Red Hat Enterprise Linux 10

Configuring and managing cloud-init for RHEL

Using cloud-init to automate the initialization and configuration of RHEL instances

Last Updated: 2026-01-21

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

Using cloud-init to automate the initialization and configuration of RHEL instances

Legal Notice

Abstract

By using the cloud-init package, you can efficiently create multiple RHEL cloud instances with consistent and repeatable deployments across various cloud platforms. The content explains how cloud-init works, how to use it to launch cloud instances, and which cloud-init use cases Red Hat supports.

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 offer computing resources as a service. Instead of using on-premise 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

Running Red Hat Enterprise Linux (RHEL) on public cloud platforms provides flexible resource allocation, cost efficiency, and software-controlled configurations to optimize your infrastructure without managing physical hardware.

RHEL as a cloud instance located on a public cloud platform has the following benefits over RHEL on-premise 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 means a cluster of remote servers maintained by the cloud service provider. Therefore, on the software level, allocating hardware resources to the instance is easily customizable, such as a specific type of CPU or storage.
In comparison to a local RHEL system, you are also not limited by the capabilities of physical host. Instead, you can select from a variety of features, based on selections offered by the cloud provider.

Space and cost efficiency

You do not need to own any on-premise servers to host 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 to use it. Therefore, you can optimize your costs based on the requirements.

Software-controlled configurations

You save the entire configuration of a cloud instance as data on the cloud platform and control it with software. Therefore, you can easily create, remove, clone, or migrate the instance. You also operate a cloud instance remotely in a cloud provider console, and it connects 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 Kernel-based Virtual Machine (KVM). This kernel is separate from the host operating system and from the *client* system that you use to connect to the instance.
Therefore, you can install any operating system on the cloud instance. This means that on a RHEL public cloud instance, you can run RHEL-specific applications not usable on your local operating system.

In addition, even if the operating system of the instance becomes unstable or compromised, it does not affect your client system.

Additional resources

- What is public cloud?

- What is a hyperscaler?
- Types of cloud computing

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 Red Hat Enterprise Linux (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. This is also known as *scaling up* and *scaling down*. You can use RHEL on public cloud 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.
- What happens in your local environment does not affect cloud instances. Therefore, you can use them for backup and disaster recovery.

Potentially problematic use cases

- You are running an existing environment that you cannot adjust. Customizing a cloud instance to fit the specific needs of an existing deployment might not be economically efficient 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.

For details on how to obtain RHEL for public cloud deployments, see 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-premise 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, might 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.
In terms of security of RHEL public cloud instances, the following applies:

- 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 might offer 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 Red Hat Enterprise Linux (RHEL) in a public cloud environment, you must select a certified cloud service provider and create a RHEL cloud instance.

- Based on your requirements and the current offerings in the market, select the optimal cloud service provider for your use case. The certified cloud service providers for running RHEL instances are:
  - Amazon Web Services (AWS)
  - Google Cloud
  - Microsoft Azure
- Create a RHEL cloud instance on your cloud platform. For details, see Methods for creating RHEL cloud instances.
- 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 Catalog

1.5. METHODS FOR CREATING RHEL CLOUD INSTANCES

You can create a Red Hat Enterprise Linux (RHEL) system image and import it to the cloud platform by using either RHEL image builder or purchasing a RHEL image directly from the cloud service provider marketplace. You can then deploy the RHEL image as a cloud instance.

To deploy a RHEL instance on a public cloud platform, you can use either 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 build the image manually.
- This method uses your existing RHEL subscription. This 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.
- Red Hat provides customer service.
- For creating many images effectively, you can use the **cloud-init** utility.

Purchase a RHEL instance directly from the cloud provider marketplace

- You post-pay an hourly rate for using the service. This method is also referred to as *pay as you go* (PAYG).
- The cloud service provider provides customer service.

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) during the first boot. You can configure **cloud-init** to set hostnames, configure network interfaces, install packages, run scripts, and change default VM actions.

Before you begin, make sure to complete the following step:

- 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, 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), such as Amazon Web Services (AWS), Google Cloud, or Microsoft Azure, you can use RHEL image builder to create the image. RHEL image builder customizes images for specific cloud providers. The following image formats come with **cloud-init** preinstalled:
  - Amazon Machine Image (AMI)
  - Virtual Hard Drive (VHD)
  - QEMU copy-on-write (QCOW2)

For details about the RHEL image builder, see Installing RHEL image builder.

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

Otherwise, 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 complete during the first boot or on subsequent boots of a virtual machine (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, metadata, and optionally vendor data. Metadata has the configuration drive created by certified cloud service provider. **cloud-init** automatically identifies the existing datasource, or you can configure a custom datasource directive.

```
# 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 by 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**, see custom datasources.

For details on configuring datasources for certified cloud service provider, 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 five boot stages: detect, local, network, config, and final. Each stage belongs to a module execution phase that controls which modules run and in what sequence.

During system boot, the **cloud-init** utility runs through five *stages* that specify 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 to 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 run any modules. Instead, it identifies the datasource for **cloud-init** configuration.

2. **Local stage**: **cloud-init** searches local data sources and applies network configuration, including the DHCP-based fallback mechanism.
   This stage belongs to the *Initialization* phase and does not run 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, and triggering configuration management plugins 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 run in their phases. At the next boot, whether a module runs within a phase depends on the *module frequency* of that module. Module frequency sets in the **cloud-init** configuration, and determines whether a module runs 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 Schema definition for module creation.

The possible *module frequency* values are as follows:

- **PER_INSTANCE** means that the module runs on the first boot of an instance. For example, if you clone an instance or create a new instance from a saved image, the modules designated according to 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

- 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. You can use these directories and files to configure **cloud-init**, examine log files, find templates, and review configuration details after **cloud-init** completes execution. 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.

Directory or File	Description
/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

To list and use modules included in the basic configuration for **cloud-init**, use the **/etc/cloud/cloud.cfg** file. You can configure or remove modules based on your requirements:

- To perform actions during one of the **cloud-init** phases, you must 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 (RHEL) are as follows:

```
users:
 - default

disable_root: true
resize_rootfs_tmp: /dev
ssh_pwauth:   false

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

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

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

cloud_final_modules:
  - 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:
    name: cloud-user
    lock_passwd: true
    gecos: Cloud User
    groups: [adm, systemd-journal]
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    shell: /bin/bash
   distro: rhel
   network:
    renderers: ['sysconfig', 'eni', 'netplan', 'network-manager', 'networkd']
  paths:
    cloud_dir: /var/lib/cloud
    templates_dir: /etc/cloud/templates
  ssh_svcname: sshd

# vim:syntax=yaml
```

- **users** specifies the default user for the system. For more information, see Users and Groups.
- **disable_root** enables or disables root login. For more information, see Authorized Keys.
- **ssh_pwauth** specifies whether **ssh** is configured to accept password authentication. For more information, see Set Passwords.
- **mount_default_fields** configures mount points. It must be a list containing six values. For more information, see Mounts.
- **ssh_deletekeys** specifies whether to remove default host SSH keys. For more information, see Host Keys.
- **ssh_genkeytypes** specifies key types to generate. For more information, see Host Keys.
- **syslog_fix_perms** configures **cloud-init** to log all boot stages to its log file. For more information, see the **cloud-config.txt** file in the **usr/share/doc/cloud-init/examples** directory.
- **disable_vmware_customization** enables or disables VMware vSphere customization.
- **cloud_init_modules:** The modules in this section are services that run when the **cloud-init** service starts, early in the boot process.
- **cloud_config_modules:** These modules run during **cloud-init** configuration, after initial boot.
- **cloud_final_modules:** These modules run in the final phase of **cloud-init**, after the configuration finishes.
- **default_user** specifies details about the default user. For more information. see Users and Groups.
- **distro:** specifies the distribution.
- **cloud_dir** specifies the main directory that has **cloud-init**-specific subdirectories. For more information, see Directory layout.
- **templates_dir** specifies the location of the templates.
- **ssh_svcname** is 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

To set logging information for **cloud-init**, use the **05_logging.cfg** file located in the **/etc/cloud/cloud.cfg.d** directory. This directory contains various **cloud-init** directives that you can add.

By default, **cloud-init** uses the logging configuration in the **05_logging.cfg** file for Red Hat Enterprise Linux (RHEL):

```
## 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 Red Hat OpenStack Platform, Red Hat Satellite, and Red Hat OpenShift to configure and initialize virtual machines.

- **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. Each module can contain multiple configuration options and has a default module frequency that decides when it runs. The following table lists the supported **cloud-init** modules.

Table 3.1. Supported cloud-init modules

cloud-init Module	Description	Default Module Frequency
ansible	Runs Ansible playbooks	per instance
bootcmd	Runs commands at the early stage of the boot process	per always
ca_certs	Adds CA certificates	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
growpart	Resizes partitions to fill the available disk space	per always
install_hotplug	Installs hot-plug devices	per instance

cloud-init Module	Description	Default Module Frequency
keys_to_console	Writes fingerprints and keys 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
rh_subscription	Registers a Red Hat Enterprise Linux (RHEL) 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
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 hostname and fully qualified domain name (FQDN)	per always
set_passwords	Sets user passwords and enables or disables SSH password authentication	per instance
spacewalk	Configures Spacewalk client	per instance

cloud-init Module	Description	Default Module Frequency
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 hostname 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

Additional resources

- Cloud-init Modules

3.3. CLOUD-INIT MODULES NOT SUPPORTED BY RED HAT

Red Hat does not support certain **cloud-init** modules, 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

Module
grub_dpkg
ubuntu_init_switch

CHAPTER 4. CREATING A VIRTUAL MACHINE WITH CLOUD-INIT

You can create a new virtual machine (VM) with **cloud-init** by creating a **meta-data** file and a **user-data** file, then including them in an ISO image attached to a VM created from a KVM Guest Image.

- 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

You can execute commands during the first start-up and initialization of a virtual machine (VM) by using 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

You can rerun the **cloud-init** process when you need additional configurations for a virtual machine (VM) configured with the **cloud-init** service.

WARNING

If you rerun the **cloud-init** process, you might lose data and overwrite credentials such as SSH keys and passwords. Avoid rerunning **cloud-init** in a production environment.

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

To change virtual machine (VM) settings after the initial boot, you can modify the **cloud-init** configuration and rerun **cloud-init**. Clean the VM data directories before rerunning **cloud-init** to apply the new configuration.

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 the **cloud-init** configuration before rerunning **cloud-init**. The exact steps vary based on your datasource, such as Red Hat OpenStack Platform or other cloud platforms.

Procedure

1. Create and launch an instance for the Red Hat OpenStack Platform. For information about creating instances for Red Hat OpenStack Platform, 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

To manage users, access rights, and passwords in a virtual machine (VM), you can use the **cloud-init** utility. You can create users, configure **sudo** access, set **root** passwords, and force password changes at first login.

- 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**
  - 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**.

  - 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.

- 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:

```
#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

To register your system with Red Hat Subscription Management, you can use the **rh_subscription** directive in **cloud-init**. Configure the directive with username and password, activation key and organization, or a custom server hostname.

Using the **default** option

Under **rh_subscription**, add your **username** and **password**.

```
rh_subscription:
  username: <example@redhat.com>
  password: <example_password>
```

Using the **activation-key** and **org** options

Under **rh_subscription**, add your **activation key** and **org** number.

```
rh_subscription:
  activation-key: <example_key>
  org: <example_id>
```

Using the **server-hostname** option

You can set a server hostname in the **/etc/rhsm/rhsm.conf** file. Under **rh_subscription**, add your **username**, **password**, and **server-hostname**.

```
rh_subscription:
  username: <example@redhat.com>
  password: <example_password>
  server-hostname: <test.example.com>
```

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. With this setting, you can configure static IP addresses, gateways, and other network settings.

Red Hat Enterprise Linux (RHEL) 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
```

NOTE

You can disable a network configuration as follows:

```
network:
  config: disabled
```

Additional resources

- Network Configuration
- NoCloud

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

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

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 SHELL SCRIPTS WITH CLOUD-INIT

You can add list values or string values to **bootcmd** or **runcmd** in **cloud-init**. You can also provide a shell script within userdata to execute custom commands during VM initialization.

- 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 locale with the **locale** module in **cloud-init** by adding the locale directive to your configuration.

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 **cloud-init** by examining configuration and log files, then rerunning **cloud-init** from the command line to apply the fixes.

- 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)