# Red Hat Enterprise Linux 10

## Configuring and managing cloud-init for RHEL

Using cloud-init to automate the initialization of cloud instances

Last Updated: 2025-05-22

# Red Hat Enterprise Linux 10 Configuring and managing cloud-init for RHEL

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 working of cloud-init, use cloud-init to initiate cloud instances, and Red Hat supported cloud-init use cases.

# Table of Contents

# 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. 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.

## 1.2. 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 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.

## 1.3. 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 certified cloud service providers (CCSP) for running RHEL instances are:
   - Amazon Web Services (AWS)
   - Google Cloud Platform (GCP)
   - Microsoft Azure
2. Create a RHEL cloud instance on your chosen cloud platform. For details, 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
- Red Hat Ecosystem Cataglogue

## 1.4. 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 RHEL system image 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.

NOTE

For detailed instructions on using various methods to deploy RHEL instances see the following chapters in this document.

**Additional resources**

- What is a golden image?

# CHAPTER 2. INTRODUCTION TO CLOUD-INIT

The **cloud-init** utility automates the initialization and configuration of virtual machines (VM), also known as instances, during the first boot. You can configure **cloud-init** to perform tasks such as:

- Configuring a host name, network interfaces, and user accounts
- Installing packages on an instance
- Running scripts
- Modifying default virtual machine (VM) actions

**Prerequisites**

- You have created a Red Hat account .

**cloud-init** is available for various types of RHEL images as follows:

- A KVM **qcow2** guest image comes preinstalled with **cloud-init**. After you launch the instance, the **cloud-init** utility is automatically enabled. You can use KVM guest images from the Red Hat Customer Portal with Red Hat Virtualization (RHV), the Red Hat OpenStack Platform (RHOSP), and Red Hat OpenShift Virtualization.
- If you download a Red Hat ISO image from the Red Hat Customer Portal to create a custom guest image, you need to install the **cloud-init** package on that customized guest image.
- If you use an image from a certified cloud service provider (CCSP), for example, AWS, GCP, or Azure, use the *RHEL image builder* to create the image. Image builder manages customized images for specific cloud providers. The following image formats are preinstalled with **cloud-init**:
  - Amazon Machine Image (AMI)
  - Virtual Hard Disk (VHD)
  - QEMU copy-on-write (qcow2)

Most cloud platforms support **cloud-init**, but configuration procedures and supported options are varied.

Alternatively, you can configure **cloud-init** for the **NoCloud** environment. By using **NoCloud**, you can manage cloud instances for both a local configuration (without network access) and configurations fetched from a remote server. Additionally, you can create a VM template by configuring **cloud-init** on one VM. By using this template, you can create additional VMs or clusters of VMs.

## 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 initiates and executes the instructions from the YAML file. These tasks may complete during the first boot or on subsequent boots of the VM, depending on the configuration.

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. This file also includes default and optional modules for **cloud-init**. The default module groups are as follows:
  - **cloud_init_modules**
  - **cloud_config_modules**
  - **cloud_final_modules**
- You can include additional directives for **cloud-init** in the **cloud.cfg.d** directory.
- While adding directives to the **cloud.cfg.d** directory, add them to a custom file named ***.cfg**, and always include **#cloud-config** at the top of the file.

For details, refer to an example file of **cloud.cfg**. See an example of cloud.cfg file.

## 2.2. DATASOURCE TYPES OF CLOUD-INIT

Datasources are the sources of configuration data for **cloud-init** in the form of user data and metadata. Metadata has the configuration drive created by certified cloud service provider (CCSP). **cloud-init** uses the user data and metadata for configuration of datasource, and optionally vendor data. In case if you need to verify vendor data is available or not:

```
# grep -i "vendor data" /var/log/cloud-init.log
```

For datasources, there are three types of configuration data: user data, metadata, and vendor data.

- User data includes directives specified in the **cloud.cfg** file and 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 CCSP determines where your instance can find user data and metadata. After adding user data and metadata to an HTTP service. In this case, when **cloud-init** runs, it consumes user data and metadata 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.

By default, **cloud-init** automatically identifies the existing datasource. **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 custom 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. After **cloud-init** runs, you can view a log file (**run/cloud-init/ds-identify.log**) that provides detailed information about the platform. For details on **datasource_list**, check custom datasources.

For details on configuring datasources for CCSP, see:

- Amazon EC2

- Azure
- Google Compute Engine

**Additional resources**

- Datasources
- How to identify the datasource in use
- Instance Metadata
- Datasources
- Vendor Data
- How can I debug my user data?
- Config drive

## 2.3. BOOT STAGES OF CLOUD-INIT

The **cloud-init** utility completes VM configuration in the following 5 boot stages. Phase is the logical grouping of the boot stages that controls the sequence of execution of modules. The default modules execute in the specific phase.

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. Each stage also belongs to a module execution *phase*, which controls what modules are run. The stages are as follows:

1. **Detect stage**: By using the **systemd** service, this stage determines whether to run **cloud-init** utility at the time of boot. The **ds-identify** tool detects whether or not to run **cloud-init** in the absence of a valid platform.
   This stage belongs to the *Initialization* phase and does not execute any modules. Instead, it identifies the datasource for **cloud-init** configuration.

2. **Local stage**: **cloud-init** searches local datasources and applies network configuration, including the DHCP-based fallback mechanism.
   This stage belongs to the *Initialization* phase and does not execute any modules.

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.
   This stage belongs to the *Networking* module execution stage.

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.
   This stage belongs to the *Configuration* module execution phase.

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.
   This stage is equivalent to the *Finalization* module execution phase.

During the first boot of VM, when the **cloud-init** service initiates, all the configured modules execute in their respective phases. On subsequent booting, whether a module executes within a phase depends on the *module frequency* of that module. Module frequency is set in the **cloud-init** configuration, and determines whether a module executes every time **cloud-init** runs on the instance, or only 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. For details, see Scename definition for module creation.

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.
- **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.

**Additional resources**

- Boot Stages of cloud-init
- Base configuration

## 2.4. FILES AND DIRECTORIES SIGNIFICANT FOR CLOUD-INIT

Red Hat supports the **cloud-init** utility, **cloud-init** modules, and default directories and files. By using directories and files, you can perform tasks such as:

- Configuring **cloud-init**
- Finding details about configuration after **cloud-init** has run
- Examining log files
- Finding templates

Depending on your requirement 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 about your datasource and the **ds-identify** script. |

### 2.4.1. The default cloud.cfg file

The **/etc/cloud/cloud.cfg** file lists the modules included in the basic configuration for **cloud-init**. These modules are the default modules for **cloud-init**. You can configure or remove modules based on your requirements.

- To perform actions during one of the **cloud-init** phases, you need to configure each module individually and list them in the **cloud.cfg** file. Modules run in the order given in **cloud.cfg**. You typically do not change this order. However, you can add additional modules to **cloud.cfg**, if Red Hat supports the modules that you want to add.
- 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.
- To ensure optimal functionality of the configuration, use module names with underscores (_) rather than dashes (**-**).
- The default contents of the file for Red Hat Enterprise Linux are as follows:

```
users: 1
 - default
```

```
disable_root: true (2)
resize_rootfs_tmp: /dev
ssh_pwauth:   false (3)

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

cloud_init_modules: (9)
 - migrator
 - seed_random
 - bootcmd
 - write_files
 - growpart
 - resizefs
 - disk_setup
 - mounts
 - set_hostname
 - update_hostname
 - update_etc_hosts
 - ca_certs
 - rsyslog
 - users_groups
 - ssh

cloud_config_modules: (10)
 - ssh_import_id
 - locale
 - set_passwords
 - rh_subscription
 - spacewalk
 - yum_add_repo
 - ntp
 - timezone
 - disable_ec2_metadata
 - runcmd

cloud_final_modules: (11)
 - package_update_upgrade_install
 - write_files_deferred
 - puppet
 - chef
 - ansible
 - mcollective
 - salt_minion
 - reset_rmc
 - rightscale_userdata
 - scripts_vendor
 - scripts_per_once
 - scripts_per_boot
 - scripts_per_instance
 - scripts_user
```

```
 - ssh_authkey_fingerprints
 - keys_to_console
 - install_hotplug
 - phone_home
 - final_message
 - power_state_change

system_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)
  network:
    renderers: ['sysconfig', 'eni', 'netplan', 'network-manager', 'networkd']
  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

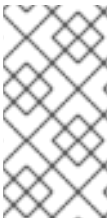
14 Specifies the main directory that contains **cloud-init**-specific subdirectories. Refer to Directory layout for more information.

15 Specifies where templates reside

16 The name of the SSH service

### 2.4.2. The default 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.

Refer to User-Data Formats for details on how to add a user script as ***.cfg** file.

### 2.4.3. The default 05_logging.cfg file

The **05_logging.cfg** file sets logging information and the default logging configuration file for **cloud-init**. The **/etc/cloud/cloud.cfg.d** directory includes this file, along with other **cloud-init** directives. The default contents of the file for 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'}
```

### 2.4.4. The /var/lib/cloud directory layout

When **cloud-init** initiates, it creates a directory layout with instance details and **cloud-init** configuration. This 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**

- Modules
- upstream example of the **cloud.cfg** file
- Cloud config examples
- Logging
- Directory layout

# CHAPTER 3. RED HAT SUPPORT FOR CLOUD-INIT

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

## 3.1. RED HAT PRODUCTS THAT USE CLOUD-INIT

You can use **cloud-init** with these Red Hat products:

- **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.

## 3.2. SUPPORTED 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.

**Table 3.1. 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 |
| salt_minion | Installs, configures, and starts **salt minion** | per instance |

| cloud-init Module | Description | Default Module Frequency |
|---|---|---|
| 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 |

Along with supported modules, refer to Modules in the **cloud-init** Documentation section for complete descriptions and options.

## 3.3. CLOUD-INIT MODULES NOT SUPPORTED BY RED HAT

The following list of modules is **not** supported by Red Hat, and using them in your **cloud-init** configuration is highly discouraged.

**Table 3.2. Modules not supported**

| Module |
| --- |
| apt_configure |
| apt_pipeline |
| byobu |
| chef |
| emit_upstart |
| grub_dpkg |
| ubuntu_init_switch |

# CHAPTER 4. CREATING A VIRTUAL MACHINE WITH CLOUD-INIT

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 **cloudinitiso** directory and set it as your working directory:

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

2. Edit the **meta-data** file:

```
$ vi meta-data
instance-id: citest
local-hostname: citest-1
```

3. Edit the **user-data** file:

```
$ vi user-data
#cloud-config
password: cilogon
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - <ssh-rsa AAA...fhHQ== sample@example.com>
```

NOTE

You can find your SSH public keys in the **~/.ssh/id_rsa.pub** file.

4. 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 rhel10.0 \
    --virt-type kvm \
    --graphics none \
    --import
```

7. Log on to your image with the default username **cloud-user** and default password **cilogon**:

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

### Verification

- Check the status of the **cloud-init** service 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.

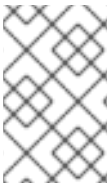
```
$ cd /var/lib/cloud/instance
$ cat datasource

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
- user-data
- meta-data

# CHAPTER 5. RUNNING FIRST-BOOT COMMANDS BY USING CLOUD-INIT

To execute commands during the first start-up and initialization of a VM, you can use the **runcmd** and **bootcmd** sections of the **cloud-init** configuration.

- 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.

### Prerequisites

- 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.

### Procedure

1. 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
```

# CHAPTER 6. RERUNNING CLOUD-INIT ON A VIRTUAL MACHINE

When additional configurations, such as modification in directives, modules, or datasources, are required for a virtual machine (VM) configured by the **cloud-init** service, you can rerun **cloud-init**.

**WARNING**

Rerunning the **cloud-init** process can result in data loss and highly discouraged for a production environment. Credentials such as SSH keys and password could be lost or replaced.

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

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. If you want to change the setting of the VM, you can modify your **cloud-init** configuration and rerun **cloud-init**.

**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.
4. Run the following commands as **root** to clean the VM data:

```
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 template image and use that image for multiple VMs. The new VMs will use the updated configuration to run **cloud-init**.

5. Rerun **cloud-init** or reboot the VM to implement the configuration changes you made. For details on re-running **cloud-init**, see Re-run cloud-init.

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

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

```
# 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 template 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 VM to implement the configuration changes you made. For details on re-running **cloud-init**, see Re-run cloud-init.

# CHAPTER 7. CONFIGURING AUTHENTICATION BY USING CLOUD-INIT

You can use the **cloud-init** utility to manage users, access rights, and passwords. Specifically, you can set up **cloud-init** to do any of the following in a VM:

- Create and describe users in a **users** section. If you add the **users** section, you must also set the default user options in that section. You can modify the section to add more users to the initial system configuration, and also set additional user options.
- Configure a user as a sudoer by adding a **sudo** and **groups** entry to the **users** section.
- Configure the user data so that only you have a **root** user access.
- Force **cloud-user** to change the **cloud-user** password at the first login to reset the password.
- Set the **root** password by creating a user list.

**Prerequisites**

- 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.

**Procedure**

- To **add users and user options**
  - Add or modify the **users** section to add users. For example:

```
#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
```

- 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.

NOTE

This example places the user **user2** into two groups: **users** and **wheel**.

- To **add a sudo user to the users list**
  - Add a **sudo** entry and specify the user access. For example, **sudo: ALL=(ALL) NOPASSWD:ALL** allows a user unrestricted user access.
  - 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
```

- To **configure an exclusive root access for a user**
  - Create an entry for the user **root** in the **users** section by modifying the **name** option:

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

  - Optional: Set up SSH keys for the **root** user:

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

- To change the default **cloud-init** user name, follow:
  - Add the line **user: <username>**, replacing <username> with the new default user name:

- 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
```

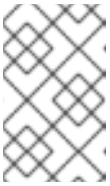
- To **reset a password for a new user**
  - Change the line **chpasswd: {expire: False}** to **chpasswd: {expire: True}**:

```
#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
```

NOTE

- This works to expire the password because **password** and **chpasswd** operate on the default user unless you indicate otherwise.
- This is a global setting. When you set **chpasswd** to **True**, all users you create need to change their passwords when they log in.

- To **set a root password**
  - Create a user list in the **chpasswd** section:



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.

# CHAPTER 8. 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
```

# CHAPTER 9. SETTING UP A STATIC NETWORKING CONFIGURATION BY USING 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.

+

NOTE

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 an existing 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.

### Prerequisites

- 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.

### Procedure

- Add a **network-interfaces** section. For example:

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

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

```
network:
  config: disabled
```

### Additional resources

- Network Configuration
- NoCloud

# CHAPTER 10. SETTING UP CONTAINER STORAGE BY USING CLOUD-INIT

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

### Prerequisites

- 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.

### Procedure

- Add or modify the **write_files** module to include the path to the **container-storage-setup** utility:

```
write_files:
  - path: /etc/sysconfig/docker-storage-setup
    permissions: 0644
    owner: root
    content: |
    ROOT_SIZE=6G
```

This example sets the size of the root logical volume to 6 GB rather than the default 3 GB.

# CHAPTER 11. USING 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 string value, the entire string runs as a shell script.
- If you use a list value for **bootcmd** or **runcmd**, each list item runs in turn using **execve**.
- If you want to use **cloud-init** to run a shell script, 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**.

# CHAPTER 12. CHANGING THE SYSTEM LOCALE WITH CLOUD-INIT

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

### Prerequisites

- 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:

### Procedure

- 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

# CHAPTER 13. PREVENTING CLOUD-INIT FROM UPDATING CONFIG FILES

When you create or restore an instance from a backup image, the instance ID changes. With an updated 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:

```
# 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**:

```
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
```

# CHAPTER 14. 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.

- For general configuration issues, 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.
- If the root partition was not automatically extended, check log messages for the **growpart** utility.

NOTE

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

  - Review the **/var/log/cloud-init.log** and **/var/log/cloud-init-output.log** files for details on any specific issue.
- If the file system was not extended, check log messages for **resizefs**

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

IMPORTANT

Rerun **cloud-init** commands as root.

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

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

- Rerun **cloud-init** with all modules in the 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
```

- 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
```

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

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

**Additional resources**

- How to debug cloud-init
- The cloud-init cli commands
- How can I simulate/debug cloud-init (Red Hat Knowledgebase)