.region }}
    sudo apt-get install my-xtra-fast-stack
{% endif %}
...
```

**Note:** Trying to reference Jinja variables that don’t exist in instance-data.json will result in warnings in `/var/log/cloud-init.log` and the following string in your rendered user-data: 

```
CI_MISSING_JINJA_VAR/<your_varname>
```

Cloud-init also surfaces a commandline tool **cloud-init query** which can assist developers or scripts with obtaining instance metadata easily. See **cloud-init query** for more information.

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

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

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

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

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

## 1.5 Directory layout

Cloudinit’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
```

(continues on next page)
/var/lib/cloud

The main directory containing the cloud-init specific subdirectories. It is typically located at /var/lib but there are certain configuration scenarios where this can be altered.

TBD, describe this overriding more.

data/

Contains information related to instance ids, datasources and hostnames of the previous and current instance if they are different. These can be examined as needed to determine any information related to a previous boot (if applicable).

handlers/

Custom part-handlers code is written out here. Files that end up here are written out with in the scheme of part-handler-XYZ where XYZ is the handler number (the first handler found starts at 0).

instance

A symlink to the current instances/ subdirectory that points to the currently active instance (which is active is dependent on the datasource loaded).

instances/

All instances that were created using this image end up with instance identifier subdirectories (and corresponding data for each instance). The currently active instance will be symlinked the instance symlink file defined previously.

scripts/

Scripts that are downloaded/created by the corresponding part-handler will end up in one of these subdirectories.

seed/

TBD

sem/

Cloud-init has a concept of a module semaphore, which basically consists of the module name and its frequency. These files are used to ensure a module is only ran per-once, per-instance, per-always. This folder contains semaphore files which are only supposed to run per-once (not tied to the instance id).
1.6 Cloud config examples

1.6.1 Including users and groups

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

# Add users to the system. Users are added after groups are added.
users:
- default
  - name: foobar
    gecos: Foo B. Bar
    primary_group: foobar
    groups: users
    selinux_user: staff_u
    expiredate: 2012-09-01
    ssh_import_id: foobar
    lock_passwd: false
    passwd: $6$j212wezy$7H/1LT4f9/N3wpgNunhsIqtMj620Kis3nyNwuizouQc3u7MbYCarYeAHWYPYb2FT.1bioDm2RrkJPb9B2MN10/
  - name: barfoo
    gecos: Bar B. Foo
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: users, admin
    ssh_import_id: None
    lock_passwd: true
    ssh Authorized Keys: 
    - <ssh pub key 1>
    - <ssh pub key 2>
  - name: cloudy
    gecos: Magic Cloud App Daemon User
    inactive: true
    system: true
  - name: fizzbuzz
    sudo: False
    ssh Authorized Keys: 
    - <ssh pub key 1>
    - <ssh pub key 2>
    - snapuser: joe@joeuser.io
  - name: nosshlogins
    ssh_redirect_user: true

# Valid Values:
# name: The user's login name
# 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
```

(continues on next page)
# SELinux user.
# lock_passwd: Defaults to true. Lock the password to disable password login
# inactive: Create the user as inactive
# passwd: The hash -- not the password itself -- of the password you want
to use for this user. You can generate a safe 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
eample 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 use SSH authentication only.

You have thus been warned.

# 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
# 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.

# ssh Authorized_keys.
# 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:
#   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]

1.6.2 Writing out arbitrary files

```yaml
#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: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHRoZSBzdGVy...!
    owner: root:root
    path: /etc/sysconfig/selinux
    permissions: '0644'
  - encoding: !!binary
    content: |
      # My new /etc/sysconfig/samba file
      SMBDOPTIONS="-D"
    path: /etc/sysconfig/samba
```

1.6. Cloud config examples
1.6.3 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:
  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
```

1.6.4 Configure an instances 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.

certs:
  # If present and set to True, the 'remove-defaults' parameter will remove
  # all the default trusted CA certificates that are normally shipped with
  # Ubuntu.
  # This is mainly for paranoid admins - most users will not need this
  # functionality.
```
Cloud-Init, Release 19.2

```yaml
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-ORG-S-TRUSTED-CA-CERT-HERE
    -----END CERTIFICATE-----
  - |
    -----BEGIN CERTIFICATE-----
    YOUR-ORG-S-TRUSTED-CA-CERT-HERE
    -----END CERTIFICATE-----
```

1.6.5 Configure an instances resolv.conf

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

```yaml
#cloud-config
#
# This is an example file to automatically configure resolv.conf when the
# instance boots for the first time.
#
# Ensure that your yaml is valid and pass this as user-data when starting
# the instance. Also be sure that your cloud.cfg file includes this
# configuration module in the appropriate section.
#
manage_resolv_conf: true

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

1.6.6 Install and run chef recipes

```yaml
#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.
#
# This example assumes the instance is 16.04 (xenial)
```

1.6. Cloud config examples
# The default is to install from packages.

# Key from https://packages.chef.io/chef.asc

apt:
  sources:
    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
    mQGiBEppC7QRBADfs0kZU6KZK+YmKw4wev5mjKJEkVG1us+NxW8wItX5sGa6kdUu
twAyj7Yr9zrf+ICFEP3gGU6+1Go0Nve7KxkhN/W7/m3G4zuk+cclKjmb8KS3qjn99
dxy64vcj9i11Vv+XXOGipDG8Eaaj7mbkixL/bMeGfDM1v8GfZPpp9vwCgn/GC
JKacfnw7MplKUKOyS1b/JeEAJqao3V1nFav83jJKEkDScf59J8kia5Op2qTk5W
ShVnNS3U51Vq1k0ZDH97Qn/YrK387H4CyhLE9hmxPXs/u118iclaars/q2MEKU2I
XKFv21eMLO9lYd6Ny/KjqBjo5WQK2J6+NahSwvthZc1EphcFignIuobP+B5wNFQpe
DbKfA/0Wn20WeFRCmmd3Hz7nHTpcnSF+4QX6yHRF/5Bgz6G6iqBIACQbZn6Hm
mSm/tV5f11izmdq5sasPcrtC0ZILfLX/m/E+Y1+CwWSHh1+YsFls1wOh1EhQD26aO
284HuV5HFRWjDLw9Lr1l1CBVQcXbobp5rP5bdr7Wh8vhqJTFqJnT3BzY29KZSBQ
YWnRvWd1cyA8cGF/a2F2zXNAAb3By29k55j2b20+iGAEExECACAFakppC7CQGwMG
CwkIBwMCBEVUCCAMiFGDIAqIEiAqIGAAKRCrPQKupg++Caj8sAARCMXmdG36WjI/K
+o+xtIbvdwnHvFYQCTCWXry2BnzL0BFFCJDK6sJqCuUIENERUYgUGFja2Fn2ZMmg
PHbHy2tzh2ZVzQGNo2WYu8w+iGIEExECAVF1AJQwYFECwMGwCwkIBwMCBhUIAgk
CwQAgMBAb4BA+aHeaAe5Ec1Ag6mD74jQx94An26z99XOHwpLNAhzm7cp13t4Xid
AJ9wVcogUBvzgg3dFq+3cmem2n7kCDQRKAgu0EAg472LCVGTtmLqBM6n32Ed
Zvb+m2BvLBSBsmqidd6u3eOH3X2uafQPHU3y2yedyDM1G+iEMTDdEeeRebC7C
SNXQ8Xvi22hRpoEBsSWLZi/XNq0n0f1+GEr+mOK0OzBD2G7D3A0nEnESWFK
OFJFerb35rSraWJj0KjDxkhse2ddU/jVZ1BY7NFt8o2mwpBmdozETM0tx3LJy1H2/
Te5FJXJMJaB2lRlyv15MVwCKJQro4MQG/7QGfIf2NfA9G3ZD2D5svj7/V0+IPrY
ICu4BQCuB6su4yYUG4j7rhH6u7Hlp99sdwagsO1pBe/vvBvc/tbwAB+eJJA9P82eU
twADBOqf2CoPrTGFuazbkeNRS0rf+boaeWPo5zK2DyfSccyAg41CYq9+0IhWq38q
DuxQv+f3jEJNvE3E3Edk1/zkXMMzDLExJGw84t5XEmhD1d2zqHqnm2zEM
Wp27juGwa3Gh15-ap004tIGacCmCNHmL1Rgiupq52kR1Q9mLEn1ECr8cWagKgS
0Ww+vMz2eM75/7/TK19HNXn4tVaqvO3FkYK54GJ2fjCqV7U5smNv4KPTYLLX/A
dC3Z46H/eM1Xp21isIfgB04v+TXn5UF3B4FAY5hevr4Dtv4MwI3sZ0w0Jfc/L
MpmzR3N2ykmO0H+vgr19IFp06D0WJhJBBcRagAJBQJKaQuahsM4AOjEC1Aq6MD
74JQ4hIaoJ5KrsY8SkCwJ26SAGz1wpv3tCAJ0bekyk56VnqoegB+y4PQDV4vK
zA==
=ImPr
-----END PGP PUBLIC KEY BLOCK-----

chef:

# 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"

# 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'}

1.6.7 Setup and run puppet

#cloud-config
#
# This is an example file to automatically setup and run puppetd
# 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.
puppet:
  # Every key present in the conf object will be added to puppet.conf:
  # [name]
  # subkey=value

1.6. Cloud config examples
# For example the configuration below will have the following section added to puppet.conf:
# server=puppetmaster.example.org
# certname=i-0123456.ip-X-Y-Z.cloud.internal
# The puppetmaster ca certificate will be available in
# /var/lib/puppet/ssl/certs/ca.pem

conf:
  agent:
    server: "puppetmaster.example.org"
    # certname supports substitutions at runtime:
    # %i: instanceid
    # %f: fqdn of the machine
    # NB: the certname will automatically be lowercased as required by puppet
    certname: "%i.%f"
  ca_cert is a special case. It won’t be added to puppet.conf.
  # It holds the puppetmaster certificate in pem format.
  # It should be a multi-line string (using the | yaml notation for
  # multi-line strings).
  # The The puppetmaster certificate is located in
  # /var/lib/puppet/ssl/ca/ca_cert.pem on the puppetmaster host.
  ca_cert: |
    -----BEGIN CERTIFICATE-----
    MIICCTCAAXkAwIBAgIBATANBgkqhkiG9w0BAQUFADANMQswCQYDVQQDDAJjYTAe
    Fw0xMDAyMTUxNzI5MjFaFw0xNTAyMTEwNzI5MjFaMA0GCSqGSIb3DQEBAQUAA4GN
    ADCBiQKgYIKwYBBQUHAwEGCCsGASCBMA0GCSqGSIb3DQEBAQEEggAeCBmgIBAQA
    b140mQoX7dtCnTvky+iNoHiUDTjawsz/tZwEdxqXQLwTGrwVvcAqbutJKi93M1LwI
    DIAQVBo3kwSLA8+ghkgBhvhCAQ0EKXyPlUHwvcGVIQIFJ1Ynkv
    T3Bb1INTC8ZH51cmF0ZWQgQ2VydGlmaWNhdGUwDwYDVR0TAQH/BAUwAwEB/zAd
    BgNVHQ4EFgQUu4+jHb+9Y5Xvo+o1l0AhevspjAwCwYDVQQGEwJBMQswCQYDVQQ
    BgNVH4EgQUu4+jHb+9Y5Xvo+o1l0AhevspjAwCwYDVQQGEwJBMQswCQYDVQQ
    -----END CERTIFICATE-----

1.6.8 Add apt repositories

#cloud-config

# Add apt repositories
# 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

(continues on next page)
# 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]
      search_dns: True

1.6.9 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

1.6.10 Alter the completion message

#cloud-config

# final_message
# default: cloud-init boot finished at $TIMESTAMP. Up SUPTIME seconds
# this message is written by cloud-final when the system is finished
# its first boot

final_message: "The system is finally up, after $SUPTIME seconds"

1.6.11 Install arbitrary packages

#cloud-config

# Install additional packages on first boot
# default: none
# if packages are specified, this apt_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]
1.6.12 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
# Aliases: apt_update
package_update: false
```

1.6.13 Run apt or yum upgrade

```cloud-config
#cloud-config
#
# Upgrade the instance on first boot
# (ie run apt-get upgrade)
#
# Default: false
# Aliases: apt_upgrade
package_upgrade: true
```

1.6.14 Adjust mount points mounted

```cloud-config
#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:
# - [ 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 ommitted 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:
```

(continues on next page)
- [ ephemeral10, /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 7 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: size in bytes

1.6.15 Call a url when finished

#cloud-config
# phone_home: if this dictionary is present, then the phone_home
# cloud-config module will post specified data back to the given
# url
# default: none
# phone_home:
# url: http://my.foo.bar/$INSTANCE/
# post: all
# tries: 10
# phone_home:
url: http://my.example.com/$INSTANCE_ID/
post: [ pub_key_dsa, pub_key_rsa, pub_key_ecdsa, instance_id ]

1.6.16 Reboot/poweroff when finished

#cloud-config
## poweroff or reboot system after finished
# default: none
#
# power_state can be used to make the system shutdown, reboot or
# halt after boot is finished. This same thing can be acheived by
# user-data scripts or by runcmd by simply invoking 'shutdown'.
#
# Doing it this way ensures that cloud-init is entirely finished with
# modules that would be executed, and avoids any error/log messages
# that may go to the console as a result of system services like
# syslog being taken down while cloud-init is running.
#
# If you delay '+5' (5 minutes) and have a timeout of
Cloud-Init, Release 19.2

(continued from previous page)

# 120 (2 minutes), then the max time until shutdown will be 7 minutes.
# cloud-init will invoke 'shutdown +5' after the process finishes, or
# when 'timeout' seconds have elapsed.
# delay: form accepted by shutdown.  default is 'now'.  other format
#    accepted is +tm (m in minutes)
# mode: required.  must be one of 'poweroff', 'halt', 'reboot'
# message: provided as the message argument to 'shutdown'.  default is none.
# timeout: the amount of time to give the cloud-init process to finish
#    before executing shutdown.
# condition: apply state change only if condition is met.
#    May be boolean True (always met), or False (never met),
#    or a command string or list to be executed.
# command's exit code indicates:
#    0: condition met
#    1: condition not met
#    other exit codes will result in 'not met', but are reserved
#    for future use.
power_state:
    delay: "+30"
    mode: poweroff
    message: Bye Bye
    timeout: 30
    condition: True

1.6.17 Configure instances 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 AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI62+nCSjUUk8EEAnnhkX1ukKoUPND/
  NRC1Wz2s5TCzk30u5+Cyz71X0XmazM315WgeErvt3wQMyT1kJNoMh0mJnWQP0t3Q8w9qG7PB19+eiHogV7NZ
  
- ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA3I7VUFv215q5n5auROs5HRDP2dqUe5q5zemNSJ87T7enqKHOEaF0U2V0p4GEWC9RyzSQVe...
    +iiD+ey3ONkZN+LQ714c9j8FRS4Hj29SCmXp5Kt5/82cD/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-----
     MIIBxwIBAAJhAKDOSHSY73nUg03XaJmd4fHiFyG+00R7VvU21V9Qcon2LSZ/x
     1cydPZ4PqfjBha6Wx2sO8ci/Ea/w0n+OHGPwax1EG29InNtj3pgrYcRztFEcb
     1j6HCib2bAzYtw1BIwJg0872Wjcmvcp280vHSv7Tawg02TkR6mPqHg0kA6GJo
     PUUn2Z6Wuba/HXOKgyhz19nPz1pG5F0fVah1MAydc13FV7k6kMBEPXTR6F9gHEg
     L0MPC7cvdAnV1NcPyA7AE1aB1mIj0eFN2a2fX0j7OMhQuc4zP7r80zaG5o6W
     p58hRncFKEvNq2CeL3vtu2A1jEawNRHf1aBYTPCHN7r2uG/iBtw8Rxhc9l5w
     kxvMgi+HpGLWzUIBS+P/Xhek1PjPAjA285rVmEP+DIR255Ls65QbgYhJm7ixQ2T9
     1uLvcMFC6135uc47g4Alt0mX1n7I2CMOMpS9SpEvu8MnayTCL+vEjwNBT+FAp0
    -----END RSA PRIVATE KEY-----
```

(continues on next page)
### 1.6.18 Additional apt configuration

```bash
# apt_pipeling (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_pipeling: 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

# 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: ...
```

(continues on next page)
# ['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, this apt_update will be set to true
packages: ['pastebinit']

apt:
  # The apt config consists of two major "areas".
  #
  # On one hand there is the global configuration for the apt feature.
  #
  # On one hand (down in this file) there is the source dictionary which allows
  # to define various entries to be considered by apt.

#############Section 1: global apt configuration
#
# The following examples number the top keys to ease identification in
# discussions.

# 1.1 preserve_sources_list
#
# Preserves the existing /etc/apt/sources.list
# Default: false - do overwrite sources_list. If set to true then any
# "mirrors" configuration will have no effect.
# Set to true to avoid affecting sources.list. In that case only
# "extra" source specifications will be written into
# /etc/apt/sources.list.d/*
preserve_sources_list: true

# 1.2 disable_suites
#
# This is an empty list by default, so nothing is disabled.
#
# If given, those suites are removed from sources.list after all other
# modifications have been made.
# Suites are even disabled if no other modification was made,
# but not if is preserve_sources_list is active.
# There is a special alias "$RELEASE" as in the sources that will be replace
# by the matching release.
#
# To ease configuration and improve readability the following common ubuntu
# suites will be automatically mapped to their full definition.
# updates => $RELEASE-updates
# backports => $RELEASE-backports
# security => $RELEASE-security
# proposed => $RELEASE-proposed
# release => $RELEASE
#
# There is no harm in specifying a suite to be disabled that is not found in
# the source.list file (just a no-op then)
#
# Note: Lines don’t get deleted, but disabled by being converted to a comment.
# The following example disables all usual defaults except $RELEASE-security.
# On top it disables a custom suite called "mysuite"

disableSuites: [$RELEASE-updates, backports, $RELEASE, mysuite]

# 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: [amd64, i386, default]
    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
    # 1. uri
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 {
    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]]/

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 '/'.
# If it doesn't end with .list it will be appended so that apt picks up it's
# 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.

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 xenial 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 and $RELEASE 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 xenial multiverse
source: deb $MIRROR $RELEASE multiverse

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 xenial 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

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

my-repo4.list:
# 2.8 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------
# Version: SKS 1.0.10
mI0ESpA3UQEALd2KVIMq0j6qWAXAyxSlF63SvPVlgsPb9Nk0D2Uixn+akqyt+xG4zKCONz6
gLJoBFHhnyyVLFt4ih9an1PqxRnTO+JKQx18NgKg6Fon569GtAo8WNKw15XKnjTDLijnj
9y96ljQrCpV9t/WsIdcJPeKFR5vc0h8EOaBE2aEADBAAG0GUxdW5jaBH7CBQUEEGz9m9y
IEFs2XNOaWOItgTAQ1AIACSpA3UQibAwYLCQghHAw1EFQIIwQWAgaMBAh4BAheAAIoJEAt7H
5Q1iCcVxWz8D/1My1+yf3JFtZUm2YoIzZasQ657vh19+pFouqFlWooyRRR9jbiyUFIhVoDQBRP
t0FwvnOFArUovUWkoADEqR8hpy3M3APUZj15K4cMzR/XaqMeQRZ5CHePS4DkBURKAKH01tS5o
uBJKQOzmlItJp15cyIKeGe8Mx18VFyVg1AZey
=Y2oI
------END PGP PUBLIC KEY BLOCK------

1.6.19 Disk setup

# Cloud-init supports the creation of simple partition tables and file systems
# 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
(continues on next page)
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

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

# The default definition is used to make sure that the ephemeral storage is
# setup properly.

# "disk_setup": disk partitioning
# --------------------------------

# The disk_setup directive instructs Cloud-init to partition a disk. The format is:
disk_setup:
  ephemeral0:
    table_type: 'mbr'
    layout: 'auto'
  /dev/xvdh:
    table_type: 'mbr'
The format is a list of dicts of dicts. The first value is the name of the device and the subsequent values define how to create and layout the partition.

The general format is:

```python
disk_setup:
    <DEVICE>:
        table_type: 'mbr'
        layout: <LAYOUT|BOOL>
        overwrite: <BOOL>
```

Where:

- `<DEVICE>`: The name of the device. 'ephemeralX' and 'swap' are special values which are specific to the cloud. For these devices Cloud-init will look up what the real device 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:
    ```python
    [[<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.

```python
disk_setup:
    /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 saftey's 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 file system
3. If either a partition of file system 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 file system
-------------------------------

fs_setup describes the how the file systems 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 file system label to be used. If set to None, no label is used.
#   - <FS_TYPE>: The file system type. It is assumed that the there will be a "mkfs.<FS_TYPE>" that behaves like "mkfs". On a standard Ubuntu Cloud Image, this means that you have the option of ext2,3,4, 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 interpreted
# as a partition value. However, ephemeralX.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
# file system already. In the absence of a partition table, it will
# put it directly on the disk.
#
# "auto": If a file system that matches the specification in terms of
# label, type and device, then cloud-init will skip the creation of
# the file system.
#
# "any": If a file system that matches the file system type and device,
# then cloud-init will skip the creation of the file system.
#
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 file system 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 file system 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 file system. Use at
You are currently reading through the Cloud-Init, Release 19.2 documentation. The following text is extracted from the document:

```
# your own peril.
#
# "false": If an existing file system exists, skip the creation.
#
# <REPLACE_FS>: This is a special directive, used for Microsoft Azure that
# instructs cloud-init to replace a file system 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 file system exists.
# If a file system matches the specification, then the operation is a no-op.
```

### 1.6.20 Register RedHat Subscription

```
#cloud-config
#
# register your Red Hat Enterprise Linux based operating system
# this cloud-init plugin is capable of registering by username
# and password *or* activation and org. Following a successfully
# registration you can:
# - auto-attach subscriptions
# - set the service level
# - add subscriptions based on its pool ID
# - enable yum repositories based on its repo id
# - disable yum repositories based on its repo id
# - alter the rhsm_baseurl and server-hostname in the
#   /etc/rhsm/rhs.conf file

rh_subscription:
  username: joe@foo.bar

  ## Quote your password if it has symbols to be safe
  password: '1234abcd'

  ## If you prefer, you can use the activation key and
  ## org instead of username and password. Be sure to
  ## comment out username and password

  #activation-key: foobar
  #org: 12345

  ## Uncomment to auto-attach subscriptions to your system
  #auto-attach: True

  ## Uncomment to set the service level for your
  ## subscriptions
  #service-level: self-support

  ## Uncomment to add pools (needs to be a list of IDs)
  #add-pool: []

  ## Uncomment to add or remove yum repos
  ## (needs to be a list of repo IDs)
  #enable-repo: []
```

(continues on next page)
## 1.6.21 Configure data sources

```
# 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://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

    # 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
```

(continues on next page)
1.6.22 Create partitions and filesystems

```yaml
# Cloud-init supports the creation of simple partition tables and file systems
# 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'}
```

(continues on next page)
```yaml
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

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

# The default definition is used to make sure that the ephemeral storage is setup properly.

# "disk_setup": disk partitioning
# -----------------------------

# The disk_setup directive instructs Cloud-init to partition a disk. The format is:

disk_setup:
  ephemeral0:
```

(continues on next page)
table_type: 'mbr'
layout: 'auto'

/dev/xvdh:
  table_type: 'mbr'
  layout:
    - 33
    - [33, 82]
    - 33
  overwrite: True

# The format is a list of dicts of dicts. The first value is the name of the
# device and the subsequent values define how to create and layout the
# partition.
# The general format is:
#   disk_setup:
#     <DEVICE>:
#       table_type: 'mbr'
#       layout: <LAYOUT|BOOL>
#       overwrite: <BOOL>

# Where:
#   <DEVICE>: The name of the device. 'ephemeralX' and 'swap' are special
#   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

(continues on next page)
Cloud-Init, Release 19.2

(continued from previous page)

# 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 saftey's 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 file system
# 3. If either a partition of file system 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 file system
# -------------------------------
#
# fs_setup describes the how the file systems are supposed to look.
#
fs_setup:
- label: ephemeral0
  filesystem: 'ext3'
  device: 'ephemeral0'
- label: mylab12
  filesystem: 'ext4'
  device: '/dev/xvdal'
- cmd: mkfs -t %{filesystem}s -L %{label}s %{device}s
  label: mylab13
  filesystem: 'btrfs'
  device: '/dev/xvdxh'

# 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 file system label to be used. If set to None, no label is
# used.
# <FS_TYPE>: The file system 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, ephemeralX.0 is the _same_ as ephemeralX.

<PART_VALUE>:
Partition definitions are overwitten 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 file system already. In the absence of a partition table, it will put it directly on the disk.

"auto": If a file system that matches the specification in terms of label, type and device, then cloud-init will skip the creation of the file system.

"any": If a file system that matches the file system type and device, then cloud-init will skip the creation of the file system.

Devices are selected based on first-detected, starting with partitions and then the raw disk. Consider the following:

<table>
<thead>
<tr>
<th>NAME</th>
<th>FSTYPE</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>xvdb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-xvdb1</td>
<td>ext4</td>
</tr>
<tr>
<td></td>
<td>-xvdb2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-xvdb3</td>
<td>btrfs</td>
</tr>
<tr>
<td>-xvdb4</td>
<td>ext4</td>
<td></td>
</tr>
</tbody>
</table>

If you ask for 'auto', label of 'test', and file system 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 formating ephemeral storage or for simple schemes.

"none": Put the file system 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 file system. Use at your own peril.

"false": If an existing file system exists, skip the creation.

<REPLACE_FS>: This is a special directive, used for Microsoft Azure that instructs cloud-init to replace a file system 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 file system exists. If a file system matches the specification, then the operation is a no-op.

## 1.6.23 Grow partitions

```python
#cloud-config
#
# growpart entry is a dict, if it is not present at all
# in config, then the default is used ("mode": 'auto', 'devices': ['/'])
#
# mode:
# values:
# * auto: use any option possible (any available)
# if none are available, do not warn, but debug.
# * growpart: use growpart to grow partitions
# if growpart is not available, this is an error.
# * off, false
#
# devices:
# a list of things to resize.
# items can be filesystem paths or devices (in /dev)
# examples:
#   devices: ['/dev/vdb1', '/dev/vdb2']
#
# ignore_growroot_disabled:
# a boolean, default is false.
# if the file /etc/growroot-disabled exists, then cloud-init will not grow
# the root partition. This is to allow a single file to disable both
# cloud-initramfs-growroot and cloud-init's growroot support.
#
# true indicates that /etc/growroot-disabled should be ignored
#
growpart:
  mode: auto
  devices: ['/']
  ignore_growroot_disabled: false
```
1.7 Boot Stages

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

There are 5 stages.

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

1.7.1 Generator

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

- A file exists: /etc/cloud/cloud-init.disabled
- 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.

This mechanism for disabling at runtime currently only exists in systemd.

1.7.2 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 bringup.
- **modules**: none

The purpose of the local stage is:

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

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

- the datasource
- 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 entirely with config like the following in /etc/cloud/cloud.cfg: ‘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 might already have been applied. That could have negative effects such as DHCP hooks or broadcast of an old hostname. It would also put the system in an odd state to recover from as it may then have to restart network devices.

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

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

### 1.7.3 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:

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

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

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

A part-handler will run at this stage, as will 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.

### 1.7.4 Config

- **systemd service:** `cloud-config.service`
- **runs:** After network stage.
- **blocks:** None.
- **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.

### 1.7.5 Final

- **systemd service:** `cloud-final.service`
- **runs:** As final part of boot (traditional “rc.local”)
• **blocks**: None.

• **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 (puppet, chef, salt-minion)

• user-scripts (including `runcmd`).

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

## 1.8 Datasources

### 1.8.1 What is a datasource?

Datasources are sources of configuration data for cloud-init that typically come from the user (aka userdata) or come from the stack that created the configuration drive (aka metadata). Typical userdata would include files, yaml, and shell scripts while typical metadata would include server name, instance id, display name and other cloud specific details. Since there are multiple ways to provide this data (each cloud solution seems to prefer its own way) internally a datasource abstract class was created to allow for a single way to access the different cloud systems methods to provide this data through the typical usage of subclasses.

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

### Datasource API

The current interface that a datasource object must provide is the following:

```python
# returns a mime multipart message that contains
# all the various fully-expanded components that
# were found from processing the raw userdata string
# - when filtering only the mime messages targeting
#   this instance id will be returned (or messages with
#   no instance id)
def get_userdata(self, apply_filter=False)

# returns the raw userdata string (or none)
def get_userdata_raw(self)

# returns a integer (or none) which can be used to identify
# this instance in a group of instances which are typically
# created from a single command, thus allowing programmatic
# filtering on this launch index (or other selective actions)
@property
def launch_index(self)

# the data sources' config_obj is a cloud-config formatted
# object that came to it from ways other than cloud-config
# because cloud-config content would be handled elsewhere
```

(continues on next page)
Adding a new Datasource

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’ll 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`. 

```python
def get_config_obj(self):
    # returns a list of public ssh keys
def get_public_ssh_keys(self):
    # translates a device 'short' name into the actual physical device
    # fully qualified name (or none if said physical device is not attached
    # or does not exist)
def device_name_to_device(self, name):
    # gets the locale string this instance should be applying
    # which typically used to adjust the instances locale settings files
def get_locale(self):
    @property
def availability_zone(self):
        # gets the instance id that was assigned to this instance by the
        # cloud provider or when said instance id does not exist in the backing
        # metadata this will return 'iid-datasource'
def get_instance_id(self):
        # gets the fully qualified domain name that this host should be using
        # when configuring network or hostname related settings, typically
        # assigned either by the cloud provider or the user creating the vm
        def get_hostname(self, fqdn=False)
        def get_package_mirror_info(self)
```

54 Chapter 1. Summary
• **Add tests for ds-identify:** Add relevant tests in a new class to `tests/unittests/test_ds_identify.py`. You can use `TestOracle` as an example.

• **Add your datasource name to the builtin list of datasources:** Add your datasource module name to the end of the `datasource_list` entry in `cloudinit/settings.py`.

• **Add your cloud platform to apport collection prompts:** Update the list of cloud platforms in `cloudinit/apport.py`. This list will be provided to the user who invokes `ubuntu-bug cloud-init`.

• **Enable datasource by default in ubuntu packaging branches:** Ubuntu packaging branches contain a template file `debian/cloud-init.templates` that ultimately sets the default `datasource_list` when installed via package. This file needs updating when the commit gets into a package.

• **Add documentation for your datasource:** You should add a new file in `doc/datasources/<cloudplatform>.rst`.

### 1.8.2 Datasource Documentation

The following is a list of the implemented datasources. Follow for more information.

**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.

**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:

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

**Metadata**

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

```bash
$ curl http://100.100.100.200/2016-01-01/meta-data
dns-conf/
eipv4
hostname
image-id
```

(continues on next page)
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
#!/bin/sh
echo "Hello World."
```

Alt Cloud

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

RHEVm

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

The format of the Custom Properties entry must be:

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

For example to pass a simple bash script:

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

% base64 < simple_script.bash
IyEvYmluL2Jhc2g2gKZWNobyAiSGVsbG8qSm9IISIgPj4gL3RtcC9KS1ZfSm91X291dC50eHQK

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:IyEvYmluL2Jhc2g2gKZWNobyAiSGVsbG8qSm9IISIgPj4gL3RtcC9KS1ZfSm91X291dC50eHQK
```
NOTE: The prefix with file name must be: `floppyinject=user-data.txt`.

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

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

### vSphere

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

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

**Note:** The ISO must contain the user data.

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

**Create the ISO**

```bash
% 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**

```bash
% 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 CDROM.

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

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

### Azure

This datasource finds metadata and user-data from the Azure cloud platform.

### walinuxagent

walinuxagent has several functions within images. For cloud-init specifically, the relevant functionality it performs is to register the instance with the Azure cloud platform at boot so networking will be permitted. For more information about the other functionality of walinuxagent, see Azure’s documentation for more details. (Note, however, that only one of walinuxagent’s provisioning and cloud-init should be used to perform instance customisation.)

If you are configuring walinuxagent yourself, you will want to ensure that you have `Provisioning.UseCloudInit` set to `y`.

### 1.8. Datasources
Builtin Agent

An alternative to using walinuxagent to register to the Azure cloud platform is to use the __builtin__ agent command. This section contains more background on what that code path does, and how to enable it.

The Azure cloud platform provides initial data to an instance via an attached CD formatted in UDF. That CD contains a ‘ovf-env.xml’ file that provides some information. Additional information is obtained via interaction with the “endpoint”.

To find the endpoint, we now leverage the dhcp client’s ability to log its known values on exit. The endpoint server is special DHCP option 245. Depending on your networking stack, this can be done by calling a script in /etc/dhcp/dhclient-exit-hooks or a file in /etc/NetworkManager/dispatcher.d. Both of these call a sub-command ‘dhclient_hook’ of cloud-init itself. This sub-command will write the client information in json format to /run/cloud-init/dhclient.hook/<interface>.json.

In order for cloud-init to leverage this method to find the endpoint, the cloud.cfg file must contain:

```
datasource:
  Azure:
    set_hostname: False
    agent_command: __builtin__
```

If those files are not available, the fallback is to check the leases file for the endpoint server (again option 245).

You can define the path to the lease file with the ‘dhclient_lease_file’ configuration.

IMDS

Azure provides the instance metadata service (IMDS) which is a REST service on 196.254.196.254 providing additional configuration information to the instance. Cloud-init uses the IMDS for:

- network configuration for the instance which is applied per boot
- a preprovisioing gate which blocks instance configuration until Azure fabric is ready to provision

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:

- **agent_command**: Either __builtin__ (default) or a command to run to getcw metadata. If __builtin__, get metadata from walinuxagent. Otherwise run the provided command to obtain metadata.
- **apply_network_config**: Boolean set to True to use network configuration described by Azure’s IMDS endpoint instead of fallback network config of dhcp on eth0. Default is True. For Ubuntu 16.04 or earlier, default is False.
- **data_dir**: Path used to read metadata files and write crawled data.
- **dhclient_lease_file**: The fallback lease file to source when looking for custom DHCP option 245 from Azure fabric.
- **disk_aliases**: A dictionary defining which device paths should be interpreted as ephemeral images. See cc_disk_setup module for more info.
- **hostname_bounce**: A dictionary Azure hostname bounce behavior to react to metadata changes. The ‘hostname_bounce: command’ entry can be either the literal string ‘builtin’ or a command to execute.
The command will be invoked after the hostname is set, and will have the ‘interface’ in its environment. If `set_hostname` is not true, then `hostname_bounce` will be ignored. An example might be:

```bash
command: ["sh", "-c", "killall dhclient; dhclient $interface"]
```

- **hostname_bounce**: A dictionary Azure hostname bounce behavior to react to metadata changes. Azure will throttle ifup/down in some cases after metadata has been updated to inform dhcp server about updated hostnames.

- **set_hostname**: Boolean set to True when we want Azure to set the hostname based on metadata.

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:

```yaml
datasource:
  Azure:
    agent_command: __builtin__
    apply_network_config: true
    data_dir: /var/lib/waagent
    dhcp_client_lease_file: /var/lib/dhcp/dhclient.eth0.leases
    disk_aliases:
      ephemeral0: /dev/disk/cloud/azure_resource
    hostname_bounce:
      interface: eth0
      command: builtin
      policy: true
    hostname_command: hostname
    set_hostname: true
```

### Userdata

Userdata is provided to cloud-init inside the ovf-env.xml file. Cloud-init expects that user-data will be provided as base64 encoded value inside the text child of a element named `UserData` or `CustomData` which is a direct child of the `LinuxProvisioningConfigurationSet` (a sibling to `UserName`). If both `UserData` and `CustomData` are provided behavior is undefined on which will be selected.

In the example below, user-data provided is ‘this is my userdata’, and the datasource config provided is `"agent_command": ["start", "walinuxagent"]`. That agent command will take affect as if it were specified in system config.

Example:

```xml
<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>
      eyJhZ2VudF9jb21tYWdlIjowLCBib3JsYXlzaWVyIjogIkJsb2FkTGlzYWxpdHkyODQ0IiwiaGVsbG8iOiAiVHJ1ZSUyNSIsICJjYXNvdW50ZXIiOjEwMCwiZGF0YSI6IjEiLCJhdWQiOiAiYnVhcnkifQ==</dscfg>
    <disable_ssh_password_authentication>true</disable_ssh_password_authentication>
    <ssh/>
  </LinuxProvisioningConfigurationSet>
</wa:ProvisioningSection>
```

(continues on next page)
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.

The interesting issue is that 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. Walinuxagent (in its current version) handles this by polling the state of hostname and bouncing (‘ifdown eth0; ifup eth0’ the network interface if it sees that a change has been made.

cloud-init handles this by setting the hostname in the DataSource’s ‘get_data’ method via ‘hostname $HostName’, and then bouncing the interface. This behavior can be configured or disabled in the datasource config. See ‘Configuration’ above.

CloudSigma

This datasource finds metadata and user-data from the CloudSigma cloud platform. Data transfer occurs through a virtual serial port of the CloudSigma’s VM and the presence of network adapter is NOT a requirement. See server context in the public documentation for more information.

Setting a hostname

By default the name of the server will be applied as a hostname on the first boot.

Providing user-data

You can provide user-data to the VM using the dedicated meta field in the server context cloudinit-user-data. By default cloud-config format is expected there and the #cloud-config header could be omitted. However since this is a raw-text field you could provide any of the valid config formats.

You have the option to encode your user-data using Base64. In order to do that you have to add the cloudinit-user-data field to the base64_fields. The latter is a comma-separated field with all the meta fields whit base64 encoded values.

If your user-data does not need an internet connection you can create a meta field in the server context cloudinit-dsmode and set “local” as value. If this field does not exist the default value is “net”.

Cloud-Init, Release 19.2

<PublicKeys>
  <PublicKey>
    <Fingerprint>6BE7A7C3C8A8F4B123CCA5D0C2F1BE4CA7B63ED7</Fingerprint>
    <Path>this-value-unused</Path>
  </PublicKey>
</PublicKeys>
</SSH>
</LinuxProvisioningConfigurationSet>
</wa:ProvisioningSection>
CloudStack

Apache CloudStack expose user-data, meta-data, user password and account sshkey thru the Virtual-Router. The datasource obtains the VR address via dhcp lease information given to the instance. For more details on meta-data and user-data, refer the CloudStack Administrator Guide.

URLs to access user-data and meta-data from the Virtual Machine. Here 10.1.1.1 is the Virtual Router IP:

```
http://10.1.1.1/latest/user-data
http://10.1.1.1/latest/meta-data
http://10.1.1.1/latest/meta-data/{metadata type}
```

Configuration

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

The settings that may be configured are:

- **max_wait**: the maximum amount of clock time in seconds that should be spent searching metadata_urls. A value less than zero will result in only one request being made, to the first in the list. (default: 120)
- **timeout**: the timeout value provided to urlopen for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 50)

An example configuration with the default values is provided below:

```
datasource:
  CloudStack:
    max_wait: 120
    timeout: 50
  datasource_list:
    - CloudStack
```

Config Drive

The configuration drive datasource supports the OpenStack configuration drive disk.

See the config drive extension and introduction in the public documentation for more information.

By default, cloud-init does *always* consider this source to be a full-fledged datasource. Instead, the typical behavior is to assume it is really only present to provide networking information. Cloud-init will copy off the network information, apply it to the system, and then continue on. The “full” datasource could then be found in the EC2 metadata service. If this is not the case then the files contained on the located drive must provide equivalents to what the EC2 metadata service would provide (which is typical of the version 2 support listed below)

Version 1

Note: Version 1 is legacy and should be considered deprecated. Version 2 has been supported in OpenStack since 2012.2 (Folsom).

The following criteria are required to as a config drive:

1. Must be formatted with vfat filesystem
2. Must contain *one* of the following files
Cloud-init, Release 19.2

/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 as a config drive:

1. Must be formatted with vfat or iso9660 filesystem or have a filesystem label of config-2

2. The files that will typically be present in the config drive are:

<table>
<thead>
<tr>
<th>openstack/</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 2012-08-10/ or latest/</td>
</tr>
<tr>
<td>- meta_data.json</td>
</tr>
<tr>
<td>- user_data (not mandatory)</td>
</tr>
<tr>
<td>- content/</td>
</tr>
<tr>
<td>- 0000 (referenced content files)</td>
</tr>
<tr>
<td>- 0001</td>
</tr>
<tr>
<td>- ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ec2</th>
</tr>
</thead>
<tbody>
<tr>
<td>- latest/</td>
</tr>
<tr>
<td>- meta-data.json {not mandatory}</td>
</tr>
</tbody>
</table>

Keys and values

Cloud-init’s behavior can be modified by keys found in the meta.js (version 1 only) file in the following ways.

```
dsmode:
  values: local, net, pass
  default: pass
```

This is what indicates if configdrive is a final data source or not. By default it is ‘pass’, meaning this datasource should not be read. Set it to ‘local’ or ‘net’ to stop cloud-init from continuing on to search for other data sources after network config.

The difference between ‘local’ and ‘net’ is that local will not require networking to be up before user-data actions (or boothooks) are run.

```
instance-id:
  default: iid-dsconfigdrive
```

Chapter 1. Summary
Cloud-Init, Release 19.2

This is utilized as the metadata’s instance-id. It should generally be unique, as it is what is used to determine “is this a new instance”.

**public-keys:**

| default | None |

If present, these keys will be used as the public keys for the instance. This value overrides the content in authorized_keys.

Note: it is likely preferable to provide keys via user-data

**user-data:**

| default | None |

This provides cloud-init user-data. See *examples* for what all can be present here.

**Digital Ocean**

The DigitalOcean datasource consumes the content served from DigitalOcean’s metadata service. This metadata service serves information about the running droplet via HTTP over the link local address 169.254.169.254. The metadata API endpoints are fully described at https://developers.digitalocean.com/metadata/.

**Configuration**

DigitalOcean’s datasource can be configured as follows:

```yaml
datasource:

    DigitalOcean: retries: 3 timeout: 2

    • retries: Determines the number of times to attempt to connect to the metadata service
    • timeout: Determines the timeout in seconds to wait for a response from the metadata service
```

**Amazon EC2**

The EC2 datasource is the oldest and most widely used datasource that cloud-init supports. This datasource interacts with a *magic* ip that is provided to the instance by the cloud provider. Typically this ip is 169.254.169.254 of which at this ip a http server is provided to the instance so that the instance can make calls to get instance userdata and instance metadata.

Metadata is accessible via the following URL:

```
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
```

(continues on next page)
Userdata is accessible via the following URL:

```plaintext
1234,fred,reboot,true | 4512,jimbo, | 173,,,
```

Note that there are multiple versions of this data provided, cloud-init by default uses **2009-04-04** but newer versions can be supported with relative ease (newer versions have more data exposed, while maintaining backward compatibility with the previous versions).

To see which versions are supported from your cloud provider use the following URL:

```plaintext
GET http://169.254.169.254/
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
...
latest
```

### Configuration

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

The settings that may be configured are:

- **metadata_urls**: This list of urls will be searched for an Ec2 metadata service. The first entry that successfully returns a 200 response for `<url>/<version>/meta-data/instance-id` will be selected. (default: `['http://169.254.169.254', 'http://instance-data:8773']`).

- **max_wait**: the maximum amount of clock time in seconds that should be spent searching `metadata_urls`. A value less than zero will result in only one request being made, to the first in the list. (default: 120)

- **timeout**: the timeout value provided to urlopen for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 50)

An example configuration with the default values is provided below:

```plaintext
datasource:
  Ec2:
    max_wait: 120
    timeout: 50
```
Notes

- There are 2 types of EC2 instances network-wise: VPC ones (Virtual Private Cloud) and Classic ones (also known as non-VPC). One major difference between them is that Classic instances have their MAC address changed on stop/restart operations, so cloud-init will recreate the network config file for EC2 Classic instances every boot. On VPC instances this file is generated only in the first boot of the instance. The check for the instance type is performed by is_classic_instance() method.

MAAS

TODO
For now see: http://maas.ubuntu.com/

NoCloud

The data source NoCloud allows the user to provide user-data and meta-data to the instance without running a network service (or even without having a network at all).

You can provide meta-data and user-data to a local vm boot via files on a vfat or iso9660 filesystem. The filesystem volume label must be cidata or CIDATA.

Alternatively, you can provide meta-data via kernel command line or SMBIOS “serial number” option. The data must be passed in the form of a string:

```plaintext
ds=nocloud[key=val;key=val]
```

or

```plaintext
ds=nocloud-net[key=val;key=val]
```

The permitted keys are:

- `h` or `local-hostname`
- `i` or `instance-id`
- `s` or `seedfrom`

With `ds=nocloud`, the `seedfrom` value must start with `/` or `file://`. With `ds=nocloud-net`, the `seedfrom` value must start with `http://`, `https://` or `ftp://`

E.g. you can pass this option to QEMU:

```plaintext
-smbios type=1,serial=ds=nocloud-net;s=http://10.10.0.1:8000/
```

to cause NoCloud to fetch the full meta-data from `http://10.10.0.1:8000/meta-data` after the network initialization is complete.

These user-data and meta-data files are expected to be in the following format.

```plaintext
/user-data
/meta-data
```

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.

Given a disk ubuntu 12.04 cloud image in ‘disk.img’, you can create a sufficient disk by following the example below.
## create user-data and meta-data files that will be used to modify image on first boot

```
$ { echo instance-id: iid-local01; echo local-hostname: cloudimg; } > meta-data

$ printf "#cloud-config
password: passw0rd
chpasswd: { expire: False }
ssh_pwauth: True
" > user-data

## create a disk to attach with some user-data and meta-data
$ genisoimage -output seed.iso -volid cidata -joliet -rock user-data meta-data
```

## alternatively, create a vfat filesystem with same files
```
$ truncate --size 2M seed.img
$ mkfs.vfat -n cidata seed.img
$ mcopy -oi seed.img user-data meta-data ::
```

## create a new qcow image to boot, backed by your original image
```
$ qemu-img create -f qcow2 -b disk.img boot-disk.img
```

## boot the image and login as 'ubuntu' with password 'passw0rd'
```
$ kvm -m 256
   -net nic -net user,hostfwd=tcp::2222-:22
   -drive file=boot-disk.img,if=virtio
   -drive file=seed.iso,if=virtio
```

Note: that the instance-id provided (iid-local01 above) is what is used to determine if this is “first boot”. So if you are making updates to user-data you will also have to change that, or start the disk fresh.

Also, you can inject an `/etc/network/interfaces` file by providing the content for that file in the `network-interfaces` field of metadata.

Example metadata:

```yaml
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.

```yaml
version: 1
config:
  - type: physical
    name: interface0
    mac_address: "52:54:00:12:34:00"
    subnets:
      - type: static
```

(continues on next page)
Cloud-Init, Release 19.2

(continued from previous page)

<table>
<thead>
<tr>
<th>address: 192.168.1.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>netmask: 255.255.255.0</td>
</tr>
<tr>
<td>gateway: 192.168.1.254</td>
</tr>
</tbody>
</table>

version: 2
ethernets:
   interface0:
      match:  
         mac_address: "52:54:00:12:34:00"
      set-name: interface0
      addresses:
         - 192.168.1.10/255.255.255.0
      gateway4: 192.168.1.254

OpenNebula

The OpenNebula (ON) datasource supports the contextualization disk.

See contextualization overview, contextualizing VMs and network configuration in the public document-
tation for more information.

OpenNebula’s virtual machines are contextualized (parametrized) by CD-ROM image, which contains a shell script context.sh with custom variables defined on virtual machine start. There are no fixed contextualization variables, but the datasource accepts many used and recommended across the documentation.

Datasource configuration

Datasource accepts following configuration options.

dsmode:
   values: local, net, disabled
   default: net

Tells if this datasource will be processed in ‘local’ (pre-networking) or ‘net’ (post-networking) stage or even com-
pletely ‘disabled’.

parseuser:
   default: nobody

Unprivileged system user used for contextualization script processing.

Contextualization disk

The following criteria are required:

1. Must be formatted with iso9660 filesystem or have a filesystem label of CONTEXT or CDROM
2. Must contain file context.sh with contextualization variables. File is generated by OpenNebula, it has a
   KEY=’VALUE’ format and can be easily read by bash

1.8. Datasources
Contextualization variables

There are no fixed contextualization variables in OpenNebula, no standard. Following variables were found on various places and revisions of the OpenNebula documentation. Where multiple similar variables are specified, only first found is taken.

**DSMODE**

Datasource mode configuration override. Values: local, net, disabled.

**DNS**

**ETH<x>_IP**

**ETH<x>_NETWORK**

**ETH<x>_MASK**

**ETH<x>_GATEWAY**

**ETH<x>_DOMAIN**

**ETH<x>_DNS**

Static network configuration.

**HOSTNAME**

Instance hostname.

**PUBLIC_IP**

**IP_PUBLIC**

**ETH0_IP**

If no hostname has been specified, cloud-init will try to create hostname from instance’s IP address in ‘local’ dsmode. In ‘net’ dsmode, cloud-init tries to resolve one of its IP addresses to get hostname.

**SSH_KEY**

**SSH_PUBLIC_KEY**

One or multiple SSH keys (separated by newlines) can be specified.

**USER_DATA**

**USERDATA**

cloud-init user data.

Example configuration

This example cloud-init configuration (*cloud.cfg*) enables OpenNebula datasource only in ‘net’ mode.

```yaml
disable_ec2_metadata: True
datasource_list: ['OpenNebula']
datasource:
  OpenNebula:
    dsmode: net
    parseuser: nobody
```
Example VM’s context section

```python
CONTEXT=[
    PUBLIC_IP="$NIC[IP]",
    SSH_KEY="$USER[SSH_KEY]
    $USER[SSH_KEY1]
    $USER[SSH_KEY2]",
    USER_DATA="#cloud-config
    # see https://help.ubuntu.com/community/CloudInit
    packages: []
    mounts:
        - [vdc, none, swap, sw, 0, 0]
    runcmd:
        - echo 'Instance has been configured by cloud-init.' | wall
]
```

OpenStack

This datasource supports reading data from the OpenStack Metadata Service.

Discovery

To determine whether a platform looks like it may be OpenStack, cloud-init checks the following environment attributes as a potential OpenStack platform:

- Maybe OpenStack if
  - non-x86 cpu architecture: because DMI data is buggy on some arches
- Is OpenStack if x86 architecture and ANY of the following
  - /proc/1/environ: Nova-lxd contains `product_name=OpenStack Nova`
  - DMI product_name: Either Openstack Nova or OpenStack Compute
  - DMI chassis_asset_tag is `OpenTelekomCloud`

Configuration

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

The settings that may be configured are:

- **metadata_urls**: This list of urls will be searched for an OpenStack metadata service. The first entry that successfully returns a 200 response for `<url>/openstack` will be selected. (default: `['http://169.254.169.254']`).
- **max_wait**: the maximum amount of clock time in seconds that should be spent searching metadata_urls. A value less than zero will result in only one request being made, to the first in the list. (default: -1)
- **timeout**: the timeout value provided to urlopen for each individual http request. This is used both when selecting a metadata_url and when crawling the metadata service. (default: 10)
- **retries**: The number of retries that should be done for an http request. This value is used only after metadata_url is selected. (default: 5)
Cloud-Init, Release 19.2

- **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)

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
package_upgrade: True
packages:
 - htop"}
```

For more general information about how cloud-init handles vendor data, including how it can be disabled by users on instances, see [Vendor Data](#).

**Oracle**

This datasource reads metadata, vendor-data and user-data from Oracle Compute Infrastructure (OCI).

**Oracle Platform**

OCI provides bare metal and virtual machines. In both cases, the platform identifies itself via DMI data in the chassis asset tag with the string ‘OracleCloud.com’.

Oracle’s platform provides a metadata service that mimics the 2013-10-17 version of OpenStack metadata service. Initially support for Oracle was done via the OpenStack datasource.

**Cloud-init has a specific datasource for Oracle in order to:**

a. allow and support future growth of the OCI platform.

b. address small differences between OpenStack and Oracle metadata implementation.

**OVF**

The OVF Datasource provides a datasource for reading data from on an Open Virtualization Format ISO transport.

For further information see a full working example in cloud-init’s source code tree in doc/sources/ovf.
SmartOS Datasource

This datasource finds metadata and user-data from the SmartOS virtualization platform (i.e. Joyent).
Please see http://smartos.org/ for information about SmartOS.

SmartOS Platform

The SmartOS virtualization platform uses meta-data to the instance via the second serial console. On Linux, this is /dev/ttyS1. The data is provided via a simple protocol: something queries for the data, the console responds with the status and if “SUCCESS” returns until a single “:n”.

New versions of the SmartOS tooling will include support for base64 encoded data.

Meta-data channels

Cloud-init supports three modes of delivering user/meta-data via the flexible channels of SmartOS.

- user-data is written to /var/db/user-data
  - per the spec, user-data is for consumption by the end-user, not provisioning tools
  - cloud-init entirely ignores this channel other than writing it to disk
  - removal of the meta-data key means that /var/db/user-data gets removed
  - a backup of previous meta-data is maintained as /var/db/user-data.<timestamp>. <timestamp> is the epoch time when cloud-init ran

- user-script is written to /var/lib/cloud/scripts/per-boot/99_user_data
  - this is executed each boot
  - a link is created to /var/db/user-script
  - previous versions of the user-script is written to /var/lib/cloud/scripts/per-boot.backup/99_user_script.<timestamp>. <timestamp> is the epoch time when cloud-init ran.
  - when the ‘user-script’ meta-data key goes missing, the user-script is removed from the file system, although a backup is maintained.
  - if the script is not shebanged (i.e. starts with #!/<executable>), then or is not an executable, cloud-init will add a shebang of “#!/bin/bash”

- cloud-init:user-data is treated like on other Clouds.
  - this channel is used for delivering _all_ cloud-init instructions
  - scripts delivered over this channel must be well formed (i.e. must have a shebang)

Cloud-init supports reading the traditional meta-data fields supported by the SmartOS tools. These are:

- root_authorized_keys
- hostname
- enable_motd_sys_info
- iptables_disable

Note: At this time iptables_disable and enable_motd_sys_info are read but are not actioned.
Disabling user-script

Cloud-init uses the per-boot script functionality to handle the execution of the user-script. If you want to prevent this use a cloud-config of:

```
#cloud-config
cloud_final_modules:
- scripts-per-once
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
- phone-home
- final-message
- power-state-change
```

Alternatively you can use the json patch method

```
#cloud-config-jsonp
{
   "op": "replace",
   "path": "/cloud_final_modules",
   "value": ["scripts-per-once",
              "scripts-per-instance",
              "scripts-user",
              "ssh-authkey-fingerprints",
              "keys-to-console",
              "phone-home",
              "final-message",
              "power-state-change"]
}
```

The default cloud-config includes “script-per-boot”. Cloud-init will still ingest and write the user-data but will not execute it, when you disable the per-boot script handling.

**Note:** Unless you have an explicit use-case, it is recommended that you not disable the per-boot script execution, especially if you are using any of the life-cycle management features of SmartOS.

The cloud-config needs to be delivered over the cloud-init:user-data channel in order for cloud-init to ingest it.

**base64**

The following are exempt from base64 encoding, owing to the fact that they are provided by SmartOS:

- root_authorised_keys
- enable_motd_sys_info
- iptables_disable
- user-data
- user-script

This list can be changed through system config of variable ‘no_base64_decode’.

This means that user-script and user-data as well as other values can be base64 encoded. Since Cloud-init can only guess as to whether or not something is truly base64 encoded, the following meta-data keys are hints as to whether or not to base64 decode something:
• base64_all: Except for excluded keys, attempt to base64 decode the values. If the value fails to decode properly, it will be returned in its text
• base64_keys: A comma delimited list of which keys are base64 encoded.
• b64-<key>: for any key, if there exists an entry in the metadata for ‘b64-<key>’ Then ‘b64-<key>’ is expected to be a plaintext boolean indicating whether or not its value is encoded.
• no_base64_decode: This is a configuration setting (i.e. /etc/cloud/cloud.cfg.d) that sets which values should not be base64 decoded.

disk_aliases and ephemeral disk

By default, SmartOS only supports a single ephemeral disk. That disk is completely empty (un-partitioned with no filesystem).

The SmartOS datasource has built-in cloud-config which instructs the ‘disk_setup’ module to partition and format the ephemeral disk.

You can control the disk_setup then in 2 ways:

1. through the datasource config, you can change the ‘alias’ of ephemeral0 to reference another device. The default is:

   ‘disk_aliases’: {‘ephemeral0’: ‘/dev/vdb’},

   Which means anywhere disk_setup sees a device named ‘ephemeral0’ then /dev/vdb will be substituted.

2. you can provide disk_setup or fs_setup data in user-data to overwrite the datasource’s built-in values.

See doc/examples/cloud-config-disk-setup.txt for information on disk_setup.

Fallback/None

This is the fallback datasource when no other datasource can be selected. It is the equivalent of a empty datasource in that it provides a empty string as userdata and a empty dictionary as metadata. It is useful for testing as well as for when you do not have a need to have an actual datasource to meet your instance requirements (ie you just want to run modules that are not concerned with any external data). It is typically put at the end of the datasource search list so that if all other datasources are not matched, then this one will be so that the user is not left with an inaccessible instance.

Note: the instance id that this datasource provides is iid-datasource-none.

Google Compute Engine

The GCE datasource gets its data from the internal compute metadata server. Metadata can be queried at the URL ‘http://metadata.google.internal/computeMetadata/v1/’ from within an instance. For more information see the GCE metadata docs.

Currently the default project and instance level metadatakeys keys project/attributes/sshKeys and instance/attributes/ssh-keys are merged to provide public-keys.

user-data and user-data-encoding can be provided to cloud-init by setting those custom metadata keys for an instance.
1.9 Logging

Cloud-init supports both local and remote logging configurable through python’s built-in logging configuration and through the cloud-init rsyslog module.

1.9.1 Command Output

Cloud-init can redirect its stdout and stderr based on config given under the `output` config key. The output of any commands run by cloud-init and any user or vendor scripts provided will also be included here. The `output` key accepts a dictionary for configuration. Output files may be specified individually for each stage (init, config, and final), or a single key `all` may be used to specify output for all stages.

The output for each stage may be specified as a dictionary of `output` and `error` keys, for stdout and stderr respectively, as a tuple with stdout first and stderr second, or as a single string to use for both. The strings passed to all of these keys are handled by the system shell, so any form of redirection that can be used in bash is valid, including piping cloud-init’s output to `tee`, or `logger`. If only a filename is provided, cloud-init will append its output to the file as though `>>` was specified.

By default, cloud-init loads its output configuration from `/etc/cloud/cloud.cfg.d/05_logging.cfg`. The default config directs both stdout and stderr from all cloud-init stages to `/var/log/cloud-init-output.log`.

The default config is given as

```yaml
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.

```yaml
output:
  init:
    output: "> /var/log/cloud-init.out"
    error: "> /var/log/cloud-init.err"
  config: "tee -a /var/log/cloud-config.log"
  final:
    - ">> /var/log/cloud-final.out"
    - "> /var/log/cloud-final.err"
```

Python Logging

Cloud-init uses the python logging module, and can accept config for this module using the standard python fileConfig format. Cloud-init looks for config for the logging module under the `logcfg` key.

**Note:** the logging configuration is not yaml, it is python `fileConfig` format, and is passed through directly to the python logging module. Please use the correct syntax for a multi-line string in yaml.

By default, cloud-init uses the logging configuration provided in `/etc/cloud/cloud.cfg.d/05_logging.cfg`. The default python logging configuration writes all cloud-init events with a priority of `WARNING` or higher to console, and writes all events with a level of `DEBUG` or higher to `/var/log/cloud-init.log` and via syslog.

Python’s `fileConfig` format consists of sections with headings in the format `[title]` and key value pairs in each section. Configuration for python logging must contain the sections `[loggers]`, `[handlers]`, and `[formatters]`.
which name the entities of their respective types that will be defined. The section name for each defined logger, handler and formatter will start with its type, followed by an underscore (_) and the name of the entity. For example, if a logger was specified with the name log01, config for the logger would be in the section [logger_log01].

Logger config entries contain basic logging 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 the python’s logging package namespace. A handler may specify a message formatter to use, a priority level, and arguments for the handler class. Common handlers are StreamHandler, which handles stream redirects (i.e. logging to stderr), and FileHandler which outputs to a log file. The logging module also supports logging over net sockets, over http, via smtp, and additional complex configurations. For full details about the handlers available for python logging, please see the documentation for python logging handlers.

Log messages are formatted using the logging.Formatter class, which is configured using formatter config entities. A default format of %(message)s is given if no formatter configs are specified. Formatter config entities accept a format string which supports variable replacements. These may also accept a datefmt string which may be used to configure the timestamp used in the log messages. The format variables %(asctime)s, %(levelname)s and %(message)s are commonly used and represent the timestamp, the priority level of the event and the event message. For additional information on logging formatters see python logging formatters.

Note: by default the format string used in the logging formatter are in python’s old style %s form. the str.format() and string.Template styles can also be used by using { or $ in place of % by setting the style parameter in formatter config.

A simple, but functional python logging configuration for cloud-init is below. It will log all messages of priority DEBUG or higher both stderr and /tmp/my.log using a StreamHandler and a FileHandler, using the default format string %(message)s:

```
logcfg: |
[loggers]
keys=root,cloudinit
[handlers]
keys=ch,cf
[formatters]
keys=
[logger_root]
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 log-
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:

```yaml
rsyslog:
  remotes:
    log_serv: "*.* @10.0.4.1:514"
```

With defaults used:

```yaml
rsyslog:
  remotes:
    log_serv: "10.0.4.1"
```

For more information on rsyslog configuration, see Rsyslog.

1.10 Modules

1.10.1 Apt Configure

Summary: configure apt

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.

Preserve sources.list:
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`.

**Note:** 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 source suites:

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`

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

### Configure primary and security mirrors:

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 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`.

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`

### Specify sources.list template:

1.10. Modules
A custom template for rendering sources.list can be specified with 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

Pass configuration to apt:

Apt configuration can be specified using conf. Configuration is specified as a string. For multiline apt configuration, make sure to follow yaml syntax.

Configure apt proxy:

Proxy configuration for apt can be specified using conf, but proxy config keys also exist for convenience. The proxy config keys, http_proxy, ftp_proxy, and https_proxy may be used to specify a proxy for http, ftp and https protocols respectively. The proxy key also exists as an alias for http_proxy. Proxy url is specified in the format <protocol>://[[user]][:pass]@[host][:port]/.

Add apt repos by regex:

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

Add source list entries:

Source list entries can be specified as a dictionary under the sources config key, with 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

The source key supports variable replacements for the following strings:

- $MIRROR
- $PRIMARY
- $SECURITY
- $RELEASE

Internal name: cc_apt_configure

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:
apt:
  preserve_sources_list: <true/false>
  disableSuites:
    - $RELEASE-updates
    - backports
    - $RELEASE
    - mysuite
  primary:
    arches:
      - amd64
      - i386
      - default
    uri: "http://us.archive.ubuntu.com/ubuntu"
  search:
    - "http://cool.but-sometimes-unreachable.com/ubuntu"
    - "http://us.archive.ubuntu.com/ubuntu"
  search_dns: <true/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
  debconfSelections:
    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 http://<url>/ xenial main"
    source2:
      source: "ppa:<ppa-name>"
    source3:
      source: "deb $MIRROR $RELEASE multiverse"
      key: |
        ------BEGIN PGP PUBLIC KEY BLOCK------
        <key data>
        ------END PGP PUBLIC KEY BLOCK------
1.10.2 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). The apt_pipelining config key may be set to false to disable pipelining altogether. This is the default behavior. If it is set to none, unchanged, or os, no change will be made to apt configuration and the default setting for the distro will be used. The pipeline depth can also be manually specified by setting apt_pipelining to a number. However, this is not recommended.

Internal name: cc_apt_pipelining
Module frequency: per instance
Supported distros: ubuntu, debian
Config keys:: apt_pipelining: <false/none/unchanged/os/number>

1.10.3 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 boohook would run. This is very similar to a boohook, but more user friendly. The environment variable INSTANCE_ID will be set to the current instance id for all run commands. Commands can be specified either as lists or strings. For invocation details, see runcmd.

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

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

Internal name: cc_bootcmd
Module frequency: always
Supported distros: all
Config schema: bootcmd: (array of (array of string)/(string))
Examples:

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

1.10.4 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: per instance

Supported distros: ubuntu, debian

Config keys:

| byobu_by_default: <user/system> |

1.10.5 CA Certs

Summary: add ca certificates

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

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

Internal name: cc_ca_certs

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:

| ca-certs: |
| remove-defaults: <true/false>
<p>| trusted: |</p>
<table>
<thead>
<tr>
<th>- &lt;single line cert&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----BEGIN CERTIFICATE-----</td>
</tr>
<tr>
<td>YOUR-ORGS-TRUSTED-CA-CERT-HERE</td>
</tr>
<tr>
<td>-----END CERTIFICATE-----</td>
</tr>
</tbody>
</table>

1.10.6 Chef

Summary: module that configures, starts and installs chef.
This module enables chef to be installed (from packages or from gems, or from omnibus). Before this occurs chef configurations are written to disk (validation.pem, client.pem, firstboot.json, client.rb), and needed chef folders/directories are created (/etc/chef and /var/log/chef and so-on). Then once installing proceeds correctly if configured chef will be started (in daemon mode or in non-daemon mode) and then once that has finished (if ran in non-daemon mode this will be when chef finishes converging, if ran in daemon mode then no further actions are possible since chef will have forked into its own process) then a post run function can run that can do finishing activities (such as removing the validation pem file).

**Internal name:** cc_chef  
**Module frequency:** per always  
**Supported distros:** all

**Config keys:**

<table>
<thead>
<tr>
<th>Chef</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>directories</strong></td>
<td>(defaulting to /etc/chef, /var/log/chef, /var/lib/chef, /var/cache/chef, /var/backups/chef, /var/run/chef)</td>
</tr>
<tr>
<td><strong>validation_cert</strong></td>
<td>(optional string to be written to file validation_key) special value 'system' means set use existing file</td>
</tr>
<tr>
<td><strong>validation_key</strong></td>
<td>(optional the path for validation_cert. default /etc/chef/validation.pem)</td>
</tr>
<tr>
<td><strong>firstboot_path</strong></td>
<td>(path to write run_list and initial_attributes keys that should also be present in this configuration, defaults to /etc/chef/firstboot.json)</td>
</tr>
<tr>
<td><strong>exec</strong></td>
<td>boolean to run or not run chef (defaults to false, unless a gem installed is requested where this will then default to true)</td>
</tr>
</tbody>
</table>

chef.rb template keys (if falsey, then will be skipped and not written to /etc/chef/client.rb)

<table>
<thead>
<tr>
<th>Chef</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>client_key</strong></td>
<td></td>
</tr>
<tr>
<td><strong>encrypted_data_bag_secret</strong></td>
<td></td>
</tr>
<tr>
<td><strong>environment</strong></td>
<td></td>
</tr>
<tr>
<td><strong>file_backup_path</strong></td>
<td></td>
</tr>
<tr>
<td><strong>file_cache_path</strong></td>
<td></td>
</tr>
<tr>
<td><strong>json_attribs</strong></td>
<td></td>
</tr>
<tr>
<td><strong>log_level</strong></td>
<td></td>
</tr>
<tr>
<td><strong>log_location</strong></td>
<td></td>
</tr>
<tr>
<td><strong>node_name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omnibus_url</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omnibus_url_retries</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omnibus_version</strong></td>
<td></td>
</tr>
<tr>
<td><strong>pid_file</strong></td>
<td></td>
</tr>
<tr>
<td><strong>server_url</strong></td>
<td></td>
</tr>
<tr>
<td><strong>show_time</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ssl_verify_mode</strong></td>
<td></td>
</tr>
<tr>
<td><strong>validation_cert</strong></td>
<td></td>
</tr>
<tr>
<td><strong>validation_key</strong></td>
<td></td>
</tr>
<tr>
<td><strong>validation_name</strong></td>
<td></td>
</tr>
</tbody>
</table>

### 1.10.7 Debug

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

Note: Log configurations are not output.

Internal name: cc_debug
Module frequency: per instance
Supported distros: all
Config keys:

```
debug:
    verbose: true/false (defaulting to true)
    output: (location to write output, defaulting to console + log)
```

### 1.10.8 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: per always
Supported distros: all
Config keys:

```
disable_ec2_metadata: <true/false>
```

### 1.10.9 Disk Setup

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

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

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

Disk partitioning is done using the disk_setup directive. This config directive accepts a dictionary where each key is either a path to a block device or an alias specified in device_aliases, and each value is the configuration options for the device. The table_type option specifies the partition table type, either mbr or gpt. The layout option specifies how partitions on the device are to be arranged. If layout is 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. 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 overwrite option controls whether this module tries to be safe about writing partition talbes 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.

**Note:** 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.

File system configuration is done using the fs_setup directive. This config directive accepts a list of filesystem configs. The device to create the filesystem on may be specified either as a path or as an alias in the format <alias name>.<y> where <y> denotes the partition number on the device. The partition can also be specified by setting partition to the desired partition number. The partition option may also be set to auto, in which this module will search for the existence of a filesystem matching the label, type and device of the fs_setup entry and will skip creating the filesystem if one is found. The partition option may also be set to any, in which case any file system that matches type and device will cause this module to skip filesystem creation for the fs_setup entry, regardless of label matching or not. To write a filesystem directly to a device, use partition: none. A label can be specified for the filesystem using label, and the filesystem type can be specified using filesystem.

**Note:** If specifying device using the <device name>.<partition number> format, the value of partition will be overwritten.

**Note:** Using overwrite: true for filesystems is dangerous and can lead to data loss, so double check the entry in fs_setup.

**Note:** replace_fs is ignored unless partition is auto or any.

**Internal name:** cc_disk_setup

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

```markdown
device_aliases:
  <alias name>: <device path>
disk_setup:
  <alias name/path>:
    table_type: <'mbr'/'gpt'>
    layout:
      - [33,82]
      - 66
    overwrite: <true/false>
fs_setup:
  - label: <label>
    filesystem: <filesystem type>
    device: <device>
    partition: <"auto"/"any"/"none"/<partition number>>
    overwrite: <true/false>
    replace_fs: <filesystem type>
```
1.10.10 Emit Upstart

**Summary:** emit upstart configuration

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

**Internal name:** cc_emit_upstart

**Module frequency:** per always

**Supported distros:** ubuntu, debian

1.10.11 Fan

**Summary:** configure ubuntu fan networking

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

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

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

**Internal name:** cc_fan

**Module frequency:** per instance

**Supported distros:** ubuntu

**Config keys:**

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

1.10.12 Final Message

**Summary:** output final message when cloud-init has finished

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

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

**Internal name:** cc_final_message

**Module frequency:** per always
Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>Config key</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_message: &lt;message&gt;</td>
</tr>
</tbody>
</table>

1.10.13 Foo

Summary: example module

Example to show module structure. Does not do anything.

Internal name: cc_foo

Module frequency: per instance

Supported distros: all

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

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

There is some functionality overlap between this module and the growroot functionality of cloud-initramfs-tools. However, there are some situations where one tool is able to function and the other is not. The default configuration for both should work for most cloud instances. To explicitly prevent cloud-initramfs-tools from running growroot, the file /etc/growroot-disabled can be created. By default, both growroot and cc_growpart will check for the existance 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: per always

Supported distros: all

Config keys:
1.10.15 Grub Dpkg

Summary: configure grub debconf installation device

Configure which device is used as the target for grub installation. This module should work correctly by default without any user configuration. It can be enabled/disabled using the enabled config key in the grub_dpkg config dict. The global config key grub-dpkg is an alias for grub_dpkg. If no installation device is specified this module will look for the first existing device in:

- /dev/sda
- /dev/vda
- /dev/xvda
- /dev/sda1
- /dev/vda1
- /dev/xvda1

Internal name: cc_grub_dpkg

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:

<table>
<thead>
<tr>
<th>grub_dpkg:</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled: &lt;true/false&gt;</td>
</tr>
<tr>
<td>grub-pc/install_devices: &lt;devices&gt;</td>
</tr>
<tr>
<td>grub-pc/install_devices_empty: &lt;devices&gt;</td>
</tr>
<tr>
<td>grub-dpkg: (alias for grub_dpkg)</td>
</tr>
</tbody>
</table>

1.10.16 Keys to Console

Summary: control which ssh keys may be written to console

For security reasons it may be desirable not to write ssh fingerprints and keys to the console. To avoid the fingerprint of types of ssh 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 keys of a key type being written to console the ssh_key_console_blacklist config key can be used. By default ssh-dss keys are not written to console.

Internal name: cc_keys_to_console

Module frequency: per instance

Supported distros: all

Config keys:
ssh_fp_console_blacklist: <list of key types>
ssh_key_console_blacklist: <list of key types>

1.10.17 Landscape

Summary: install and configure landscape client

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

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

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

Note: see landscape documentation for client config keys

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

Internal name: cc_landscape
Module frequency: per instance
Supported distros: ubuntu
Config keys:

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

1.10.18 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: per instance
Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>locale: &lt;locale str&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale_configfile: &lt;path to locale config file&gt;</td>
</tr>
</tbody>
</table>

### 1.10.19 LXD

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

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

**Internal name:** cc_lxd

**Module frequency:** per instance

**Supported distros:** ubuntu

**Config keys:**

```
1.10.20 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`.

1.10. Modules

89```
Cloud-Init, Release 19.2

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

Internal name: cc_mcollective
Module frequency: per instance
Supported distros: all
Config keys:

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

1.10.21 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: per always
Supported distros: all
Config keys:

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

1.10.22 Mounts

Summary: configure mount points and swap files

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

In order to remove a previously listed mount, an entry can be added to the mounts list containing fs_spec for the device to be removed but no mountpoint (i.e. [ sdal ] or [ sdal, 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 7 values. It defaults to:
mount_default_fields: [none, none, "auto", "defaults,nobootwait", "0", "2"]

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

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

Note that nobootwait is an upstart specific boot option that somewhat equates to the more standardnofail.

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

Internal name: cc_mounts
Module frequency: per instance
Supported distros: all

Config keys:

mounts:
- [ /dev/ephemeral0, /mnt, auto, "defaults,noexec" ]
- [ sdc, /opt/data ]
- [ xvdb, /opt/data, "auto", "defaults,nofail", "0", "0" ]

mount_default_fields: [None, None, "auto", "defaults,nofail", "0", "2"]

swap:
  filename: <file>
  size: <"auto"/size in bytes>
  maxsize: <size in bytes>

1.10.23 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 /etc/ntp.conf.dist 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: centos, debian, fedora, opensuse, rhel, sles, ubuntu

Config schema: ntp: (object/null)

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

  config: (object) Configuration settings or overrides for the ntp_client specified.
    service_name: (string) The systemd or sysvinit service name used to start and stop the ntp_client service.
    packages: (array of string) List of packages needed to be installed for the selected ntp_client.
**check_exe**: (string) The executable name for the `ntp_client`. For example, ntp service `check_exe` is ‘ntpd’ because it runs the ntpd binary.

**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.

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

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

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

**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`.

**Examples**:

```plaintext
# 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 %}
            {% for server in servers -%}
            server {{server}} iburst
            {% endfor %}
        pools: [0.int.pool.ntp.org, 1.int.pool.ntp.org, ntp.myorg.org]
        servers:
        - ntp.server.local
        - ntp.ubuntu.com
        - 192.168.23.2
```

### 1.10.24 Package Update Upgrade Install

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

**Internal name:** cc_package_update_upgrade_install

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

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

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

### 1.10.25 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
- instance_id
- hostname
- fdqn

**Internal name:** cc_phone_home

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

```plaintext
phone_home:
  url: http://example.com/$INSTANCE_ID/
  post:
    - pub_key_dsa
    - instance_id
    - fdqn
  tries: 10
```
1.10.26 Power State Change

Summary: change power state

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

The system can be shutdown before cloud-init has finished using the timeout option. The delay key specifies a duration to be added onto any shutdown command used. Therefore, if a 5 minute delay and a 120 second shutdown are specified, the maximum amount of time between cloud-init starting and the system shutting down is 7 minutes, and the minimum amount of time is 5 minutes. The delay key must have an argument in a form that the shutdown utility recognizes. The most common format is the form +5 for 5 minutes. See man shutdown for more options.

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

Internal name: cc_power_state_change
Module frequency: per instance
Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>power_state:</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay: &lt;now/+minutes'&gt;</td>
</tr>
<tr>
<td>mode: &lt;poweroff/halt/reboot&gt;</td>
</tr>
<tr>
<td>message: &lt;shutdown message&gt;</td>
</tr>
<tr>
<td>timeout: &lt;seconds&gt;</td>
</tr>
<tr>
<td>condition: &lt;true/false/command&gt;</td>
</tr>
</tbody>
</table>

1.10.27 Puppet

Summary: install, configure and start puppet

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

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

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

Module frequency: per instance

Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet</td>
<td>install: &lt;true/false&gt;</td>
</tr>
<tr>
<td></td>
<td>version: &lt;version&gt;</td>
</tr>
<tr>
<td></td>
<td>conf_file: '/etc/puppet/puppet.conf'</td>
</tr>
<tr>
<td></td>
<td>ssl_dir: '/var/lib/puppet/ssl'</td>
</tr>
<tr>
<td></td>
<td>package_name: 'puppet'</td>
</tr>
<tr>
<td>conf</td>
<td>agent:</td>
</tr>
<tr>
<td></td>
<td>server: &quot;puppetmaster.example.org&quot;</td>
</tr>
<tr>
<td></td>
<td>certname: &quot;%i.%f&quot;</td>
</tr>
<tr>
<td></td>
<td>ca_cert:</td>
</tr>
<tr>
<td></td>
<td>-------BEGIN CERTIFICATE-------</td>
</tr>
<tr>
<td></td>
<td>&lt;cert_data&gt;</td>
</tr>
<tr>
<td></td>
<td>-------END CERTIFICATE-------</td>
</tr>
</tbody>
</table>

1.10.28 Resizefs

Summary: Resize filesystem

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

Internal name: cc_resizefs

Module frequency: always

Supported distros: all

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

Examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resize_rootfs: false</td>
<td># disable root filesystem resize operation</td>
</tr>
</tbody>
</table>

1.10.29 Resolv Conf

Summary: configure resolv.conf

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

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

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

Internal name: cc_resolv_conf
Module frequency: per instance
Supported distros: fedora, rhel, sles
Config keys:

<table>
<thead>
<tr>
<th>Config keys</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage_resolv_conf</td>
<td>&lt;true/false&gt;</td>
</tr>
<tr>
<td>resolv_conf</td>
<td></td>
</tr>
<tr>
<td>nameservers</td>
<td>['8.8.4.4', '8.8.8.8']</td>
</tr>
<tr>
<td>searchdomains</td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>example.com</td>
</tr>
<tr>
<td>options</td>
<td></td>
</tr>
<tr>
<td>rotate</td>
<td>&lt;true/false&gt;</td>
</tr>
<tr>
<td>timeout</td>
<td>1</td>
</tr>
</tbody>
</table>

1.10.30 RedHat Subscription

Summary: register red hat enterprise linux based system

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

Internal name: cc_rh_subscription
Module frequency: per instance
Supported distros: rhel, fedora
Config keys:

<table>
<thead>
<tr>
<th>Config keys</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>rh_subscription</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>&lt;username&gt;</td>
</tr>
<tr>
<td>password</td>
<td>&lt;password&gt;</td>
</tr>
<tr>
<td>activation-key</td>
<td>&lt;activation key&gt;</td>
</tr>
<tr>
<td>org</td>
<td>&lt;org number&gt;</td>
</tr>
<tr>
<td>auto-attach</td>
<td>&lt;true/false&gt;</td>
</tr>
<tr>
<td>service-level</td>
<td>&lt;service level&gt;</td>
</tr>
<tr>
<td>add-pool</td>
<td>&lt;list of pool ids&gt;</td>
</tr>
<tr>
<td>enable-repo</td>
<td>&lt;list of yum repo ids&gt;</td>
</tr>
<tr>
<td>disable-repo</td>
<td>&lt;list of yum repo ids&gt;</td>
</tr>
<tr>
<td>rhsm-baseurl</td>
<td>&lt;url&gt;</td>
</tr>
<tr>
<td>server-hostname</td>
<td>&lt;hostname&gt;</td>
</tr>
</tbody>
</table>

1.10.31 Rightscale Userdata

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

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

Internal name: cc_rightscale_userdata
Module frequency: per instance
Supported distros: all
Config keys:

CLOUD_INIT_REMOTE_HOOK=<url>

1.10.32 Rsyslog

Summary: configure system logging via rsyslog

This module configures remote system logging using rsyslog.

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

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

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

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

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

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

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

- rsyslog-> rsyslog/configs
Cloud-Init, Release 19.2

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

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

**Internal name:** cc_rsyslog

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

```yaml
rsyslog:
  config_dir: config_dir
  config_filename: config_filename
  configs:
  - ".* @@192.158.1.1"
  - content: ".* @@192.0.2.1:10514"
    filename: 01-example.conf
  - content: |
    ".* @@syslogd.example.com"
  remotes:
  - maas: "192.168.1.1"
  - juju: "10.0.4.1"
  service_reload_command: [your, syslog, restart, command]
```

**Legacy config keys:**

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

### 1.10.33 Runcmd

**Summary:** Run arbitrary commands

Run arbitrary commands at a `rc.local` like level with output to the console. Each item can be either a list or a string. If the item is a list, it will be properly executed as if passed to `execve()` (with the first arg as the command). If the item is a string, it will be written to a file and interpreted using `sh`.

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

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

**Internal name:** cc_runcmd

**Module frequency:** once-per-instance

**Supported distros:** all

**Config schema:** runcmd: (array of (array of string)/(string))
Examples:

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

## 1.10.34 Salt Minion

### Summary:
set up and run salt minion

This module installs, configures and starts salt minion. If the `salt_minion` key is present in the config parts, then salt minion will be installed and started. Configuration for salt minion can be specified in the `conf` key under `salt_minion`. Any conf values present there will be assigned in `/etc/salt/minion`. The public and private keys to use for salt minion can be specified with `public_key` and `private_key` respectively. Optionally if you have a custom package name, service name or config directory you can specify them with `pkg_name`, `service_name` and `config_dir`.

**Internal name:** cc_salt_minion

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

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

## 1.10.35 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:** per always

**Supported distros:** all
1.10.36 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.

Internal name: cc_scripts_per_instance

Module frequency: per instance

Supported distros: all

1.10.37 Scripts Per Once

Summary: run one time scripts

Any scripts in the scripts/per-once directory on the datasource will be run only once. Scripts will be run in alphabetical order. This module does not accept any config keys.

Internal name: cc_scripts_per_once

Module frequency: per once

Supported distros: all

1.10.38 Scripts User

Summary: run user scripts

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

Internal name: cc_scripts_user

Module frequency: per instance

Supported distros: all

1.10.39 Scripts Vendor

Summary: run vendor scripts

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

Internal name: cc_scripts_vendor

Module frequency: per instance

Supported distros: all

Config keys:

vendor_data:
  prefix: <vendor data prefix>
1.10.40 Seed Random

Summary: provide random seed data

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

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

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

Instead of specifying a data string, a command can be run to generate/collect the data to be written. The command should be specified as a list of args in the command key. If a command is specified that cannot be run, no error will be reported unless command_required is set to true.

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

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

Internal name: cc_seed_random

Module frequency: per instance

Supported distros: all

Config keys:

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

1.10.41 Set Hostname

Summary: set hostname and fqdn

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

A hostname and fqdn can be provided by specifying a full domain name under the fqdn key. Alternatively, a hostname can be specified using the hostname key, and the fqdn of the cloud will be used. If a fqdn specified with the hostname key, it will be handled properly, although it is better to use the fqdn config key. If both fqdn and hostname are set, fqdn will be used.

Internal name: per instance

Supported distros: all
Config keys:

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

### 1.10.42 Set Passwords

**Summary:** Set user passwords

Set system passwords and enable or disable ssh password authentication. The `chpasswd` config key accepts a dictionary containing a single one of two keys, either `expire` or `list`. If `expire` is specified and is set to `false`, then the `password` global config key is used as the password for all user accounts. If the `expire` key is specified and is set to `true` then user passwords will be expired, preventing the default system passwords from being used.

If the `list` key is provided, a list of `username:password` pairs can be specified. The usernames specified must already exist on the system, or have been created using the `cc_users_groups` module. A password can be randomly generated using `username:RANDOM` or `username:R`. A hashed password can be specified using `username:$6$salt$hash`. Password ssh authentication can be enabled, disabled, or left to system defaults using `ssh_pwauth`.

**Note:** if using `expire: true` then a ssh authkey should be specified or it may not be possible to login to the system

**Internal name:** `cc_set_passwords`

**Module frequency:** per instance

**Supported distros:** all

Config keys:

```
ssh_pwauth: <yes/no/unchanged>
password: password1
chpasswd:
    expire: <true/false>
chpasswd:
    list: |
        user1:password1
        user2:RANDOM
        user3:password3
        user4:R

##
# or as yaml list
##
```

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

**Summary:** Install, configure and manage snapd and snap packages

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

---

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

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

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

---

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

---

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

**Internal name:** cc_snap

**Module frequency:** once-per-instance

**Supported distros:** ubuntu

**Config schema:** `snap`: (object)

- `squashfuse_in_container`: (boolean)
- `commands`: (object/array of (array of string)/(string))
- `assertions`: (object/array of string)

**Examples:**

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

# --- Example2 ---
# LXC-based containers require squashfuse before snaps can be installed

snap:
  commands:
    00: apt-get install squashfuse -y
    11: snap install emoji
```

(continues on next page)
# Example3

Convenience: the snap command can be omitted when specifying commands as a list and 'snap' will automatically be prepended. The following commands are equivalent:

```bash
snap:
  commands:
    00: ['install', 'vlc']
    01: ['snap', 'install', 'vlc']
    02: snap install vlc
    03: 'snap install vlc'
```

## 1.10.44 Snappy

**Summary:** snappy modules allows configuration of snappy.

**Deprecated:** Use Snap module instead. This module will not exist in cloud-init 18.3.

The below example config config would install etcd, and then install pkg2.smoser with a `<config-file>` argument where config-file has config-blob inside it. If pkgname is installed already, then snappy config pkgname `<file>` will be called where file has pkgname-config-blob as its content.

Entries in config can be namespaced or non-namespaced for a package. In either case, the config provided to snappy command is non-namespaced. The package name is provided as it appears.

If `packages_dir` has files in it that end in .snap, then they are installed. Given 3 files:

- `<packages_dir>/foo.snap`
- `<packages_dir>/foo.config`
- `<packages_dir>/bar.snap`

cloud-init will invoke:

- snappy install `<packages_dir>/foo.snap `<packages_dir>/foo.config
- snappy install `<packages_dir>/bar.snap`

**Note:** that if provided a config entry for ubuntu-core, then cloud-init will invoke: snappy config ubuntu-core `<config>` Allowing you to configure ubuntu-core in this way.

The `ssh_enabled` key controls the system’s ssh service. The default value is `auto`. Options are:

- **True:** enable ssh service
- **False:** disable ssh service
- **auto:** enable ssh service if either ssh keys have been provided or user has requested password authentication (ssh_pwauth).

**Internal name:** cc_snappy

**Module frequency:** per instance

**Supported distros:** ubuntu

**Config keys:**
#cloud-config

snappy:
    system_snappy: auto
    ssh_enabled: auto
    packages: [etcd, pkg2.smser]
    config:
        pkgname:
            key2: value2
        pkg2:
            key1: value1
        packages_dir: '/writable/user-data/cloud-init/snaps'

1.10.45 Snap Config

Summary: snap_config modules allows configuration of snapd.

Deprecated: Use Snap module instead. This module will not exist in cloud-init 18.3.

This module uses the same snappy namespace for configuration but acts only only a subset of the configuration.

If assertions is set and the user has included a list of assertions then cloud-init will collect the assertions into a single assertion file and invoke snap ack <path to file with assertions> which will attempt to load the provided assertions into the snapd assertion database.

If email is set, this value is used to create an authorized user for contacting and installing snaps from the Ubuntu Store. This is done by calling snap create-user command.

If known is set to True, then it is expected the user also included an assertion of type system-user. When snap create-user is called cloud-init will append ‘–known’ flag which instructs snapd to look for a system-user assertion with the details. If known is not set, then snap create-user will contact the Ubuntu SSO for validating and importing a system-user for the instance.

Note: If the system is already managed, then cloud-init will not attempt to create a system-user.

Internal name: cc_snap_config

Module frequency: per instance

Supported distros: any with ‘snapd’ available

Config keys:

#cloud-config
snappy:
    assertions:
    - |
      <assertion 1>
    - |
      <assertion 2>
    email: user@user.org
    known: true

1.10.46 Spacewalk

Summary: install and configure spacewalk
Cloud-Init, Release 19.2

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

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

**Internal name:** cc_spacewalk

**Module frequency:** per instance

**Supported distros:** redhat, fedora

**Config keys:**

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

### 1.10.47 SSH

**Summary:** configure ssh and ssh keys

This module handles most configuration for ssh and ssh keys. Many images have default ssh keys, which can be removed using `ssh_deletekeys`. Since removing default keys is usually the desired behavior this option is enabled by default.

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

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

**Note:** to specify multiline private keys, use yaml multiline syntax

If no keys are specified using `ssh_keys`, then keys will be generated using `ssh-keygen`. By default one public/private pair of each supported key type will be generated. The key types to generate can be specified using the `ssh_genkeytypes` config flag, which accepts a list of key types to use. For each 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 key types for the `ssh_keys` and the `ssh_genkeytypes` config flags are:

- rsa
- dsa
- ecdsa
- ed25519

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

Internal name: cc\_ssh

Module frequency: per instance

Supported distros: all

Config keys:

```yaml
ssh\_deletekeys: <true/false>

ssh\_keys:
  rsa\_private: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIBxwIBAAJhAKD0YSHy73nUgysOl3XsJmd4fHiFyQ+00R7VVu2iV9Qco ...
    -----END RSA PRIVATE KEY-----
  rsa\_public: ssh-rsa AAAAB3NzaC1yc2EAAAAB1wAAAGEAoPRhIfLvedSDKw7Xd ...

  dsa\_private: |
    -----BEGIN DSA PRIVATE KEY-----
    MIIBxwIBAAJhAKD0YSHy73nUgysOl3XsJmd4fHiFyQ+00R7VVu2iV9Qco ...
    -----END DSA PRIVATE KEY-----
  dsa\_public: ssh-dsa AAAAB3NzaC1yc2EAAAAB1wAAAGEAoPRhIfLvedSDKw7Xd ...

ssh\_genkeytypes: <key type>

disable\_root: <true/false>

disable\_root\_opts: <disable root options string>

ssh\_authorized\_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAAB1wAAAGEA3FSyQwB16z+nCSjUU ...
  - ssh-rsa AAAAB3NzaC1yc2EAAAAB1wAAAGEA317VUE215gSn5uavROsc5HRDpZ ...
```

### 1.10.48 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 md5.

**Internal name:** ‘cc\_ssh\_authkey\_fingerprints‘

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

```yaml
no\_ssh\_fingerprints: <true/false>

authkey\_hash: <hash type>
```
1.10.49 SSH Import Id

Summary: import ssh id

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

Internal name: cc_ssh_import_id

Module frequency: per instance

Supported distros: ubuntu, debian

Config keys:

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

1.10.50 Timezone

Summary: set system timezone

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

Internal name: cc_timezone

Module frequency: per instance

Supported distros: all

Config keys:

```
timezone: <timezone>
```

1.10.51 Ubuntu Advantage

Summary: Configure Ubuntu Advantage support services

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

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

Internal name: cc_ubuntu_advantage

Module frequency: once-per-instance

Supported distros: ubuntu

Config schema: `ubuntu_advantage`: (object)
token: (string) A contract token obtained from https://ubuntu.com/advantage.

enable: (array of string)

Examples:

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

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

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

### 1.10.52 Ubuntu Drivers

**Summary:** Interact with third party drivers in Ubuntu.

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

**Internal name:** cc_ubuntu_drivers

**Module frequency:** once-per-instance

**Supported distros:** ubuntu

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

  nvidia: (object)

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

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

**Examples:**

```plaintext
drivers:
  nvidia:
    license-accepted: true
```
1.10.53 Update Etc Hosts

Summary: update /etc/hosts

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

If set to true or template, cloud-init will generate /etc/hosts 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 /etc/hosts entirely, but rather will ensure that a entry for the fqdn with a distribution dependent ip is present in /etc/hosts (i.e. ping <hostname> will ping 127.0.0.1 or 127.0.1.1 or other ip).

Note: if manage_etc_hosts is set true or template, the contents of /etc/hosts will be updated every boot. to make any changes to /etc/hosts 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: per always

Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>manage_etc_hosts:</th>
<th>&lt;true/&quot;template&quot;/false/&quot;localhost&quot;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fqdn:</td>
<td>&lt;fqdn&gt;</td>
</tr>
<tr>
<td>hostname:</td>
<td>&lt;fqdn/hostname&gt;</td>
</tr>
</tbody>
</table>

1.10.54 Update Hostname

Summary: update hostname and fqdn

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

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

Internal name: cc_update_hostname

Module frequency: per always

Supported distros: all

Config keys:

<table>
<thead>
<tr>
<th>preserve_hostname:</th>
<th>&lt;true/false&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fqdn:</td>
<td>&lt;fqdn&gt;</td>
</tr>
<tr>
<td>hostname:</td>
<td>&lt;fqdn/hostname&gt;</td>
</tr>
</tbody>
</table>
1.10.55 Users and Groups

**Summary:** configure users and groups

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

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

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

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

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

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

Internal name: cc_users_groups
Module frequency: per instance
Supported distros: all
Config keys:

<table>
<thead>
<tr>
<th>groups:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- &lt;group&gt;: [&lt;user&gt;, &lt;user&gt;]</td>
</tr>
<tr>
<td>- &lt;group&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>users:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- default</td>
</tr>
<tr>
<td># User explicitly omitted from sudo permission; also default behavior.</td>
</tr>
<tr>
<td>- name: &lt;some_restricted_user&gt;</td>
</tr>
<tr>
<td>sudo: false</td>
</tr>
<tr>
<td>- name: &lt;username&gt;</td>
</tr>
<tr>
<td>expiredate: &lt;date&gt;</td>
</tr>
<tr>
<td>gecos: &lt;comment&gt;</td>
</tr>
<tr>
<td>groups: &lt;additional groups&gt;</td>
</tr>
<tr>
<td>homeimdir: &lt;home directory&gt;</td>
</tr>
<tr>
<td>inactive: &lt;true/false&gt;</td>
</tr>
<tr>
<td>lock_passwd: &lt;true/false&gt;</td>
</tr>
<tr>
<td>no_create_home: &lt;true/false&gt;</td>
</tr>
<tr>
<td>no_log_init: &lt;true/false&gt;</td>
</tr>
<tr>
<td>no_user_group: &lt;true/false&gt;</td>
</tr>
<tr>
<td>passwd: &lt;password&gt;</td>
</tr>
<tr>
<td>primary_group: &lt;primary group&gt;</td>
</tr>
<tr>
<td>selinux_user: &lt;selinux username&gt;</td>
</tr>
<tr>
<td>shell: &lt;shell path&gt;</td>
</tr>
<tr>
<td>snapuser: &lt;email&gt;</td>
</tr>
<tr>
<td>ssh_redirect_user: &lt;true/false&gt;</td>
</tr>
<tr>
<td>ssh Authorized_keys:</td>
</tr>
<tr>
<td>- &lt;key&gt;</td>
</tr>
<tr>
<td>- &lt;key&gt;</td>
</tr>
<tr>
<td>ssh_import_id: &lt;id&gt;</td>
</tr>
<tr>
<td>sudo: &lt;sudo config&gt;</td>
</tr>
<tr>
<td>system: &lt;true/false&gt;</td>
</tr>
<tr>
<td>uid: &lt;user id&gt;</td>
</tr>
</tbody>
</table>

1.10.56 Write Files

Summary: write arbitrary files
Write out arbitrary content to files, optionally setting permissions. 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.
Note: if multiline data is provided, care should be taken to ensure that it follows yaml formatting standards. to specify binary data, use the yaml option `!!binary`

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

**Internal name:** cc_write_files

**Module frequency:** per instance

**Supported distros:** all

**Config keys:**

```yaml
classified_files:
  - encoding: b64
    content: C1MgVGhpcyBmaWxlIGNvbnRyb2xzb2xzdGF0ZW50eBTRUxpbnV4...
    owner: root:root
    path: /etc/sysconfig/selinux
    permissions: '0644'
  - content: !binary |
    # My new /etc/sysconfig/samba file
    path: /etc/sysconfig/samba
    permissions: '0777'
    --
    content: !binary |
    path: /etc/sysconfig/samba
    permissions: '0755'
  - content: |
    15 * * * * root ship_logs
    path: /etc/crontab
    append: true
```

### 1.10.57 Yum Add Repo

**Summary:** add yum repository configuration to the system

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

**Internal name:** cc_yum_add_repo

**Module frequency:** per always

**Supported distros:** fedora, rhel

**Config keys:**

```yaml
classified_files:
  <repo-name>:
```

(continues on next page)
1.11 Merging User-Data Sections

1.11.1 Overview

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

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

For example.

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

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

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

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

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

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

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

1.11.2 Built-in Mergers

Cloud-init provides merging for the following built-in types:
• Dict
• List
• String

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

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

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

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

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

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

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

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

### 1.11.3 Customizability

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

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

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

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

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

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

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

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

### 1.11.4 How to activate

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

1. The first way involves the usage of MIME messages in cloud-init to specify multipart documents (this is one way in which multiple cloud-config is joined together into a single cloud-config). Two new headers are looked for, both of which can define the way merging is done (the first header to exist wins). These new headers (in lookup order) are ‘Merge-Type’ and ‘X-Merge-Type’. The value should be a string which will satisfy the new merging format definition (see below for this format).

2. The second way is actually specifying the merge-type in the body of the cloud-config dictionary. There are 2 ways to specify this, either as a string or as a dictionary (see format below). The keys that are looked up for this definition are the following (in order), ‘merge_how’, ‘merge_type’.

#### String format

The string format that is expected is the following.

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

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

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

```
list()+dict()+str()
```
Dictionary format

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

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

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

1.11.5 Specifying multiple types and its effect

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

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

1.11.6 Other uses

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

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

1.11.7 Example cloud-config

A common request is to include multiple `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

```yaml
#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

## 1.12 Network Configuration

- Default Behavior
- Disabling Network Configuration
- Fallback Networking
- Network Configuration Sources
- Network Configuration Outputs
- Network Output Policy
- Network Configuration Tools
- Examples

### 1.12.1 Default Behavior

Cloud-init’s searches for network configuration in order of increasing precedence; each item overriding the previous.

**Datasource**

For example, OpenStack may provide network config in the MetaData Service.

**System Config**

A `network:` entry in /etc/cloud/cloud.cfg.d/* configuration files.

**Kernel Command Line**

`ip=or network-config=<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.

### 1.12.2 Disabling Network Configuration

Users may disable Cloud-init’s network configuration capability and rely on other methods, such as embedded configuration or other customizations.

Cloud-init supports the following methods for disabling cloud-init.

**Kernel Command Line**
Cloud-init will check for a parameter `network-config` and the value is expected to be YAML string in the *Networking Config Version 1* format. The YAML string may optionally be Base64 encoded, and optionally compressed with gzip.

Example disabling kernel command line entry:

```
network-config={config: disabled}
```

```
ccloud config

In the combined cloud-init configuration dictionary.
```

```
  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.

### 1.12.3 Fallback Network Configuration

Cloud-init will attempt to determine which of any attached network devices is most likely to have a connection and then generate a network configuration to issue a DHCP request on that interface.

Cloud-init runs during early boot and does not expect composed network devices (such as Bridges) to be available. Cloud-init does not consider the following interface devices as likely ‘first’ network interfaces for fallback configuration; they are filtered out from being selected.

- *loopback*: name=lo
- *Virtual Ethernet*: name=veth*
- *Software Bridges*: type=bridge
- *Software VLANs*: type=_vlan

Cloud-init will prefer network interfaces that indicate they are connected via the Linux carrier flag being set. If no interfaces are marked connected, then all unfiltered interfaces are potential connections.

Of the potential interfaces, Cloud-init will attempt to pick the “right” interface given the information it has available.

Finally after selecting the “right” interface, a configuration is generated and applied to the system.

### 1.12.4 Network Configuration Sources

Cloud-init accepts a number of different network configuration formats in support of different cloud substrates. The Datasource for these clouds in Cloud-init will detect and consume Datasource-specific network configuration formats for use when writing an instance’s network configuration.

The following Datasources optionally provide network configuration:

- *Config Drive*
  - OpenStack Metadata Service Network
  - *Network Configuration ENI (Legacy)*
- *Digital Ocean*
  - DigitalOcean JSON metadata
- *NoCloud*
Cloud-init supports reading and writing network config in the ENI format which is consumed by the ifupdown tool to parse and apply network configuration.

As an input format this is legacy. In cases where ENI format is available and another format is also available, it will prefer to use the other format. This can happen in either NoCloud or OpenStack datasources.

Please reference existing documentation for the /etc/network/interfaces(5) format.

Network Configuration Version 1

This network configuration format lets users customize their instance’s networking interfaces by assigning subnet configuration, virtual device creation (bonds, bridges, vlans) routes and DNS configuration.

Required elements of a Network Config Version 1 are config and version.

Cloud-init will read this format from system config. For example the following could be present in /etc/cloud/cloud.cfg.d/custom-networking.cfg:

```yaml
network:
  version: 1
  config:
    - type: physical
      name: eth0
      subnets:
        - type: dhcp
```

The NoCloud datasource can also provide cloud-init networking configuration in this Format.

Configuration Types

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

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

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

• Subnet/IP (subnets)

Physical

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

name: <desired device name>

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

mac_address: <MAC Address>

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

Note: Cloud-init will handle the persistent mapping between a device’s name and the mac_address.

mtu: <MTU SizeBytes>

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

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

Physical Example:

```
network:
  version: 1
  config:
    # Simple network adapter
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
    # Second nic with Jumbo frames
    - type: physical
      name: jumbo0
      mac_address: aa:11:22:33:44:55
      mtu: 9000
    # 10G pair
    - type: physical
      name: gbe0
      mac_address: cd:11:22:33:44:00
```
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 devices name must be less than 15 characters. Names exceeding the maximum will be truncated. This is a limitation of the Linux kernel network-device structure.

**mac_address**: `<MAC Address>`

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

**bond_interfaces**: `<List of network device names>`

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

**mtu**: `<MTU SizeBytes>`

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

**Note**: The possible supported values of a device’s MTU is not available at configuration time. It’s possible to specify a value too large or 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:**

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

(continues on next page)
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:

```yaml
network:
  version: 1
config:
  # Simple network adapter
  - type: physical
    name: interface0
    mac_address: 00:11:22:33:44:55
  # Second nic with Jumbo frames
  - type: physical
    name: jumbo0
    mac_address: aa:11:22:33:44:55
    mtu: 9000
  - type: bridge
    name: br0
    bridge_interfaces:
    - jumbo0
    params:
      bridge_ageing: 250
```

(continues on next page)
**VLAN**

Type **vlan** requires the following keys:

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

The following optional keys are supported:

**mtu**: `<MTU SizeBytes>`

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

**Note**: The possible supported values of a device’s MTU is not available at configuration time. It’s possible to specify a value too large or 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.

**Nameserver Example:**

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

**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:**

```yaml
network:
  version: 1
  config:
    - type: physical
      name: interface0
      mac_address: 00:11:22:33:44:55
    subnets:
      - type: static
        address: 192.168.23.14/24
        gateway: 192.168.23.1
      - type: route
        destination: 192.168.24.0/24
        gateway: 192.168.24.1
        metric: 3
```

**Subnet/IP**

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

Valid keys for **subnets** include the following:

- **type**: Specify the subnet type.
• **control**: Specify manual, auto or hotplug. Indicates how the interface will be handled during boot.

• **address**: IPv4 or IPv6 address. It may include CIDR netmask notation.

• **netmask**: IPv4 subnet mask in dotted format or CIDR notation.

• **gateway**: IPv4 address of the default gateway for this subnet.

• **dns_nameservers**: Specify a list of IPv4 dns server IPs to end up in resolv.conf.

• **dns_search**: Specify a list of search paths to be included in resolv.conf.

• **routes**: Specify a list of routes for a given interface

Subnet types are one of the following:

• **dhcp4**: Configure this interface with IPv4 dhcp.

• **dhcp**: Alias for dhcp4

• **dhcp6**: Configure this interface with IPv6 dhcp.

• **static**: Configure this interface with a static IPv4.

• **static6**: Configure this interface with a static IPv6.

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

**Subnet DHCP Example**:

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

**Subnet Static Example**:

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

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

**Multiple subnet Example**:

```yaml
network:
  version: 1
```

(continues on next page)
Subnet with routes Example:

```yaml
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
```

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:

```yaml
network:
  version: 1
  config:
    # 10G pair
    - type: physical
      name: gbe0
      mac_address: cd:11:22:33:44:00
    - type: physical
      name: gbe1
      mac_address: cd:11:22:33:44:02
    # Bond.
    - type: bond
```
name: bond0
bond_interfaces:
- gbe0
- gbe1
params:
  bond-mode: 802.3ad
  bond-lacp-rate: fast
# A Bond VLAN.
  - type: vlan
    name: bond0.200
    vlan_link: bond0
    vlan_id: 200
    subnets:
      - type: dhcp4

More Examples

Some more examples to explore the various options available.

Multiple VLAN example:

    network:
      version: 1
      config:
        - id: eth0
          mac_address: d4:be:d9:a8:49:13
          mtu: 1500
          name: eth0
          subnets:
            - address: 10.245.168.16/21
              dns_nameservers: []
              gateway: 10.245.168.1
              type: static
              type: physical
            - id: eth1
              mac_address: d4:be:d9:a8:49:15
              mtu: 1500
              name: eth1
              subnets:
                - address: 10.245.188.2/24
                  dns_nameservers: []
                  type: static
                  type: physical
                - id: eth1.2667
                  mtu: 1500
                  name: eth1.2667
                  subnets:
                    - address: 10.245.184.2/24
                      dns_nameservers: []
                      type: static
                      type: vlan
                      vlan_id: 2667
                      vlan_link: eth1
                    - id: eth1.2668
Networking Config Version 2

Cloud-init’s support for Version 2 network config is a subset of the version 2 format defined for the netplan tool. Cloud-init supports both reading and writing of Version 2; the latter support requires a distro with netplan present.

The network key has at least two required elements. First it must include version: 2 and one or more of possible device types..

Cloud-init will read this format from system config. For example the following could be present in /etc/cloud/cloud.cfg.d/custom-networking.cfg:

```yaml
network:
  version: 2
  ethernets: []
```

It may also be provided in other locations including the NoCloud, see Default Behavior for other places.

Supported device types values are as follows:

- Ethernets (ethernets)
- Bonds (bonds)
- Bridges (bridges)
Cloud-Init, Release 19.2

- VLANs (vlans)

Each type block contains device definitions as a map where the keys (called “configuration IDs”). Each entry under the types may include IP and/or device configuration.

Cloud-init does not currently support wifis type that is present in native netplan.

**Device configuration IDs**

The key names below the per-device-type definition maps (like ethernets:) are called “ID’s. They must be unique throughout the entire set of configuration files. Their primary purpose is to serve as anchor names for composite devices, for example to enumerate the members of a bridge that is currently being defined.

There are two physically/structurally different classes of device definitions, and the ID field has a different interpretation for each:

**Physical devices**

- (Examples: ethernet, wifi) These can dynamically come and go between reboots and even during runtime (hot-plugging). In the generic case, they can be selected by match: rules on desired properties, such as name/name pattern, MAC address, driver, or device paths. In general these will match any number of devices (unless they refer to properties which are unique such as the full path or MAC address), so without further knowledge about the hardware these will always be considered as a group.

  It is valid to specify no match rules at all, in which case the ID field is simply the interface name to be matched. This is mostly useful if you want to keep simple cases simple, and it’s how network device configuration has been done for a long time.

  If there are match: rules, then the ID field is a purely opaque name which is only being used for references from definitions of compound devices in the config.

**Virtual devices**

- (Examples: veth, bridge, bond) These are fully under the control of the config file(s) and the network stack. I.e. these devices are being created instead of matched. Thus match: and set-name: are not applicable for these, and the ID field is the name of the created virtual device.

**Common properties for physical device types**

**match: <(mapping)>**

This selects a subset of available physical devices by various hardware properties. The following configuration will then apply to all matching devices, as soon as they appear. *All* specified properties must match. The following properties for creating matches are supported:

**name: <(scalar)>**

Current interface name. Globs are supported, and the primary use case for matching on names, as selecting one fixed name can be more easily achieved with having no match: at all and just using the ID (see above). Note that currently only networkd supports globbing, NetworkManager does not.

**macaddress: <(scalar)>**

Device’s MAC address in the form “XX:XX:XX:XX:XX”. Globs are not allowed.

**driver: <(scalar)>**

Kernel driver name, corresponding to the DRIVER udev property. Globs are supported. Matching on driver is only supported with networkd.
Examples:

```plaintext
# all cards on second PCI bus
match:
  name: enp2*

# fixed MAC address
match:

# first card of driver `ixgbe`
match:
  driver: ixgbe
  name: en*s0

set-name: <(scalar)>

When matching on unique properties such as path or MAC, or with additional assumptions such as “there will only ever be one wifi device”, match rules can be written so that they only match one device. Then this property can be used to give that device a more specific/desirable/nicer name than the default from udev’s ifnames. Any additional device that satisfies the match rules will then fail to get renamed and keep the original kernel name (and dmesg will show an error).

wakeonlan: <(bool)>

Enable wake on LAN. Off by default.

Common properties for all device types

renderer: <(scalar)>

Use the given networking backend for this definition. Currently supported are networkd and NetworkManager. This property can be specified globally in networks:, for a device type (in e.g. ethernets:) or for a particular device definition. Default is networkd.

Note: Cloud-init only supports networkd backend if rendering version2 config to the instance.

dhcp4: <(bool)>

Enable DHCP for IPv4. Off by default.

dhcp6: <(bool)>

Enable DHCP for IPv6. Off by default.

addresses: <(sequence of scalars)>

Add static addresses to the interface in addition to the ones received through DHCP or RA. Each sequence entry is in CIDR notation, i.e. of the form addr/prefixlen. addr is an IPv4 or IPv6 address as recognized by inet_pton``(3) and ``prefixlen the number of bits of the subnet.

Example: addresses: [192.168.14.2/24, 2001:1::1/64]

gateway4 or gateway6: <(scalar)>

Set default gateway for IPv4/6, for manual address configuration. This requires setting addresses too. Gateway IPs must be in a form recognized by inet_pton(3)

Example for IPv4: gateway4: 172.16.0.1 Example for IPv6: gateway6: 2001:4::1
**mtu**: `<MTU SizeBytes>`

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

**nameservers**: `<(mapping)>`

Set DNS servers and search domains, for manual address configuration. There are two supported fields: `addresses:` is a list of IPv4 or IPv6 addresses similar to `gateway*`, and `search:` is a list of search domains.

Example:

```yaml
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:

```yaml
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:

```yaml
ethernets:
    switchports:
        match: {name: "enp2*"}
    bonds:
        bond0:
            interfaces: [switchports]
```

**parameters**: `<(mapping)>`

Customization parameters for special bonding options. Time values are specified in seconds unless otherwise specified.

**mode**: `<(scalar)>`

Set the bonding mode used for the interfaces. The default is `balance-rr` (round robin). Possible values are `balance-rr`, `active-backup`, `balance-xor`, `broadcast`, `802.3ad`, `balance-tlb`, and `balance-alb`.

**lacp-rate**: `<(scalar)>`
Set the rate at which 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 behavior in most situations.

**arp-interval:** `<(scalar)>`

Set the interval value for how frequently ARP link monitoring should happen. The default value is 0, which disables ARP monitoring.

**arp-ip-targets:** `<(sequence of scalars)>`

IPs of other hosts on the link which should be sent ARP requests in order to validate that a slave is up. This option is only used when arp-interval is set to a value other than 0. At least one IP address must be given for ARP link monitoring to function. Only IPv4 addresses are supported. You can specify up to 16 IP addresses. The default value is an empty list.

**arp-validate:** `<(scalar)>`

Configure how ARP replies are to be validated when using ARP link monitoring. Possible values are none, active, backup, and all.

**arp-all-targets:** `<(scalar)>`

Specify whether to use any ARP IP target being up as sufficient for a slave to be considered up; or if all the targets must be up. This is only used for active-backup mode when arp-validate is enabled. Possible values are any and all.

**up-delay:** `<(scalar)>`

Specify the delay before enabling a link once the link is physically up. The default value is 0.

**down-delay:** `<(scalar)>`

Specify the delay before disabling a link once the link has been lost. The default value is 0.

**fail-over-mac-policy:** `<(scalar)>`

Set whether to set all slaves to the same MAC address when adding them to the bond, or how else the system should handle MAC addresses. The possible values are none, active, and follow.

**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:

```yaml
ethernets:
  switchports:
    match: {name: "enp2*"}

bridges:
  br0:
    interfaces: [switchports]
```

**parameters**: `<(mapping)>`

Customization parameters for special bridging options. Time values are specified in seconds unless otherwise specified.

**ageing-time**: `<(scalar)>`

Set the period of time to keep a MAC address in the forwarding database after a packet is received.

**priority**: `<(scalar)>`

Set the priority value for the bridge. This value should be an number between 0 and 65535. Lower values mean higher priority. The bridge with the higher priority will be elected as the root bridge.

**forward-delay**: `<(scalar)>`

Specify the period of time the bridge will remain in Listening and Learning states before getting to the Forwarding state. This value should be set in seconds for the systemd backend, and in milliseconds for the NetworkManager backend.

**hello-time**: `<(scalar)>`

Specify the interval between two hello packets being sent out from the root and designated bridges. Hello packets communicate information about the network topology.

**max-age**: `<(scalar)>`
Set the maximum age of a hello packet. If the last hello packet is older than that value, the bridge will attempt to become the root bridge.

**path-cost**: `<(scalar)>`

Set the cost of a path on the bridge. Faster interfaces should have a lower cost. This allows a finer control on the network topology so that the fastest paths are available whenever possible.

**stp**: `<(bool)>`

Define whether the bridge should use Spanning Tree Protocol. The default value is “true”, which means that Spanning Tree should be used.

### VLANs

**id**: `<(scalar)>`

VLAN ID, a number between 0 and 4094.

**link**: `<(scalar)>`

ID of the underlying device definition on which this VLAN gets created.

Example:

```
ethernets:
  enol: {...}
vlans:
  en-intra:
    id: 1
    link: enol
    dhcp4: yes
  en-vpn:
    id: 2
    link: enol
    address: ...
```

### Examples

Configure an ethernet device with networkd, identified by its name, and enable DHCP:

```
network:
  version: 2
  ethernets:
    enol:
      dhcp4: true
```

This is a complex example which shows most available features:

```
network:
  version: 2
  ethernets:
    id0:
      match:
        macaddress: 00:11:22:33:44:55
      wakeonlan: true
```

(continues on next page)
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]
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
  # static routes
routes:
  - to: 0.0.0.0/0
    via: 11.0.0.1
    metric: 3

1.12.5 Network Configuration Outputs

Cloud-init converts various forms of user supplied or automatically generated configuration into an internal network configuration state. From this state Cloud-init delegates rendering of the configuration to Distro supported formats. The following renderers are supported in cloud-init:

- ENI

/etc/network/interfaces or ENI is supported by the `ifupdown` package found in Ubuntu and Debian.

- Netplan

Introduced in Ubuntu 16.10 (Yakkety Yak), netplan has been the default network configuration tool in Ubuntu since 17.10 (Artful Aardvark). netplan consumes `Networking Config Version 2` input and renders network configuration for
supported backends such as `systemd-networkd` and `NetworkManager`.

- **Sysconfig**

  Sysconfig format is used by RHEL, CentOS, Fedora and other derivatives.

### 1.12.6 Network Output Policy

The default policy for selecting a network renderer in order of preference is as follows:

- ENI
- Sysconfig
- Netplan

When applying the policy, Cloud-init checks if the current instance has the correct binaries and paths to support the renderer. The first renderer that can be used is selected. Users may override the network renderer policy by supplying an updated configuration in cloud-config.

```
system_info:
    network:
        renderers: ['netplan', 'eni', 'sysconfig']
```

### 1.12.7 Network Configuration Tools

Cloud-init contains one tool used to test input/output conversion between formats. The `tools/net-convert.py` in the Cloud-init source repository is helpful for examining expected output for a given input format.

**CLI Interface**:

```
$ tools/net-convert.py --help
usage: net-convert.py [-h] --network-data PATH --kind
    {eni,network_data.json,yaml} -d PATH [-m name,mac]
    --output-kind {eni,netplan,sysconfig}
optional arguments:
  -h, --help show this help message and exit
  --network-data PATH, -p PATH
  --kind {eni,network_data.json,yaml}, -k {eni,network_data.json,yaml}
  -d PATH, --directory PATH
directory to place output in
  -m name,mac, --mac name,mac
  interface name to mac mapping
  --output-kind {eni,netplan,sysconfig}, -ok {eni,netplan,sysconfig}
```

Example output converting V2 to sysconfig:

```
$ tools/net-convert.py --network-data v2.yaml --kind yaml
  --output-kind sysconfig -d target
$ cat target/etc/sysconfig/network-scripts/ifcfg-eth*
# Created by cloud-init on instance boot automatically, do not edit.
#
BOOTPROTO=static
DEVICE=eth7
IPADDR=192.168.1.5/255.255.255.0
NM_CONTROLLED=no
```

(continues on next page)
1.13 Vendor Data

1.13.1 Overview

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

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 multipart 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 multipart input with a cloud-config-jsonp part like:

```
#cloud-config-jsonp
[ { "op": "add", "path": "/runcmd", "value": ["my", "command", "here"] } ]
```

Further, we strongly advise vendors to not ‘be evil’. By evil, we mean any action that could compromise a system. Since users trust you, please take care to make sure that any vendor data is safe, atomic, idempotent and does not put your users at risk.

1.13.2 Input Formats

cloud-init will download and cache to filesystem any vendor-data that it finds. Vendor data is handled exactly like user-data. That means that the vendor can supply multipart input and have those parts acted on in the same way as user-data.

The only differences are:

- user-scripts are stored in a different location than user-scripts (to avoid namespace collision)
- user can disable part handlers by cloud-config settings. For example, to disable handling of ‘part-handlers’ in vendor-data, the user could provide user-data like this:
1.13.3 Examples

There are examples in the examples subdirectory.
Additionally, the ‘tools’ directory contains ‘write-mime-multipart’, which can be used to easily generate mime-multi-part files from a list of input files. That data can then be given to an instance.
See ‘write-mime-multipart –help’ for usage.

1.14 Testing and debugging cloud-init

1.14.1 Overview

This topic will discuss general approaches for test and debug of cloud-init on deployed instances.

1.14.2 Boot Time Analysis - cloud-init analyze

Occasionally instances don’t appear as performant as we would like and cloud-init packages a simple facility to inspect what operations took cloud-init the longest during boot and setup.

The script /usr/bin/cloud-init has an analyze sub-command analyze which parses any cloud-init.log file into formatted and sorted events. It allows for detailed analysis of the most costly cloud-init operations to determine the long-pole in cloud-init configuration and setup. These subcommands default to reading /var/log/cloud-init.log.

- analyze show Parse and organize cloud-init.log events by stage and include each sub-stage granularity with time delta reports.

```
$ cloud-init analyze show -i my-cloud-init.log
-- Boot Record 01 --
The total time elapsed since completing an event is printed after the "@" character.
The time the event takes is printed after the "+" character.
Starting stage: modules-config
  |`->config-emit_upstart ran successfully @05.47600s +00.00100s
  |`->config-snap_config ran successfully @05.47700s +00.00100s
  |`->config-ssh-import-id ran successfully @05.47800s +00.00200s
  |`->config-locale ran successfully @05.48000s +00.00100s
  ...
```

- analyze dump Parse cloud-init.log into event records and return a list of dictionaries that can be consumed for other reporting needs.

```
$ cloud-init analyze dump -i my-cloud-init.log
[
  {
    "description": "running config modules",
    "event_type": "start",
    "name": "modules-config",
```
- **analyze blame** Parse cloud-init.log into event records and sort them based on highest time cost for quick assessment of areas of cloud-init that may need improvement.

```bash
$ cloud-init analyze blame -i my-cloud-init.log
```

-- Boot Record 11 --

- 00.01300s (modules-final/config-scripts-per-boot)
- 00.00400s (modules-final/config-final-message)
- 00.00100s (modules-final/config-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.

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

-- Most Recent Boot Record --

- Kernel ended boot at: 2019-06-13 16:00:00.944740
- Kernel time to boot (seconds): 5.135355
- Cloud-init start: 2019-06-13 16:00:05.738396
- Time between Kernel boot and Cloud-init start (seconds): 4.793656

**Analyze quickstart - LXC**

To quickly obtain a cloud-init log try using lxc on any ubuntu system:

```bash
$ lxc init ubuntu-daily:xenial 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 - | \ cl
$ lxc file pull x1/var/log/cloud-init.log - | \ python3 -m cloudinit.analyze dump -i -
```

**Analyze quickstart - KVM**

To quickly analyze a KVM a cloud-init log:

1. Download the current cloud image

```bash
$ wget https://cloud-images.ubuntu.com/daily/server/xenial/current/xenial-server-
    → cloudimg-amd64.img
```

2. Create a snapshot image to preserve the original cloud-image

```bash
$ qemu-img create -b xenial-server-cloudimg-amd64.img -f qcow2 \ test-cloudinit.qcow2
```

3. Create a seed image with metadata using `cloud-localsd`
4. Launch your modified VM

```bash
$ 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)

```bash
$ ssh -p 2222 ubuntu@localhost 'cat /var/log/cloud-init.log' | \ 
  cloud-init analyze blame -i -
```

### 1.14.3 Running single cloud config modules

This subcommand is not called by the init system. It can be called manually to load the configured datasource and run a single cloud-config module once using the cached userdata and metadata after the instance has booted. Each cloud-config module has a module FREQUENCY configured: PER_INSTANCE, PER_BOOT, PER_ONCE or PER_ALWAYS. When a module is run by cloud-init, it stores a semaphore file in `/var/lib/cloud/instance/sem/config_<module_name>_<frequency>` which marks when the module last successfully ran. Presence of this semaphore file prevents a module from running again if it has already been run. To ensure that a module is run again, the desired frequency can be overridden on the commandline:

```bash
$ sudo cloud-init single --name cc_ssh --frequency always 
... 
Generating public/private ed25519 key pair 
... 
```

Inspect cloud-init.log for output of what operations were performed as a result.

### 1.15 More information

#### 1.15.1 Useful external references

- The beauty of cloudinit
- Introduction to cloud-init (video)

### 1.16 Hacking on cloud-init

This document describes how to contribute changes to cloud-init. It assumes you have a Launchpad account, and refers to your launchpad user as LP_USER throughout.
1.16.1 Do these things once

- To contribute, 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-agreementcanonical group. Unfortunately there is no easy way to check if an organization or company you are doing work for has signed. If you are unsure or have questions, email Josh Powers or ping powersj in #cloud-init channel via freenode.

When prompted for ‘Project contact’ or ‘Canonical Project Manager’ enter ‘Josh Powers’.

- 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:

```bash
git config user.name "Your Name"
git config user.email "Your Email"
```

- Clone the upstream repository on Launchpad:

```bash
git clone https://git.launchpad.net/cloud-init
cd cloud-init
```

There is more information on Launchpad as a git hosting site in Launchpad git documentation.

- Create a new remote pointing to your personal Launchpad repository. This is equivalent to ‘fork’ on github.

```bash
git remote add LP_USER ssh://LP_USER@git.launchpad.net/~LP_USER/cloud-init
git push LP_USER master
```

1.16.2 Do these things for each feature or bug

- Create a new topic branch for your work:

```bash
git checkout -b my-topic-branch
```

- Make and commit your changes (note, you can make multiple commits, fixes, more commits.):

```bash
git commit
```

- Run unit tests and lint/formatting checks with tox:

```bash
tox
```

- Push your changes to your personal Launchpad repository:

```bash
git push -u LP_USER my-topic-branch
```

- Use your browser to create a merge request:
  - Open the branch on Launchpad.
  - You can see a web view of your repository and navigate to the branch at:
    `https://code.launchpad.net/~LP_USER/cloud-init/`
Cloud-Init, Release 19.2

* It will typically be at:
  
  https://code.launchpad.net/~LP_USER/cloud-init/+git/cloud-init/
  +ref/BRANCHNAME

  for example, here is larsks move-to-git branch: https://code.launchpad.net/~larsks/cloud-init/+git/
  cloud-init/+ref/feature/move-to-git

  – Click ‘Propose for merging’
  – Select ‘lp:cloud-init’ as the target repository
  – Type ‘master’ as the Target reference path
  – Click ‘Propose Merge’
  – On the next page, hit ‘Set commit message’ and type a git combined git style commit message like:

    Activate the frobnicator.

    The frobnicator was previously inactive and now runs by default. This may save the world some day. Then, list the bugs you fixed as footers with syntax as shown here.

    The commit message should be one summary line of less than 74 characters followed by a blank line, and then one or more paragraphs describing the change and why it was needed.

    This is the message that will be used on the commit when it is squshed and merged into trunk.

    LP: #1

  Then, someone in the cloud-init-dev group will review your changes and follow up in the merge request.

  Feel free to ping and/or join #cloud-init on freenode irc if you have any questions.

1.17 Integration Testing

1.17.1 Overview

This page describes the execution, development, and architecture of the cloud-init integration tests:

  • Execution explains the options available and running of tests
  • Development shows how to write test cases
  • Architecture explains the internal processes

1.17.2 Execution

Overview

In order to avoid the need for dependencies and ease the setup and configuration users can run the integration tests via tox:
Everything after the double dash will be passed to the integration tests. Executing tests has several options:

- **run** an alias to run both **collect** and **verify**. The `tree_run` command does the same thing, except uses a deb built from the current working tree.
- **collect** deploys on the specified platform and distro, patches with the requested deb or rpm, and finally collects output of the arbitrary commands. Similarly, `tree_collect` will collect output using a deb built from the current working tree.
- **verify** given a directory of test data, run the Python unit tests on it to generate results.
- **bddeb** will build a deb of the current working tree.

### Run

The first example will provide a complete end-to-end run of data collection and verification. There are additional examples below explaining how to run one or the other independently.

```
$ git clone https://git.launchpad.net/cloud-init
$ cd cloud-init
$ tox -e citest -- -h
```

The above command will do the following:

- **run** both collect output and run tests the output
- **--verbose** verbose output
- **--os-name** `stretch` on the Debian Stretch release
- **--os-name** `xenial` on the Ubuntu Xenial release
- **--deb** `cloud-init_0.7.8-my_patch_all.deb` use this deb as the version of cloud-init to run with
- **--preserve-data** always preserve collected data, do not remove data after successful test run
- **--preserve-instance** do not destroy the instance after test to allow for debugging the stopped instance during integration test development. By default, test instances are destroyed after the test completes.
- **--data-dir** `~/collection` write collected data into `~/collection`, rather than using a temporary directory

For a more detailed explanation of each option see below.

**Note:** By default, data collected by the run command will be written into a temporary directory and deleted after a successful. If you would like to preserve this data, please use the option **--preserve-data**.

### Collect

If developing tests it may be necessary to see if cloud-config works as expected and the correct files are pulled down. In this case only a collect can be run by running:
$ tox -e citest -- collect -n xenial --data-dir /tmp/collection

The above command will run the collection tests on xenial and place all results into /tmp/collection.

**Verify**

When developing tests it is much easier to simply rerun the verify scripts without the more lengthy collect process. This can be done by running:

$ tox -e citest -- verify --data-dir /tmp/collection

The above command will run the verify scripts on the data discovered in /tmp/collection.

**TreeRun and TreeCollect**

If working on a cloud-init feature or resolving a bug, it may be useful to run the current copy of cloud-init in the integration testing environment. The integration testing suite can automatically build a deb based on the current working tree of cloud-init and run the test suite using this deb.

The tree_run and tree_collect commands take the same arguments as the run and collect commands. These commands will build a deb and write it into a temporary file, then start the test suite and pass that deb in. To build a deb only, and not run the test suite, the bddeb command can be used.

Note that code in the cloud-init working tree that has not been committed when the cloud-init deb is built will still be included. To build a cloud-init deb from or use the tree_run command using a copy of cloud-init located in a different directory, use the option --cloud-init /path/to/cloud-init.

$ tox -e citest -- tree_run --verbose \
--os-name xenial --os-name stretch \
--test modules/final_message --test modules/write_files \
--result /tmp/result.yaml

**Bddeb**

The bddeb command can be used to generate a deb file. This is used by the tree_run and tree_collect commands to build a deb of the current working tree. It can also be used a user to generate a deb for use in other situations and avoid needing to have all the build and test dependencies installed locally.

- --bddeb-args: arguments to pass through to bddeb
- --build-os: distribution to use as build system (default is xenial)
- --build-platform: platform to use for build system (default is lxd)
- --cloud-init: path to base of cloud-init tree (default is ‘.’)
- --deb: path to write output deb to (default is ‘.’)

**Setup Image**

By default an image that is used will remain unmodified, but certain scenarios may require image modification. For example, many images may use a much older cloud-init. As a result tests looking at newer functionality will fail because a newer version of cloud-init may be required. The following options can be used for further customization:

- --deb: install the specified deb into the image
Cloud-Init, Release 19.2

- **--rpm**: install the specified rpm into the image
- **--repo**: enable a repository and upgrade cloud-init afterwards
- **--ppa**: enable a ppa and upgrade cloud-init afterwards
- **--upgrade**: upgrade cloud-init from repos
- **--upgrade-full**: run a full system upgrade
- **--script**: execute a script in the image. This can perform any setup required that is not covered by the other options

### 1.17.3 Test Case Development

#### Overview

As a test writer you need to develop a test configuration and a verification file:

- The test configuration specifies a specific cloud-config to be used by cloud-init and a list of arbitrary commands to capture the output of (e.g. my_test.yaml)
- The verification file runs tests on the collected output to determine the result of the test (e.g. my_test.py)

The names must match, however the extensions will of course be different, yaml vs py.

#### Configuration

The test configuration is a YAML file such as *ntp_server.yaml* below:

```yaml
# Empty NTP config to setup using defaults
# NOTE: this should not require apt feature, use 'which' rather than 'dpkg -l'
# NOTE: this should not require no_ntpdate feature, use 'which' to check for
# installation rather than 'dpkg -l', as 'grep ntp' matches 'ntpd'
# NOTE: the verifier should check for any ntp server not 'ubuntu.pool.ntp.org'

cloud_config: |
  #cloud-config
  ntp:
  servers:
    - pool.ntp.org

required_features:
- apt
- no_ntpdate
- ubuntu_ntp

collect_scripts:
  ntp_installed_servers: |
    #!/bin/bash
dpkg -l | grep ntp | wc -l
  ntp_conf_dist_servers: |
    #!/bin/bash
  ls /etc/ntp.conf.dist | wc -l
  ntp_conf_servers: |
    #!/bin/bash
cat /etc/ntp.conf | grep '^server'
```

There are several keys, 1 required and some optional, in the YAML file:

1.17. Integration Testing
1. The required key is `cloud_config`. This should be a string of valid YAML that is exactly what would normally be placed in a cloud-config file, including the cloud-config header. This essentially sets up the scenario under test.

2. One optional key is `collect_scripts`. This key has one or more sub-keys containing strings of arbitrary commands to execute (e.g. `cat /var/log/cloud-config-output.log`). In the example above the output of `dpkg` is captured, grep for `ntp`, and the number of lines reported. The name of the sub-key is important. The sub-key is used by the verification script to recall the output of the commands ran.

3. The optional `enabled` key enables or disables the test case. By default the test case will be enabled.

4. The optional `required_features` key may be used to specify a list of features flags that an image must have to be able to run the test case. For example, if a test case relies on an image supporting `apt`, then the config for the test case should include `required_features: [ apt ]`.

Default Collect Scripts

By default the following files will be collected for every test. There is no need to specify these items:

- `/var/log/cloud-init.log`
- `/var/log/cloud-init-output.log`
- `/run/cloud-init/.instance-id`
- `/run/cloud-init/result.json`
- `/run/cloud-init/status.json`
- `dpkg-query -W -f='${Version}' cloud-init`

Verification

The verification script is a Python file with unit tests like the one, `ntp_server.py`, below:

```python
# This file is part of cloud-init. See LICENSE file for license information.

""
"""cloud-init Integration Test Verify Script"
""
from tests.cloud_tests.testcases import base

class TestNtp(base.CloudTestCase):
    """Test ntp module""
    def test_ntp_installed(self):
        """Test ntp installed""
        out = self.get_data_file('ntp_installed_empty')
        self.assertEqual(1, int(out))

    def test_ntp_dist_entries(self):
        """Test dist config file has one entry""
        out = self.get_data_file('ntp_conf_dist_empty')
        self.assertEqual(1, int(out))

    def test_ntp_entires(self):
        """Test config entries""
        out = self.get_data_file('ntp_conf_empty')
        self.assertIn('pool 0.ubuntu.pool.ntp.org iburst', out)
```

(continues on next page)
Here is a breakdown of the unit test file:

- The import statement allows access to the output files.
- The class can be named anything, but must import the `base.CloudTestCase`, either directly or via another test class.
- There can be 1 to N number of functions with any name, however only functions starting with `test_` will be executed.
- There can be 1 to N number of classes in a test module, however only classes inheriting from `base.CloudTestCase` will be loaded.
- Output from the commands can be accessed via `self.get_data_file('key')` where key is the sub-key of `collect_scripts` above.
- The cloud config that the test ran with can be accessed via `self.cloud_config`, or any entry from the cloud config can be accessed via `self.get_config_entry('key')`.
- See the base `CloudTestCase` for additional helper functions.

**Layout**

Integration tests are located under the `tests/cloud_tests` directory. Test configurations are placed under `configs` and the test verification scripts under `testcases`:

```
$ cloud-init$ tree -d tests/cloud_tests/
tests/cloud_tests/
  configs
  |__ bugs
  |    |__ examples
  |    |__ main
  |    |__ modules
  testcases
  |__ bugs
  |__ examples
  |__ main
  |__ modules
```

The sub-folders of bugs, examples, main, and modules help organize the tests. View the README.md in each to understand in more detail each directory.

**Test Creation Helper**

The integration testing suite has a built in helper to aid in test development. Help can be invoked via `tox -e citest -- create --help`. It can create a template test case config file with user data passed in from the command line, as well as a template test case verifier module.

The following would create a test case named `example` under the `modules` category with the given description, and cloud config data read in from `/tmp/user_data`. 

```bash
# vi: ts=4 expandtab
```
Development Checklist

- **Configuration File**
  - Named `your_test.yaml`
  - Contains at least a valid cloud-config
  - Optionally, commands to capture additional output
  - Valid YAML
  - Placed in the appropriate sub-folder in the configs directory
  - Any image features required for the test are specified

- **Verification File**
  - Named `your_test.py`
  - Valid unit tests validating output collected
  - Passes pylint & pep8 checks
  - Placed in the appropriate sub-folder in the test cases directory

- **Tested by running the test:**

  ```
  $ tox -e citest -- run -verbose \
  --os-name <release target> \
  --test modules/your_test.yaml \
  [--deb <build of cloud-init>]
  ```

### 1.17.4 Platforms

**EC2**

To run on the EC2 platform it is required that the user has an AWS credentials configuration file specifying his or her access keys and a default region. These configuration files are the standard that the AWS cli and other AWS tools utilize for interacting directly with AWS itself and are normally generated when running `aws configure`:

```
$ cat $HOME/.aws/credentials
[default]
aws_access_key_id = <KEY HERE>
aws_secret_access_key = <KEY HERE>

$ cat $HOME/.aws/config
[default]
region = us-west-2
```

### 1.17.5 Architecture

The following section outlines the high-level architecture of the integration process.
Overview

The process flow during a complete end-to-end LXD-backed test.

1. Configuration
   - The back end and specific distro releases are verified as supported
   - The test or tests that need to be run are determined either by directory or by individual yaml

2. Image Creation
   - Acquire the request LXD image
   - Install the specified cloud-init package
   - Clean the image so that it does not appear to have been booted
   - A snapshot of the image is created and reused by all tests

3. Configuration
   - For each test, the cloud-config is injected into a copy of the snapshot and booted
   - The framework waits for /var/lib/cloud/instance/boot-finished (up to 120 seconds)
   - All default commands are ran and output collected
   - Any commands the user specified are executed and output collected

4. Verification
   - The default commands are checked for any failures, errors, and warnings to validate basic functionality of cloud-init completed successfully
   - The user generated unit tests are then ran validating against the collected output

5. Results
   - If any failures were detected the test suite returns a failure
   - Results can be dumped in yaml format to a specified file using the \texttt{-r <result\_file\_name>}. \texttt{yaml} option

Configuring the Test Suite

Most of the behavior of the test suite is configurable through several yaml files. These control the behavior of the test suite’s platforms, images, and tests. The main config files for platforms, images and test cases are \texttt{platforms.yaml}, \texttt{releases.yaml} and \texttt{testcases.yaml}.

Config handling

All configurable parts of the test suite use a defaults + overrides system for managing config entries. All base config items are dictionaries.

Merging is done on a key-by-key basis, with all keys in the default and override represented in the final result. If a key exists both in the defaults and the overrides, then the behavior depends on the type of data the key refers to. If it is atomic data or a list, then the overrides will replace the default. If the data is a dictionary then the value will be the result of merging that dictionary from the default config and that dictionary from the overrides.

Merging is done using the function \texttt{tests.cloud_tests.config.merge_config}, which can be examined for more detail on config merging behavior.
The following demonstrates merge behavior:

```yaml
defaults:
    list_item:
        - list_entry_1
        - list_entry_2
    int_item_1: 123
    int_item_2: 234
    dict_item:
        subkey_1: 1
        subkey_2: 2
        subkey_dict:
            subsubkey_1: a
            subsubkey_2: b

overrides:
    list_item:
        overridden_list_entry
    int_item_1: 0
    dict_item:
        subkey_2: false
        subkey_dict:
            subsubkey_2: 'new value'

result:
    list_item:
        overridden_list_entry
    int_item_1: 0
    int_item_2: 234
    dict_item:
        subkey_1: 1
        subkey_2: false
        subkey_dict:
            subsubkey_1: a
            subsubkey_2: 'new value'
```

**Image Config**

Image configuration is handled in `releases.yaml`. The image configuration controls how platforms locate and acquire images, how the platforms should interact with the images, how platforms should detect when an image has fully booted, any options that are required to set the image up, and features that the image supports.

Since settings for locating an image and interacting with it differ from platform to platform, there are 4 levels of settings available for images on top of the default image settings. The structure of the image config file is:

```yaml
default_release_config:
    default:
        ...
        <platform>:
            ...
        <platform>:
            ...

releases:
    <release name>:
        <default>:
            ...
```

(continues on next page)
The base config is created from the overall defaults and the overrides for the platform. The overrides are created from the default config for the image and the platform specific overrides for the image.

**System Boot**

The test suite must be able to test if a system has fully booted and if cloud-init has finished running, so that running collect scripts does not race against the target image booting. This is done using the `system_ready_script` and `cloud_init_ready_script` image config keys.

Each of these keys accepts a small bash test statement as a string that must return 0 or 1. Since this test statement will be added into a larger bash statement it must be a single statement using the `test` syntax.

The default image config provides a system ready script that works for any systemd based image. If the image is not systemd based, then a different test statement must be provided. The default config also provides a test for whether or not cloud-init has finished which checks for the file `/run/cloud-init/result.json`. This should be sufficient for most systems as writing this file is one of the last things cloud-init does.

The setting `boot_timeout` controls how long, in seconds, the platform should wait for an image to boot. If the system ready script has not indicated that the system is fully booted within this time an error will be raised.

**Feature Flags**

Not all test cases can work on all images due to features the test case requires not being present on that image. If a test case requires features in an image that are not likely to be present across all distros and platforms that the test suite supports, then the test can be skipped everywhere it is not supported.

Feature flags, which are names for features supported on some images, but not all that may be required by test cases. Configuration for feature flags is provided in `releases.yaml` under the `features` top level key. The features config includes a list of all currently defined feature flags, their meanings, and a list of feature groups.

Feature groups are groups of features that many images have in common. For example, the `Ubuntu_specific` feature group includes features that should be present across most Ubuntu releases, but may or may not be for other distros. Feature groups are specified for an image as a list under the key `feature_groups`.

An image’s feature flags are derived from the feature groups that that image has and any feature overrides provided. Feature overrides can be specified under the `features` key which accepts a dictionary of `{<feature_name>: true/false}` mappings. If a feature is omitted from an image’s feature flags or set to false in the overrides then the test suite will skip any tests that require that feature when using that image.

Feature flags may be overridden at run time using the `--feature-override` command line argument. It accepts a feature flag and value to set in the format `<feature name>=true/false`. Multiple `--feature-override` flags can be used, and will all be applied to all feature flags for images used during a test.

**Setup Overrides**

If an image requires some of the options for image setup to be used, then it may specify overrides for the command line arguments passed into setup image. These may be specified as a dictionary under the `setup_overrides` key. When an image is set up, the arguments that control how it is set up will be the arguments from the command line, with any entries in `setup_overrides` used to override these arguments.
For example, images that do not come with cloud-init already installed should have `setup_overrides:
(upgrade: true)` specified so that in the event that no additional setup options are given, cloud-init will be installed from the image’s repos before running tests. Note that if other options such as `--deb` are passed in on the command line, these will still work as expected, since apt’s policy for cloud-init would prefer the locally installed deb over an older version from the repos.

**Platform Specific Options**

There are many platform specific options in image configuration that allow platforms to locate images and that control additional setup that the platform may have to do to make the image usable. For information on how these work, please consult the documentation for that platform in the integration testing suite and the `releases.yaml` file for examples.

**Error Handling**

The test suite makes an attempt to run as many tests as possible even in the event of some failing so that automated runs collect as much data as possible. In the event that something goes wrong while setting up for or running a test, the test suite will attempt to continue running any tests which have not been affected by the error.

For example, if the test suite was told to run tests on one platform for two releases and an error occurred setting up the first image, all tests for that image would be skipped, and the test suite would continue to set up the second image and run tests on it. Or, if the system does not start properly for one test case out of many to run on that image, that test case will be skipped and the next one will be run.

Note that if any errors occur, the test suite will record the failure and where it occurred in the result data and write it out to the specified result file.

**Results**

The test suite generates result data that includes how long each stage of the test suite took and which parts were and were not successful. This data is dumped to the log after the collect and verify stages, and may also be written out in yaml format to a file. If part of the setup failed, the traceback for the failure and the error message will be included in the result file. If a test verifier finds a problem with the collected data from a test run, the class, test function and test will be recorded in the result data.

**Exit Codes**

The test suite counts how many errors occur throughout a run. The exit code after a run is the number of errors that occurred. If the exit code is non-zero then something is wrong either with the test suite, the configuration for an image, a test case, or cloud-init itself.

Note that the exit code does not always directly correspond to the number of failed test cases, since in some cases, a single error during image setup can mean that several test cases are not run. If run is used, then the exit code will be the sum of the number of errors in the collect and verify stages.

**Data Dir**

When using run, the collected data is written into a temporary directory. In the event that all tests pass, this directory is deleted, but if a test fails or an error occurs, this data will be left in place, and a message will be written to the log giving the location of the data.
cloudinit.config.cc_apt_configure, 76
cloudinit.config.cc_apt_pipelining, 79
cloudinit.config.cc_bootcmd, 80
cloudinit.config.cc_byobu, 80
cloudinit.config.cc_ca_certs, 81
cloudinit.config.cc_chef, 81
cloudinit.config.cc_debug, 82
cloudinit.config.cc_disable_ec2_metadata, 83
cloudinit.config.cc_disk_setup, 83
cloudinit.config.cc_emit_upstart, 84
cloudinit.config.cc_fan, 85
cloudinit.config.cc_final_message, 85
cloudinit.config.cc_foo, 86
cloudinit.config.cc_growpart, 86
cloudinit.config.cc_grub_dpkg, 87
cloudinit.config.cc_keys_to_console, 87
cloudinit.config.cc_landscape, 88
cloudinit.config.cc_locale, 88
cloudinit.config.cc_lxd, 89
cloudinit.config.cc_mcollective, 89
cloudinit.config.cc_migrator, 90
cloudinit.config.cc_mounts, 90
cloudinit.config.cc_ntp, 91
cloudinit.config.cc_package_update_upgrade_install, 92
cloudinit.config.cc_phone_home, 93
cloudinit.config.cc_power_state_change, 93
cloudinit.config.cc_puppet, 94
cloudinit.config.cc_resizesfs, 95
cloudinit.config.cc_resolv_conf, 95
cloudinit.config.cc_rh_subscription, 96
cloudinit.config.cc_rightscale_userdata, 96
cloudinit.config.cc_rsyslog, 97
cloudinit.config.cc_runcmd, 98
cloudinit.config.cc_salt_minion, 99
cloudinit.config.cc_scripts_per_boot, 99
cloudinit.config.cc_scripts_per_instance, 99
cloudinit.config.cc_scripts_per_once, 100
cloudinit.config.cc_scripts_user, 100
cloudinit.config.cc_scripts_vendor, 100
cloudinit.config.cc_seed_random, 100
cloudinit.config.cc_set_hostname, 101
cloudinit.config.cc_set_passwords, 102
cloudinit.config.cc_snap, 102
cloudinit.config.cc_snap_config, 105
cloudinit.config.cc_snappy, 104
cloudinit.config.cc_spacewalk, 105
cloudinit.config.cc_ssh, 106
cloudinit.config.cc_ssh_authkey_fingerprints, 107
cloudinit.config.cc_ssh_import_id, 107
cloudinit.config.cc_timezone, 108
cloudinit.config.cc_ubuntu_advantage, 108
cloudinit.config.cc_ubuntu_drivers, 109
cloudinit.config.cc_update_etc_hosts, 109
cloudinit.config.cc_update_hostname, 110
cloudinit.config.cc_users_groups, 110
cloudinit.config.cc_write_files, 112
cloudinit.config.cc_yum_add_repo, 113
<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudinit.config.cc_apt_configure</td>
<td>module</td>
<td>76</td>
</tr>
<tr>
<td>cloudinit.config.cc_apt_pipelining</td>
<td>module</td>
<td>79</td>
</tr>
<tr>
<td>cloudinit.config.cc_bootcmd</td>
<td>module</td>
<td>80</td>
</tr>
<tr>
<td>cloudinit.config.cc_byobu</td>
<td>module</td>
<td>80</td>
</tr>
<tr>
<td>cloudinit.config.cc_ca_certs</td>
<td>module</td>
<td>81</td>
</tr>
<tr>
<td>cloudinit.config.cc_chef</td>
<td>module</td>
<td>81</td>
</tr>
<tr>
<td>cloudinit.config.cc_debug</td>
<td>module</td>
<td>82</td>
</tr>
<tr>
<td>cloudinit.config.cc_disable_ec2_metadata</td>
<td>module</td>
<td>83</td>
</tr>
<tr>
<td>cloudinit.config.cc_disk_setup</td>
<td>module</td>
<td>83</td>
</tr>
<tr>
<td>cloudinit.config.cc_emit_upstart</td>
<td>module</td>
<td>84</td>
</tr>
<tr>
<td>cloudinit.config.cc_fan</td>
<td>module</td>
<td>85</td>
</tr>
<tr>
<td>cloudinit.config.cc_final_message</td>
<td>module</td>
<td>85</td>
</tr>
<tr>
<td>cloudinit.config.cc_foo</td>
<td>module</td>
<td>86</td>
</tr>
<tr>
<td>cloudinit.config.cc_growpart</td>
<td>module</td>
<td>86</td>
</tr>
<tr>
<td>cloudinit.config.cc_grub_dpkg</td>
<td>module</td>
<td>87</td>
</tr>
<tr>
<td>cloudinit.config.cc_keys_to_console</td>
<td>module</td>
<td>87</td>
</tr>
<tr>
<td>cloudinit.config.cc_landscape</td>
<td>module</td>
<td>88</td>
</tr>
<tr>
<td>cloudinit.config.cc_locale</td>
<td>module</td>
<td>88</td>
</tr>
<tr>
<td>cloudinit.config.cc_lxd</td>
<td>module</td>
<td>89</td>
</tr>
<tr>
<td>cloudinit.config.cc_mcollective</td>
<td>module</td>
<td>89</td>
</tr>
<tr>
<td>cloudinit.config.cc_migrator</td>
<td>module</td>
<td>90</td>
</tr>
<tr>
<td>cloudinit.config.cc_mounts</td>
<td>module</td>
<td>90</td>
</tr>
<tr>
<td>cloudinit.config.cc_ntp</td>
<td>module</td>
<td>91</td>
</tr>
<tr>
<td>cloudinit.config.cc_package_update_upgrade_install</td>
<td>module</td>
<td>92</td>
</tr>
<tr>
<td>cloudinit.config.cc_phone_home</td>
<td>module</td>
<td>93</td>
</tr>
<tr>
<td>cloudinit.config.cc_power_state_change</td>
<td>module</td>
<td>93</td>
</tr>
<tr>
<td>cloudinit.config.cc_puppet</td>
<td>module</td>
<td>94</td>
</tr>
<tr>
<td>cloudinit.config.cc_resizefs</td>
<td>module</td>
<td>95</td>
</tr>
<tr>
<td>cloudinit.config.cc_resolv_conf</td>
<td>module</td>
<td>95</td>
</tr>
<tr>
<td>cloudinit.config.cc_rh_subscription</td>
<td>module</td>
<td>96</td>
</tr>
<tr>
<td>cloudinit.config.cc_rightscale_userdata</td>
<td>module</td>
<td>96</td>
</tr>
<tr>
<td>cloudinit.config.cc_rsyslog</td>
<td>module</td>
<td>97</td>
</tr>
<tr>
<td>cloudinit.config.cc_runcmd</td>
<td>module</td>
<td>98</td>
</tr>
<tr>
<td>cloudinit.config.cc_salt_minion</td>
<td>module</td>
<td>99</td>
</tr>
<tr>
<td>cloudinit.config.cc_scripts_per_boot</td>
<td>module</td>
<td>99</td>
</tr>
<tr>
<td>cloudinit.config.cc_scripts_per_instance</td>
<td>module</td>
<td>99</td>
</tr>
<tr>
<td>cloudinit.config.cc_scripts_per_once</td>
<td>module</td>
<td>100</td>
</tr>
<tr>
<td>cloudinit.config.cc_scripts_user</td>
<td>module</td>
<td>100</td>
</tr>
<tr>
<td>cloudinit.config.cc_scripts_vendor</td>
<td>module</td>
<td>100</td>
</tr>
<tr>
<td>cloudinit.config.cc_seed_random</td>
<td>module</td>
<td>100</td>
</tr>
<tr>
<td>cloudinit.config.cc_set_hostname</td>
<td>module</td>
<td>101</td>
</tr>
<tr>
<td>cloudinit.config.cc_set_passwords</td>
<td>module</td>
<td>102</td>
</tr>
<tr>
<td>cloudinit.config.cc_snapsnap</td>
<td>module</td>
<td>102</td>
</tr>
<tr>
<td>cloudinit.config.cc_snap</td>
<td>module</td>
<td>103</td>
</tr>
<tr>
<td>cloudinit.config.cc_snap_config</td>
<td>module</td>
<td>105</td>
</tr>
<tr>
<td>cloudinit.config.cc_snappy</td>
<td>module</td>
<td>104</td>
</tr>
<tr>
<td>cloudinit.config.cc_snapconf</td>
<td>module</td>
<td>106</td>
</tr>
<tr>
<td>cloudinit.config.cc_spacewalk</td>
<td>module</td>
<td>105</td>
</tr>
<tr>
<td>cloudinit.config.cc_ssh</td>
<td>module</td>
<td>106</td>
</tr>
<tr>
<td>cloudinit.config.cc_ssh_authkey_fingerprints</td>
<td>module</td>
<td>107</td>
</tr>
<tr>
<td>cloudinit.config.cc_ssh_import_id</td>
<td>module</td>
<td>107</td>
</tr>
<tr>
<td>cloudinit.config.cc_timezone</td>
<td>module</td>
<td>108</td>
</tr>
<tr>
<td>cloudinit.config.cc_ubuntu_advantage</td>
<td>module</td>
<td>108</td>
</tr>
</tbody>
</table>
Cloud-Init, Release 19.2

(module), 108
cloudinit.config.cc_ubuntu_drivers (module), 109
cloudinit.config.cc_update_etc_hosts (module), 109
cloudinit.config.cc_update_hostname (module), 110
cloudinit.config.cc_users_groups (module), 110
cloudinit.config.cc_write_files (module), 112
cloudinit.config.cc_yum_add_repo (module), 113