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: 2024-03-06

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 efficiently create multiple cloud instances of RHEL by using the cloud-init package. This allows for consistent and repeatable deployment of RHEL on a variety of cloud platforms. In the following chapters, you can learn more about: How cloud-init works How to use cloud-init to initiate cloud instances What uses of cloud-init Red Hat supports

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 feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCING RHEL ON PUBLIC CLOUD PLATFORMS

Public cloud platforms provide computing resources as a service. Instead of using on-premises hardware, you can run your IT workloads, including Red Hat Enterprise Linux (RHEL) systems, as public cloud instances.

1.1. BENEFITS OF USING RHEL IN A PUBLIC CLOUD

RHEL as a cloud instance located on a public cloud platform has the following benefits over RHEL on-premises physical systems or virtual machines (VMs):

- **Flexible and fine-grained allocation of resources**
  A cloud instance of RHEL runs as a VM on a cloud platform, which typically means a cluster of remote servers maintained by the provider of the cloud service. Therefore, allocating hardware resources to the instance, such as a specific type of CPU or storage, happens on the software level and is easily customizable.

  In comparison to a local RHEL system, you are also not limited by the capabilities of your physical host. Instead, you can choose from a variety of features, based on selection offered by the cloud provider.

- **Space and cost efficiency**
  You do not need to own any on-premises servers to host your cloud workloads. This avoids the space, power, and maintenance requirements associated with physical hardware.

  Instead, on public cloud platforms, you pay the cloud provider directly for using a cloud instance. The cost is typically based on the hardware allocated to the instance and the time you spend using it. Therefore, you can optimize your costs based on your requirements.

- **Software-controlled configurations**
  The entire configuration of a cloud instance is saved as data on the cloud platform, and is controlled by software. Therefore, you can easily create, remove, clone, or migrate the instance. A cloud instance is also operated remotely in a cloud provider console and is connected to remote storage by default.

  In addition, you can back up the current state of a cloud instance as a snapshot at any time. Afterwards, you can load the snapshot to restore the instance to the saved state.

- **Separation from the host and software compatibility**
  Similarly to a local VM, the RHEL guest operating system on a cloud instance runs on a virtualized kernel. This kernel is separate from the host operating system and from the *client* system that you use to connect to the instance.

  Therefore, any operating system can be installed on the cloud instance. This means that on a RHEL public cloud instance, you can run RHEL-specific applications that cannot be used on your local operating system.

  In addition, even if the operating system of the instance becomes unstable or is compromised, your client system is not affected in any way.

Additional resources

- What is public cloud?

- What is a hyperscaler?
- Types of cloud computing
- Public cloud use cases for RHEL
- Obtaining RHEL for public cloud deployments

1.2. PUBLIC CLOUD USE CASES FOR RHEL

Deploying on a public cloud provides many benefits, but might not be the most efficient solution in every scenario. If you are evaluating whether to migrate your RHEL deployments to the public cloud, consider whether your use case will benefit from the advantages of the public cloud.

Beneficial use cases

- Deploying public cloud instances is very effective for flexibly increasing and decreasing the active computing power of your deployments, also known as *scaling up* and *scaling down*. Therefore, using RHEL on public cloud is recommended in the following scenarios:
  - Clusters with high peak workloads and low general performance requirements. Scaling up and down based on your demands can be highly efficient in terms of resource costs.
  - Quickly setting up or expanding your clusters. This avoids high upfront costs of setting up local servers.
- Cloud instances are not affected by what happens in your local environment. Therefore, you can use them for backup and disaster recovery.

Potentially problematic use cases

- You are running an existing environment that cannot be adjusted. Customizing a cloud instance to fit the specific needs of an existing deployment may not be cost-effective in comparison with your current host platform.
- You are operating with a hard limit on your budget. Maintaining your deployment in a local data center typically provides less flexibility but more control over the maximum resource costs than the public cloud does.

Next steps

- Obtaining RHEL for public cloud deployments

Additional resources

- Should I migrate my application to the cloud? Here's how to decide.

1.3. FREQUENT CONCERNS WHEN MIGRATING TO A PUBLIC CLOUD

Moving your RHEL workloads from a local environment to a public cloud platform might raise concerns about the changes involved. The following are the most commonly asked questions.

Will my RHEL work differently as a cloud instance than as a local virtual machine?

In most respects, RHEL instances on a public cloud platform work the same as RHEL virtual machines on a local host, such as an on-premises server. Notable exceptions include:

- Instead of private orchestration interfaces, public cloud instances use provider-specific console interfaces for managing your cloud resources.
- Certain features, such as nested virtualization, may not work correctly. If a specific feature is critical for your deployment, check the feature's compatibility in advance with your chosen public cloud provider.

Will my data stay safe in a public cloud as opposed to a local server?

The data in your RHEL cloud instances is in your ownership, and your public cloud provider does not have any access to it. In addition, major cloud providers support data encryption in transit, which improves the security of data when migrating your virtual machines to the public cloud.

The general security of your RHEL public cloud instances is managed as follows:

- Your public cloud provider is responsible for the security of the cloud hypervisor
- Red Hat provides the security features of the RHEL guest operating systems in your instances
- You manage the specific security settings and practices in your cloud infrastructure

What effect does my geographic region have on the functionality of RHEL public cloud instances?

You can use RHEL instances on a public cloud platform regardless of your geographical location. Therefore, you can run your instances in the same region as your on-premises server.

However, hosting your instances in a physically distant region might cause high latency when operating them. In addition, depending on the public cloud provider, certain regions may provide additional features or be more cost-efficient. Before creating your RHEL instances, review the properties of the hosting regions available for your chosen cloud provider.

1.4. OBTAINING RHEL FOR PUBLIC CLOUD DEPLOYMENTS

To deploy a RHEL system in a public cloud environment, you need to:

1. Select the optimal cloud provider for your use case, based on your requirements and the current offer on the market.
   The cloud providers currently certified for running RHEL instances are:
   - Amazon Web Services (AWS)
     - For more information, see Deploying RHEL 9 on Amazon Web Services .
   - Google Cloud Platform (GCP)
     - For more information, see Deploying RHEL 9 on Google Cloud Platform .
   - Microsoft Azure
     - For more information, see Deploying RHEL 9 on Microsoft Azure .
2. Create a RHEL cloud instance on your chosen cloud platform. For more information, see Methods for creating RHEL cloud instances .
3. To keep your RHEL deployment up-to-date, use Red Hat Update Infrastructure (RHUI).

Additional resources

- RHUI documentation
- Red Hat Open Hybrid Cloud

1.5. METHODS FOR CREATING RHEL CLOUD INSTANCES

To deploy a RHEL instance on a public cloud platform, you can use one of the following methods:

Create a system image of RHEL and import it to the cloud platform.

- To create the system image, you can use the RHEL image builder or you can build the image manually.
- This method uses your existing RHEL subscription, and is also referred to as *bring your own subscription* (BYOS).
- You pre-pay a yearly subscription, and you can use your Red Hat customer discount.
- Your customer service is provided by Red Hat.
- For creating multiple images effectively, you can use the **cloud-init** tool.

Purchase a RHEL instance directly from the cloud provider marketplace.

- You post-pay an hourly rate for using the service. Therefore, this method is also referred to as *pay as you go* (PAYG).
- Your customer service is provided by the cloud platform provider.

Additional resources

- What is a golden image?

CHAPTER 2. INTRODUCTION TO CLOUD-INIT

The **cloud-init** utility automates the initialization of cloud instances during system boot. You can configure **cloud-init** to perform a variety of tasks:

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

Prerequisites

- Sign up for a Red Hat Customer Portal account.

The **cloud-init** is available in various types of RHEL images. For example:

- If you download a KVM guest image from the Red Hat Customer Portal , the image comes preinstalled with the **cloud-init** package. After you launch the instance, the **cloud-init** package becomes enabled. KVM guest images on the Red Hat Customer Portal are intended to use with Red Hat Virtualization (RHV), the Red Hat OpenStack Platform (RHOSP), and Red Hat OpenShift Virtualization.
- You can also download the RHEL ISO image from the Red Hat Customer Portal to create a custom guest image. In this case, you need to install the **cloud-init** package on the customized guest image.
- If you require to use an image from a cloud service provider (for example, AWS or Azure), use the *RHEL image builder* to create the image. Image builder images are customized for specific cloud providers. The following image types include **cloud-init** already installed:
  - Amazon Machine Image (AMI)
  - Virtual Hard Disk (VHD)
  - QEMU copy-on-write (qcow2)
    For details about the RHEL image builder, see Composing a customized RHEL system image.

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

In addition, you can configure **cloud-init** on one VM and then use that VM as a template to create additional VMs or clusters of VMs.

Specific Red Hat products, for example, Red Hat Virtualization, have documented procedures to configure **cloud-init** for those products.

2.1. OVERVIEW OF THE CLOUD-INIT CONFIGURATION

The **cloud-init** utility uses YAML-formatted configuration files to apply user-defined tasks to instances. When an instance boots, the **cloud-init** service starts and executes the instructions from the YAML file. Depending on the configuration, tasks complete either during the first boot or on subsequent boots of the VM.

To define the specific tasks, configure the **/etc/cloud/cloud.cfg** file and add directives under the **/etc/cloud/cloud.cfg.d/** directory.

- The **cloud.cfg** file includes directives for various system configurations, such as user access, authentication, and system information.
  The file also includes default and optional modules for **cloud-init**. These modules execute in order in the following phases: .. The **cloud-init** initialization phase .. The configuration phase .. The final phase.

  + In the **cloud.cfg** file, the modules for the three phases are listed under **cloud_init_modules**, **cloud_config_modules**, and **cloud_final_modules** respectively.

- You can add additional directives for **cloud-init** in the **cloud.cfg.d** directory. When adding directives to the **cloud.cfg.d** directory, you need to add them to a custom file named ***.cfg**, and always include **#cloud-config** at the top of the file.

2.2. CLOUD-INIT OPERATES IN STAGES

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

1. **Generator stage**: By using the **systemd** service, this phase determines whether to run **cloud-init** utility at the time of boot.

2. **Local stage**: **cloud-init** searches local datasources and applies network configuration, including the DHCP-based fallback mechanism.

3. **Network stage**: **cloud-init** processes user data by running modules listed under **cloud_init_modules** in the **/etc/cloud/cloud.cfg** file. You can add, remove, enable, or disable modules in the **cloud_init_modules** section.

4. **Config stage**: **cloud-init** runs modules listed under **cloud_config_modules** section in the **/etc/cloud/cloud.cfg** file. You can add, remove, enable, or disable modules in the **cloud_config_modules** section.

5. **Final stage**: **cloud-init** runs modules and configurations included in the **cloud_final_modules** section of the **/etc/cloud/cloud.cfg** file. It can include the installation of specific packages, as well as triggering configuration management plug-ins and user-defined scripts. You can add, remove, enable, or disable modules in the **cloud_final_modules** section.

Additional resources

- Boot Stages of cloud-init

2.3. CLOUD-INIT MODULES EXECUTE IN PHASES

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

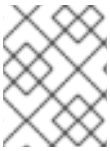
1. The network phase (**cloud_init_modules**)

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.

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

The datasources that **cloud-init** consumes are 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 about 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 about vendor data.

2.5. 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
- How to identify the datasource I'm using
- How can I debug my user data?

2.6. ADDITIONAL RESOURCES

- Upstream documentation for cloud-init

CHAPTER 3. RED HAT SUPPORT FOR CLOUD-INIT

Red Hat supports the **cloud-init** utility, **cloud-init** modules, and default directories and files across various Red Hat products.

3.1. CLOUD-INIT SIGNIFICANT DIRECTORIES AND FILES

By using directories and files in the following table, you can perform tasks such as:

- Configuring **cloud-init**
- Finding information about 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 3.1. 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 about your datasource and the **ds-identify** script.

3.2. RED HAT PRODUCTS THAT USE CLOUD-INIT

You can use **cloud-init** with these 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 about 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 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.

3.3. RED HAT SUPPORTS THESE CLOUD-INIT MODULES

Red Hat supports most **cloud-init** modules. Individual modules can contain multiple configuration options. In the following table, you can find 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 3.2. 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
growpart	Resizes partitions to fill the available disk space	per always

cloud-init Module	Description	Default Module Frequency
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
resolv_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
runcmd	Runs arbitrary commands	per instance

cloud-init Module	Description	Default Module Frequency
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 list of modules is **not** supported by Red Hat:

Table 3.3. Modules not supported

Module
apt_configure
apt_pipeline
byobu
chef
emit_upstart
grub_dpkg
ubuntu_init_switch

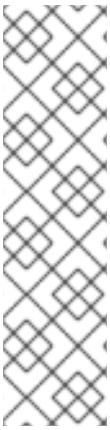
3.4. THE DEFAULT CLOUD.CFG FILE

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 about 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

- Modules

3.5. 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

- Cloud config examples

3.6. 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

3.7. 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 4. CONFIGURING CLOUD-INIT

By using **cloud-init**, you can perform a variety of configuration tasks.

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.

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

To create a new virtual machine (VM) that includes **cloud-init**, create a **meta-data** file and a **user-data** file.

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

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

Procedure

1. Create a directory named **cloudinitiso** and set is as your working directory:

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

2. Create the **meta-data** file and add the following information:

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

3. Create the **user-data** file and add the following information:

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

NOTE

The last 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** utility and attach the downloaded image to the existing 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 with username **cloud-user** and password **cilogon**:

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

Verification

- Check the **cloud-init** status to confirm that the utility has completed its defined tasks:

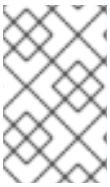
```
[cloud-user@citest-1 instance]$ cloud-init status
status: done
```

- The **cloud-init** utility 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
```



NOTE

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

Additional resources

- Upstream documentation for the NoCloud data source

4.2. EXPIRING A CLOUD USER PASSWORD WITH CLOUD-INIT

To force **cloud-user** to change the **cloud-user** password at the first login, you can set their password as expired.

Procedure

1. Depending on the requirements of your datasource, edit the **user-data** file or 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.

4.3. CHANGING A DEFAULT USER NAME WITH CLOUD-INIT

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

Procedure

1. Depending on the requirements of your datasource, edit the **user-data** file or 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:

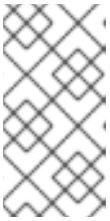
```
#cloud-config
user: username
password: mypassword
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...SDvz user1@yourdomain.com
  - ssh-rsa AAB...QTuo user2@yourdomain.com
```

4.4. SETTING A ROOT PASSWORD WITH CLOUD-INIT

To set the root password, create a user list.

Procedure

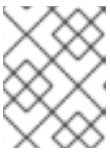
1. Depending on the requirements of your datasource, edit the **user-data** file or add the following directive to the **cloud.cfg.d** directory:



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:



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.

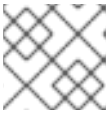
4.5. MANAGING RED HAT SUBSCRIPTIONS WITH CLOUD-INIT

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

Example 1

- You can use 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.

Example 2

- You can use the **activation-key** and **org** options:
  Under **rh_subscription**, add your **activation key** and **org** number and set **auto-attach** to **True**.

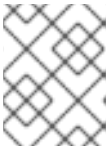
```
rh_subscription:
  activation-key: example_key
  org: 12345
  auto-attach: True
```

Example 3

- You can add 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.

Example 4

- You can set 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
```

4.6. 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 on the requirements of your datasource, edit the **user-data** file or add the following directive to the **cloud.cfg.d** directory:

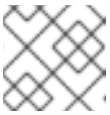
NOTE

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**.

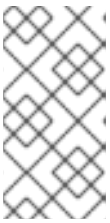
4.7. RUNNING FIRST BOOT COMMANDS WITH CLOUD-INIT

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 on the requirements of your datasource, edit the **user-data** file or 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
```

4.8. 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 on the requirements of your datasource, edit the **user-data** file or 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
```

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

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

Red Hat Enterprise Linux provides its default networking service through **NetworkManager**, a dynamic network control and configuration daemon that keeps network devices and connections up and active when they are available.

Your datasource might provide a network configuration. For details, see the **cloud-init** section Network Configuration Sources.

If you do not specify 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.

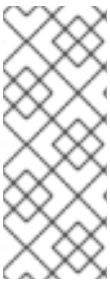
1. Depending on the requirements of your datasource, edit the **user-data** file or add the following directive to the **cloud.cfg.d** directory:

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:
  version: 1
  config:
    - type: physical
      name: eth0
      subnets:
        - type: static
          address: 192.0.2.1/24
          gateway: 192.0.2.254
```



NOTE

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

```
network:
  config: disabled
```

Additional resources

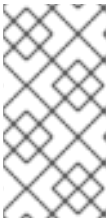
- Network Configuration
- NoCloud

4.10. 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 on the requirements of your datasource, edit the **user-data** file or 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
```

4.11. 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 on the requirements of your datasource, edit the **user-data** file or add the following directive to the **cloud.cfg.d** directory:

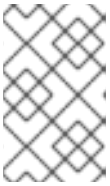
NOTE

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 6 GB rather than the default 3 GB.

```
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.

4.12. CHANGING THE SYSTEM LOCALE WITH CLOUD-INIT

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

Procedure

1. Depending on the requirements of your datasource, edit the **meta-data** file. You can also 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

- Set system locale

4.13. 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 userdata.

- If you use a list value for **bootcmd** or **runcmd**, each list item runs in turn using **execve**.
- If you use a string value, then the entire string runs 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**.

4.14. PREVENTING CLOUD-INIT FROM UPDATING CONFIG FILES

When you create or restore an instance from a backup image, the instance ID changes. With the change in the instance ID, the **cloud-init** utility updates configuration files. However, you can ensure that **cloud-init** does not update certain configuration files when you create or restore from backup.

Procedure

1. Edit the **/etc/cloud/cloud.cfg** file, for example:

```
# vi /etc/cloud/cloud.cfg
```

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

- To check the configuration files updated by **cloud-init**, 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
```

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

You can modify your **cloud-init** configuration before rerunning the **cloud-init** utility. When you launch a VM with the **cloud-init** package installed and enabled, **cloud-init** runs in its default state on the initial boot of the VM.

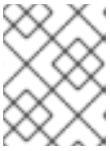
Procedure

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 will use updated **cloud-init** configuration to run **cloud-init**.

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

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

You can modify your **cloud-init** configuration before rerunning **cloud-init**. This procedure uses OpenStack as an example datasource. Note that the exact steps you need to perform vary based on your datasource.

Procedure

1. Create and launch an instance for the OpenStack Platform. For information about creating instances for OpenStack, see Creating an instance. In this example, the virtual machine (VM) includes **cloud-init**, which runs upon boot of the VM.

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.

4.17. TROUBLESHOOTING CLOUD-INIT

After running the **cloud-init** utility, you can troubleshoot the instance by examining the configuration and log files. After identifying the issue, rerun **cloud-init** on your instance. You can run **cloud-init** from the command line. For details, run the **cloud-init --help** command.

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 root partition was not automatically extended, check log messages for the **growpart** utility. 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** commands as root:

a. Rerun **cloud-init** with only the init modules:

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

b. Rerun **cloud-init** with all modules in the configuration:

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

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

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

d. 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
```

e. Rerun the **cloud-init** utility:

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

Additional resources

- The cloud-init cli commands