Red Hat Enterprise Linux 9

Configuring and managing cloud-init for RHEL 9

Using cloud-init to automate the initialization of cloud instances

Last Updated: 2022-05-17

Red Hat Enterprise Linux 9 Configuring and managing cloud-init for RHEL 9

Using cloud-init to automate the initialization of cloud instances

Legal Notice

Abstract

You can use cloud-init to automate the initialization of cloud instances. You can install the cloud-init package on your virtual machine, or you can choose a Red Hat Enterprise Linux image that includes cloud-init already installed. You can use cloud-init with a number of Red Hat products.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better.

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting feedback via Bugzilla, create a new ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. INTRODUCTION TO CLOUD-INIT

cloud-init is a software package that automates the initialization of cloud instances during system boot. You can configure **cloud-init** to perform a variety of tasks. Some sample tasks that **cloud-init** can perform include:

- Configuring a host name
- Installing packages on an instance
- Running scripts
- Suppressing default virtual machine (VM) behavior

Where you obtain your image for configuring **cloud-init** depends on how you intend to use it.

- The **cloud-init** package is installed on KVM Guest Images that you download from the Red Hat Customer Portal. When you launch an instance, **cloud-init** is enabled. KVM Guest Images that you download from the Red Hat Customer Portal are intended for use with Red Hat Virtualization (RHV) and the Red Hat OpenStack Platform (RHOSP). You can also create an image from scratch for RHV and RHOSP.
- Another option is to download an ISO image from the Red Hat Customer Portal or create one. In this case, you need to install **cloud-init** on your ISO image.
- If you plan to use an image with a cloud provider (for example, AWS or Azure), use Red Hat Image Builder to create the image. Image Builder images are customized for use for specific cloud providers. The image types AMI, VHD, and qcow2 include **cloud-init** already installed. Refer to Composing a Customized RHEL System Image for information on Image Builder.

Most cloud platforms support **cloud-init**, though configuration procedures and supported options vary. Alternatively, you can configure **cloud-init** for a NoCloud environment.

You can configure **cloud-init** on one VM and then use that VM as a template for additional VMs or clusters of VMs.

Specific Red Hat products (for example, Red Hat Virtualization) have documented procedures for configuring **cloud-init** for use with those products.

This document refers to the **cloud-init** documentation in a number of places. Refer to the referenced **cloud-init** documentation for complete information on **cloud-init**.

Prerequisites

- Sign up for a Red Hat Customer Portal account.

1.1. ADDITIONAL RESOURCES

- Documentation for cloud-init

1.2. CLOUD-INIT CONFIGURATION

cloud-init uses YAML-formatted file instructions to perform tasks. You decide the initial configuration you want **cloud-init** to perform by providing instructions within the YAML files. When an instance boots, the **cloud-init** service starts and searches for and executes the instructions. Tasks complete during the

first boot or on subsequent boots of your VM, based on your **cloud-init** configuration.

You define the tasks by configuring the **/etc/cloud/cloud.cfg** file and adding directives under the **/etc/cloud/cloud.cfg.d/** directory.

- The **cloud.cfg** file includes directives, such as those for user access and authentication and system information.
  The file also includes default and optional modules for **cloud-init**. The modules are executed in order within three phases that include the **cloud-init** initialization phase, the configuration phase, and the final phase. Within the **cloud.cfg** file, modules for the three phases are listed under **cloud_init_modules**, **cloud_config_modules**, and **cloud_final_modules**, respectively.

- The **cloud.cfg.d** directory is where you can add additional directives for **cloud-init**. When you add directives to the **cloud.cfg.d** directory, you typically add them to a file named ***.cfg**, and you always include **#cloud-config** at the top of the file.

1.3. CLOUD-INIT OPERATES IN STAGES

cloud-init operates in five stages during a system boot. Those stages determine whether **cloud-init** runs and where it finds its datasources, among other tasks. The stages are as follows:

1. The **cloud-init** generator stage, through the **systemd** service, determines whether to run **cloud-init** upon the boot.

2. During the local stage, **cloud-init** finds local datasources and applies network configuration.

3. During the network stage, **cloud-init** processes user data and runs the modules listed under **cloud_init_modules** in your **cloud.cfg** file. You can enable, disable, or add modules to the **cloud_init_modules** section.

4. During the config stage, **cloud-init** runs the modules listed under **cloud_config_modules** in your **cloud.cfg** file. You can enable, disable, or add modules to the **cloud_config_modules** section.

5. During the final stage, **cloud-init** can run what you have included under **cloud_final_modules** in your **cloud.cfg** file. You can include package installations that you would typically run after a system boots and can also include configuration management plug-ins and user scripts. You can enable, disable, or add modules to the **cloud_final_modules** section.

The five boot stages are described in the **cloud-init** Documentation section Boot Stages.

1.4. CLOUD-INIT MODULES EXECUTE IN PHASES

When **cloud-init** runs, it executes the modules within **cloud.cfg** in order within three phases:

1. The network phase (**cloud_init_modules**)

2. The configuration phase (**cloud_config_modules**)

3. The final phase (**cloud_final_modules**)

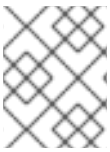
When **cloud-init** runs for the first time on a VM, all the modules you have configured run in their respective phases. On a subsequent running of **cloud-init**, whether a module runs within a phase depends on the *module frequency* of the individual module. Some modules run every time **cloud-init** runs; some modules only run the first time **cloud-init** runs, even if the instance ID changes.

NOTE

An instance ID uniquely identifies an instance. When an instance ID changes, **cloud-init** treats the instance as a new instance.

The possible *module frequency* values are as follows:

- **Per instance** means that the module runs on first boot of an instance. For example, if you clone an instance or create a new instance from a saved image, the modules designated as per instance run again.
- **Per once** means that the module runs only once. For example, if you clone an instance or create a new instance from a saved image, the modules designated per once do not run again on those instances.
- **Per always** means the module runs on every boot.



NOTE

You can override a module's frequency when you configure the module or by using the command line.

1.5. CLOUD-INIT ACTS UPON USER DATA, METADATA, AND VENDOR DATA

cloud-init consumes and acts upon user data, metadata, and vendor data.

- User data includes directives you specify in the **cloud.cfg** file and in the **cloud.cfg.d** directory, for example, user data can include files to run, packages to install, and shell scripts. Refer to the **cloud-init** Documentation section User-Data Formats for information on the types of user data that **cloud-init** allows.
- Metadata includes data associated with a specific datasource, for example, metadata can include a server name and instance ID. If you are using a specific cloud platform, the platform determines where your instances find user data and metadata. Your platform may require that you add metadata and user data to an HTTP service; in this case, when **cloud-init** runs it consumes metadata and user data from the HTTP service.
- Vendor data is optionally provided by the organization (for example, a cloud provider) and includes information that can customize the image to better fit the environment where the image runs. **cloud-init** acts upon optional vendor data and user data after it reads any metadata and initializes the system. By default, vendor data runs on the first boot. You can disable vendor data execution.
  Refer to the **cloud-init** Documentation section Instance Metadata for a description of metadata; Datasources for a list of datasources; and Vendor Data for more information on vendor data.

1.6. CLOUD-INIT IDENTIFIES THE CLOUD PLATFORM

cloud-init attempts to identify the cloud platform using the script **ds-identify**. The script runs on the first boot of an instance.

Adding a datasource directive can save time when **cloud-init** runs. You would add the directive in the **/etc/cloud/cloud.cfg** file or in the **/etc/cloud/cloud.cfg.d** directory. For example:

```
datasource_list:[Ec2]
```

Beyond adding the directive for your cloud platform, you can further configure **cloud-init** by adding additional configuration details, such as metadata URLs.

```
datasource_list: [Ec2]
datasource:
  Ec2:
    metadata_urls: ['http://169.254.169.254']
```

After **cloud-init** runs, you can view a log file (**run/cloud-init/ds-identify.log**) that provides detailed information about the platform.

Additional resources

- Datasources
- What datasource am I using?

CHAPTER 2. RED HAT SUPPORT FOR CLOUD-INIT

This chapter covers Red Hat support for **cloud-init**. It includes information on Red Hat products that use **cloud-init**, **cloud-init** modules that Red Hat supports, and default directories and files.

2.1. CLOUD-INIT SIGNIFICANT DIRECTORIES AND FILES

The following table includes important directories and files. Review these directories and files; they allow you to perform tasks like:

- Configuring **cloud-init**
- Finding information on your configuration after **cloud-init** has run
- Examining log files
- Finding templates

Depending on your scenario and datasource, there can be additional files and directories important to your configuration.

Table 2.1. cloud-init directories and files

Directory or File	Description
/etc/cloud/cloud.cfg	The **cloud.cfg** file includes the basic **cloud-init** configuration and lets you know in what phase each module runs.
/etc/cloud/cloud.cfg.d	The **cloud.cfg.d** directory is where you can add additional directives for **cloud-init**.
/var/lib/cloud	When **cloud-init** runs, it creates a directory layout under **/var/lib/cloud**. The layout includes directories and files that give specifics on your instance configuration.
/usr/share/doc/cloud-init/examples	The **examples** directory includes multiple examples. You can use them to help model your own directives.
/etc/cloud/templates	This directory includes templates that you can enable in **cloud-init** for certain scenarios. The templates provide direction for enabling.
/var/log/cloud-init.log	The **cloud-init.log** file provides log information helpful for debugging.
/run/cloud-init	The **/run/cloud-init** directory includes logged information on your datasource and the **ds-identify** script.

2.2. RED HAT PRODUCTS THAT USE CLOUD-INIT

You can use **cloud-init** with the following Red Hat products.

- **Red Hat Virtualization.** Once you install **cloud-init** on a VM, you can create a template and leverage **cloud-init** functions for all VMs created from that template. Refer to Using Cloud-Init to Automate the Configuration of Virtual Machines for information on using **cloud-init** with VMs.
- **Red Hat OpenStack Platform.** You can use **cloud-init** to help configure images for OpenStack. Refer to the Instances and Images Guide for more information.
- **Red Hat CloudForms.** You can use **cloud-init** to provision VMs for Red Hat CloudForms. Refer to Customization Templates for Virtual Machine and Instance Provisioning for more information.
- **Red Hat Satellite.** You can use **cloud-init** with Red Hat Satellite. Refer to Preparing Cloud-init Images in Red Hat Virtualization for more information.
- **Red Hat OpenShift.** You can use **cloud-init** when you create VMs for OpenShift. Refer to Creating Virtual Machines for more information.

2.3. RED HAT SUPPORTS THESE CLOUD-INIT MODULES

Red Hat supports most **cloud-init** modules. Individual modules can contain multiple configuration options. The following table lists all of the **cloud-init** modules that Red Hat currently supports and provides a brief description and the default module frequency. Refer to Modules in the cloud-init Documentation section for complete descriptions and options for these modules.

Table 2.2. Supported cloud-init modules

cloud-init Module	Description	Default Module Frequency
bootcmd	Runs commands early in the boot process	per always
ca_certs	Adds CA certificates	per instance
debug	Enables or disables output of internal information to assist with debugging	per instance
disable_ec2_metadata	Enables or disables the AWS EC2 metadata	per always
disk_setup	Configures simple partition tables and file systems	per instance
final_message	Specifies the output message once **cloud-init** completes	per always
foo	Example shows module structure (Module does nothing)	per instance

cloud-init Module	Description	Default Module Frequency
growpart	Resizes partitions to fill the available disk space	per always
keys_to_console	Allows controls of fingerprints and keys that can be written to the console	per instance
landscape	Installs and configures a landscape client	per instance
locale	Configures the system locale and applies it system-wide	per instance
mcollective	Installs, configures, and starts **mcollective**	per instance
migrator	Moves old versions of **cloud-init** to newer versions	per always
mounts	Configures mount points and swap files	per instance
phone_home	Posts data to a remote host after boot completes	per instance
power_state_change	Completes shutdown and reboot after all configuration modules have run	per instance
puppet	Installs and configures puppet	per instance
resizefs	Resizes a file system to use all available space on a partition	per always
resolve_conf	Configures **resolv.conf**	per instance
rh_subscription	Registers a Red Hat Enterprise Linux system	per instance
rightscale_userdata	Adds support for RightScale configuration hooks to **cloud-init**	per instance
rsyslog	Configures remote system logging using **rsyslog**	per instance

cloud-init Module	Description	Default Module Frequency
runcmd	Runs arbitrary commands	per instance
salt_minion	Installs, configures, and starts **salt minion**	per instance
scripts_per_boot	Runs per boot scripts	per always
scripts_per_instance	Runs per instance scripts	per instance
scripts_per_once	Runs scripts once	per once
scripts_user	Runs user scripts	per instance
scripts_vendor	Runs vendor scripts	per instance
seed_random	Provides random seed data	per instance
set_hostname	Sets host name and fully qualified domain name (FQDN)	per always
set_passwords	Sets user passwords and enables or disables SSH password authentication	per instance
ssh_authkey_fingerprints	Logs fingerprints of user SSH keys	per instance
ssh_import_id	Imports SSH keys	per instance
ssh	Configures SSH, and host and authorized SSH keys	per instance
timezone	Sets the system time zone	per instance
update_etc_hosts	Updates **/etc/hosts**	per always
update_hostname	Updates host name and FQDN	per always
users_groups	Configures users and groups	per instance
write_files	Writes arbitrary files	per instance
yum_add_repo	Adds dnf repository configuration to the system	per always

The following table lists modules that Red Hat does not currently support.

Table 2.3. Modules not supported

Module
apt_configure
apt_pipeline
byobu
chef
emit_upstart
grub_dpkg
ubuntu_init_switch

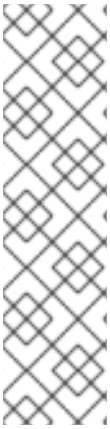
2.4. THE DEFAULT CLOUD.CFG FILE

The **/etc/cloud/cloud.cfg** file lists the modules comprising the basic configuration for **cloud-init**.

The modules in the file are the default modules for **cloud-init**. You can configure the modules for your environment or remove modules you do not need. Modules that are included in **cloud.cfg** do not necessarily do anything by being listed in the file. You need to configure them individually if you want them to perform actions during one of the **cloud-init** phases.

The **cloud.cfg** file provides the chronology for running individual modules. You can add additional modules to **cloud.cfg** as long as Red Hat supports the modules you want to add.

The default contents of the file for Red Hat Enterprise Linux (RHEL) are as follows:



NOTE

- Modules run in the order given in **cloud.cfg**. You typically do not change this order.
- The **cloud.cfg** directives can be overridden by user data.
- When running **cloud-init** manually, you can override **cloud.cfg** with command line options.
- Each module includes its own configuration options, where you can add specific information.

```
users: 1
 - default

disable_root: 1 2
```

```
ssh_pwauth:   0 3

mount_default_fields: [~, ~, 'auto', 'defaults,nofail,x-systemd.requires=cloud-init.service', '0', '2'] 4
ssh_deletekeys:   1 5
ssh_genkeytypes: ['rsa', 'ecdsa', 'ed25519'] 6
syslog_fix_perms: ~ 7
disable_vmware_customization: false 8

cloud_init_modules: 9
 - disk_setup
 - migrator
 - bootcmd
 - write-files
 - growpart
 - resizefs
 - set_hostname
 - update_hostname
 - update_etc_hosts
 - rsyslog
 - users-groups
 - ssh

cloud_config_modules: 10
 - mounts
 - locale
 - set-passwords
 - rh_subscription
 - dnf-add-repo
 - package-update-upgrade-install
 - timezone
 - puppet
 - chef
 - salt-minion
 - mcollective
 - disable-ec2-metadata
 - runcmd

cloud_final_modules: 11
 - rightscale_userdata
 - scripts-per-once
 - scripts-per-boot
 - scripts-per-instance
 - scripts-user
 - ssh-authkey-fingerprints
 - keys-to-console
 - phone-home
 - final-message
 - power-state-change

system_info:
  default_user: 12
    name: cloud-user
    lock_passwd: true
    gecos: Cloud User
    groups: [adm, systemd-journal]
```

```
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    shell: /bin/bash
  distro: rhel (13)
  paths:
    cloud_dir: /var/lib/cloud (14)
    templates_dir: /etc/cloud/templates (15)
  ssh_svcname: sshd (16)

# vim:syntax=yaml
```

1. Specifies the default user for the system. Refer to Users and Groups for more information.
2. Enables or disables root login. Refer to Authorized Keys for more information.
3. Specifies whether **ssh** is configured to accept password authentication. Refer to Set Passwords for more information.
4. Configures mount points; must be a list containing six values. Refer to Mounts for more information.
5. Specifies whether to remove default host SSH keys. Refer to Host Keys for more information.
6. Specifies key types to generate. Refer to Host Keys for more information. Note that for RHEL 8.4 and earlier, the default value of this line is ~.
7. **cloud-init** runs at multiple stages of boot. Set this option so that **cloud-init** can log all stages to its log file. Find more information on this option in the **cloud-config.txt** file in the **usr/share/doc/cloud-init/examples** directory.
8. Enables or disables VMware vSphere customization
9. The modules in this section are services that run when the **cloud-init** service starts, early in the boot process.
10. These modules run during **cloud-init** configuration, after initial boot.
11. These modules run in the final phase of **cloud-init**, after the configuration finishes.
12. Specifies details about the default user. Refer to Users and Groups for more information.
13. Specifies the distribution
14. Specifies the main directory that contains **cloud-init**-specific subdirectories. Refer to Directory layout for more information.
15. Specifies where templates reside
16. The name of the SSH service

Additional resources

- Where are the Configuration Files?
- Modules

2.5. THE CLOUD.CFG.D DIRECTORY

cloud-init acts upon directives that you provide and configure. Typically, those directives are included in the **cloud.cfg.d** directory.

NOTE

While you can configure modules by adding user data directives within the **cloud.cfg** file, as a best practice consider leaving **cloud.cfg** unmodified. Add your directives to the **/etc/cloud/cloud.cfg.d** directory. Adding directives to this directory can make future modifications and upgrades easier.

There are multiple ways to add directives. You can include directives in a file named ***.cfg**, which includes the heading **#cloud-config**. Typically, the directory would contain multiple ***cfg** files. There are other options for adding directives, for example, you can add a user data script. Refer to User-Data Formats for more information.

Additional resources

- Where are the Configuration Files?
- Cloud config examples

2.6. THE DEFAULT 05_LOGGING.CFG FILE

The **05_logging.cfg** file sets logging information for **cloud-init**. The **/etc/cloud/cloud.cfg.d** directory includes this file, along with other **cloud-init** directives that you add.

cloud-init uses the logging configuration in **05_logging.cfg** by default. The default contents of the file for Red Hat Enterprise Linux (RHEL) are as follows:

```
## This yaml formatted config file handles setting
## logger information.  The values that are necessary to be set
## are seen at the bottom.  The top '_log' are only used to remove
## redundancy in a syslog and fallback-to-file case.
##
## The 'log_cfgs' entry defines a list of logger configs
## Each entry in the list is tried, and the first one that
## works is used.  If a log_cfg list entry is an array, it will
## be joined with '\n'.
_log:
 - &log_base |
   [loggers]
   keys=root,cloudinit

   [handlers]
   keys=consoleHandler,cloudLogHandler

   [formatters]
   keys=simpleFormatter,arg0Formatter

   [logger_root]
   level=DEBUG
   handlers=consoleHandler,cloudLogHandler
```

```
    [logger_cloudinit]
    level=DEBUG
    qualname=cloudinit
    handlers=
    propagate=1

    [handler_consoleHandler]
    class=StreamHandler
    level=WARNING
    formatter=arg0Formatter
    args=(sys.stderr,)

    [formatter_arg0Formatter]
    format=%(asctime)s - %(filename)s[%(levelname)s]: %(message)s

    [formatter_simpleFormatter]
    format=[CLOUDINIT] %(filename)s[%(levelname)s]: %(message)s
  - &log_file |
    [handler_cloudLogHandler]
    class=FileHandler
    level=DEBUG
    formatter=arg0Formatter
    args=('/var/log/cloud-init.log',)
  - &log_syslog |
    [handler_cloudLogHandler]
    class=handlers.SysLogHandler
    level=DEBUG
    formatter=simpleFormatter
    args=("/dev/log", handlers.SysLogHandler.LOG_USER)

log_cfgs:
# Array entries in this list will be joined into a string
# that defines the configuration.
#
# If you want logs to go to syslog, uncomment the following line.
# - [ *log_base, *log_syslog ]
#
# The default behavior is to just log to a file.
# This mechanism that does not depend on a system service to operate.
 - [ *log_base, *log_file ]
# A file path can also be used.
# - /etc/log.conf

# This tells cloud-init to redirect its stdout and stderr to
# 'tee -a /var/log/cloud-init-output.log' so the user can see output
# there without needing to look on the console.
output: {all: '| tee -a /var/log/cloud-init-output.log'}
```

Additional resources

- Logging

2.7. THE CLOUD-INIT /VAR/LIB/CLOUD DIRECTORY LAYOUT

When **cloud-init** first runs, it creates a directory layout that includes information about your instance and **cloud-init** configuration.

The directory can include optional directories, such as **/scripts/vendor**.

The following is a sample directory layout for **cloud-init**.

```
/var/lib/cloud/
  - data/
    - instance-id
    - previous-instance-id
    - previous-datasource
    - previous-hostname
    - result.json
    - set-hostname
    - status.json
  - handlers/
  - instance
    - boot-finished
    - cloud-config.txt
    - datasource
    - handlers/
    - obj.pkl
    - scripts/
    - sem/
    - user-data.txt
    - user-data.txt.i
    - vendor-data.txt
    - vendor-data.txt.i
  - instances/
     f111ee00-0a4a-4eea-9c17-3fa164739c55/
       - boot-finished
       - cloud-config.txt
       - datasource
       - handlers/
       - obj.pkl
       - scripts/
       - sem/
       - user-data.txt
       - user-data.txt.i
       - vendor-data.txt
       - vendor-data.txt.i
  - scripts/
    - per-boot/
    - per-instance/
    - per-once/
    - vendor/
  - seed/
  - sem/
    - config_scripts_per_once.once
```

Additional resources

- Directory layout

CHAPTER 3. CONFIGURING CLOUD-INIT

This chapter includes examples of the most common configuration tasks for **cloud-init**.

Your **cloud-init** configuration can require that you add directives to the **cloud.cfg** file and the **cloud.cfg.d** directory. Alternatively, your specific data source might require that you add directives to files, such as a user data file and a metadata file. A data source might require that you upload your directives to an HTTP server. Check the requirements of your data source and add directives accordingly.

3.1. CREATING A VIRTUAL MACHINE THAT INCLUDES CLOUD-INIT FOR A NOCLOUD DATASOURCE

This section provides a sample procedure for creating a new virtual machine (VM) that includes **cloud-init**. In this procedure, you create a **meta-data** and **user-data** file.

- Your **meta-data** file includes instance details.
- Your **user-data** file includes information to create a user and grant access.

Then, you include these files in a new ISO image, and you attach the ISO file to a new VM you create from a KVM Guest Image. In this scenario, the datasource is NoCloud.

Procedure

1. Create a directory named **cloudinitiso** and move into it.

```
$ mkdir cloudinitiso
$ cd cloudinitiso
```

2. Create a file named **meta-data**. Add the following information to the file.

```
instance-id: citest
local-hostname: citest-1
```

3. Create a file named **user-data**. Include the following information in the file.

```
#cloud-config
password: cilogon
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...fhHQ== sample@redhat.com
```

NOTE

The final line of the **user-data** file references an SSH public key. Find your SSH public keys in **~/.ssh/id_rsa.pub**. When trying this sample procedure, modify the line to include one of your public keys.

4. Use the **genisoimage** command to create an ISO image that includes **user-data** and **meta-data**.

```
# genisoimage -output ciiso.iso -volid cidata -joliet -rock user-data meta-data

I: -input-charset not specified, using utf-8 (detected in locale settings)
Total translation table size: 0
Total rockridge attributes bytes: 331
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 0
183 extents written (0 MB)
```

5. Download a KVM Guest Image from the Red Hat Customer Portal to the **/var/lib/libvirt/images** directory.

6. Create a new VM from the KVM Guest Image using the **virt-install** command. Include the ISO image you created as an attachment to the image.

```
virt-install \
   --memory 4096 \
   --vcpus 4 \
   --name mytestcivm \
   --disk /var/lib/libvirt/images/rhel-8.1-x86_64-
kvm.qcow2,device=disk,bus=virtio,format=qcow2 \
   --disk /home/sample/cloudinitiso/ciiso.iso,device=cdrom \
   --os-type Linux \
   --os-variant rhel9.0 \
   --virt-type kvm \
   --graphics none \
   --import
```

7. Log on to your image as **cloud-user**. Your password is **cilogon**.

```
citest-1 login: cloud-user
Password:
[cloud-user@citest-1 ~]$
```

Verification

- Check the **cloud-init** status to see that it has completed its tasks.

```
[cloud-user@citest-1 instance]$ cloud-init status
status: done
```

- **cloud-init** creates the **cloud-init** directory layout under **/var/lib/cloud** when it runs, and it updates or changes certain directory contents based upon the directives you have specified. For example, you can confirm that the datasource is **NoCloud** by checking the datasource file.

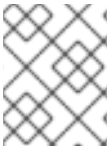
```
$ cd /var/lib/cloud/instance
$ cat datasource
DataSourceNoCloud: DataSourceNoCloud [seed=/dev/sr0][dsmode=net]
```

cloud-init copies user-data into **/var/lib/cloud/instance/user-data.txt**.

```
$cat user-data.txt
```

```
#cloud-config
password: cilogon
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...fhHQ== sample@redhat.com
```

These are samples. The **cloud-init** directory layout includes much more information.



NOTE

For OpenStack, the Instances and Images Guide includes information for configuring an instance using **cloud-init**. See Creating a customized instance for specific procedures.

Additional resources

- NoCloud

3.2. EXPIRING A CLOUD USER PASSWORD WITH CLOUD-INIT

You can force **cloud-user** to change the **cloud-user** password at the first login. Perform the following procedure to expire a password.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

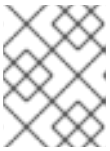
NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Change the line **chpasswd: {expire: False}** to **chpasswd: {expire: True}**.

```
#cloud-config
password: mypassword
chpasswd: {expire: True}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...SDvz user1@yourdomain.com
  - ssh-rsa AAB...QTuo user2@yourdomain.com
```

This works to expire the password because **password** and **chpasswd** operate on the default user unless you indicate otherwise.



NOTE

This is a global setting. When you set **chpasswd** to **True**, all users you create need to change their passwords when they log in.

3.3. CHANGING A DEFAULT USER NAME WITH CLOUD-INIT

You can change the default user name to something other than **cloud-user**.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add the line **user: <username>**, replacing <username> with the new default user name.

```
#cloud-config
user: username
password: mypassword
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...SDvz user1@yourdomain.com
  - ssh-rsa AAB...QTuo user2@yourdomain.com
```

3.4. SETTING A ROOT PASSWORD WITH CLOUD-INIT

To set the root password, create a user list.

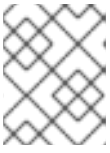
Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Create a user list in the **chpasswd** section of the file. The format is shown in the following sample.



NOTE

White space is significant. Do not include white space before or after the colon in your user list. If you include white space, the password is set with a space in it.

```
#cloud-config
```

```
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...SDvz user1@yourdomain.com
  - ssh-rsa AAB...QTuo user2@yourdomain.com
chpasswd:
  list: |
     root:myrootpassword
     cloud-user:mypassword
  expire: False
```

NOTE

If you use this method to set the user password, you must set **all passwords** in this section.

3.5. MANAGING RED HAT SUBSCRIPTIONS WITH CLOUD-INIT

You can use the **rh_subscription** directive to register your system. Samples follow. For each subscription, you would edit your user data.

Procedure

The following example uses the **auto-attach** and **service-level** options.

- Under **rh_subscription**, add your **username** and **password**, set **auto-attach** to **True**, and set **service-level** to **self-support**.

```
rh_subscription:
  username: sample@redhat.com
  password: 'mypassword'
  auto-attach: True
  service-level: self-support
```

NOTE

The **service-level** option requires that you use the **auto-attach** option.

The following example uses the **activation-key** and **org** options.

- Under **rh_subscription**, add your **activation key** and **org** number and set **auto-attach** to **True**.

```
rh_subscription:
  activation-key: example_key
  org: 12345
  auto-attach: True
```

The following example adds a subscription pool.

- Under **rh_subscription**, add your **username**, **password**, and pool number.

```
rh_subscription:
  username: sample@redhat.com
  password: 'password'
  add-pool: XYZ01234567
```

-

NOTE

This sample is the equivalent of the **subscription-manager attach --pool=XYZ01234567** command.

The following example sets a server host name in the **/etc/rhsm/rhsm.conf** file.

- Under **rh_subscription**, add your **username**, **password**, **server-hostname**, and set **auto-attach** to **True**.

```
rh_subscription:
  username: sample@redhat.com
  password: 'password'
  server-hostname: test.example.com
  auto-attach: True
```

3.6. ADDING USERS AND USER OPTIONS WITH CLOUD-INIT

You create and describe users in a **users** section. You can modify the section to add more users to your initial system configuration, and you can set additional user options.

If you add the **users** section, you must also set the default user options in this section.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add or modify the **users** section to add users.

   - If you want **cloud-user** to be the default user created along with the other users you specify, ensure that you add **default** as the first entry in the section. If it is not the first entry, **cloud-user** is not created.

   - By default, users are labeled as **unconfined_u** if there is not an **selinux-user** value.

```
#cloud-config
users:
  - default
  - name: user2
    gecos: User N. Ame
    selinux-user: staff_u
    groups: users,wheel
    ssh_pwauth: True
    ssh_authorized_keys:
```

```
    - ssh-rsa AA..vz user@domain.com
chpasswd:
  list: |
    root:password
    cloud-user:mypassword
    user2:mypassword2
  expire: False
```

NOTE

- The example places the user **user2** into two groups, **users** and **wheel**.

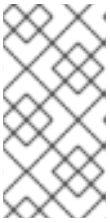
3.7. RUNNING FIRST BOOT COMMANDS WITH CLOUD-INIT

You can use the **runcmd** and **bootcmd** sections to execute commands during startup and initialization.

The **bootcmd** section executes early in the initialization process and by default runs on every boot. The **runcmd** section executes near the end of the process and is only executed during the first boot and initialization.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.



NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add the sections for **bootcmd** and **runcmd**; include commands you want **cloud-init** to execute.

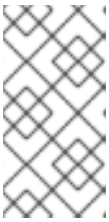
```
#cloud-config
users:
  - default
  - name: user2
    gecos: User N. Ame
    groups: users
chpasswd:
  list: |
    root:password
    fedora:myfedpassword
    user2:mypassword2
  expire: False
bootcmd:
 - echo New MOTD >> /etc/motd
runcmd:
 - echo New MOTD2 >> /etc/motd
```

3.8. ADDING ADDITIONAL SUDOERS WITH CLOUD-INIT

You can configure a user as a sudoer by adding a **sudo** and **groups** entry to the **users** section.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.



NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add a **sudo** entry and specify the user access. For example, **sudo: ALL=(ALL) NOPASSWD:ALL** allows a user unrestricted user access.

3. Add a **groups** entry and specify the groups that include the user.

```
#cloud-config
users:
  - default
  - name: user2
    gecos: User D. Two
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    groups: wheel,adm,systemd-journal
    ssh_pwauth: True
    ssh_authorized_keys:
      - ssh-rsa AA...vz user@domain.com
chpasswd:
  list: |
    root:password
    cloud-user:mypassword
    user2:mypassword2
  expire: False
```

3.9. SETTING UP A STATIC NETWORKING CONFIGURATION WITH CLOUD-INIT

You can set up your network configuration with **cloud-init** by adding a **network-interfaces** section to your metadata.

Red Hat Enterprise Linux provides its default networking service through **NetworkManager**, which is a dynamic network control and configuration daemon that keeps network devices and connections up and active when they are available. Refer to Getting Started with NetworkManager for more information on **NetworkManager**.

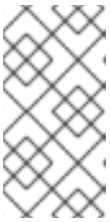
Your datasource might provide a network configuration. Refer to the **cloud-init** documentation section Network Configuration Sources for more information.

If you specify no network configuration for **cloud-init** and have not disabled network configuration, **cloud-init** tries to determine if any attached devices have a connection. If it finds a connected device, it generates a network configuration that issues a DHCP request on the interface. Refer to the **cloud-init** documentation section Fallback Network Configuration for more information.

Procedure

The following example adds a static networking configuration.

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.



NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add a **network-interfaces** section.

```
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
bootcmd:
  - ifdown eth0
  - ifup eth0
```

NOTE

You can disable a network configuration by adding the following information to your metadata.

```
network
  config: disabled
```

Additional resources

- Network Configuration
- NoCloud

3.10. CONFIGURING ONLY A ROOT USER WITH CLOUD-INIT

You can configure your user data so that you have a root user and no other users.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Create an entry for the user **root** in the **users** section.
   The simple example that follows includes a **users** section with only the **name** option.

```
users:
  - name: root
chpasswd:
  list: |
    root:password
  expire: False
```

3. Optionally, set up SSH keys for the root user.

```
users:
  - name: root
    ssh_pwauth: True
    ssh_authorized_keys:
      - ssh-rsa AA..vz user@domain.com
```

3.11. SETTING UP STORAGE WITH CONTAINER-STORAGE-SETUP IN CLOUD-INIT

You can set up storage by referencing the **container-storage-setup** utility within the **write_files** module.

Procedure

1. Depending upon the requirements of your datasource, open your user-data file for editing, or otherwise add the following directive to the **cloud.cfg.d** directory.

NOTE

All user directives include **#cloud-config** at the top of the file so that **cloud-init** recognizes the file as containing user directives. When you include directives in the **cloud.cfg.d** directory, name the file ***.cfg**, and always include **#cloud-config** at the top of the file.

2. Add or modify the **write_files** module to include the path to the **container-storage-setup** utility.
   The following example sets the size of the root logical volume to 6GB rather than the default 3GB.

```
write_files:
  - path: /etc/sysconfig/docker-storage-setup
    permissions: 0644
```

```
owner: root
content: |
ROOT_SIZE=6G
```

NOTE

Prior to RHEL 7.4, **container-storage-setup** was called **docker-storage-setup**. If you are using OverlayFS for storage, as of RHEL 7.4 you can now use that type of file system with SELinux in enforcing mode.

3.12. CHANGING THE SYSTEM LOCALE WITH CLOUD-INIT

You can configure the system location with the **locale** module.

Procedure

1. Depending upon the requirements of your datasource, open your meta-data file for editing, or otherwise add the following directive to the **cloud.cfg** file or the **cloud.cfg.d** directory.
2. Add the **locale** directive, specifying the location. The following sample sets the **locale** to **ja_JP** (Japan) with **UTF-8** encoding.

```
#cloud-config
locale: ja_JP.UTF-8
```

Additional resources

- Locale

3.13. CLOUD-INIT AND SHELL SCRIPTS

You can add list values or string values to **bootcmd** or **runcmd**. You can also provide a shell script within your userdata.

- If you use a list value for **bootcmd** or **runcmd**, each list item is run in turn using **execve**.
- If you use a string value, then the entire string is run as a shell script.
- If you want to use **cloud-init** to run a shell script, you can provide a shell script (complete with shebang (#!)) instead of providing **cloud-init** with a **.yaml** file.

Refer to Run commands on first boot for examples of how to put shell scripts in **bootcmd** and **runcmd**.

3.14. PREVENTING CLOUD-INIT FROM UPDATING CONFIG FILES

When you create or restore an instance from a backup image, the instance ID changes. The change in instance ID can cause **cloud-init** to update configuration files.

Perform the following procedure to ensure that **cloud-init** does not update certain configuration files when you create or restore from backup.

Procedure

1. Open the **/etc/cloud/cloud.cfg** file for editing.

2. Comment out or remove the configuration that you do not want **cloud-init** to update when you restore your instance.
   For example, to avoid updating the SSH key file, remove **-ssh** from the **cloud_init_modules** section.

```
cloud_init_modules:
 - disk_setup
 - migrator
 - bootcmd
 - write-files
 - growpart
 - resizefs
 - set_hostname
 - update_hostname
 - update_etc_hosts
 - rsyslog
 - users-groups
 # - ssh
```

Verification

You can check to see which configuration files **cloud-init** has updated. To do so, examine the **/var/log/cloud/cloud-init.log** file. Updated files are logged during instance startup with messages beginning with **Writing to**. For example:

```
2019-09-03 00:16:07,XXX - util.py[DEBUG]: Writing to /root/.ssh/authorized_keys - wb: [XXX] 554 bytes
2019-09-03 00:16:08,XXX - util.py[DEBUG]: Writing to /etc/ssh/sshd_config - wb: [XXX] 3905 bytes
```

3.15. MODIFYING A VM CREATED FROM A KVM GUEST IMAGE AFTER CLOUD-INIT HAS RUN

This section provides a sample procedure for when you want to modify your **cloud-init** configuration before rerunning **cloud-init**. When you launch a VM that includes the **cloud-init** package installed and enabled, **cloud-init** runs in its default state on that initial boot of your VM.

Procedure

1. Log in to your VM.

2. Add or change directives, for example, modify the **cloud.cfg** file in the **/etc/cloud** directory or add directives to the **/etc/cloud/cloud.cfg.d** directory.

3. Run the **cloud-init clean** command to clean directories so that **cloud-init** can rerun. You can also run the following commands as root to clean the VM.

```
`rm -Rf /var/lib/cloud/instances/*`
`rm -Rf /var/lib/cloud/instance`
`rm -Rf /var/lib/cloud/data/*`
```

NOTE

You can save the cleaned image as a new image and use that image for multiple VMs. The new VMs run **cloud-init** using your updated **cloud-init** configuration.

4. Rerun **cloud-init** or reboot the VM.
cloud-init reruns, implementing the configuration changes you made.

3.16. MODIFYING A VM FOR A SPECIFIC DATASOURCE AFTER CLOUD-INIT HAS RUN

This section provides a sample procedure for when you want to modify your **cloud-init** configuration before rerunning **cloud-init**. The following procedure uses OpenStack as an example. Note that the procedure varies based upon your datasource.

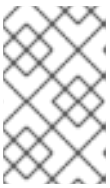
Procedure

1. Create and launch an instance for the OpenStack Platform. Refer to Virtual Machine Instances for information on creating instances for OpenStack. In this example, our virtual machine includes **cloud-init**, which runs upon boot of the virtual machine.

2. Add or change directives. For example, modify the **user-data.file** file that is stored on the OpenStack HTTP server.

3. Clean the virtual machine. Run the following commands as root.

```
`rm -rf /etc/resolv.conf /run/cloud-init`
`userdel -rf cloud-user`
`hostnamectl set-hostname localhost.localdomain`
`rm /etc/NetworkManager/conf.d/99-cloud-init.conf`
```



NOTE

You can save the cleaned image as a new image and use that image for multiple virtual machines. The new virtual machines run **cloud-init** using your updated **cloud-init** configuration.

4. Rerun **cloud-init** or reboot the virtual machine.
Cloud-init reruns, implementing the configuration changes you made.

3.17. TROUBLESHOOTING CLOUD-INIT

You can troubleshoot your instance after **cloud-init** has run by examining your configuration and log files. Once you have identified the issue, you can rerun **cloud-init** on your instance.

You can run **cloud-init** from the command line using the **cloud-init** command. To view the command syntax, along with a description of the optional arguments and subcommands, run the **cloud-init --help** command. The basic syntax follows.

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

The procedure that follows offers ideas for identifying issues with **cloud-init** and samples for rerunning the program.

Procedure

1. Review the **cloud-init** configuration files.
   a. Examine the **/etc/cloud/cloud.cfg** configuration file. Check which modules are included under **cloud_init_modules**, **cloud_config_modules**, and **cloud_final_modules**.
   b. Check directives (***.cfg** files) in the **/etc/cloud/cloud.cfg.d** directory.
2. Review the **/var/log/cloud-init.log** and **/var/log/cloud-init-output.log** files for details on a specific issue. For example, if the issue was that the root partition was not automatically extended, check log messages for **growpart**. If the file system was not extended, check log messages for **resizefs**. For example:

```
# grep resizefs /var/log/cloud-init.log
```

NOTE

growpart does not support LVM. If your root partition is based in LVM, the root partition is not automatically extended upon first boot.

3. Rerun **cloud-init**. Sample scenarios follow. Run commands as root.
   - Rerun **cloud-init** with only the init modules.

```
/usr/bin/cloud-init -d init
```

   - Rerun **cloud-init** with all modules in your configuration.

```
/usr/bin/cloud-init -d modules
```

   - Delete the **cloud-init** cache and force **cloud-init** to run after boot.

```
rm -rf /var/lib/cloud/* && /usr/bin/cloud-init -d init
```

   - Run the following commands to clean directories and simulate a clean instance.

```
rm -Rf /var/lib/cloud/instances/*
rm -Rf /var/lib/cloud/instance
rm -Rf /var/lib/cloud/data/*
reboot
```

   - Run the following commands to rerun **cloud-init**.

```
cloud-init init --local
cloud-init init
```

Additional resources

- CLI Interface