Red Hat Enterprise Linux 10

Composing a customized RHEL system image

Creating customized system images with RHEL image builder on RHEL 10.0

Last Updated: 2025-06-10

Red Hat Enterprise Linux 10 Composing a customized RHEL system image

Creating customized system images with RHEL image builder on RHEL 10.0

Legal Notice

Abstract

RHEL image builder is a tool for creating deployment-ready customized system images: installation disks, virtual machines, cloud vendor-specific images, and others. By using RHEL image builder, you can create these images faster if compared to manual procedures, because it eliminates the specific configurations required for each output type.

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. RHEL IMAGE BUILDER DESCRIPTION

To deploy a system, create a system image. To create RHEL system images, use the RHEL image builder tool. You can use RHEL image builder to create customized system images of Red Hat Enterprise Linux, including system images prepared for deployment on cloud platforms. RHEL image builder automatically handles the setup details for each output type and is therefore easier to use and faster to work with than manual methods of image creation. You can access the RHEL image builder functionalities by using the command line in the **composer-cli** tool, or the graphical user interface in the RHEL web console.

1.1. RHEL IMAGE BUILDER TERMINOLOGY

RHEL image builder uses the following concepts:

Blueprint

A blueprint is a description of a customized system image. It lists the packages and customizations that will be part of the system. You can edit blueprints with customizations and save them as a particular version. When you create a system image from a blueprint, the image is associated with the blueprint in the RHEL image builder interface.
Create blueprints in the TOML format.

Compose

Composes are individual builds of a system image, based on a specific version of a particular blueprint. Compose as a term refers to the system image, the logs from its creation, inputs, metadata, and the process itself.

Customizations

Customizations are specifications for the image that are not packages. This includes users, groups, and SSH keys.

1.2. RHEL IMAGE BUILDER OUTPUT FORMATS

RHEL image builder can create images in multiple output formats shown in the following table.

Table 1.1. RHEL image builder output formats

Description	CLI name	File extension
QEMU Image	**qcow2**	**.qcow2**
Disk Archive	**tar**	**.tar**
Amazon Web Services	**raw**	**.raw**
Microsoft Azure	**vhd**	**.vhd**
Google Cloud Platform	**gce**	**.tar.gz**
VMware vSphere	**vmdk**	**.vmdk**
VMware vSphere	**ova**	**.ova**

Description	CLI name	File extension
Openstack	**qcow2**	**.qcow2**
RHEL Installer	**image-installer**	**.iso**
Oracle Cloud Infrastructure	**.oci**	**.qcow2**

To check the supported types, run the command:

```
# composer-cli compose types
```

1.3. SUPPORTED ARCHITECTURES FOR IMAGE BUILDS

RHEL image builder supports building images for the following architectures:

- AMD and Intel 64-bit (**x86_64**)
- ARM64 (**aarch64**)
- IBM Z (**s390x**)
- IBM POWER systems (ppc64)

However, RHEL image builder does not support multi-architecture builds. It only builds images of the same system architecture that it is running on. For example, if RHEL image builder is running on an **x86_64** system, it can only build images for the **x86_64** architecture.

1.4. ADDITIONAL RESOURCES

- RHEL image builder additional documentation index

CHAPTER 2. INSTALLING RHEL IMAGE BUILDER

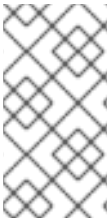
RHEL image builder is a tool for creating custom system images. Before using RHEL image builder, you must install it.

2.1. RHEL IMAGE BUILDER SYSTEM REQUIREMENTS

The host that runs RHEL image builder must meet the following requirements:

Table 2.1. RHEL image builder system requirements

Parameter	Minimal Required Value
System type	A dedicated host or virtual machine. Note that RHEL image builder is not supported in containers, including Red Hat Universal Base Images (UBI).
Processor	2 cores
Memory	4 GiB
Disk space	20 GiB of free space in the ` /var/cache/` filesystem
Access privileges	root
Network	Internet connectivity to the Red Hat Content Delivery Network (CDN).



NOTE

If you do not have internet connectivity, use RHEL image builder in isolated networks. For that, you must override the default repositories to point to your local repositories to not connect to Red Hat Content Delivery Network (CDN). Ensure that you have your content mirrored internally or use Red Hat Satellite.

Additional resources

- Configuring RHEL image builder repositories
- Provisioning to Satellite using a Red Hat image builder image

2.2. INSTALLING RHEL IMAGE BUILDER

Install RHEL image builder to have access to all the **osbuild-composer** package functionalities.

Prerequisites

- You are logged in to the Red Hat Enterprise Linux host on which you want to install RHEL image builder.
- The Red Hat Enterprise Linux host is subscribed to Red Hat Subscription Manager (RHSM) or Red Hat Satellite.

- You have enabled the **BaseOS** and **AppStream** repositories to be able to install the RHEL image builder packages.

Procedure

1. Install RHEL image builder and other necessary packages:

```
# dnf install osbuild-composer composer-cli cockpit-image-builder
```

   - **osbuild-composer** - A service to build customized RHEL operating system images.
   - **composer-cli**- This package enables access to the CLI interface.
   - **cockpit-image-builder** - This package enables access to the Web UI interface. The web console is installed as a dependency of the **cockpit-image-builder** package.

2. Enable and start RHEL image builder socket:

```
# systemctl enable --now osbuild-composer.socket
```

3. If you want to use RHEL image builder in the web console, enable and start it.

```
# systemctl enable --now cockpit.socket
```

   The **osbuild-composer** and **cockpit** services start automatically on first access.

4. Load the shell configuration script so that the autocomplete feature for the **composer-cli** command starts working immediately without logging out and in:

```
$ source /etc/bash_completion.d/composer-cli
```

5. Restart the running **osbuild-composer** service on your Red Hat Enterprise Linux host.

```
# systemctl restart osbuild-composer
```

Verification

- Verify that the installation works by running **composer-cli**:

```
# composer-cli status show
```

Troubleshooting

You can use a system journal to track RHEL image builder activities. Additionally, you can find the log messages in the file.

- To find the journal output for traceback, run the following commands:

```
$ journalctl | grep osbuild
```

- To show both remote or local workers:

```
$ journalctl -u osbuild-worker
```

- To show the running services:

```
$ journalctl -u osbuild-composer.service
```

CHAPTER 3. CONFIGURING RHEL IMAGE BUILDER REPOSITORIES

To use RHEL image builder, you must ensure that the repositories are configured. You can use the following types of repositories in RHEL image builder:

Official repository overrides

Use these if you want to download base system RPMs from elsewhere than the Red Hat Content Delivery Network (CDN) official repositories, for example, a custom mirror in your network. Using official repository overrides disables the default repositories, and your custom mirror must contain all the necessary packages.

Custom third-party repositories

Use these to include packages that are not available in the official RHEL repositories.

3.1. ADDING CUSTOM THIRD-PARTY REPOSITORIES TO RHEL IMAGE BUILDER

You can add custom third-party sources to your repositories and manage these repositories by using the **composer-cli**.

Prerequisites

- You have the URL of the custom third-party repository.

Procedure

1. Create a repository source file, such as **/root/repo.toml**. For example:

```
id = "k8s"
name = "Kubernetes"
type = "yum-baseurl"
url = "https://server.example.com/repos/<company_internal_packages>/"
check_gpg = false
check_ssl = false
system = false
```

   The **type** field accepts the following valid values: **yum-baseurl**, **yum-mirrorlist**, and **yum-metalink**.

2. Save the file in the TOML format.

3. Add the new third-party source to RHEL image builder:

```
$ composer-cli sources add <file_name>.toml
```

Verification

1. Check if the new source was successfully added:

```
$ composer-cli sources list
```

2. Check the new source content:

```
$ composer-cli sources info <source_id>
```

3.2. ADDING THIRD-PARTY REPOSITORIES WITH SPECIFIC DISTRIBUTIONS TO RHEL IMAGE BUILDER

You can specify a list of distributions in the custom third-party source file by using the optional field **distro**. The repository file uses the distribution string list while resolving dependencies during the image building.

Any request that specifies **rhel-10.0** uses this source. For example, if you list packages and specify **rhel-10.0**, it includes this source. However, listing packages for the host distribution do not include this source.

Prerequisites

- You have the URL of the custom third-party repository.
- You have the list of distributions that you want to specify.

Procedure

1. Create a repository source file, such as **/root/repo.toml**. For example, to specify the distribution:

```
check_gpg = true
check_ssl = true
distros = ["rhel-10.0"]
id = "rhel-10.0-local"
name = "packages for RHEL"
system = false
type = "yum-baseurl"
url = "https://local/repos/rhel10/<project_repo>/"
```

2. Save the file in the TOML format.
3. Add the new third-party source to RHEL image builder:

```
$ composer-cli sources add <file-name>.toml
```

Verification

1. Check if the new source was successfully added:

```
$ composer-cli sources list
```

2. Check the new source content:

```
$ composer-cli sources info <source_id>
```

3.3. CHECKING REPOSITORIES METADATA WITH GPG

To detect and avoid corrupted packages, you can use the DNF package manager to check the GNU Privacy Guard (GPG) signature on RPM packages, and also to check if the repository metadata has been signed with a GPG key.

You can either enter the **gpgkey** that you want to do the check over **https** by setting the **gpgkeys** field with the key URL. Alternatively, to improve security, you can also embed the whole key into the **gpgkeys** field, to import it directly instead of fetching the key from the URL.

Prerequisites

- The directory that you want to use as a repository exists and contains packages.

Procedure

1. Access the folder where you want to create a repository:

```
$ cd repo/
```

2. Run the **createrepo_c** to create a repository from RPM packages:

```
$ createrepo_c .
```

3. Access the directory where the repodata is:

```
$ cd repodata/
```

4. Sign your **repomd.xml** file:

```
$ gpg -u gpg-key-email --yes --detach-sign --armor /srv/repo/example/repomd.xml
```

5. To enable GPG signature checks in the repository:

   a. Set **check_repogpg = true** in the repository source.

   b. Enter the **gpgkey** that you want to do the check. If your key is available over **https**, set the **gpgkeys** field with the key URL for the key. You can add as many URL keys as you need. The following is an example:

```
check_gpg = true
check_ssl = true
id = "<signed_local_packages>"
name = "<repository_name>"
type = "yum-baseurl"
url = "\https://local/repos/projectrepo/"
check_repogpg = true
gpgkeys=["\https://local/keys/repokey.pub"]
```

   As an alternative, add the GPG key directly in the **gpgkeys** field, for example:

```
check_gpg = true
check_ssl = true
check_repogpg
id = "custom-local"
name = "signed local packages"
```

```
type = "yum-baseurl"
url = "https://local/repos/projectrepo/"
gpgkeys=["https://remote/keys/other-repokey.pub",
'''-----BEGIN PGP PUBLIC KEY BLOCK-----
…
-----END PGP PUBLIC KEY BLOCK-----''']
```

- If the test does not find the signature, the GPG tool shows an error similar to the following one:

```
$ GPG verification is enabled, but GPG signature is not available.
This may be an error or the repository does not support GPG verification:
Status code: 404 for \http://repo-server/rhel/repodata/repomd.xml.asc (IP:
192.168.1.3)
```

- If the signature is invalid, the GPG tool shows an error similar to the following one:

```
repomd.xml GPG signature verification error: Bad GPG signature
```

Verification

- Test the signature of the repository manually:

```
$ gpg --verify /srv/repo/example/repomd.xml.asc
```

3.4. RHEL IMAGE BUILDER DEFAULT SYSTEM REPOSITORIES

RHEL image builder **osbuild-composer** back end does not inherit the system repositories located in the **/etc/yum.repos.d/** directory. Instead, it has its own set of official repositories defined in the **/usr/share/osbuild-composer/repositories** directory. This includes the Red Hat official repository, which contains the base system RPMs to install additional software or update already installed programs to newer versions. If you want to override the official repositories, you must define overrides in **/etc/osbuild-composer/repositories/**. This directory is for user defined overrides and the files located there take precedence over those in the **/usr/share/osbuild-composer/repositories/** directory.

The configuration files are not in the usual RPM repository format known from the files in **/etc/yum.repos.d/**. Instead, they are JSON files.

3.5. OVERRIDING A SYSTEM REPOSITORY

You can configure your own repository override for RHEL image builder in the **/etc/osbuild-composer/repositories** directory.

Prerequisites

- You have a custom repository that is accessible from your host system.

Procedure

1. Create the **/etc/osbuild-composer/repositories/** directory to store your repository overrides:

```
$ sudo mkdir -p /etc/osbuild-composer/repositories
```

2. Create a JSON file, using a name corresponding to your RHEL version. Alternatively, you can copy the file for your distribution from **/usr/share/osbuild-composer/** and modify its content. For RHEL 10, use **/etc/osbuild-composer/repositories/rhel-10.json**.

3. Add the following structure to your JSON file. Specify only one of the following attributes, in the string format:

   - **baseurl** - The base URL of the repository.
   - **metalink** - The URL of a metalink file that contains a list of valid mirror repositories.
   - **mirrorlist** - The URL of a mirrorlist file that contains a list of valid mirror repositories. The remaining fields, such as **gpgkey**, and **metadata_expire**, are optional.
     For example:

```
{
    "x86_64": [
      {
          "name": "baseos",
          "baseurl": "http://mirror.example.com/composes/released/RHEL-
10.0/10.0/BaseOS/x86_64/os/",
          "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\n (…)",
          "check_gpg": true
      }
   ]
}
```

     Alternatively, you can copy the JSON file for your distribution, by replacing **rhel-version.json** with your RHEL version, for example: **rhel-10.0.json**.

```
$ cp /usr/share/osbuild-composer/repositories/rhel-10.0.json /etc/osbuild-
composer/repositories/
```

4. Optional: Verify the JSON file:

```
$ json_verify /etc/osbuild-composer/repositories/file.json
```

5. Edit the **baseurl** paths in the **rhel-10.0.json** file and save it. For example:

```
$ /etc/osbuild-composer/repositories/rhel-10.0.json
```

6. Restart the **osbuild-composer.service**:

```
$ sudo systemctl restart osbuild-composer.service
```

Verification

- Check if the repository points to the correct URLs:

```
$ cat /etc/yum.repos.d/redhat.repo
```

  You can see that the repository points to the correct URLs which are copied from the **/etc/yum.repos.d/redhat.repo** file.

Additional resources

- The latest RPMs version available in repository not visible for **osbuild-composer** (Red Hat Knowledgebase)

3.6. OVERRIDING A SYSTEM REPOSITORY THAT REQUIRES SUBSCRIPTIONS

You can set up the **osbuild-composer** service to use system subscriptions that are defined in the **/etc/yum.repos.d/redhat.repo** file. To use a system subscription in **osbuild-composer**, define a repository override that has the following details:

- The same **baseurl** as the repository defined in **/etc/yum.repos.d/redhat.repo**.
- The value of **"rhsm": true** defined in the JSON object.

NOTE

osbuild-composer does not automatically use repositories defined in **/etc/yum.repos.d/**. You need to manually specify them either as a system repository override or as an additional **source** by using **composer-cli**. The "BaseOS" and "AppStream" repositories usually use system repository overrides, whereas all the other repositories use **composer-cli** sources.

Prerequisites

- Your system has a subscription defined in **/etc/yum.repos.d/redhat.repo**
- You have created a repository override.

Procedure

1. Get the **baseurl** from the **/etc/yum.repos.d/redhat.repo** file:

```
# cat /etc/yum.repos.d/redhat.repo
[AppStream]
name = AppStream mirror example
baseurl = \https://mirror.example.com/RHEL-10.0/10.0/AppStream/x86_64/os/
enabled = 1
gpgcheck = 0
sslverify = 1
sslcacert = /etc/pki/ca1/ca.crt
sslclientkey = /etc/pki/ca1/client.key
sslclientcert = /etc/pki/ca1/client.crt
metadata_expire = 86400
enabled_metadata = 0
```

2. Configure the repository override to use the same **baseurl** and set **rhsm** to true:

```
{
   "x86_64": [
      {
         "name": "AppStream mirror example",
         "baseurl": "https://mirror.example.com/RHEL-10.0/10.0/AppStream/x86_64/os/",
```

```
            "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\n (...)",
            "check_gpg": true,
            "rhsm": true
        }
    ]
}
```

3. Restart the **osbuild-composer.service**:

```
$ sudo systemctl restart osbuild-composer.service
```

Additional resources

- RHEL image builder uses CDN repositories when host is registered to Satellite 6 (Red Hat Knowledgebase)

3.7. CONFIGURING AND USING SATELLITE CV AS A CONTENT SOURCE

You can use Satellite's content views (CV) as repositories to build images with RHEL image builder. For that, on your host registered to Satellite, manually configure the repository references to be able to retrieve from the Satellite repositories, instead of the Red Hat Content Delivery Network (CDN) official repositories.

Prerequisites

- You have installed RHEL image builder. See Installing RHEL image builder .
- You are using RHEL image builder on a host registered to Satellite 6. See linkhttps://docs.redhat.com/en/documentation/red_hat_satellite/6.7/html/provisioning_guide/inc an-image-builder-image-for-provisioning[Using a RHEL image builder image for Provisioning].

Procedure

1. Find the repository URL from your currently configured repositories:

```
$ $ sudo yum -v repolist "-baseos-rpms" | grep -i repo-baseurl
```

The following output is an example:

```
https://satellite6.example.com/pulp/content/YourOrg/YourEnv/YourCV/content/dist/rhel10/10/x86_64/baseos/os
```

2. Modify the hard-coded repositories to a Satellite Server.
   a. Create a repository directory with the **0755** permission:

```
$ sudo mkdir -pvm 0755 /etc/osbuild-composer/repositories
```

   b. Copy the content from **/usr/share/osbuild-composer/repositories/*.json** to the directory that you created:

```
$ sudo cp /usr/share/osbuild-composer/repositories/*.json /etc/osbuild-composer/repositories/
```

c. Update the Satellite URL and the file contents through the **/content/dist/*** line:

```
$ sudo sed -i -e 's|cdn.redhat.com|satellite6.example.com/pulp/content/YourOrg/YourEnv/YourCV|' /etc/osbuild-composer/repositories/.json
```

d. Verify that the configuration was correctly replaced:

```
$ sudo vi /etc/osbuild-composer/repositories/rhel-10.json
```

3. Restart the services:

```
$ sudo systemctl restart osbuild-worker@1.service osbuild-composer.service
```

4. Override the required system repository in Red Hat image builder configuration and use the URL of your Satellite repository as a baseurl. See Overriding a system repository .

Additional resources

- Composer RHEL image builder fails when multiple custom repositories are defined on the Satellite (Red Hat Knowledgebase)

3.8. USING SATELLITE CV AS REPOSITORIES TO BUILD IMAGES IN RHEL IMAGE BUILDER

Configure RHEL image builder to use Satellite's content views (CV) as repositories to build your custom images.

Prerequisites

- You have integrated Satellite with RHEL web console. See Enabling the RHEL web console on Satellite

Procedure

1. In the Satellite web UI, navigate to **Content > Products**, select your **Product** and click the repository you want to use.
2. Search for the secured URL (HTTPS) in the Published field and copy it.
3. Use the URL that you copied as a baseurl for the Red Hat image builder repository. See Adding custom third-party repositories to RHEL image builder.

Next steps

- Build the image. See Creating a system image by using RHEL image builder in the web console interface.

CHAPTER 4. CREATING SYSTEM IMAGES BY USING THE RHEL IMAGE BUILDER CLI

RHEL image builder is a tool for creating custom system images. To control RHEL image builder and create your custom system images, you can use the command line (CLI) or the web console interface.

4.1. INTRODUCING THE RHEL IMAGE BUILDER COMMAND-LINE INTERFACE

You can use the RHEL image builder command-line interface (CLI) to create blueprints, by running the **composer-cli** command with the suitable options and subcommands.

The workflow for the command line can be summarized as follows:

1. Create a blueprint or export (*save*) an existing blueprint definition to a plain text file
2. Edit this file in a text editor
3. Import the blueprint text file back into image builder
4. Run a compose to build an image from the blueprint
5. Export the image file to download it

Apart from the basic subcommands to create a blueprint, the **composer-cli** command offers many subcommands to examine the state of configured blueprints and composes.

4.2. USING RHEL IMAGE BUILDER AS A NON-ROOT USER

To run the **composer-cli** commands as non-root, the user must be in the **weldr** group.

Prerequisites

- You have created a user.

Procedure

- To add a user to the **weldr** or **root** groups, run the following commands:

```
$ sudo usermod -a -G weldr user
$ newgrp weldr
```

4.3. CREATING A BLUEPRINT BY USING THE COMMAND LINE

You can create a new RHEL image builder blueprint by using the command line (CLI). The blueprint describes the final image and its customizations, such as packages, and kernel customizations.

Prerequisites

- You are logged in as the root user or a user who is a member of the **weldr** group

Procedure

1. Create a plain text file with the following contents:

```
name = "<blueprint_name>"
description = "<long_form_description>"
version = "<0.0.1>"
modules = []
groups = []
```

Replace *<blueprint_name>* and *<long_form_description>* with a name and description for your blueprint.

Replace *<0.0.1>* with a version number according to the Semantic Versioning scheme.

2. For every package that you want to be included in the blueprint, add the following lines to the file:

```
[[packages]]
name = "<package_name>"
version = "<package_version>"
```

Replace *<package_name>* with the name of the package, such as **httpd**, **gdb-doc**, or **coreutils**.

Optionally, replace *<package_version>* with the version to use. This field supports **dnf** version specifications:

- For a specific version, use the exact version number such as **8.7.0**.
- For the latest available version, use the asterisk *****
- For the latest minor version, use formats such as **8.***.

3. Customize your blueprints to suit your needs. For example, disable Simultaneous Multi Threading (SMT), add the following lines to the blueprint file:

```
[customizations.kernel]
append = "nosmt=force"
```

For additional customizations available, see Supported image customizations.

Note that **[]** and **[[]]** are different data structures expressed in TOML.

- The **[customizations.kernel]** header represents a single table that is defined by a collection of keys and their corresponding value pairs, for example: **append = "nosmt=force"**.
- The **[[packages]]** header represents an array of tables. The first instance defines the array and its first table element, for example, **name = "package-name"** and **version = "package-version"**, and each subsequent instance creates and defines a new table element in that array, in the order that you defined them.

4. Save the file, for example, as ***<blueprint_name>.toml*** and close the text editor.

5. Optional: Check if all settings from the Blueprint TOML file have been correctly parsed. Save the blueprint and compare the saved output with the input file:

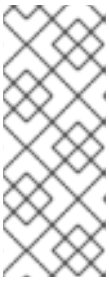
```
# composer-cli blueprints save <blueprint_name>.toml
```

a. Compare the *<blueprint_name>.toml* saved file with the input file.

6. Push the blueprint:

```
# composer-cli blueprints push <blueprint_name>.toml
```

Replace *<blueprint_name>* with the value you used in previous steps.



NOTE

To create images using **composer-cli** as non-root, add your user to the **weldr** or **root** groups.

```
# usermod -a -G weldr user
$ newgrp weldr
```

Verification

- List the existing blueprints to verify that the blueprint has been pushed and exists:

```
# composer-cli blueprints list
```

- Display the blueprint configuration you have just added:

```
# composer-cli blueprints show <blueprint_name>
```

- Check whether the components and versions listed in the blueprint and their dependencies are valid:

```
# composer-cli blueprints depsolve <blueprint_name>
```

If RHEL image builder is unable to solve the dependencies of a package from your custom repositories, remove the **osbuild-composer** cache:

```
$ sudo rm -rf /var/cache/osbuild-composer/
$ sudo systemctl restart osbuild-composer
```

Additional resources

- osbuild-composer is unable to depsolve a package from my custom repository (Red Hat Knowledgebase)
- Composing a customized RHEL system image with proxy server (Red Hat Knowledgebase)

4.4. EDITING A BLUEPRINT WITH COMMAND-LINE INTERFACE

You can edit an existing blueprint in the command-line (CLI) interface to, for example, add a new package, or define a new group, and to create your customized images. For that, follow the steps:

Prerequisites

- You have created a blueprint.

Procedure

1. List the existing blueprints:

```
# composer-cli blueprints list
```

2. Save the blueprint to a local text file:

```
# composer-cli blueprints save <blueprint_name>
```

3. Edit the ***<blueprint_name>.toml*** file with a text editor and make your changes.

4. Before finishing the edits, verify that the file is a valid blueprint:

   a. Remove the following line from the blueprint, if present:

```
packages = []
```

   b. Increase the version number, for example, from 0.0.1 to 0.1.0. Remember that RHEL image builder blueprint versions must use the Semantic Versioning scheme. Note also that if you do not change the version, the **patch** version component increases automatically.

5. Save the file and close the text editor.

6. Push the blueprint back into RHEL image builder:

```
# composer-cli blueprints push <blueprint_name>.toml
```

NOTE

To import the blueprint back into RHEL image builder, supply the file name including the **.toml** extension, while in other commands use only the blueprint name.

Verification

1. To verify that the contents uploaded to RHEL image builder match your edits, list the contents of blueprint:

```
# composer-cli blueprints show <blueprint_name>
```

2. Check whether the components and versions listed in the blueprint and their dependencies are valid:

```
# composer-cli blueprints depsolve <blueprint_name>
```

Additional resources

- Supported Image Customizations

4.5. CREATING A SYSTEM IMAGE WITH RHEL IMAGE BUILDER ON THE COMMAND LINE

You can build a customized RHEL image by using the RHEL image builder command-line interface. For that, you must specify a blueprint and an image type. Optionally, you can also specify a distribution. If you do not specify a distribution, it will use the same distribution and version as the host system. The architecture is also the same as the one on the host.

Prerequisites

- You have a blueprint prepared for the image.

Procedure

1. Optional: List the image formats you can create:

```
# composer-cli compose types
```

2. Start the compose:

```
# composer-cli compose start <blueprint_name> <image_type>
```

Replace *<blueprint_name>* with the name of the blueprint, and *<image_type>* with the type of the image. For the available values, see the output of the **composer-cli compose types** command.

The compose process starts in the background and shows the composer Universally Unique Identifier (UUID).

3. The image creation can take up to ten minutes to complete.
To check the status of the compose:

```
# composer-cli compose status
```

A finished compose shows the **FINISHED** status value. To identify your compose in the list, use its UUID.

4. After the compose process is finished, download the resulting image file:

```
# composer-cli compose image <uuid>
```

Replace *<uuid>* with the UUID value shown in the previous steps.

Verification

After you create your image, you can check the image creation progress by using the following commands:

- Download the metadata of the image to get a **.tar** file of the metadata for the compose:

```
$ sudo composer-cli compose metadata <uuid>
```

- Download the logs of the image:

```
$ sudo composer-cli compose logs <uuid>
```

The command creates a **.tar** file that contains the logs for the image creation. If the logs are empty, you can check the journal.

- Check the journal:

```
$ journalctl | grep osbuild
```

- Check the manifest of the image:

```
$ sudo cat /var/lib/osbuild-composer/jobs/<job_uuid>.json
```

You can find the *<job_uuid>.json* in the journal.

Additional resources

- Tracing RHEL image builder (Red Hat Knowledgebase)

4.6. BASIC RHEL IMAGE BUILDER COMMAND-LINE COMMANDS

The RHEL image builder command-line interface offers the following subcommands.

Blueprint manipulation

List all available blueprints

```
# composer-cli blueprints list
```

Show a blueprint contents in the TOML format

```
# composer-cli blueprints show <blueprint_name>
```

Save (export) blueprint contents in the TOML format into a file *<blueprint_name>.toml*

```
# composer-cli blueprints save <blueprint_name>
```

Remove a blueprint

```
# composer-cli blueprints delete <blueprint_name>
```

Push (import) a blueprint file in the TOML format into RHEL image builder

```
# composer-cli blueprints push <blueprint_name>
```

Composing images from blueprints

List the available image types

```
# composer-cli compose types
```

Start a compose

```
# composer-cli compose start <blueprint> <compose_type>
```

List all composes

```
# composer-cli compose list
```

List all composes and their status

```
# composer-cli compose status
```

Cancel a running compose

```
# composer-cli compose cancel <compose_uuid>
```

Delete a finished compose

```
# composer-cli compose delete <compose_uuid>
```

Show detailed information about a compose

```
# composer-cli compose info <compose_uuid>
```

Download image file of a compose

```
# composer-cli compose image <compose_uuid>
```

See more subcommands and options

```
# composer-cli help
```

Additional resources

- The *composer-cli*(1) man page on your system

4.7. PACKAGES INSTALLED BY RHEL IMAGE BUILDER

When you create a system image by using RHEL image builder, the system installs a set of base package groups.

NOTE

When you add additional components to your blueprint, ensure that the packages in the components you added do not conflict with any other package components. Otherwise, the system fails to solve dependencies and creating your customized image fails. You can check if there is no conflict between the packages by running the command:

```
# composer-cli blueprints depsolve BLUEPRINT-NAME
```

Table 4.1. Default packages to support image type creation

Image type	Default Packages
ami	**checkpolicy**, **chrony**, **cloud-init**, **cloud-utils-growpart**, **@Core**, **dhcp-client**, **insights-client**, **kernel**, **langpacks-en**, **net-tools**, **NetworkManager**, **redhat-release**, **redhat-release-eula**, **rng-tools**, **rsync**, **selinux-policy-targeted**, **tar**, **yum-utils**
qcow2	**@core, chrony, dnf, kernel, dnf, nfs-utils, dnf-utils, cloud-init, python3-jsonschema, qemu-guest-agent, cloud-utils-growpart, dracut-norescue, tar, tcpdump, rsync, dnf-plugin-spacewalk, rhn-client-tools, rhnlib, rhnsd, rhn-setup, NetworkManager, dhcp-client, cockpit-ws, cockpit-system, subscription-manager-cockpit, redhat-release, redhat-release-eula, rng-tools, insights-client**
tar	**policycoreutils, selinux-policy-targeted**
vhd	**@core, langpacks-en**
vmdk	**@core, chrony, cloud-init, firewalld, langpacks-en, open-vm-tools, selinux-policy-targeted**
image-installer	**anaconda-dracut, curl, dracut-config-generic, dracut-network, hostname, iwl100-firmware, iwl1000-firmware, iwl105-firmware, iwl135-firmware, iwl2000-firmware, iwl2030-firmware, iwl3160-firmware, iwl5000-firmware, iwl5150-firmware, iwl6000-firmware, iwl6050-firmware, iwl7260-firmware, kernel, less, nfs-utils, openssh-clients, ostree, plymouth, prefixdevname, rng-tools, rpcbind, selinux-policy-targeted, systemd, tar, xfsprogs, xz**
gce	**@core, langpacks-en, acpid, dhcp-client, dnf-automatic, net-tools, python3, rng-tools, tar, vim**

4.8. ENABLED SERVICES ON CUSTOM IMAGES

When you use image builder to configure a custom image, the default services that the image uses are determined by the following:

- The RHEL release on which you use the **osbuild-composer** utility

- The image type

For example, the **ami** image type enables the **sshd**, **chronyd**, and **cloud-init** services by default. If these services are not enabled, the custom image does not boot.

Table 4.2. Enabled services to support image type creation

Image type	Default enabled Services
ami	sshd, cloud-init, cloud-init-local, cloud-config, cloud-final
openstack	sshd, cloud-init, cloud-init-local, cloud-config, cloud-final
qcow2	cloud-init
rhel-edge-commit	No extra service enables by default
tar	No extra service enables by default
vhd	sshd, chronyd, waagent, cloud-init, cloud-init-local, cloud-config, cloud-final
vmdk	sshd, chronyd, vmtoolsd, cloud-init

Note: You can customize which services to enable during the system boot. However, the customization does not override services enabled by default for the mentioned image types.

Additional resources

- Supported Image Customizations

CHAPTER 5. SUPPORTED IMAGE CUSTOMIZATIONS

You can customize your image by adding customizations to your blueprint, such as:

- Adding an additional RPM package
- Enabling a service
- Customizing a kernel command line parameter.

Between others. You can use several image customizations within blueprints. By using the customizations, you can add packages and groups to the image that are not available in the default packages. To use these options, configure the customizations in the blueprint and import (push) it to RHEL image builder.

5.1. SELECTING A DISTRIBUTION

You can use the **distro** field to specify the distribution to use when composing your images or solving dependencies in the blueprint. If the **distro** field is left blank, the blueprint automatically uses the host's operating system distribution. If you do not specify a distribution, the blueprint uses the host distribution. When you upgrade the host operating system, blueprints without a specified distribution build images by using the upgraded operating system version.

You can build images for older major versions on a newer system. For example, you can use a RHEL 10 host to create RHEL 9 and RHEL 8 images. However, you cannot build images for newer major versions on an older system.

IMPORTANT

You cannot build an operating system image that differs from the RHEL image builder host. For example, you cannot use a RHEL system to build Fedora or CentOS images.

- Customize the blueprint with the RHEL distribution to always build the specified RHEL image:

```
name = "blueprint_name"
description = "blueprint_version"
version = "0.1"
distro = "different_minor_version"
```

For example:

```
name = "tmux"
description = "tmux image with openssh"
version = "1.2.16"
distro = "rhel-9.6"
```

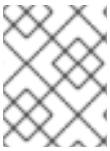
Replace **"*different_minor_version*"** to build a different minor version, for example, if you want to build a RHEL 9.6 image, use **distro** = "rhel-96". On RHEL 9.5 image, you can build minor versions such as RHEL 9.4, RHEL 9.3, and earlier releases.

5.2. SELECTING A PACKAGE GROUP

Customize the blueprint with package groups. The **groups** list describes the groups of packages that

you want to install into the image. The package groups are defined in the repository metadata. Each group has a descriptive name that is used primarily for display in user interfaces, and an ID that is commonly used in Kickstart files. In this case, you must use the ID to list a group. Groups have three different ways of categorizing their packages: mandatory, default, and optional. Only mandatory and default packages are installed in the blueprints. It is not possible to select optional packages.

The **name** attribute is a required string and must match exactly the package group id in the repositories.



NOTE

Currently, there are no differences between packages and modules in **osbuild-composer**. Both are treated as an RPM package dependency.

- Customize your blueprint with a package:

```
[[groups]]
name = "group_name"
```

Replace ***group_name*** with the name of the group. For example, **anaconda-tools**:

```
[[groups]]
name = "anaconda-tools"
```

5.3. SELECTING A PACKAGE

Customize the blueprint with packages and modules.

- The **name** attribute is a required string and can be an exact match, or a filesystem glob that uses ***** for wildcards, and **?** for character matching.
- The **version** attribute is an optional string can be an exact match or a filesystem glob of the version that uses ***** for wildcards, and **?** for character matching. If you do not enter a version, the system uses the latest version in the repositories.

When you use a virtual provides as the package name, the version glob must be *****. Consequently, you will be unable to freeze the blueprint, because the provides will expand into multiple packages with their own names and versions.

NOTE

Currently, there are no differences between packages and modules in **osbuild-composer**. Both are treated as an RPM package dependency. .Procedure

- Customize your blueprint with a package:

```
[[packages]]
name = "package_name"
```

Replace ***package_name*** with the name of the group. For example, the **tmux-2.9a**, and the **openssh-server-8.*** packages.

```
[[packages]]
name = "tmux"
version = "2.9a"
```

name = "openssh-server" version = "8.*"

5.4. EMBEDDING A CONTAINER

You can customize your blueprint to embed the latest RHEL container. The containers list contains objects with a source, and optionally, the **tls-verify** attribute.

The container list entries describe the container images to be embedded into the image.

- **source** - Mandatory field. It is a reference to the container image at a registry. This example uses the **registry.access.redhat.com** registry. You can specify a tag version. The default tag version is latest.
- **name** - The name of the container in the local registry.
- **tls-verify** - Boolean field. The tls-verify boolean field controls the transport layer security. The default value is true.

The embedded containers do not start automatically. To start it, create **systemd** unit files or **quadlets** with the files customization.

- To embed a container from **registry.access.redhat.com/ubi10/ubi:latest** and a container from your host, add the following customization to your blueprint:

```
[[containers]]
source = "registry.access.redhat.com/ubi10/ubi:latest"
name =  "local-name"
tls-verify = true

[[containers]]
source = "localhost/test:latest"
local-storage = true
```

You can access protected container resources by using a **containers-auth.json** file. See Container registry credentials.

5.5. SETTING THE IMAGE HOSTNAME

The **customizations.hostname** is an optional string that you can use to configure the final image hostname. This customization is optional, and if you do not set it, the blueprint uses the default hostname.

- Customize the blueprint to configure the hostname:

```
[customizations]
hostname = "baseimage"
```

5.6. SPECIFYING ADDITIONAL USERS

Add a user to the image, and optionally, set their SSH key. All fields for this section are optional except for the **name**.

Procedure

- Customize the blueprint to add a user to the image:

```
[[customizations.user]]
name = "USER-NAME"
description = "USER-DESCRIPTION"
password = "PASSWORD-HASH"
key = "PUBLIC-SSH-KEY"
home = "/home/USER-NAME/"
shell = "/usr/bin/bash"
groups = ["users", "wheel"]
uid = NUMBER
gid = NUMBER
```

```
[[customizations.user]]
name = "admin"
description = "Administrator account"
password = "$6$CHO2$3rN8eviE2t50lmVyBYihTgVRHcaecmeCk31L..."
key = "PUBLIC SSH KEY"
home = "/srv/widget/"
shell = "/usr/bin/bash"
groups = ["widget", "users", "wheel"]
uid = 1200
gid = 1200
expiredate = 12345
```

The GID is optional and must already exist in the image. Optionally, a package creates it, or the blueprint creates the GID by using the **[[customizations.group]]** entry.

Replace *PASSWORD-HASH* with the actual **password hash**. To generate the **password hash**, use a command such as:

```
$ python3 -c 'import crypt,getpass;pw=getpass.getpass();print(crypt.crypt(pw) if
(pw==getpass.getpass("Confirm: ")) else exit())'
```

Replace the other placeholders with suitable values.

Enter the **name** value and omit any lines you do not need.

Repeat this block for every user to include.

5.7. SPECIFYING ADDITIONAL GROUPS

Specify a group for the resulting system image. Both the **name** and the **gid** attributes are mandatory.

- Customize the blueprint with a group:

```
[[customizations.group]]
name = "GROUP-NAME"
gid = NUMBER
```

Repeat this block for every group to include. For example:

```
[[customizations.group]]
name = "widget"
gid = 1130
```

5.8. SETTING SSH KEY FOR EXISTING USERS

You can use **customizations.sshkey** to set an SSH key for the existing users in the final image. Both **user** and **key** attributes are mandatory.

- Customize the blueprint by setting an SSH key for existing users:

```
[[customizations.sshkey]]
user = "root"
key = "PUBLIC-SSH-KEY"
```

For example:

```
[[customizations.sshkey]]
user = "root"
key = "SSH key for root"
```

5.9. APPENDING A KERNEL ARGUMENT

You can append arguments to the boot loader kernel command line. By default, RHEL image builder builds a default kernel into the image. However, you can customize the kernel by configuring it in the blueprint.

- Append a kernel boot parameter option to the defaults:

```
[customizations.kernel]
append = "<kernel_option>"
```

For example:

```
[customizations.kernel]
name = "kernel-debug"
append = "nosmt=force"
```

5.10. BUILDING RHEL IMAGES BY USING THE REAL-TIME KERNEL

To build a RHEL image by using the real-time kernel (**kernel-rt**), you need to override a repository so that you can then build an image in which **kernel-rt** is correctly selected as the default kernel. Use the **.json** from the **/usr/share/osbuild-composer/repositories/** directory. Then, you can deploy the image that you built to a system and use the real time kernel features.

NOTE

The real-time kernel runs on AMD64 and Intel 64 server platforms that are certified to run Red Hat Enterprise Linux.

Prerequisites

- Your system is registered and RHEL is attached to a RHEL for Real Time subscription.

Procedure

1. Create the following directory:

```
# mkdir /etc/osbuild-composer/repositories/
```

2. Copy the content from the **/usr/share/osbuild-composer/repositories/rhel-10.json** file to the new directory:

```
# cp /usr/share/osbuild-composer/repositories/rhel-10.json /etc/osbuild-composer/repositories
```

3. Edit the **/etc/osbuild-composer/repositories/rhel-10.json** file to include the RT kernel repo:

```
# grep -C 6 kernel-rt /etc/osbuild-composer/repositories/rhel-10.json
    {
      "baseurl": "https://cdn.redhat.com/content/dist/rhel10/10/x86_64/appstream/os",
      "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\nm……..=\n=UZd/\n-----END
PGP PUBLIC KEY BLOCK-----\n",
      "rhsm": true,
      "check_gpg": true
    },
    {
      "name": "kernel-rt",
      "baseurl": "https://cdn.redhat.com/content/dist/rhel10/10/x86_64/rt/os",
      "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\nmQINBEr……fg==\n=UZd/\n-
----END PGP PUBLIC KEY BLOCK-----\n",
      "rhsm": true,
      "check_gpg": true
    },
```

4. Restart the service:

```
# systemctl restart osbuild-composer
```

5. Confirm that the **kernel-rt** has been included into the **.json** file:

```
# composer-cli sources list
# composer-cli sources info kernel-rt
```

You will see the URL that you have previously configured.

6. Create a blueprint. In the blueprint, add the "[customizations.kernel]" customization. The following is an example that contains the "[customizations.kernel]" in the blueprint:

```
name = "rt-kernel-image"
description = ""
version = "2.0.0"
modules = []
groups = []
distro = "rhel-10.0"
```

```
[[customizations.user]]
name = "admin"
password = "admin"
groups = ["users", "wheel"]
[customizations.kernel]
name = "kernel-rt"
append = ""
```

7. Push the blueprint to the server:

```
# composer-cli blueprints push rt-kernel-image.toml
```

8. Build your image from the blueprint you created. The following example builds a (**.qcow2**) image:

```
# composer-cli compose start rt-kernel-image qcow2
```

9. Deploy the image that you built to the system where you want to use the real time kernel features.

Verification

- After booting a VM from the image, verify that the image was built with the **kernel-rt** correctly selected as the default kernel.

```
$ cat /proc/cmdline
BOOT_IMAGE=(hd0,got3)/vmlinuz-6.12.0-0.el10_0_.x86_64+rt...
```

5.11. SETTING TIME ZONE AND NTP

You can customize your blueprint to configure the time zone and the *Network Time Protocol* (NTP). Both **timezone** and **ntpservers** attributes are optional strings. If you do not customize the time zone, the system uses *Universal Time, Coordinated* (UTC). If you do not set NTP servers, the system uses the default distribution.

- Customize the blueprint with the **timezone** and the **ntpservers** you want:

```
[customizations.timezone]
timezone = "TIMEZONE"
ntpservers = "NTP_SERVER"
```

For example:

```
[customizations.timezone]
timezone = "US/Eastern"
ntpservers = ["0.north-america.pool.ntp.org", "1.north-america.pool.ntp.org"]
```

NOTE

Some image types, such as Google Cloud, already have NTP servers set up. You cannot override it because the image requires the NTP servers to boot in the selected environment. However, you can customize the time zone in the blueprint.

5.12. CUSTOMIZING THE LOCALE SETTINGS

You can customize the locale settings for your resulting system image. Both **language** and the **keyboard** attributes are mandatory. You can add many other languages. The first language you add is the primary language and the other languages are secondary.

Procedure

- Set the locale settings:

```
[customizations.locale]
languages = ["<language>"]
keyboard = "<keyboard>"
```

  For example:

```
[customizations.locale]
languages = ["en_US.UTF-8"]
keyboard = "us"
```

- To list the values supported by the languages, run the following command:

```
$ localectl list-locales
```

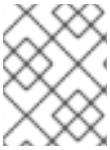
- To list the values supported by the keyboard, run the following command:

```
$ localectl list-keymaps
```

5.13. CUSTOMIZING FIREWALL

Set the firewall for the resulting system image. By default, the firewall blocks incoming connections, except for services that enable their ports explicitly, such as **sshd**.

If you do not want to use the **[customizations.firewall]** or the **[customizations.firewall.services]**, either remove the attributes, or set them to an empty list []. If you only want to use the default firewall setup, you can omit the customization from the blueprint.



NOTE

The Google template explicitly disable the firewall for their environment. You cannot override this behavior by setting the blueprint.

Procedure

- Customize the blueprint with the following settings to open other ports and services:

```
[customizations.firewall]
ports = ["<ports>"]
```

  Where **ports** is an optional list of strings that contain ports or a range of ports and protocols to open. You can configure the ports by using the following format: **port:protocol** format. You can configure the port ranges by using the **portA-portB:protocol** format. For example:

```
[customizations.firewall]
ports = ["22:tcp", "80:tcp", "imap:tcp", "53:tcp", "53:udp", "30000-32767:tcp", "30000-
32767:udp"]
```

You can use numeric ports, or their names from the **/etc/services** to enable or disable port lists.

- Specify which firewall services to enable or disable in the **customizations.firewall.service** section:

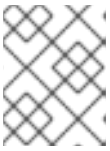
```
[customizations.firewall.services]
enabled = ["<services>"]
disabled = ["<services>"]
```

- You can check the available firewall services:

```
$ firewall-cmd --get-services
```

For example:

```
[customizations.firewall.services]
enabled = ["ftp", "ntp", "dhcp"]
disabled = ["telnet"]
```



NOTE

The services listed in **firewall.services** are different from the **service-names** available in the **/etc/services** file.

5.14. ENABLING OR DISABLING SERVICES

You can control which services to enable during the boot time. Some image types already have services enabled or disabled to ensure that the image works correctly and you cannot override this setup. The **[customizations.services]** settings in the blueprint do not replace these services, but add the services to the list of services already present in the image templates.

- Customize which services to enable during the boot time:

```
[customizations.services]
enabled = ["SERVICES"]
disabled = ["SERVICES"]
```

For example:

```
[customizations.services]
enabled = ["sshd", "cockpit.socket", "httpd"]
disabled = ["postfix", "telnetd"]
```

5.15. INJECTING A KICKSTART FILE IN AN ISO IMAGE

You can use the **[customization.installer]** blueprint customization to add your own Kickstart file in your builds for ISO installers such as, **image installer** or **edge installer**, and gain more flexibility when building ISO images for bare metal deployments.

WARNING

Booting the ISO on a machine with an existing operating system or data can be destructive, because the Kickstart is configured to automatically reformat the first disk on the system.

You can choose the following options to add your own Kickstart file:

- Setting all values during the installation process.
- Enabling the **unattended = true** field in the Kickstart, and getting a fully unattended installation with defaults.
- Injecting your own Kickstart by using the Kickstart field. This can result in both a fully unattended installation if you specify all required fields, or the installation program prompts you for some fields that might be missing.

The Anaconda installer ISO image types support the following blueprint customization:

```
[customizations.installer]
unattended = true
sudo-nopasswd = ["user", "%wheel"]
```

unattended: Creates a Kickstart file that makes the installation fully automatic. This includes setting the following options by default:

- text display mode
- en_US.UTF-8 language/locale
- us keyboard layout
- UTC time zone
- zerombr, clearpart, and autopart to automatically wipe and partition the first disk
- network options to enable dhcp and auto-activation

The following is an example:

```
liveimg --url file:///run/install/repo/liveimg.tar.gz
lang en_US.UTF-8
keyboard us
timezone UTC
zerombr
clearpart --all --initlabel
text
autopart --type=plain --fstype=xfs --nohome
reboot --eject
network --device=link --bootproto=dhcp --onboot=on --activate
```

sudo-nopasswd: Adds a snippet to the Kickstart file that, after installation, creates drop-in files in **/etc/sudoers.d** to allow the specified users and groups to run sudo without a password. The groups must be prefixed with **%**. For example, setting the value to **["user", "%wheel"]** creates the following Kickstart **%post** section:

```
%post
echo -e "user\tALL=(ALL)\tNOPASSWD: ALL" > "/etc/sudoers.d/user"
chmod 0440 /etc/sudoers.d/user
echo -e "%wheel\tALL=(ALL)\tNOPASSWD: ALL" > "/etc/sudoers.d/%wheel"
chmod 0440 /etc/sudoers.d/%wheel
restorecon -rvF /etc/sudoers.d
%end
```

Installer Kickstart

As an alternative, you can include a custom Kickstart by using the following customization:

```
[customizations.installer.kickstart]
contents = """
text --non-interactive
zerombr
clearpart --all --initlabel --disklabel=gpt
autopart --noswap --type=lvm
network --bootproto=dhcp --device=link --activate --onboot=on
"""
```

osbuild-composer automatically adds the command that installs the system: **liveimg** or **ostreesetup**, if it is relevant for the **image-installer**, or **edge-installer** image types. You cannot use the **[customizations.installer.kickstart]** customization in combination with any other installer customizations.

5.16. SPECIFYING A PARTITION MODE

Use the **partitioning_mode** variable to select how to partition the disk image that you are building. You can customize your image with the following supported modes:

- **auto-lvm**: It uses the raw partition mode, unless there are one or more filesystem customizations. In that case, it uses the LVM partition mode.
- **lvm**: It always uses the LVM partition mode, even when there are no extra mountpoints.
- **raw**: It uses raw partitions even when there are one or more mountpoints.
- You can customize your blueprint with the **partitioning_mode** variable by using the following customization:

  ```
  [customizations]
  partitioning_mode = "lvm"
  ```

5.17. SPECIFYING A CUSTOM FILESYSTEM CONFIGURATION

You can specify a custom filesystem configuration in your blueprints and therefore create images with a specific disk layout, instead of the default layout configuration. By using the non-default layout configuration in your blueprints, you can benefit from:

- Security benchmark compliance
- Protection against out-of-disk errors
- Improved performance
- Consistency with existing setups

NOTE

The OSTree systems do not support the filesystem customizations, because OSTree images have their own mount rule, such as read-only. The following image types are not supported:

- **image-installer**
- **edge-installer**
- **edge-simplified-installer**

Additionally, the following image types do not support filesystem customizations, because these image types do not create partitioned operating system images:

- **edge-commit**
- **edge-container**
- **tar**
- **container**

For release distributions before 9.4, the blueprint supports the following **mountpoints** and their sub-directories:

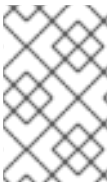
- / - the root mount point
- **/var**
- **/home**
- **/opt**
- **/srv**
- **/usr**
- **/app**
- **/data**
- **/tmp**

From the RHEL 9.4 and 8.10 release distributions onward, you can specify arbitrary custom mountpoints, except for specific paths that are reserved for the operating system.

You cannot specify arbitrary custom mountpoints on the following mountpoints and their sub-directories:

- **/bin**
- **/dev**
- **/etc**
- **/lib**
- **/lib64**
- **/lost+found**
- **/proc**
- **/run**
- **/sbin**
- **/sys**
- **/sysroot**
- **/var/lock**
- **/var/run**

You can customize the filesystem in the blueprint for the /**usr** custom mountpoint, but its subdirectory is not allowed.



NOTE

Customizing mount points is only supported from RHEL 9.0 distributions onward, by using the CLI. In earlier distributions, you can only specify the **root** partition as a mount point and specify the **size** argument as an alias for the image size.

If you have more than one partition in the customized image, you can create images with a customized file system partition on LVM and resize those partitions at runtime. To do this, you can specify a customized filesystem configuration in your blueprint and therefore create images with the required disk layout. The default filesystem layout remains unchanged - if you use plain images without file system customization, and **cloud-init** resizes the root partition.

The blueprint automatically converts the file system customization to an LVM partition.

You can use the custom file blueprint customization to create new files or to replace existing files. The parent directory of the file you specify must exist, otherwise, the image build fails. Ensure that the parent directory exists by specifying it in the **[[customizations.directories]]** customization.

WARNING

If you combine the files customizations with other blueprint customizations, it might affect the functioning of the other customizations, or it might override the current files customizations.

5.17.1. Specifying customized files in the blueprint

With the **[[customizations.files]]** blueprint customization you can:

- Create new text files.
- Modifying existing files. WARNING: this can override the existing content.
- Set user and group ownership for the file you are creating.
- Set the mode permission in the octal format.

You cannot create or replace the following files:

- **/etc/fstab**
- **/etc/shadow**
- **/etc/passwd**
- **/etc/group**

You can create customized files and directories in your image, by using the **[[customizations.files]]** and the **[[customizations.directories]]** blueprint customizations. You can use these customizations only in the **/etc** directory.

NOTE

These blueprint customizations are supported by all image types, except the image types that deploy OSTree commits, such as **edge-raw-image**, **edge-installer**, and **edge-simplified-installer**.

WARNING

If you use the **customizations.directories** with a directory path which already exists in the image with **mode**, **user** or **group** already set, the image build fails to prevent changing the ownership or permissions of the existing directory.

5.17.2. Specifying customized directories in the blueprint

With the **[[customizations.directories]]** blueprint customization you can:

- Create new directories.
- Set user and group ownership for the directory you are creating.
- Set the directory mode permission in the octal format.
- Ensure that parent directories are created as needed.

With the **[[customizations.files]]** blueprint customization you can:

- Create new text files.
- Modifying existing files. WARNING: this can override the existing content.
- Set user and group ownership for the file you are creating.
- Set the mode permission in the octal format.

NOTE

You cannot create or replace the following files:

- **/etc/fstab**
- **/etc/shadow**
- **/etc/passwd**
- **/etc/group**

The following customizations are available:

- Customize the filesystem configuration in your blueprint:

```
[[customizations.filesystem]]
mountpoint = "MOUNTPOINT"
minsize = MINIMUM-PARTITION-SIZE
```

The **MINIMUM-PARTITION-SIZE** value has no default size format. The blueprint customization supports the following values and units: kB to TB and KiB to TiB. For example, you can define the mount point size in bytes:

```
[[customizations.filesystem]]
mountpoint = "/var"
minsize = 1073741824
```

- Define the mount point size by using units. For example:

```
[[customizations.filesystem]]
mountpoint = "/opt"
minsize = "20 GiB"
```

```
[[customizations.filesystem]]
mountpoint = "/var"
minsize = "1 GiB"
```

- Define the minimum partition by setting **minsize**. For example:

```
[[customizations.filesystem]]
mountpoint = "/var"
minsize = 2147483648
```

- Create customized directories under the **/etc** directory for your image by using **[[customizations.directories]]**:

```
[[customizations.directories]]
path = "/etc/directory_name"
mode = "octal_access_permission"
user = "user_string_or_integer"
group = "group_string_or_integer"
ensure_parents = boolean
```

The blueprint entries are described as following:

- **path** - Mandatory - enter the path to the directory that you want to create. It must be an absolute path under the /**etc** directory.
- **mode** - Optional - set the access permission on the directory, in the octal format. If you do not specify a permission, it defaults to 0755. The leading zero is optional.
- **user** - Optional - set a user as the owner of the directory. If you do not specify a user, it defaults to **root**. You can specify the user as a string or as an integer.
- **group** - Optional - set a group as the owner of the directory. If you do not specify a group, it defaults to **root**. You can specify the group as a string or as an integer.
- **ensure_parents** - Optional - Specify whether you want to create parent directories as needed. If you do not specify a value, it defaults to **false**.

- Create customized file under the /**etc** directory for your image by using **[[customizations.directories]]**:

```
[[customizations.files]]
path = "/etc/directory_name"
mode = "octal_access_permission"
user = "user_string_or_integer"
group = "group_string_or_integer"
data = "Hello world!"
```

The blueprint entries are described as following:

- **path** - Mandatory - enter the path to the file that you want to create. It must be an absolute path under the /**etc** directory.
- **mode** Optional - set the access permission on the file, in the octal format. If you do not specify a permission, it defaults to 0644. The leading zero is optional.
- **user** - Optional - set a user as the owner of the file. If you do not specify a user, it defaults to **root**. You can specify the user as a string or as an integer.
- **group** - Optional - set a group as the owner of the file. If you do not specify a group, it defaults to **root**. You can specify the group as a string or as an integer.
- **data** - Optional - Specify the content of a plain text file. If you do not specify a content, it creates an empty file.

5.18. SPECIFYING VOLUME GROUPS AND LOGICAL VOLUMES NAMING IN THE BLUEPRINT

You can use RHEL image builder for the following operations:

- Create RHEL disk images with advanced partitioning layout. You can create disk images with custom mount points, LVM-based partitions and LVM-based SWAP. For example, change the size of the / and the /**boot** directories by using the **config.toml** file.
- Select which file system to use. You can choose between **ext4** and **xfs**.
- Add swap partitions and LVs. The disk images can contain LV-based SWAP.
- Change names of LVM entities. The Logical Volumes (LV) and Volume Groups (VG) inside the images can have custom names.

The following options are not supported:

- Multiple PVs or VGs in one image.
- SWAP files
- Mount options for non-physical partitions, such as /**dev**/**shm**, and /**tmp**.

Example: Adding the VG and LG customization names where the file systems reside.

```
[[customizations.disk.partitions]]
type = "plain"
label = "data"
mountpoint = "/data"
fs_type = "ext4"
minsize = "50 GiB"

[[customizations.disk.partitions]]
type = "lvm"
name = "mainvg"
minsize = "20 GiB"

[[customizations.disk.partitions.logical_volumes]]
name = "rootlv"
mountpoint = "/"
label = "root"
fs_type = "ext4"
minsize = "2 GiB"

[[customizations.disk.partitions.logical_volumes]]
name = "homelv"
mountpoint = "/home"
label = "home"
fs_type = "ext4"
minsize = "2 GiB"

[[customizations.disk.partitions.logical_volumes]]
name = "swaplv"
fs_type = "swap"
minsize = "1 GiB"
```

CHAPTER 6. CREATING SYSTEM IMAGES BY USING RHEL IMAGE BUILDER WEB CONSOLE INTERFACE

RHEL image builder is a tool for creating custom system images. To control RHEL image builder and create your custom system images, you can use the web console interface.

6.1. ACCESSING THE RHEL IMAGE BUILDER DASHBOARD IN THE RHEL WEB CONSOLE

With the **cockpit-image-builder** plugin for the RHEL web console, you can manage image builder blueprints and composes by using a graphical interface.

Prerequisites

- You must have root access to the system.
- You installed RHEL image builder.
- You installed the **cockpit-image-builder** package.

Procedure

1. On the host, open **https://localhost**:**9090**/ in a web browser.
2. Log in to the web console as the root user.
3. To display the RHEL image builder controls, click the **Image Builder** button, in the upper-left corner of the window.
   The RHEL image builder dashboard opens, listing existing blueprints, if any.

Additional resources

- Managing systems using the RHEL web console

6.2. CREATING A BLUEPRINT IN THE WEB CONSOLE INTERFACE

Creating a blueprint is a necessary step before creating the customized RHEL system image. All the customizations are optional.

Prerequisites

- You have opened the RHEL image builder app from the web console in a browser. See Accessing RHEL image builder GUI in the RHEL web console .

Procedure

1. Click **Create Blueprint** in the upper-right corner.
   A dialog wizard with fields for the blueprint name and description opens.
2. On the **Details** page:
   a. Enter the name of the blueprint and, optionally, its description.

b. Click **Next**.

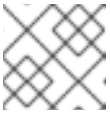
3. Optional: In the **Packages** page:

   a. On the **Available packages** search, enter the package name.

   b. Click the button to move it to the **Chosen packages** field.

   c. Repeat the previous steps to search and include as many packages as you want.

   d. Click **Next**.



NOTE

These customizations are all optional unless otherwise specified.

4. On the **Kernel** page, enter a kernel name and the command-line arguments.

5. On the **File system** page, you can select **Use automatic partitioning** or **Manually configure partitions** for your image file system. For manually configuring the partitions, complete the following steps:

   a. Click the **Manually configure partitions** button.
   The **Configure partitions** section opens, showing the configuration based on Red Hat standards and security guides.

   b. From the dropdown menu, provide details to configure the partitions:

      i. For the **Mount point** field, select one of the following mount point type options:

      - / - the root mount point
      - **/var**
      - **/home**
      - **/opt**
      - **/srv**
      - **/usr**
      - **/app**
      - **/data**
      - **/tmp**
      - **/usr/local**
      You can also add an additional path to the **Mount point**, such as **/tmp**. For example: **/var** as a prefix and **/tmp** as an additional path results in **/var/tmp**.

NOTE

Depending on the Mount point type you choose, the file system type changes to **xfs**.

ii. For the **Minimum size partition** field of the file system, enter the needed minimum partition size. In the Minimum size dropdown menu, you can use common size units such as **GiB**, **MiB**, or **KiB**. The default unit is **GiB**.

NOTE

Minimum size means that RHEL image builder can still increase the partition sizes, in case they are too small to create a working image.

c. To add more partitions, click the **Add partition** button. If you see the following error message: **Duplicate partitions: Only one partition at each mount point can be created.**, you can:

i. Click the **Remove** button to remove the duplicated partition.

ii. Choose a new mount point for the partition you want to create.

d. After you finish the partitioning configuration, click **Next**.

6. On the **Services** page, you can enable or disable services:

a. Enter the service names you want to enable or disable, separating them by a comma, by space, or by pressing the **Enter** key. Click **Next**.

7. On the **Firewall** page, set up your firewall setting:

a. Enter the **Ports**, and the firewall services you want to enable or disable.

b. Click the **Add zone** button to manage your firewall rules for each zone independently. Click **Next**.

8. On the **Users** page, add a users by following the steps:

a. Click **Add user**.

b. Enter a **Username**, a **password**, and a **SSH key**. You can also mark the user as a privileged user, by clicking the **Server administrator** checkbox. Click **Next**.

9. On the **Groups** page, add groups by completing the following steps:

a. Click the **Add groups** button:

i. Enter a **Group name** and a **Group ID**. You can add more groups. Click **Next**.

10. On the **SSH keys** page, add a key:

a. Click the **Add key** button.

i. Enter the SSH key.

ii. Enter a **User**. Click **Next**.

11. On the **Timezone** page, set your time zone settings:

a. On the **Timezone** field, enter the time zone you want to add to your system image. For example, add the following time zone format: "US/Eastern".
If you do not set a time zone, the system uses Universal Time, Coordinated (UTC) as default.

b. Enter the **NTP** servers. Click **Next**

12. On the **Locale** page, complete the following steps:

    a. On the **Keyboard** search field, enter the package name you want to add to your system image. For example: ["en_US.UTF-8"].

    b. On the **Languages** search field, enter the package name you want to add to your system image. For example: "us". Click **Next**.

13. On the **Others** page, complete the following steps:

    a. On the **Hostname** field, enter the hostname you want to add to your system image. If you do not add a hostname, the operating system determines the hostname.

    b. Mandatory only for the Simplifier Installer image: On the **Installation Devices** field, enter a valid node for your system image. For example: **dev/sda1**. Click **Next**.

14. Mandatory only when building FIDO images: On the **FIDO device onboarding** page, complete the following steps:

    a. On the **Manufacturing server URL** field, enter the following information:

        i. On the **DIUN public key insecure** field, enter the insecure public key.

        ii. On the **DIUN public key hash** field, enter the public key hash.

        iii. On the **DIUN public key root certs** field, enter the public key root certs. Click **Next**

15. On the **OpenSCAP** page, complete the following steps:

    a. On the **Datastream** field, enter the **datastream** remediation instructions you want to add to your system image.

    b. On the **Profile ID** field, enter the **profile_id** security profile you want to add to your system image. Click **Next**

16. Mandatory only when building Ignition images: On the **Ignition** page, complete the following steps:

    a. On the **Firstboot URL** field, enter the package name you want to add to your system image.

    b. On the **Embedded Data** field, drag or upload your file. Click **Next**.

17. . On the **Review** page, review the details about the blueprint. Click **Create**.

The RHEL image builder view opens, listing existing blueprints.

6.3. IMPORTING A BLUEPRINT IN THE RHEL IMAGE BUILDER WEB CONSOLE INTERFACE

You can import and use an already existing blueprint. The system automatically resolves all the dependencies.

Prerequisites

- You have opened the RHEL image builder app from the web console in a browser.

- You have a blueprint that you want to import to use in the RHEL image builder web console interface.

Procedure

1. On the RHEL image builder dashboard, click **Import blueprint**. The **Import blueprint** wizard opens.
2. From the **Upload** field, either drag or upload an existing blueprint. This blueprint can be in either **TOML** or **JSON** format.
3. Click **Import**. The dashboard lists the blueprint you imported.

Verification

When you click the blueprint you imported, you have access to a dashboard with all the customizations for the blueprint that you imported.

- To verify the packages that have been selected for the imported blueprint, navigate to the **Packages** tab.
  - To list all the package dependencies, click **All**. The list is searchable and can be ordered.

Next steps

- Optional: To modify any customization:
  - From the **Customizations** dashboard, click the customization you want to make a change. Optionally, you can click **Edit blueprint** to navigate to all the available customization options.

Additional resources

- Creating a system image by using RHEL image builder in the web console interface

6.4. EXPORTING A BLUEPRINT FROM THE RHEL IMAGE BUILDER WEB CONSOLE INTERFACE

You can export a blueprint to use the customizations in another system. You can export the blueprint in the **TOML** or in the **JSON** format. Both formats work on the CLI and also in the API interface.

Prerequisites

- You have opened the RHEL image builder app from the web console in a browser.
- You have a blueprint that you want to export.

Procedure

1. On the image builder dashboard, select the blueprint you want to export.
2. Click **Export blueprint**. The **Export blueprint** wizard opens.
3. Click the **Export** button to download the blueprint as a file or click the **Copy** button to copy the blueprint to the clipboard.

a. Optional: Click the **Copy** button to copy the blueprint.

Verification

- Open the exported blueprint in a text editor to inspect and review it.

6.5. CREATING A SYSTEM IMAGE BY USING RHEL IMAGE BUILDER IN THE WEB CONSOLE INTERFACE

You can create a customized RHEL system image from a blueprint by completing the following steps.

Prerequisites

- You opened the RHEL image builder app from the web console in a browser.
- You created a blueprint.

Procedure

1. In the RHEL image builder dashboard, click the blueprint tab.
2. On the blueprint table, find the blueprint you want to build an image.
3. On the right side of the chosen blueprint, click **Create Image**. The **Create image** dialog wizard opens.
4. On the **Image output** page, complete the following steps:
   a. From the **Select a blueprint** list, select the image type you want.
   b. From the **Image output type** list, select the image output type you want. Depending on the image type you select, you need to add further details.
5. Click **Next**.
6. On the **Review** page, review the details about the image creation and click **Create image**. The image build starts and takes up to 20 minutes to complete.

Verification

After the image finishes building, you can:

- Download the image.
  - On the RHEL image builder dashboard, click the **Node options (▮)** menu and select **Download image**.
- Download the logs of the image to inspect the elements and verify if any issue is found.
  - On the RHEL image builder dashboard, click the **Node options (▮)** menu and select **Download logs**.

CHAPTER 7. CREATING A BOOT ISO INSTALLER IMAGE WITH RHEL IMAGE BUILDER

You can use RHEL image builder to create bootable ISO Installer images. These images consist of a **.tar** file that has a root file system. You can use the bootable ISO image to install the file system to a bare metal server.

RHEL image builder builds a manifest that creates a boot ISO that contains a root file system. To create the ISO image, select the image type **image-installer**. RHEL image builder builds a **.tar** file with the following content:

- a standard Anaconda installer ISO
- an embedded RHEL system tar file
- a default Kickstart file that installs the commit with minimal default requirements

The created installer ISO image includes a pre-configured system image that you can install directly to a bare metal server.

7.1. CREATING A BOOT ISO INSTALLER IMAGE USING THE RHEL IMAGE BUILDER CLI

You can create a customized boot ISO installer image by using the RHEL image builder command-line interface. As a result, image builder builds an **.iso** file that contains a **.tar** file, which you can install for the Operating system. The **.iso** file is set up to boot Anaconda and install the **.tar** file to set up the system. You can use the created ISO image file on a hard disk or to boot in a virtual machine, for example, in an HTTP Boot or a USB installation.

WARNING

The Installer (**.iso**) image type does not accept partitions customization. If you try to manually configure the filesystem customization, it is not applied to any system built by the Installer image. Mounting an ISO image built with RHEL image builder file system customizations causes an error in the Kickstart, and the installation does not reboot automatically. For more information, see Automate a RHEL ISO installation generated by image builder.

Prerequisites

- You have created a blueprint for the image and customized it with a user included and pushed it back into RHEL image builder. See Supported blueprint customizations.

Procedure

1. Create the ISO image:

```
# composer-cli compose start <blueprint_name> <image_installer>
```

- *<blueprint_name>* with name of the blueprint you created
- *<image_installer>* is the image type
  The compose process starts in the background and the UUID of the compose is shown. Wait until the compose is finished. This might take several minutes.

2. Check the status of the compose:

```
# composer-cli compose status
```

   A finished compose shows a status value of **FINISHED**.

3. Identify the compose in the list by its UUID.

```
# composer-cli compose list
```

4. After the compose is finished, download the created image file to the current directory:

```
# composer-cli compose image <uuid>
```

   Replace *<uuid>* with the UUID value obtained in the previous steps.

   RHEL image builder builds a **.iso** file that contains a **.tar** file. The **.tar** file is the image that will be installed for the Operating system. The **.iso** is set up to boot Anaconda and install the **.tar** file to set up the system.

Next steps

In the directory where you downloaded the image file.

1. Locate the **.iso** image you downloaded.
2. Mount the ISO.

```
$ mount -o ro <path_to_iso> /mnt
```

   You can find the **.tar** file at the **/mnt/liveimg.tar.gz** directory.

3. List the **.tar** file content:

```
$ tar ztvf /mnt/liveimg.tar.gz
```

Additional resources

- Creating system images with RHEL image builder command-line interface
- Creating a bootable installation medium for RHEL

7.2. CREATING A BOOT ISO INSTALLER IMAGE BY USING RHEL IMAGE BUILDER IN THE GUI

You can build a customized boot ISO installer image by using the RHEL image builder GUI. You can use the resulting ISO image file on a hard disk or boot it in a virtual machine. For example, in an HTTP Boot or a USB installation.

WARNING

The Installer (**.iso**) image type does not accept partitions customization. If you try to manually configure the filesystem customization, it is not applied to any system built by the Installer image. Mounting an ISO image built with RHEL image builder file system customizations causes an error in the Kickstart, and the installation does not reboot automatically. For more information, see Automate a RHEL ISO installation generated by image builder.

Prerequisites

- You have opened the RHEL image builder app from the web console in a browser.
- You have created a blueprint for your image. See Creating a RHEL image builder blueprint in the web console interface.

Procedure

1. On the RHEL image builder dashboard, locate the blueprint that you want to use to build your image. Optionally, enter the blueprint name or a part of it into the search box at upper left, and click **Enter**.
2. On the right side of the blueprint, click the corresponding **Create Image** button.
   The **Create image** dialog wizard opens.
3. On the **Create image** dialog wizard:
   a. In the **Image Type** list, select **"RHEL Installer (.iso)"**.
   b. Click **Next**.
   c. On the **Review** tab, click **Create**.
   RHEL image builder adds the compose of a RHEL ISO image to the queue.

   After the process is complete, you can see the image build complete status. RHEL image builder creates the ISO image.

Verification

After the image is successfully created, you can download it.

1. Click **Download** to save the **"RHEL Installer (.iso)"** image to your system.
2. Navigate to the folder where you downloaded the **"RHEL Installer (.iso)"** image.
3. Locate the **.tar** image you downloaded.
4. Extract the **"RHEL Installer (.iso)"** image content.

```
$ tar -xf <content>.tar
```

Additional resources

- Creating a bootable installation medium for RHEL

7.3. INSTALLING A BOOTABLE ISO TO A MEDIA AND BOOTING IT

Install the bootable ISO image you created by using RHEL image builder to a bare metal system.

Prerequisites

- You created the bootable ISO image by using RHEL image builder.
- You have downloaded the bootable ISO image.
- You installed the **dd** tool.
- You have a USB flash drive with enough capacity for the ISO image. The required size varies depending on the packages you selected in your blueprint, but the minimum size is 8 GB.

Procedure

1. Write the bootable ISO image directly to the USB drive using the **dd** tool. For example:

   ```
   dd if=installer.iso of=/dev/sdX
   ```

   Where **installer.iso** is the ISO image file name and **/dev/sdX** is your USB flash drive device path.

2. Insert the flash drive into a USB port of the computer you want to boot.
3. Boot the ISO image from the USB flash drive.
   When the installation environment starts, you might need to complete the installation manually, similarly to the default Red Hat Enterprise Linux installation.

Additional resources

- Booting the installation media
- Customizing your installation
- Creating a bootable USB device on Linux

CHAPTER 8. CREATING PRE-HARDENED IMAGES WITH RHEL IMAGE BUILDER OPENSCAP INTEGRATION

RHEL image builder on-premise supports the OpenSCAP integration. This integration enables the production of pre-hardened RHEL images. By setting up a blueprint, you can perform the following actions:

- Customize it with a set of predefined security profiles
- Add a set of packages or add-on files
- Build a customized RHEL image ready to deploy on your chosen platform that is more suitable to your environment

Red Hat provides regularly updated versions of the security hardening profiles that you can choose when you build your systems so that you can meet your current deployment guidelines.

8.1. THE OPENSCAP BLUEPRINT CUSTOMIZATION

With the OpenSCAP support for blueprint customization, you can generate blueprints from the `scap-security-guide` content for specific security profiles and then use the blueprints to build your own pre-hardened images.

Creating a customized blueprint with OpenSCAP involves the following high level steps:

- Modify the mount points and configure the file system layout according to your specific requirements.
- In the blueprint, Select the OpenSCAP profile. This configures the image to trigger the remediation during the image build in accordance with the selected profile. Also during the image build, OpenSCAP applies a pre-first-boot remediation.

To use the **OpenSCAP** blueprint customization in your image blueprints, you need to provide the following information:

- The data stream path to the **datastream** remediation instructions. The data stream files from **scap-security-guide** package are located in the **/usr/share/xml/scap/ssg/content/** directory.
- The **profile_id** of the required security profile. The value of the **profile_id** field accepts both the long and short forms, for example, the following are acceptable: **cis** or **xccdf_org.ssgproject.content_profile_cis**. See SCAP Security Guide profiles supported in RHEL 10 for more details.

The following example is a snippet with the **OpenSCAP** remediation stage:

```
[customizations.openscap]
# If you want to use the data stream from the 'scap-security-guide' package
# the 'datastream' key could be omitted.
# datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel10-ds.xml"
profile_id = "xccdf_org.ssgproject.content_profile_cis"
```

You can find more details about the **SCAP** source data stream from the **scap-security-guide** package, including the list of security profiles it provides, by using the command:

```
# oscap info /usr/share/xml/scap/ssg/content/ssg-rhel10-ds.xml
```

For your convenience the **OpenSCAP** tool can generate the hardening blueprint for any profile available in **scap-security-guide** data streams.

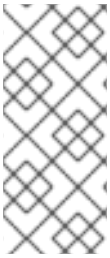
For example, the command:

```
# oscap xccdf generate fix --profile=cis --fix-type=blueprint /usr/share/xml/scap/ssg/content/ssg-rhel10-ds.xml
```

generates a blueprint for CIS profile similar to:

```
# Blueprint for CIS Red Hat Enterprise Linux 10.0 Benchmark for Level 2 - Server

# Profile Description:
# This profile defines a baseline that aligns to the "Level 2 - Server"
# configuration from the Center for Internet Security® Red Hat Enterprise
# Linux 10 Benchmark™, v3.0.0, released 2023-10-30.
# This profile includes Center for Internet Security®
# Red Hat Enterprise Linux 10.0 CIS Benchmarks™ content.
#
# Profile ID:  xccdf_org.ssgproject.content_profile_cis
# Benchmark ID:  xccdf_org.ssgproject.content_benchmark_RHEL-10.0
# Benchmark Version:  0.1.74
# XCCDF Version:  1.2

name = "hardened_xccdf_org.ssgproject.content_profile_cis"
description = "CIS Red Hat Enterprise Linux 10.0 Benchmark for Level 2 - Server"
version = "0.1.74"

[customizations.openscap]
profile_id = "xccdf_org.ssgproject.content_profile_cis"
# If your hardening data stream is not part of the 'scap-security-guide' package
# provide the absolute path to it (from the root of the image filesystem).
# datastream = "/usr/share/xml/scap/ssg/content/ssg-xxxxx-ds.xml"

[[customizations.filesystem]]
mountpoint = "/home"
size = 1073741824

[[customizations.filesystem]]
mountpoint = "/tmp"
size = 1073741824

[[customizations.filesystem]]
mountpoint = "/var"
size = 3221225472

[[customizations.filesystem]]
mountpoint = "/var/tmp"
size = 1073741824

[[packages]]
name = "aide"
version = "*"

[[packages]]
```

```
name = "libselinux"
version = "*"

[[packages]]
name = "audit"
version = "*"

[customizations.kernel]
append = "audit_backlog_limit=8192 audit=1"

[customizations.services]
enabled = ["auditd","crond","firewalld","systemd-journald","rsyslog"]
disabled = []
masked = ["nfs-server","rpcbind","autofs","bluetooth","nftables"]
```



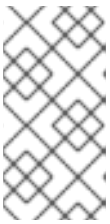
NOTE

Do not use this exact blueprint snippet for image hardening. It does not reflect a complete profile. As Red Hat constantly updates and refines security requirements for each profile in the **scap-security-guide** package, it makes sense to always re-generate the initial template using the most up-to-date version of the data stream provided for your system.

Now you can customize the blueprint or use it as it is to build an image.

RHEL image builder generates the necessary configurations for the **osbuild** stage based on your blueprint customization. Additionally, RHEL image builder adds two packages to the image:

- **openscap-scanner** - the **OpenSCAP** tool.
- **scap-security-guide** - the package which contains the remediation and evaluation instructions.



NOTE

The remediation stage uses the **scap-security-guide** package for the datastream because this package is installed on the image by default. If you want to use a different datastream, add the necessary package to the blueprint, and specify the path to the datastream in the **oscap** configuration.

Additional resources

- SCAP Security Guide profiles supported in RHEL 10

8.2. CREATING A PRE-HARDENED IMAGE WITH RHEL IMAGE BUILDER

With the OpenSCAP and RHEL image builder integration, you can create images that are pre-hardened in compliance with a specific profile, and you can deploy them in a VM, or a bare-metal environment, for example.

Prerequisites

- You are logged in as the root user or a user who is a member of the **weldr** group.
- The **openscap** and **scap-security-guide** packages are installed.

Procedure

1. Create a hardening blueprint in the TOML format, using **OpenSCAP** tool and **scap-security-guide** content, and modify it if necessary:

```
# oscap xccdf generate fix --profile=<profileID> --fix-type=<blueprint>
/usr/share/xml/scap/ssg/content/ssg-rhel10-ds.xml > cis.toml
```

   Replace ***<profileID>*** with the profile ID with which the system should comply, for example, **cis**.

2. Push the blueprint to **osbuild-composer** by using the **composer-cli** tool:

```
# composer-cli blueprints push <blueprint_name>.toml
```

3. Start the build of hardened image:

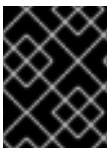
```
# composer-cli compose start <blueprint_name> <image_type>
```

   Replace ***<image_type>*** with any image type, for example, **qcow2**.

   After the image build is ready, you can use your pre-hardened image on your deployments. See Creating a virtual machine.

Verification

After you deploy your pre-hardened image you can perform a configuration compliance scan to verify that the image is aligned with the selected security profile.



IMPORTANT

Performing a configuration compliance scanning does not guarantee the system is compliant. For more information, see Configuration compliance scanning.

Additional resources

- Scanning the system for configuration compliance and vulnerabilities

8.3. CUSTOMIZING A PRE-HARDENED IMAGE WITH RHEL IMAGE BUILDER

You can customize a security profile by changing parameters in certain rules, for example, minimum password length, removing rules that you cover in a different way, and selecting additional rules, to implement internal policies. You cannot define new rules by customizing a profile.

When you build an image from that blueprint, it creates a tailoring file with a new tailoring profile ID and saves it to the image as **/usr/share/xml/osbuild-oscap-tailoring/tailoring.xml**. The new profile ID will have **_osbuild_tailoring** appended as a suffix to the base profile. For example, if you use the CIS (**cis**) base profile, the profile ID will be **xccdf_org.ssgproject.content_profile_cis_osbuild_tailoring**.

Prerequisites

- You are logged in as the root user or a user who is a member of the **weldr** group.
- The **openscap** and **scap-security-guide** packages are installed.

Procedure

1. Create a hardening blueprint in the TOML format from a selected profile. For example:

```
# oscap xccdf generate fix --profile=<profileID> --fix-type=blueprint
/usr/share/xml/scap/ssg/content/ssg-rhel10-ds.xml > <profileID>tailored.toml
```

2. Append the tailoring section with the customized rule set to the blueprint. Note that the tailoring customization will only affect the default selected or unselected state of the rules in the profile in which the customization is based on, by selecting or deselecting a rule, without changing the state of other rules.

```
# Blueprint for CIS Red Hat Enterprise Linux 10.0 Benchmark for Level 2 - Server
# ...
[customizations.openscap.tailoring]
selected = [ "xccdf_org.ssgproject.content_bind_crypto_policy" ]
unselected = [ "grub2_password" ]
```

3. Push the blueprint to **osbuild-composer** by using the **composer-cli** tool:

```
# composer-cli blueprints push <blueprintProfileID>tailored.toml
```

4. Start the build of hardened image:

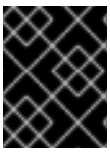
```
# composer-cli compose start <blueprintProfileID> image_type
```

Replace ***<image_type>*** with any image type, for example, **qcow2**.

After the image build is ready, use your pre-hardened image on your deployments.

Verification

After you deploy your pre-hardened image, you can perform a configuration compliance scan to verify that the image is aligned with the selected security profile.



IMPORTANT

Performing a configuration compliance scanning does not guarantee the system is compliant. For more information, see Configuration compliance scanning.

Additional resources

- Scanning the system for configuration compliance and vulnerabilities

CHAPTER 9. ENABLING FIPS MODE WITH RHEL IMAGE BUILDER

You can create a customized image and boot a FIPS-enabled RHEL image. Before you compose the image, you must change the value of the **fips** directive in your blueprint.

Prerequisites

- You are logged in as the root user or a user who is a member of the **weldr** group.

Procedure

1. Create a plain text file in the Tom's Obvious, Minimal Language (TOML) format with the following content:

```
name = "system-fips-mode-enabled"
description = "blueprint with FIPS enabled "
version = "0.0.1"

[customizations]
fips = true

[[customizations.user]]
name = "admin"
password = "admin"
groups = ["users", "wheel"]
```

2. Import the blueprint to the RHEL image builder server:

```
# composer-cli blueprints push blueprint-name.toml
```

3. List the existing blueprints to check whether the created blueprint is successfully imported and exists:

```
# composer-cli blueprints show blueprint-name
```

4. Check whether the components and versions listed in the blueprint and their dependencies are valid:

```
# composer-cli blueprints depsolve blueprint-name
```

5. Build the customized RHEL image:

```
# composer-cli compose start \ blueprint-name \ image-type \
```

6. Review the image status:

```
# composer-cli compose status
…
$ UUID FINISHED date blueprint-name blueprint-version image-type
…
```

7. Download the image:

```
# composer-cli compose image UUID
```

RHEL image builder downloads the image to the current directory path. The UUID number and the image size are displayed alongside:

```
$ UUID-image-name.type: size MB
```

Verification

1. Log in to the system image with the username and password that you configured in your blueprint.
2. Check if FIPS mode is enabled:

```
$ cat /proc/sys/crypto/fips_enabled
1
```

CHAPTER 10. PREPARING AND DEPLOYING A KVM GUEST IMAGE BY USING RHEL IMAGE BUILDER

Use RHEL image builder to create a **.qcow2** purpose-built that you can deploy on a Kernel-based Virtual Machine (KVM) based hypervisor.

Creating a customized KVM guest image involves the following high-level steps:

1. Create a blueprint for the **.qcow2** image.
2. Create a **.qcow2** image by using RHEL image builder.
3. Create a virtual machine from the KVM guest image.

10.1. CREATING CUSTOMIZED KVM GUEST IMAGES BY USING RHEL IMAGE BUILDER

You can create a customized **.qcow2** KVM guest image by using RHEL image builder. The following procedure shows the steps on the GUI, but you can also use the CLI.

Prerequisites

- You must be in the **root** or **weldr** group to access the system.
- The **cockpit-image-builder** package is installed.
- On a RHEL system, you have opened the RHEL image builder dashboard of the web console.
- You have created a blueprint. See Creating a blueprint in the web console interface .

Procedure

1. Click the blueprint name you created.
2. Select the tab **Images**.
3. Click **Create Image** to create your customized image. The **Create Image** window opens.
4. From the **Type** drop-down menu list, select **QEMU Image(.qcow2)**.
5. Set the size that you want the image to be when instantiated and click **Create**.
6. A small pop-up on the upper right side of the window informs you that the image creation has been added to the queue. After the image creation process is complete, you can see the **Image build complete** status.

Verification

- Click the breadcrumbs icon and select the **Download** option. RHEL image builder downloads the KVM guest image **.qcow2** file at your default download location.

Additional resources

- Creating a blueprint in the web console interface

10.2. CREATING A VIRTUAL MACHINE FROM A KVM GUEST IMAGE

RHEL image builder already has **cloud-init** installed and enabled.

Prerequisites

- You created a **.qcow2** image by using RHEL image builder.
- You have the **qemu-kvm** package installed on your system. You can check if the **/dev/kvm** device is available on your system, and virtualization features are enabled in the BIOS.
- You have the **libvirt** and **virt-install** packages installed on your system.
- You have the **genisoimage** utility, that is provided by the **xorriso** package, installed on your system.

Procedure

1. Move the **.qcow2** image that you created by using RHEL image builder to the **/var/lib/libvirt/images/** directory.
2. Create a directory, for example, **cloudinitiso** and navigate to this newly created directory:

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

3. Create a file named **meta-data**. Add the following information to this file:

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

4. Create a file named **user-data**. Add the following information to the file:

```
#cloud-config
user: admin
password: password
chpasswd: {expire: False}
ssh_pwauth: True
ssh_authorized_keys:
  - ssh-rsa AAA...fhHQ== your.email@example.com
```

ssh_authorized_keys is your SSH public key. You can find your SSH public key in **~/.ssh/*id_rsa.pub***\.

5. Use the **genisoimage** utility to create an ISO image that includes the **user-data** and **meta-data** files.

```
# genisoimage -output cloud-init.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)
```

6. Create a new VM from the KVM Guest Image by using the **virt-install** command. Include the ISO image you created on step 4 as an attachment to the VM image.

```
# virt-install \
   --memory 4096 \
   --vcpus 4 \
   --name myvm \
   --disk rhel-10-x86_64-kvm.qcow2,device=disk,bus=virtio,format=qcow2 \
   --disk cloud-init.iso,device=cdrom \
   --os-variant rhel 10 \
   --virt-type kvm \
   --graphics none \
   --import
```

- --graphics none - means it is a headless RHEL 10 VM.
- --vcpus 4 - means that it uses 4 virtual CPUs.
- --memory 4096 - means it uses 4096 MB RAM.

7. The VM installation starts:

```
Starting install...
Connected to domain mytestcivm
...
[  OK  ] Started Execute cloud user/final scripts.
[  OK  ] Reached target Cloud-init target.

Red Hat Enterprise Linux 10 (Ootpa)
Kernel 4.18.0-221.el8.x86_64 on an x86_64
```

Verification

After the boot is complete, the VM shows a text login interface. To log in to the local console of the VM, use user the details from the **user-data** file:

1. Enter **admin** as a username and press **Enter**.
2. Enter **password** as password and press **Enter**.
   After the login authentication is complete, you have access to the VM by using the CLI.

Additional resources

- The Enabling virtualization documentation
- The Files and directories significant for cloud-init documentation

CHAPTER 11. PUSHING A CONTAINER TO A REGISTRY AND EMBEDDING IT INTO AN IMAGE

With RHEL image builder, you can build security hardened images using the OpenSCAP tool. You can take advantage of the support for container customization in the blueprints to create a container and embed it directly into the image you create.

11.1. BLUEPRINT CUSTOMIZATION TO EMBED A CONTAINER INTO AN IMAGE

To embed a container from registry.access.redhat.com registry, you must add a container customization to your blueprint. For example:

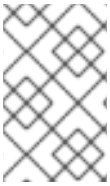
```
[[containers]]
source = "registry.access.redhat.com/ubi10/ubi:latest"
name =  "<local_name>"
tls-verify = true
```

- **source** - Mandatory field. It is a reference to the container image at a registry. This example uses the **registry.access.redhat.com** registry. You can specify a tag version. The default tag version is **latest**.
- **name** - The name of the container in the local registry.
- **tls-verify** - Boolean field. The **tls-verify** boolean field controls the transport layer security. The default value is **true**.
  RHEL image builder pulls the container during the image build and stores the container into the image. The default local container storage location depends on the image type, so that all support **container-tools**, such as Podman, are able to work with it. The embedded containers are not started.

  To access protected container resources, you can use a **containers-auth.json** file.

11.2. THE CONTAINER REGISTRY CREDENTIALS

The **osbuild-worker@.service** is a template service that can start multiple service instances. By default, the **osbuild-composer** service always starts with only one local **osbuild-worker**, specifically **osbuild-worker@1.service**. The **osbuild-worker** service is responsible for the communication with the container registry. To enable the service, set up the **/etc/osbuild-worker/osbuild-worker.toml** configuration file.



NOTE

After setting the **/etc/osbuild-worker/osbuild-worker.toml** configuration file, you must restart the **osbuild-worker** service, because it reads the **/etc/osbuild-worker/osbuild-worker.toml** configuration file only once, during the **osbuild-worker** service start.

To stop the service instance, restart the systemd service with the following command:

```
$ systemctl restart osbuild-worker@*
```

With that, you restart all the started instances of **osbuild-worker**, specifically **osbuild-worker@1.service**, the only service that might be running.

The **/etc/osbuild-worker/osbuild-worker.toml** configuration file has a containers section with an **auth_field_path** entry that is a string referring to a path of a **containers-auth.json** file to be used for accessing protected resources. The container registry credentials are only used to pull a container image from a registry, when embedding the container into the image.

For example:

```
[containers]
auth_file_path = "/etc/osbuild-worker/containers-auth.json"
```

Additional resources

- The **containers-auth.json** man page on your system

11.3. PUSHING A CONTAINER ARTIFACT DIRECTLY TO A CONTAINER REGISTRY

You can push container artifacts, such as RHEL for Edge container images directly, directly to a container registry after you build it, by using the RHEL image builder CLI.

Prerequisites

- Access to quay.io registry. This example uses the **quay.io** container registry as a target registry, but you can use a container registry of your choice.

Procedure

1. Set up a **registry-config.toml** file to select the container provider. The credentials are optional.

```
provider = "container_provider"
[settings]
tls_verify = false
username = "admin"
password = "your_password"
```

2. Create a blueprint in the **.toml** format. This is a blueprint for the container in which you install an **nginx** package into the blueprint.

```
name = "simple-container"
description = "Simple RHEL container"
version = "0.0.1"
[[packages]]
name = "nginx"
version = "*"
```

3. Push the blueprint:

```
# composer-cli blueprints push blueprint.toml
```

4. Build the container image, by passing the registry and the repository to the **composer-cli** tool as arguments.

```
# composer-cli compose start simple-container container "quay.io:8080/osbuild/repository"
registry-config.toml
```

   - simple-container - is the blueprint name.
   - container - is the image type.
   - "quay.io:8080/osbuild/*repository*" - **quay.io** is the target registry, **osbuild** is the organization and **repository** is the location to push the container when it finishes building. Optionally, you can set a **tag**. If you do not set a value for **:tag**, it uses **:latest** tag by default.

NOTE

Building the container image takes time because of resolving dependencies of the customized packages.

5. After the image build finishes, the container you created is available in quay.io.

Verification

1. Open quay.io. and click **Repository Tags**.

```
You can see details about the container you created, such as:
- last modified
- image size
- the `manifest ID`, that you can copy to the clipboard.
```

2. Copy the **manifest ID** value to build the image in which you want to embed a container.

Additional resources

- The Image tags overview documentation

11.4. BUILDING AN IMAGE AND PULLING THE CONTAINER INTO THE IMAGE

After you have created the container image, you can build your customized image and pull the container image into it. For that, you must specify a **container customization** in the blueprint, and the **container name** for the final image. During the build process, the container image is fetched and placed in the local Podman container storage.

Prerequisites

- You created a container image and pushed it into your local **quay.io** container registry instance. See Pushing a container artifact directly to a container registry .
- You have access to registry.access.redhat.com.
- You have a container **manifest ID**.
- You have the **qemu-kvm** and **qemu-img** packages installed.

Procedure

1. Create a blueprint to build a **qcow2** image. The blueprint must contain the " " customization.

```
name = "image"
description = "A qcow2 image with a container"
version = "0.0.1"
distro = "rhel-10"
[[packages]]
name = "podman"
version = "*"
[[containers]]
source = "registry.access.redhat.com/ubi10:8080/osbuild/container/container-image@sha256:manifest-ID-from-Repository-tag: tag-version"
name =  "source-name"
tls-verify = true
```

2. Push the blueprint:

```
# composer-cli blueprints push <blueprint_image>.toml
```

3. Build the container image:

```
# composer-cli start compose <image> qcow2
```

- *image* is the blueprint name.
- **qcow2** is the image type.

NOTE

Building the image takes time because it checks the container on **quay.io** registry.

4. To check the status of the compose:

```
# composer-cli compose status
```

A finished compose shows the **FINISHED** status value. To identify your compose in the list, use its UUID.

5. After the compose process is finished, download the resulting image file to your default download location:

```
# composer-cli compose image <uuid>
```

Replace UUID with the UUID value shown in the previous steps.

You can use the **qcow2** image you created and downloaded to create a VM.

Verification

From the resulting **qcow2** image that you downloaded, perform the following steps:

1. Start the **qcow2** image in a VM. See Creating a virtual machine from a KVM guest image .

2. The **qemu** wizard opens. Login in to the **qcow2** image.

   a. Enter the username and password. These can be the username and password you set up in the **.qcow2** blueprint in the "customizations.user" section, or created at boot time with **cloud-init**.

3. Run the container image and open a shell prompt inside the container:

   ```
   # podman run -it registry.access.redhat.com/ubi10:8080/osbuild/<repository> /bin/bash/
   ```

   registry.access.redhat.com is the target registry, **osbuild** is the organization and **repository** is the location to push the container when it finishes building.

4. Check that the packages you added to the blueprint are available:

   ```
   # type -a nginx
   ```

   The output shows you the **nginx** package path.

Additional resources

- Red Hat Container Registry Authentication
- Accessing and Configuring the Red Hat Registry
- Basic Podman commands
- Building, running, and managing containers

CHAPTER 12. PREPARING AND UPLOADING CLOUD IMAGES BY USING RHEL IMAGE BUILDER

RHEL image builder can create custom system images ready for use on various cloud platforms. To use your customized RHEL system image in a cloud, create the system image with RHEL image builder by using the chosen output type, configure your system for uploading the image, and upload the image to your cloud account.

You can push customized image clouds through the **Image Builder** application in the RHEL web console, available for a subset of the service providers that we support, such as **AWS** and **Microsoft Azure** clouds.

CHAPTER 13. PREPARING AND UPLOADING AMI IMAGES TO AWS

You can create custom images and can update them, either manually or automatically, to the AWS cloud with RHEL image builder.

13.1. PREPARING TO MANUALLY UPLOAD AWS AMI IMAGES

Before uploading an AWS AMI image, you must configure a system for uploading the images.

Prerequisites

- You must have an Access Key ID configured in the AWS IAM account manager.
- You must have a writable S3 bucket prepared.

Procedure

1. Install Python 3 and the **pip** tool:

```
# dnf install python3 python3-pip
```

2. Install the AWS command-line tools with **pip**:

```
# pip3 install awscli
```

3. Set your profile. The terminal prompts you to provide your credentials, region and output format:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

4. Define a name for your bucket and create a bucket:

```
$ BUCKET=bucketname
$ aws s3 mb s3://$BUCKET
```

Replace ***bucketname*** with the actual bucket name. It must be a globally unique name. As a result, your bucket is created.

5. To grant permission to access the S3 bucket, create a **vmimport** S3 Role in the AWS Identity and Access Management (IAM), if you have not already done so in the past:

   a. Create a **trust-policy.json** file with the trust policy configuration, in the JSON format. For example:

```
{
   "Version": "2022-10-17",
   "Statement": [{
      "Effect": "Allow",
```

```
        "Principal": {
            "Service": "vmie.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "sts:Externalid": "vmimport"
            }
        }
    }]
}
```

b. Create a **role-policy.json** file with the role policy configuration, in the JSON format. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:GetBucketLocation", "s3:GetObject", "s3:ListBucket"],
        "Resource": ["arn:aws:s3:::%s", "arn:aws:s3:::%s/"] }, { "Effect": "Allow", "Action":
["ec2:ModifySnapshotAttribute", "ec2:CopySnapshot", "ec2:RegisterImage",
"ec2:Describe"],
        "Resource": "*"
    }]
}
$BUCKET $BUCKET
```

c. Create a role for your Amazon Web Services account, by using the **trust-policy.json** file:

```
$ aws iam create-role --role-name vmimport --assume-role-policy-document file://trust-policy.json
```

d. Embed an inline policy document, by using the **role-policy.json** file:

```
$ aws iam put-role-policy --role-name vmimport --policy-name vmimport --policy-document file://role-policy.json
```

Additional resources

- Using high-level (s3) commands with the AWS CLI

13.2. MANUALLY UPLOADING AN AMI IMAGE TO AWS BY USING THE CLI

You can use RHEL image builder to build **ami** images and manually upload them directly to Amazon AWS Cloud service provider, by using the CLI.

Prerequisites

- You have an **Access Key ID** configured in the AWS IAM account manager.
- You have a writable S3 bucket prepared.

- You have a defined blueprint.

Procedure

1. Using the text editor, create a configuration file with the following content:

```
provider = "aws"
[settings]
accessKeyID = "AWS_ACCESS_KEY_ID"
secretAccessKey = "AWS_SECRET_ACCESS_KEY"
bucket = "AWS_BUCKET"
region = "AWS_REGION"
key = "IMAGE_KEY"
```

   Replace values in the fields with your credentials for **accessKeyID**, **secretAccessKey**, **bucket**, and **region**. The **IMAGE_KEY** value is the name of your VM Image to be uploaded to EC2.

2. Save the file as *CONFIGURATION-FILE*.toml and close the text editor.

3. Start the compose to upload it to AWS:

```
# composer-cli compose start blueprint-name image-type image-key configuration-file.toml
```

   Replace:

   - *blueprint-name* with the name of the blueprint you created
   - *image-type* with the **ami** image type.
   - *image-key* with the name of your VM Image to be uploaded to EC2.
   - *configuration-file*.toml with the name of the configuration file of the cloud provider.

NOTE

You must have the correct AWS Identity and Access Management (IAM) settings for the bucket you are going to send your customized image to. You have to set up a policy to your bucket before you are able to upload images to it.

4. Check the status of the image build:

```
# composer-cli compose status
```

   After the image upload process is complete, you can see the "FINISHED" status.

Verification

To confirm that the image upload was successful:

1. Access EC2 on the menu and select the correct region in the AWS console. The image must have the **available** status, to indicate that it was successfully uploaded.

2. On the dashboard, select your image and click **Launch**.

Additional resources

- Required service role to import a VM

13.3. CREATING AND AUTOMATICALLY UPLOADING IMAGES TO THE AWS CLOUD AMI

You can create a **.raw** image by using RHEL image builder, and choose to check the **Upload to AWS** checkbox to automatically push the output image that you create directly to the **Amazon AWS Cloud AMI** service provider.

Prerequisites

- You must have **root** or **wheel** group user access to the system.
- You have opened the RHEL image builder interface of the RHEL web console in a browser.
- You have created a blueprint. See Creating a blueprint in the web console interface .
- You must have an Access Key ID configured in the AWS IAM account manager.
- You must have a writable S3 bucket prepared.

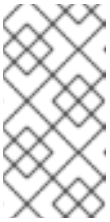
Procedure

1. In the RHEL image builder dashboard, click the **blueprint name** that you previously created.
2. Select the tab **Images**.
3. Click **Create Image** to create your customized image.
   The **Create Image** window opens.
   a. From the **Type** drop-down menu list, select **Amazon Machine Image Disk (.raw)**.
   b. Check the **Upload to AWS** checkbox to upload your image to the AWS Cloud and click **Next**.
   c. To authenticate your access to AWS, type your **AWS access key ID** and **AWS secret access key** in the corresponding fields. Click **Next**.

NOTE

You can view your AWS secret access key only when you create a new Access Key ID. If you do not know your Secret Key, generate a new Access Key ID.

   d. Type the name of the image in the **Image name** field, type the Amazon bucket name in the **Amazon S3 bucket name** field and type the **AWS region** field for the bucket you are going to add your customized image to. Click **Next**.
   e. Review the information and click **Finish**.
   Optionally, click **Back** to modify any incorrect detail.



NOTE

You must have the correct IAM settings for the bucket you are going to send your customized image. This procedure uses the IAM Import and Export, so you have to set up **a policy** to your bucket before you are able to upload images to it. For more information, see Required Permissions for IAM Users .

4. A pop-up on the upper right informs you of the saving progress. It also informs that the image creation has been initiated, the progress of this image creation and the subsequent upload to the AWS Cloud.
   After the process is complete, you can see the **Image build complete** status.

5. In a browser, access Service→EC2.

   a. On the AWS console dashboard menu, choose the correct region. The image must have the **Available** status, to indicate that it is uploaded.

   b. On the AWS dashboard, select your image and click **Launch**.

6. A new window opens. Choose an instance type according to the resources you need to start your image. Click **Review and Launch**.

7. Review your instance start details. You can edit each section if you need to make any changes. Click **Launch**.

8. Before you start the instance, select a public key to access it.
   You can either use the key pair you already have or you can create a new key pair.

   Follow the next steps to create a new key pair in EC2 and attach it to the new instance.

   a. From the drop-down menu list, select **Create a new key pair**.

   b. Enter the name to the new key pair. It generates a new key pair.

   c. Click **Download Key Pair** to save the new key pair on your local system.

9. Then, you can click **Launch Instance** to start your instance.
   You can check the status of the instance, which displays as **Initializing**.

10. After the instance status is **running**, the **Connect** button becomes available.

11. Click **Connect**. A window is displayed with instructions on how to connect by using SSH.

    a. Select **A standalone SSH client** as the preferred connection method to and open a terminal.

    b. In the location you store your private key, ensure that your key is publicly viewable for SSH to work. To do so, run the command:

    ```
    $ chmod 400 <your-instance-name.pem>
    ```

    c. Connect to your instance by using its Public DNS:

    ```
    $ ssh -i <your-instance-name.pem> ec2-user@<your-instance-IP-address>
    ```

    d. Type **yes** to confirm that you want to continue connecting.

As a result, you are connected to your instance over SSH.

Verification

- Check if you are able to perform any action while connected to your instance by using SSH.

Additional resources

- Open a case on Red Hat Customer Portal
- Connecting to your Linux instance by using SSH

CHAPTER 14. PREPARING AND UPLOADING VHD IMAGES TO MICROSOFT AZURE

You can create custom images and can update them, either manually or automatically, to the Microsoft Azure cloud with RHEL image builder.

14.1. PREPARING TO MANUALLY UPLOAD MICROSOFT AZURE VHD IMAGES

To create a VHD image that you can manually upload to **Microsoft Azure** cloud, you can use RHEL image builder.

Prerequisites

- You must have a Microsoft Azure resource group and storage account.
- You have Python installed. The **AZ CLI** tool depends on python.

Procedure

1. Import the Microsoft repository key:

```
$ sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

2. Create a local **azure-cli.repo** repository with the following information. Save the **azure-cli.repo** repository under **/etc/yum.repos.d/**:

```
[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/vscode
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
```

3. Install Microsoft Azure CLI. The downloaded version of the Microsoft Azure CLI package can vary depending on the current available version.

```
$ sudo dnf downloader azure-cli
$ sudo rpm -ivh --nodeps azure-cli-2.0.64-1.el7.x86_64.rpm
```

4. Run Microsoft Azure CLI:

```
$ az login
```

   The terminal shows the following message **Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code**. Then, the terminal opens a browser with a link to "https://microsoft.com/devicelogin" from where you can login.

NOTE

If you are running a remote (SSH) session, the login page link will not open in the browser. In this case, you can copy the link to a browser and login to authenticate your remote session. To sign in, use a web browser to open the "https://microsoft.com/devicelogin" page and enter the device code to authenticate.

5. List the keys for the storage account in Microsoft Azure and make note of value **key1** from the output of the previous command.

```
$ az storage account keys list --resource-group resource-group-name --account-name account-name
```

Replace ***resource-group-name*** with name of your Microsoft Azure resource group and ***storage-account-name*** with name of your Microsoft Azure storage account.

a. To list the available resources using the following command:

```
$ az resource list
```

6. Create a storage container:

```
$ az storage container create --account-name storage-account-name \
--account-key key1-value --name storage-account-name
```

Replace ***storage-account-name*** with name of the storage account.

Additional resources

- Microsoft Azure CLI

14.2. MANUALLY UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

After you have created your customized VHD image, you can manually upload it to the Microsoft Azure cloud. When you create a **.vhd** image by using the CLI, RHEL image builder writes temporary files to the **/var** subdirectory. To prevent the **.vhd** image creation from failing, increase the **/var** subdirectory capacity to at least 15 to 20 GB free space to ensure availability.

Prerequisites

- Your system must be set up for uploading Microsoft Azure VHD images.
- You must have a Microsoft Azure VHD image created by RHEL image builder.
  - In the GUI, use the **Azure Disk Image (.vhd)** image type.
  - In the CLI, use the **vhd** output type.

Procedure

1. Push the image to Microsoft Azure and create an instance from it:

```
$ az storage blob upload --account-name account_name --container-name container_name -
-file image-disk.vhd --name image-disk.vhd --type page
```

2. After the upload to the Microsoft Azure Blob storage completes, create a Microsoft Azure image from it. Because the images that you create with RHEL image builder generate hybrid images that support to both the **V1 = BIOS** and **V2 = UEFI** instances types, you can specify the **--hyper-v-generation** argument. The default instance type is **V1**.

```
$ az image create --resource-group resource_group_name --name image-disk.vhd --os-type
linux --location location \
--source https://$account_name.blob.core.windows.net/container_name/image-disk.vhd
- Running
```

Verification

1. Create an instance either with the Microsoft Azure portal, or a command similar to the following:

```
$ az vm create --resource-group resource_group_name --location location --name vm_name
--image image-disk.vhd --admin-username azure-user --generate-ssh-keys
- Running
```

2. Use your private key via SSH to access the resulting instance. Log in as **azure-user**. This username was set on the previous step.

Additional resources

- Composing an image for the **.vhd** format fails (Red Hat Knowledgebase)

14.3. CREATING AND AUTOMATICALLY UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

You can create **.vhd** images by using RHEL image builder that will be automatically uploaded to a Blob Storage of the Microsoft Azure Cloud service provider.

Prerequisites

- You have root access to the system.
- You have access to the RHEL image builder interface of the RHEL web console.
- You created a blueprint. See Creating a RHEL image builder blueprint in the web console interface.
- You have a Microsoft Storage Account created.
- You have a writable Blob Storage prepared.

Procedure

1. In the RHEL image builder dashboard, select the blueprint you want to use.
2. Click the **Images** tab.

3. Click **Create Image** to create your customized **.vhd** image.
   The **Create image** wizard opens.

   a. Select **Microsoft Azure (.vhd)** from the **Type** drop-down menu list.

   b. Check the **Upload to Azure** checkbox to upload your image to the Microsoft Azure Cloud.

   c. Enter the **Image Size** and click **Next**.

4. On the **Upload to Azure** page, enter the following information:

   a. On the Authentication page, enter:

      i. Your **Storage account** name. You can find it on the **Storage account** page, in the Microsoft Azure portal.

      ii. Your **Storage access key**. You can find it on the **Access Key** Storage page.

      iii. Click **Next**.

   b. On the **Authentication** page, enter:

      i. The image name.

      ii. The **Storage container**. It is the blob container to which you will upload the image. Find it under the **Blob service** section, in the Microsoft Azure portal.

      iii. Click **Next**.

5. On the **Review** page, click **Create**. The RHEL image builder and upload processes start.
   Access the image you pushed into **Microsoft Azure Cloud**.

6. Access the Microsoft Azure portal.

7. In the search bar, type "storage account" and click **Storage accounts** from the list.

8. On the search bar, type "Images" and select the first entry under **Services**. You are redirected to the **Image dashboard**.

9. On the navigation panel, click **Containers**.

10. Find the container you created. Inside the container is the **.vhd** file you created and pushed by using RHEL image builder.

Verification

1. Verify that you can create a VM image and launch it.

   a. In the search bar, type images account and click **Images** from the list.

   b. Click **+Create**.

   c. From the dropdown list, choose the resource group you used earlier.

   d. Enter a name for the image.

   e. For the **OS type**, select **Linux**.

f. For the **VM generation**, select **Gen 2**.

g. Under **Storage Blob**, click **Browse** and click through the storage accounts and container until you reach your VHD file.

h. Click **Select** at the end of the page.

i. Choose an Account Type, for example, **Standard SSD**.

j. Click **Review + Create** and then **Create**. Wait a few moments for the image creation.

2. To launch the VM, follow the steps:

a. Click **Go to resource**.

b. Click **+ Create VM** from the menu bar on the header.

c. Enter a name for your virtual machine.

d. Complete the **Size** and **Administrator account** sections.

e. Click **Review + Create** and then **Create**. You can see the deployment progress. After the deployment finishes, click the virtual machine name to retrieve the public IP address of the instance to connect by using SSH.

f. Open a terminal to create an SSH connection to connect to the VM.

Additional resources

- Microsoft Azure Storage Documentation
- Create a Microsoft Azure Storage account

CHAPTER 15. PREPARING AND UPLOADING VMDK CUSTOM IMAGES TO VSPHERE

You can create custom images and can update them, either manually or automatically, to the VMware vSphere cloud with RHEL image builder.

15.1. CREATING AND AUTOMATICALLY UPLOADING CUSTOMIZED RHEL VMDK IMAGES BY USING IMAGE BUILDER

With RHEL image builder, you can create customized system images in the Open virtualization format (**.ova**), and automatically upload these images to the VMware vSphere client. The Open virtualization format (**.ova**) is a **.vmdk** image with additional metadata about the virtual hardware, which contains a minimal template to make it easier to import images into vSphere. The **.ovf** (Open Virtualization Format) package is part of the vSphere **.ova** image. After RHEL image builder finishes importing the **.ova** image to the vSphere client, you can configure it with any additional hardware, such as network, disks, and CD ROMS.

You can import the Open virtualization format (**.ova**) image by using either the vSphere GUI or the **govc** client. To upload the image by using the **govc** client, see Uploading VMDK images and creating a RHEL virtual machine in vSphere.

Prerequisites

- You opened the RHEL image builder app from the web console in a browser.
- You created a blueprint.

Procedure

1. In the RHEL image builder dashboard, click the **Blueprint** tab.
2. On the blueprint table, find the blueprint you want to build an image.
3. On the right side of the chosen blueprint, click **Create Image**. The **Create image** dialog wizard opens.
4. On the **Image output** page, complete the following steps:
    a. From the **Select a blueprint** list, select the image type you want.
    b. From the **Image output type** list, select the image output type you want.
    c. Optional: Check **Upload to VMware** checkbox to upload the image directly to VMware.
    d. Enter a size for the image.
    e. Click **Next**.
5. On the **Upload to VMware** page, enter the following information:
    a. **Image name**: Enter the image name.
    b. **Host**: Enter the VMware vSphere instance URL where the image file will be uploaded.
    c. **Cluster**: Enter the cluster name page to which the image will be uploaded.

d. **Datacenter**: The data center name to which the image will be uploaded.

e. **Datastore**: The data store name to which the image will be uploaded.

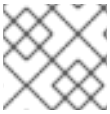
f. **Folder**: Folder name to which the image will be uploaded.

g. Click **Next**.

6. On the **Review** page, review the details about the image creation and click **Create**.
The image creation starts, the progress of this image creation and the subsequent upload to the VMware vSphere client.

15.2. UPLOADING VMDK IMAGES AND CREATING A RHEL VIRTUAL MACHINE IN VSPHERE

With RHEL image builder, you can create customized VMware vSphere system images, either in the Open virtualization format (**.ova**) or in the Virtual disk (**.vmdk**) format. You can upload customized images to the VMware vSphere client. You can upload the **.vmdk** or **.ova** image to VMware vSphere using the **govc import.vmdk** CLI tool. The **vmdk** you create contains the **cloud-init** package installed and you can use it to provision users by using user data, for example.



NOTE

Uploading **vmdk** images by using the VMware vSphere GUI is not supported.

Prerequisites

- You created a blueprint with username and password customizations.
- You created a VMware vSphere image either in the **.ova** or the **.vmdk** format by using RHEL image builder and downloaded it to your host system.
- You installed and configured the **govc** CLI tool, to be able use the **import.vmdk** command.

Procedure

1. Configure the following values in the user environment with the GOVC environment variables:

```
GOVC_URL
GOVC_DATACENTER
GOVC_FOLDER
GOVC_DATASTORE
GOVC_RESOURCE_POOL
GOVC_NETWORK
```

2. Navigate to the directory where you downloaded your VMware vSphere image.
3. Launch the VMware vSphere image on vSphere by following the steps:
   a. Import the VMware vSphere image in to vSphere:

```
$ govc import.vmdk ./composer-api.vmdk folder-name
```

   For the **.ova** format:

```
$ govc import.ova ./composer-api.ova folder-name
```

b. Create the VM in vSphere without powering it on:

```
govc vm.create \
-net.adapter=vmxnet3 \
-m=4096 -c=2 -g=rhel8_64Guest \
-firmware=efi -disk="folder-name/composer-api.vmdk" \
-disk.controller=scsi -on=false \
 vmname
```

For the **.ova** format, replace the line **-firmware=efi -disk="*folder-name*/composer-api.vmdk"** with **-firmware=efi -disk="*folder-name*/composer-api.ova"**

c. Power-on the VM:

```
$ govc vm.power -on vm-name
```

d. Retrieve the VM IP address:

```
$ govc vm.ip vm-name
```

e. Use SSH to log in to the VM, using the username and password you specified in your blueprint:

```
$ ssh admin@ vm-ip-address
```

NOTE

If you copied the **.vmdk** image from your local host to the destination using the **govc datastore.upload** command, using the resulting image is not supported. There is no option to use the **import.vmdk** command in the vSphere GUI and as a result, the vSphere GUI does not support the direct upload. The **.vmdk** image is not usable from the vSphere GUI.

15.3. CREATING AND AUTOMATICALLY UPLOADING VMDK IMAGES TO VSPHERE USING IMAGE BUILDER GUI

You can build VMware images by using the RHEL image builder GUI tool and automatically push the images directly to your vSphere instance. This avoids the need to download the image file and push it manually. The **vmdk** that you create contains the **cloud-init** package installed and you can use it to provision users by using user data, for example. To build **.vmdk** images by using RHEL image builder and push them directly to vSphere instances service provider, follow the steps:

Prerequisites

- You are a member of the **root** or the **weldr** group.
- You have opened link:https://localhost:9090/RHEL image builder in a browser.
- You have created a blueprint.
- You have a vSphere Account.

Procedure

1. For the blueprint you created, click the **Images** tab .
2. Click **Create Image** to create your customized image.
   The **Image type** window opens.
3. In the **Image type** window:
   a. From the dropdown menu, select the Type: VMware vSphere (**.vmdk**).
   b. Check the **Upload to VMware** checkbox to upload your image to the vSphere.
   c. Optional: Set the size of the image you want to instantiate. The minimal default size is 2 GB.
   d. Click **Next**.
4. In the **Upload to VMware** window, under **Authentication**, enter the following details:
   a. **Username**: username of the vSphere account.
   b. **Password**: password of the vSphere account.
5. In the **Upload to VMware** window, under **Destination**, enter the following details about the image upload destination:
   a. **Image name**: a name for the image.
   b. **Host**: The URL of your VMware vSphere.
   c. **Cluster**: The name of the cluster.
   d. **Data center**: The name of the data center.
   e. **Data store**:The name of the Data store.
   f. Click **Next**.
6. In the **Review** window, review the details of the image creation and click **Finish**.
   You can click **Back** to modify any incorrect detail.

RHEL image builder adds the compose of a RHEL vSphere image to the queue, and creates and uploads the image to the Cluster on the vSphere instance you specified.

After the process is complete, you can see the **Image build complete** status.

Verification

After the image status upload is completed successfully, you can create a Virtual Machine (VM) from the image you uploaded and login into it. To do so:

1. Access VMware vSphere Client.
2. Search for the image in the Cluster on the vSphere instance you specified.
3. Select the image you uploaded.
4. Right-click the selected image.

5. Click **New Virtual Machine**.
   A **New Virtual Machine** window opens.

   In the **New Virtual Machine** window, provide the following details:

   a. Select **New Virtual Machine**.

   b. Select a name and a folder for your VM.

   c. Select a computer resource: choose a destination computer resource for this operation.

   d. Select storage: For example, select NFS-Node1

   e. Select compatibility: The image should be BIOS only.

   f. Select a guest operating system: For example, select *Linux* and *Red Hat Fedora (64-bit)* .

   g. **Customize hardware**: When creating a VM, on the **Device Configuration** button on the upper right, delete the default New Hard Disk and use the drop-down to select an Existing Hard Disk disk image:

   h. Ready to complete: Review the details and click **Finish** to create the image.

6. Navigate to the **VMs** tab.

   a. From the list, select the VM you created.

   b. Click the **Start** button from the panel. A new window appears, showing the VM image loading.

   c. Log in with the credentials you created for the blueprint.

   d. You can verify if the packages you added to the blueprint are installed. For example:

   ```
   $ rpm -qa | grep firefox
   ```

Additional resources

- Introduction to vSphere Installation and Setup

CHAPTER 16. PREPARING AND UPLOADING CUSTOM GCE IMAGES TO GCP

With RHEL image builder, you can build a **gce** image, provide credentials for your user or GCP service account, and then upload the **gce** image directly to the GCP environment.

16.1. CONFIGURING AND UPLOADING A GCE IMAGE TO GCP BY USING THE CLI

Set up a configuration file with credentials to upload your **gce** image to GCP by using the RHEL image builder CLI.

WARNING

You cannot manually import **gce** image to GCP, because the image will not boot. You must use either **gcloud** or RHEL image builder to upload it.

Prerequisites

- You have a valid Google account and credentials to upload your image to GCP. The credentials can be from a user account or a service account. The account associated with the credentials must have at least the following IAM roles assigned:
  - **roles/storage.admin** - to create and delete storage objects
  - **roles/compute.storageAdmin** - to import a VM image to Compute Engine.
- You have an existing GCP bucket.

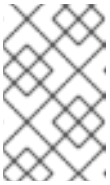
Procedure

1. Use a text editor to create a **gcp-config.toml** configuration file with the following content:

```
provider = "gcp"
[settings]
bucket = "<gcp_bucket>"
region = "<gcp_storage_region>"
object = "<object_key>"
credentials = "<gcp_credentials>"
```

   - ***<gcp_bucket>*** points to an existing bucket. It is used to store the intermediate storage object of the image which is being uploaded.
   - ***<gcp_storage_region>*** is both a regular Google storage region and a dual or multi region.
   - ***<object_key>*** is the name of an intermediate storage object. It must not exist before the upload, and it is deleted when the upload process is done. If the object name does not end with **.tar.gz**, the extension is automatically added to the object name.

- ***<gcp_credentials>*** is a **Base64-encoded** scheme of the credentials JSON file downloaded from GCP. The credentials determine which project the GCP uploads the image to.



NOTE

Specifying ***<gcp_credentials>*** in the **gcp-config.toml** file is optional if you use a different mechanism to authenticate with GCP. For other authentication methods, see Authenticating with GCP.

2. Retrieve the ***<gcp_credentials>*** from the JSON file downloaded from GCP.

```
$ sudo base64 -w 0 cee-gcp-nasa-476a1fa485b7.json
```

3. Create a compose with an additional image name and cloud provider profile:

```
$ sudo composer-cli compose start <blueprint_name> gce <image_key> gcp-config.toml
```

The image build, upload, and cloud registration processes can take up to ten minutes to complete.

Verification

- Verify that the image status is FINISHED:

```
$ sudo composer-cli compose status
```

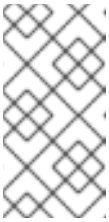
Additional resources

- Identity and Access Management
- Create storage buckets

16.2. HOW RHEL IMAGE BUILDER SORTS THE AUTHENTICATION ORDER OF DIFFERENT GCP CREDENTIALS

You can use several different types of credentials with RHEL image builder to authenticate with GCP. If RHEL image builder configuration is set to authenticate with GCP by using multiple sets of credentials, it uses the credentials in the following order of preference:

1. Credentials specified with the **composer-cli** command in the configuration file.
2. Credentials configured in the **osbuild-composer** worker configuration.
3. **Application Default Credentials** from the **Google GCP SDK** library, which tries to automatically find a way to authenticate by using the following options:
   a. If the *GOOGLE_APPLICATION_CREDENTIALS* environment variable is set, Application Default Credentials tries to load and use credentials from the file pointed to by the variable.
   b. Application Default Credentials tries to authenticate by using the service account attached to the resource that is running the code. For example, Google Compute Engine VM.



NOTE

You must use the GCP credentials to determine which GCP project to upload the image to. Therefore, unless you want to upload all of your images to the same GCP project, you always must specify the credentials in the **gcp-config.toml** configuration file with the **composer-cli** command.

16.3. SPECIFYING GCP CREDENTIALS WITH THE COMPOSER-CLI COMMAND

You can configure GCP authentication credentials to be used for GCP globally for all image builds. This way, if you want to import images to the same GCP project, you can use the same credentials for all image uploads to GCP.

Procedure

- In the **/etc/osbuild-worker/osbuild-worker.toml** worker configuration, set the following credential value:

```
[gcp]
credentials = "PATH_TO_GCP_ACCOUNT_CREDENTIALS"
```

16.4. SPECIFYING CREDENTIALS IN THE OSBUILD-COMPOSER WORKER CONFIGURATION

You can configure GCP authentication credentials to be used for GCP globally for all image builds. This way, if you want to import images to the same GCP project, you can use the same credentials for all image uploads to GCP.

Procedure

- In the **/etc/osbuild-worker/osbuild-worker.toml** worker configuration, set the following credential value:

```
[gcp]
credentials = "PATH_TO_GCP_ACCOUNT_CREDENTIALS"
```

CHAPTER 17. PREPARING AND UPLOADING CUSTOM IMAGES DIRECTLY TO OCI

You can create custom images and then automatically update them to the Oracle Cloud Infrastructure (OCI) instance with RHEL image builder.

17.1. CREATING AND AUTOMATICALLY UPLOADING CUSTOM IMAGES TO OCI

With RHEL image builder, build customized images and automatically push them directly to your Oracle Cloud Infrastructure (OCI) instance. Then, you can start an image instance from the OCI dashboard.

Prerequisites

- You have **root** or **weldr** group user access to the system.
- You have an Oracle Cloud account.
- You must be granted security access in an **OCI policy** by your administrator.
- You have created an OCI Bucket in the **OCI_REGION** of your choice.

Procedure

1. Open the RHEL image builder interface of the web console in a browser.
2. Click **Create blueprint**. The **Create blueprint** wizard opens.
3. On the **Details** page, enter a name for the blueprint, and optionally, a description. Click **Next**.
4. On the **Packages** page, select the components and packages that you want to include in the image. Click **Next**.
5. On the **Customizations** page, configure the customizations that you want for your blueprint. Click **Next**.
6. On the **Review** page, click **Create**.
7. To create an image, click **Create Image**. The **Create image** wizard opens.
8. On the **Image output** page, complete the following steps:
    a. From the **"Select a blueprint"** drop-down menu, select the blueprint you want.
    b. From the **"Image output type"** drop-down menu, select **Oracle Cloud Infrastructure (.qcow2)**.
    c. Check the **"Upload OCI** checkbox to upload your image to the OCI.
    d. Enter the **"image size"**. Click **Next**.
9. On the **Upload to OCI - Authentication** page, enter the following mandatory details:
    a. User OCID: you can find it in the Console on the page showing the user's details.

b. Private key

10. On the **Upload to OCI - Destination** page, enter the following mandatory details and click **Next**.

    a. Image name: a name for the image to be uploaded.

    b. OCI bucket

    c. Bucket namespace

    d. Bucket region

    e. Bucket compartment

    f. Bucket tenancy

11. Review the details in the wizard and click **Finish**.

RHEL image builder adds the compose of a RHEL **.qcow2** image to the queue.

Verification

1. Access the OCI dashboard and click **Custom Images**.
2. Select the **Compartment** you specified for the image and locate the image in the **Import image** table.
3. Click the image name and verify the image information.

Additional resources

- Managing custom images in the OCI.
- Managing buckets in the OCI.

CHAPTER 18. PREPARING AND UPLOADING CUSTOMIZED QCOW2 IMAGES DIRECTLY TO OPENSTACK

You can create custom **.qcow2** images with RHEL image builder, and manually upload them to the OpenStack cloud deployments.

18.1. UPLOADING QCOW2 IMAGES TO OPENSTACK

With the RHEL image builder tool, you can create customized **.qcow2** images that are suitable for uploading to OpenStack cloud deployments, and starting instances there. RHEL image builder creates images in the QCOW2 format, but with further changes specific to OpenStack.

Prerequisites

- You have created a blueprint.

Procedure

1. Start the compose of a **QCOW2** image.

```
# composer-cli compose start blueprint_name qcow2
```

2. Check the status of the building.

```
# composer-cli compose status
```

After the image build finishes, you can download the image.

3. Download the **QCOW2** image:

```
# composer-cli compose image UUID
```

4. Access the OpenStack dashboard and click **+Create Image**.
5. On the left menu, select the **Admin** tab.
6. From the **System Panel**, click **Image**.
The **Create An Image** wizard opens.
7. In the **Create An Image** wizard:
   a. Enter a name for the image
   b. Click **Browse** to upload the **QCOW2** image.
   c. From the **Format** dropdown list, select the **QCOW2 - QEMU Emulator**.
   d. Click **Create Image**.
8. On the left menu, select the **Project** tab.
   a. From the **Compute** menu, select **Instances**.
   b. Click the **Launch Instance** button.

The **Launch Instance** wizard opens.

c. On the **Details** page, enter a name for the instance. Click **Next**.

d. On the **Source** page, select the name of the image you uploaded. Click **Next**.

e. On the **Flavor** page, select the machine resources that best fit your needs. Click **Launch**.

9. You can run the image instance using any mechanism (CLI or OpenStack web UI) from the image. Use your private key via SSH to access the resulting instance. Log in as **cloud-user**.

CHAPTER 19. PREPARING AND UPLOADING CUSTOMIZED RHEL IMAGES TO THE ALIBABA CLOUD

You can upload customized **.ami** images that you created by using RHEL image builder to the Alibaba Cloud.

19.1. CREATING AN INSTANCE OF A CUSTOMIZED RHEL IMAGE USING ALIBABA CLOUD

You can create instances of a customized RHEL image by using the Alibaba ECS Console.

Prerequisites

- You have activated OSS and uploaded your custom image.
- You have successfully imported your image to ECS Console.

Procedure

1. Log in to the ECS console.
2. On the left-side menu, select **Instances**.
3. In the upper-right corner, click **Create Instance**. You are redirected to a new window.
4. Complete all the required information. See Creating an instance by using the wizard for more details.
5. Click **Create Instance** and confirm the order.

NOTE

You can see the option **Create Order** instead of **Create Instance**, depending on your subscription.

As a result, you have an active instance ready for deployment from the **Alibaba ECS Console**.

Additional resources

- Creating an instance by using a custom image
- Create an instance by using the wizard

19.2. IMPORTING IMAGES TO ALIBABA CLOUD

To import a customized Alibaba RHEL image that you created by using RHEL image builder to the Elastic Compute Service (ECS), follow the steps:

Prerequisites

- Your system is set up for uploading Alibaba images.
- You have created an **ami** image by using RHEL image builder.

- You have a bucket. See Creating a bucket.
- You have an active Alibaba Account.
- You activated OSS.
- You have uploaded the image to Object Storage Service (OSS).

Procedure

1. Log in to the ECS console.
    i. On the left-side menu, click **Images**.
    ii. On the upper right side, click **Import Image**. A dialog window opens.
    iii. Confirm that you have set up the correct region where the image is located. Enter the following information:
        a. **OSS Object Address**: See how to obtain OSS Object Address.
        b. **Image Name**
        c. **Operating System**
        d. **System Disk Size**
        e. **System Architecture**
        f. **Platform**: Red Hat
    iv. Optional: Provide the following details:
        g. **Image Format**: **qcow2** or **ami**, depending on the uploaded image format.
        h. **Image Description**
        i. **Add Images of Data Disks**
        The address can be determined in the OSS management console. After selecting the required bucket in the left menu:
2. Select **Files** section.
3. Click the **Details** link on the right for the appropriate image.
A window appears on the right side of the screen, showing image details. The **OSS** object address is in the **URL** box.
4. Click **OK**.

NOTE

The importing process time can vary depending on the image size.

The customized image is imported to the **ECS** Console.

Additional resources

- Notes for importing images
- Creating an instance from custom images
- Upload an object

19.3. UPLOADING CUSTOMIZED RHEL IMAGES TO ALIBABA

You can upload a customized **AMI** image you created by using RHEL image builder to the Object Storage Service (OSS).

Prerequisites

- Your system is set up for uploading Alibaba images.
- You have created an **ami** image by using RHEL image builder.
- You have a bucket. See Creating a bucket.
- You have an active Alibaba Account.
- You activated OSS.

Procedure

1. Log in to the OSS console.
2. In the Bucket menu on the left, select the bucket to which you want to upload an image.
3. In the upper right menu, click the **Files** tab.
4. Click **Upload**. A dialog window opens on the right side. Configure the following:
   - **Upload To**: Choose to upload the file to the **Current** directory or to a **Specified** directory.
   - **File ACL**: Choose the type of permission of the uploaded file.
5. Click **Upload**.
6. Select the image you want to upload to the OSS Console..
7. Click **Open**.

Additional resources

- Upload an object
- Creating an instance from custom images
- Importing images

19.4. PREPARING TO UPLOAD CUSTOMIZED RHEL IMAGES TO ALIBABA CLOUD

To deploy a customized RHEL image to the Alibaba Cloud, first you need to verify the customized image. The image needs a specific configuration to boot successfully, because Alibaba Cloud requests the custom images to meet certain requirements before you use it.

NOTE

RHEL image builder generates images that conform to Alibaba's requirements. However, Red Hat recommends also using the Alibaba **image_check tool** to verify the format compliance of your image.

Prerequisites

- You must have created an Alibaba image by using RHEL image builder.

Procedure

1. Connect to the system containing the image that you want to check by using the Alibaba **image_check** tool.
2. Download the **image_check** tool:

```
$ curl -O https://docs-aliyun.cn-hangzhou.oss.aliyun-
inc.com/assets/attach/73848/cn_zh/1557459863884/image_check
```

3. Change the file permission of the image compliance tool:

```
# chmod +x image_check
```

4. Run the command to start the image compliance tool checkup:

```
# ./image_check
```

The tool verifies the system configuration and generates a report that is displayed on your screen. The **image_check** tool saves this report in the same folder where the image compliance tool is running.

Troubleshooting

If any of the **Detection Items** fail, follow the instructions in the terminal to correct it.

Additional resources

- Image Compliance Tool