Red Hat Enterprise Linux 9

Composing a customized RHEL system image

Creating customized system images with image builder on Red Hat Enterprise Linux 9

Last Updated: 2023-09-20

Red Hat Enterprise Linux 9 Composing a customized RHEL system image

Creating customized system images with image builder on Red Hat Enterprise Linux 9

Legal Notice

Abstract

Image builder is a tool for creating deployment-ready customized system images: installation disks, virtual machines, cloud vendor-specific images, and others. Using image builder, you can create these images faster if compared to manual procedures, because it eliminates the specific configurations required for each output type. This document describes how to set up image builder and create images with it.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Jira (account required)

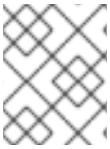
1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the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

CHAPTER 1. IMAGE BUILDER DESCRIPTION

To deploy a system on a cloud platform, create a system image. To create RHEL system images, use the image builder tool.

1.1. WHAT IS IMAGE BUILDER?

You can use image builder to create customized system images of RHEL, including system images prepared for deployment on cloud platforms. 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 image builder functionality through a command-line interface in the **composer-cli** tool, or a graphical user interface in the RHEL web console.



NOTE

From RHEL 8.3 onward, the **osbuild-composer** back end replaces **lorax-composer**. The new service provides REST APIs for image building.

1.2. IMAGE BUILDER TERMINOLOGY

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 image builder interface of the RHEL web console.
You can 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.

1.3. IMAGE BUILDER OUTPUT FORMATS

Image builder can create images in multiple output formats shown in the following table. To check the supported types, run the command:

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

Table 1.1. Image builder output formats

Description	CLI name	file extension
QEMU QCOW2 Image	**qcow2**	**.qcow2**
TAR Archive	**tar**	**.tar**

Description	CLI name	file extension
Amazon Machine Image Disk	**ami**	**.raw**
Azure Disk Image	**vhd**	**.vhd**
Google Cloud Platform	**gce**	**.vhd**
VMware Virtual Machine Disk	**vmdk**	**.vmdk**
Openstack	**openstack**	**.qcow2**
RHEL for Edge Commit	**edge-commit**	**.tar**
RHEL for Edge Container	**edge-container**	**.tar**
RHEL for Edge Installer	**edge-installer**	**.iso**
RHEL for Edge Raw	**edge-raw-image**	**.tar**
RHEL for Edge Simplified Installer	**edge-simplified-installer**	**.iso**
ISO image	**image-installer**	**.iso**

1.4. IMAGE BUILDER SYSTEM REQUIREMENTS

The environment where image builder runs, for example a dedicated virtual machine, must meet requirements listed in the following table.

Table 1.2. Image builder system requirements

Parameter	Minimal Required Value
System type	A dedicated virtual machine. Note that image builder is not supported on containers, including Red Hat Universal Base Images (UBI).
Processor	2 cores
Memory	4 GiB
Disk space	20 GiB of free space in the **/var** filesystem
Access privileges	Administrator level (root)
Network	Internet connectivity

NOTE

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

Additional resources

- Provisioning to Satellite using a Red Hat image builder image

CHAPTER 2. INSTALLING IMAGE BUILDER

Before using image builder, you must install image builder in a virtual machine.

2.1. INSTALLING IMAGE BUILDER IN A VIRTUAL MACHINE

To install image builder on a dedicated virtual machine (VM), follow these steps:

Prerequisites

- You must be connected to a RHEL VM.
- The VM for image builder must be running and 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 the image builder and other necessary packages on the VM:
   - **osbuild-composer** - supported from RHEL 8.3 onward
   - **composer-cli**
   - **cockpit-composer**
   - **bash-completion**

   ```
   # dnf install osbuild-composer composer-cli cockpit-composer bash-completion
   ```

   The web console is installed as a dependency of the *cockpit-composer* package.
2. Enable image builder to start after each reboot:

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

   The **osbuild-composer** and **cockpit** services start automatically on first access.
3. Load the shell configuration script so that the autocomplete feature for the **composer-cli** command starts working immediately without reboot:

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

IMPORTANT

The **osbuild-composer** package is the new backend engine that will be the preferred default and focus of all new functionality beginning with Red Hat Enterprise Linux 8.3 and later. The previous backend **lorax-composer** package is considered deprecated, will only receive select fixes for the remainder of the Red Hat Enterprise Linux 8 life cycle and will be omitted from future major releases. It is recommended to uninstall **lorax-composer** in favor of osbuild-composer.

Verification

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. MANAGING IMAGE BUILDER REPOSITORIES

You can use the following types of repositories in image builder:

Custom third-party repositories

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

Official repository overrides

Use these if you want to download base system RPMs from elsewhere than the official repositories, for example, a custom mirror in your network. Because using official repository overrides disables the default repositories, your custom mirror must contain all the necessary packages.

3.1. ADDING CUSTOM THIRD-PARTY REPOSITORIES

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:

```
id = "repository_id"
name = "repository_name"
type = "repository_type"
url = "repository-url"
check_gpg = false
check_ssl = false
```

   For example:

```
id = "k8s"
name = "Kubernetes"
type = "yum-baseurl"
url = "https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64"
check_gpg = false
check_ssl = false
system = false
```

2. Save the file in the TOML format.

3. Add the new third-party source with the following command:

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

Verification

- Check if the new source was successfully added:

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

- Check the new source content:

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

3.2. ADDING THIRD-PARTY REPOSITORIES WITH SPECIFIC DISTRIBUTIONS

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-9** will use this source. For example, if you list packages and specify **rhel-9**, it will include this source. However, listing packages for the host distribution will not include this source.

Procedure

1. Create a repository source file:

```
check_gpg = true
check_ssl = true
distros = ["list_of_distributions"]
id = "repository_id"
name = "repository-name"
system = false
type = ""repository_type"
url = "repository-url"
```

For example, to specify the distribution:

```
check_gpg = true
check_ssl = true
distros = ["rhel-9"]
id = "rh9-local"
name = "packages for RHEL"
system = false
type = "yum-baseurl"
url = "http://local/repos/rhel9/projectrepo/"
```

2. Save the file in the TOML format.

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.

For security reasons, you can distribute the key in a separate channel from the RPMs, by making your GPG key available over **https**. You can indicate which GPG key to use to do the check, by setting **check_repogpg = true** in the source. If the key is available over **https**, set the **gpgkeys** entry to the URL for the key. Optionally, you can also embed the whole key into the source **gpgkeys** entry to import it directly instead of fetching it from the URL.

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. Set up a repository by signing your **repomd.xml** file:

```
$ gpg -u YOUR-GPG-KEY-EMAIL --yes --detach-sign --armor repomd.xml
```

5. Check the GPG signature.

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

   b. 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 = "repository_id"
name = "repository_name"
system = false
type = "repository_type"
url = "repository_URL"
check_repogpg = true
gpgkeys=["_GPG_key_URL"]
```

   c. Optional: You can embed the whole key into the **gpgkeys** field. You can add as many keys as you need. For example, add the GPG key directly in the **gpgkeys** field:

```
check_gpg = true
check_ssl = true
check_repogpg
id = "repository_id"
name = "repository_name"
system = false
type = "repository_type"
url = "repository_URL"
gpgkeys=["GPG_key"]
```

Verification

- Test the signature of the repository manually:

```
$ gpg --verify repomd.xml.asc
```

- If the test does not find the signature, you will be prompt with 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, you will be prompt with an error similar to the following one:

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

3.4. IMAGE BUILDER DEFAULT SYSTEM REPOSITORIES

The **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** directory.

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

3.5. OVERRIDING A SYSTEM REPOSITORY

You can configure a repository override for image builder in the **/etc/osbuild-composer/repositories** directory with the following steps.

Prerequisites

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

Procedure

1. Create a directory where you want to store your repository overrides:

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

2. You can create your own JSON file structure.

3. 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 9, use **/etc/osbuild-composer/repositories/rhel-92.json**.

4. Add the following structure to your JSON file, for example:

```
{
   "<ARCH>": [
      {
         "name": "baseos",
         "baseurl": "http://mirror.example.com/composes/released/RHEL-
9/9.0/BaseOS/x86_64/os/",
```

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

Specify only one of the following attributes:

- **baseurl** - string: a base URL of the repository.
- **metalink** - string: a URL of a metalink file that contains a list of valid mirror repositories.
- **mirrorlist** - string: a URL of a mirrorlist file that contains a list of valid mirror repositories The remaining fields are optional.

a. Alternatively, you can copy the JSON file for your distribution.

i. Copy the repository file to the directory you created. In the following command, replace **rhel-version.json** with your RHEL version, for example: rhel-9.json.

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

5. Using a text editor, edit the **baseurl** paths in the **rhel-9.json** file and save it. For example:

```
$ vi /etc/osbuild-composer/repositories/rhel-version.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**.

3.6. OVERRIDING A SYSTEM REPOSITORY WITH SUPPORT FOR SUBSCRIPTIONS

The **osbuild-composer** service can 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 same **baseurl** as the repository defined in **/etc/yum.repos.d/redhat.repo**.

- The value of **"rhsm": true** defined in the JSON object.

Prerequisites

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

Procedure

1. Obtain 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-9/9.0/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-9/9.0/AppStream/x86_64/os/",
         "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\n (…)",
         "check_gpg": true,
         "rhsm": true
      }
   ]
}
```

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** using **composer-cli**. System repository overrides are usually used for "BaseOS" and "AppStream" repositories, whereas **composer-cli** sources are used for all the other repositories.

As a result, image builder reads the **/etc/yum.repos.d/redhat.repo** file from the host system and uses it as a source of subscriptions.

Additional resources

- Image builder uses CDN repositories when host is registered to Satellite 6

CHAPTER 4. CREATING SYSTEM IMAGES USING THE IMAGE BUILDER COMMAND-LINE INTERFACE

Image builder is a tool for creating custom system images. To control image builder and create your custom system images, you can use the command-line interface (CLI) or the web console interface. Currently, however, the CLI is the preferred method to use image builder.

WARNING

The image builder tool does not support building RHEL 9 images on a RHEL 8 system. You can only build an image from the earlier version of the image you use in the host distribution.

4.1. INTRODUCING THE IMAGE BUILDER COMMAND-LINE INTERFACE

The image builder command-line interface (CLI) is currently the preferred method to use image builder. It offers more functionality than the web console interface. To use the CLI, run the **composer-cli** command with the suitable options and subcommands.

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

1. Export (*save*) the blueprint definition to a plain text file
2. Edit this file in a text editor
3. Import (*push*) 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 achieve this procedure, the **composer-cli** command offers many subcommands to examine the state of configured blueprints and composes.

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

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

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

4.2. CREATING AN IMAGE BUILDER BLUEPRINT USING THE COMMAND-LINE INTERFACE

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

Prerequisite

- Access to the image builder tool.

Procedure

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

```
name = "BLUEPRINT-NAME"
description = "LONG FORM DESCRIPTION TEXT"
version = "0.0.1"
modules = []
groups = []
```

   Replace *BLUEPRINT-NAME* and *LONG FORM DESCRIPTION TEXT* 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**.

   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.

4. Save the file, for example, as *BLUEPRINT-NAME*.toml and close the text editor.

5. Push (import) 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 image builder is unable to depsolve a package from your custom repositories, follow the steps:

- 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 .
- Composing a customized RHEL system image with proxy server .

4.3. EDITING AN IMAGE BUILDER BLUEPRINT WITH COMMAND-LINE INTERFACE

You can edit an existing image builder 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. Save (export) the blueprint to a local text file:

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

2. Edit the *BLUEPRINT-NAME*.toml file with a text editor and make your changes.
3. Before finishing the edits, verify that the file is a valid blueprint:
   a. Remove this line, if present:

   ```
   packages = []
   ```

   b. Increase the version number, for example, fro 0.0.1 to 0.1.0. Remember that 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.
   c. Check if the contents are valid TOML specifications. See the TOML documentation for more information.

NOTE

TOML documentation is a community product and is not supported by Red Hat. You can report any issues with the tool at https://github.com/toml-lang/toml/issues

4. Save the file and close the text editor.
5. Push (import) the blueprint back into image builder:

```
# composer-cli blueprints push BLUEPRINT-NAME.toml
```

NOTE

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

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

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

7. 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.

4.4. CREATING A SYSTEM IMAGE WITH IMAGE BUILDER IN THE COMMAND-LINE INTERFACE

You can build a custom image using the image builder command-line interface.

Prerequisites

- You have a blueprint prepared for the image. See Creating an image builder blueprint using the command-line interface.

Procedure

1. Start the compose:

```
# composer-cli compose start BLUEPRINT-NAME IMAGE-TYPE
```

   Replace *BLUEPRINT-NAME* with 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).

2. Wait until the compose process is finished. 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.

3. 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 using the following commands:

- Check the compose status:

```
$ sudo composer-cli compose status
```

- Download the metadata of the image:

```
$ 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:

```
$ sudo cat /var/lib/osbuild-composer/jobs/job_UUID.json
```

You can find the *job_UUID*.json in the journal.

Additional resources

- Tracing image builder

4.5. BASIC IMAGE BUILDER COMMAND-LINE COMMANDS

The 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 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
```

Replace *BLUEPRINT* with the name of the blueprint to build, and *COMPOSE-TYPE* with the output image 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

- *composer-cli*(1) man page

4.6. IMAGE BUILDER BLUEPRINT FORMAT

Image builder blueprints are presented to the user as plain text in the TOML format.

The elements of a typical blueprint file include the following:

The blueprint metadata

```
name = "BLUEPRINT-NAME"
description = "LONG FORM DESCRIPTION TEXT"
version = "VERSION"
```

The *BLUEPRINT-NAME* and *LONG FORM DESCRIPTION TEXT* field are a name and description for your blueprint.

The *VERSION* is a version number according to the Semantic Versioning scheme.

This part is present only once for the entire blueprint file.

The *modules* entry lists the package names and versions of packages to be installed into the image.

The *group* entry describes a group of packages to be installed into the image. Groups use the following package categories:

- Mandatory
- Default
- Optional
  Blueprints install the mandatory and default packages. There is no mechanism for selecting optional packages.

Groups to include in the image

```
[[groups]]
name = "group-name"
```

The *group-name* is the name of the group, for example, **anaconda-tools**, **widget**, **wheel** or **users**.

Packages to include in the image

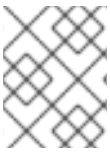
```
[[packages]]
name = "package-name"
version = "package-version"
```

package-name is the name of the package, such as **httpd**, **gdb-doc**, or **coreutils**.

package-version is a version to use. This field supports **dnf** version specifications:

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

Repeat this block for every package to include.



NOTE

Currently there are no differences between packages and modules in the image builder tool. Both are treated as RPM package dependencies.

4.7. SUPPORTED IMAGE CUSTOMIZATIONS

You can customize your image by adding to your blueprint an additional RPM package, by enabling a service, or by customizing a kernel command line parameter. You can use several image customizations within blueprints. To make use of these options, you must configure the customizations in the blueprint and import (push) it to image builder.

NOTE

These customizations are not supported when using image builder in the web console.

Select a distribution

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

Replace "*different_minor_version*" to build a different minor version, for example, if you want to build a RHEL 9.0, use **distro** = "rhel-90". On a RHEL 9.1, you can build minor versions such as RHEL 9.1, RHEL 8.7 and earlier releases. If you do not specify a distribution, the blueprint uses the host distribution. In case you upgrade the host operating system, the blueprints with no distribution set will build images using the new OS version.

Select a package group

```
[[packages]]
name = "package_group_name"
```

Replace "*package_group_name*" with the name of the group. For example, "@server with gui".

Set the image hostname

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

User specifications for the resulting system 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
```

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.

IMPORTANT

To generate the **password hash**, you must install **python3** on your system.

```
# dnf install python3
```

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 *PUBLIC-SSH-KEY* with the actual public key.

Replace the other placeholders with suitable values.

You must enter the **name**. You can omit any of the lines that you do not need.

Repeat this block for every user to include.

Group specifications for the resulting system image

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

Repeat this block for every group to include.

Set an existing users SSH key

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

NOTE

The "Set an existing user SSH key" customization is only applicable for existing users. To create a user and set an SSH key, see the **User specifications for the resulting system image** customization.

Append a kernel boot parameter option to the defaults

```
[customizations.kernel]
append = "KERNEL-OPTION"
```

By default, image builder builds a default kernel into the image. But, you can customize the kernel with the following configuration in blueprint

```
[customizations.kernel]
name = "KERNEL-rt"
```

Define a kernel name to use in an image

```
[customizations.kernel.name]
name = "KERNEL-NAME"
```

Set the timezone and the*Network Time Protocol*(NTP) servers for the resulting system image

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

If you do not set a timezone, the system uses *Universal Time, Coordinated* **(UTC)** as default. Setting NTP servers is optional.

Set the locale settings for the resulting system image

```
[customizations.locale]
languages = ["LANGUAGE"]
keyboard = "KEYBOARD"
```

Setting both the language and the keyboard options is mandatory. You can add many other languages. The first language you add will be the primary language and the other languages will be secondary. For example:

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

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

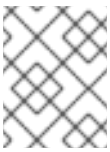
```
$ localectl list-locales
```

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

```
$ localectl list-keymaps
```

Firewall customization

Set the firewall for the resulting system image. By default, the firewall blocks all access, except for services that enable their ports explicitly, such as **sshd**. The following blueprint can be used to open other ports or services.
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 and OpenStack templates explicitly disable the firewall for their environment. This cannot be overridden by the blueprint.

```
[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 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.

Customize the firewall services

services is an optional object with the following attributes containing services to enable or disable for **firewalld**:

- **enabled** - An optional list of strings for services to enable.
- **disabled** - An optional list of strings for services to disable.
  Check the available firewall services.

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

In the blueprint, under section **customizations.firewall.service**, specify the firewall services that you want to customize.

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

For example:

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

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

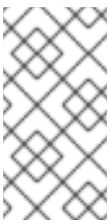
NOTE

If you do not want to customize the firewall services, omit the **[customizations.firewall]** and **[customizations.firewall.services]** sections from the blueprint.

Set which services to enable during the boot time

```
[customizations.services]
enabled = ["SERVICES"]
disabled = ["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 this setup cannot be overridden. The **[customizations.services]** customization in the blueprint do not replace these services, but add them to the list of services already present in the image templates.



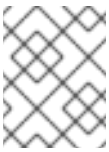
NOTE

Each time a build starts, it clones the repository of the host system. If you refer to a repository with a large amount of history, it might take some time to clone and it uses a significant amount of disk space. Also, the clone is temporary and the build removes it after it creates the RPM package.

Specify 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
  To customize the filesystem configuration in your blueprint:



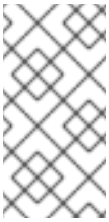
NOTE

The filesystem customization is not supported for OSTree systems, because OSTree images have their own mount rule, such as read-only.

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

The blueprint supports the following **mountpoints** and their sub-directories:

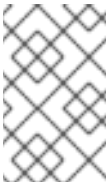
- / - the root mount point
- /**var**
- /**home**
- /**opt**
- /**srv**
- /**usr**
- /**app**
- /**data**
- /**boot** - Supported from RHEL 8.7 and RHEL 9.1 onward.



NOTE

Customizing mount points is only supported from RHEL 8.5 and 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 desired disk layout. The default filesystem layout remains unchanged - if you use plain images without file system customization, and **cloud-init** resizes the root partition.



NOTE

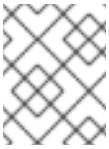
From 8.6 onward, for the **osbuild-composer-46.1-1.el8** RPM and later version, the physical partitions are no longer available and filesystem customizations create logical volumes.

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

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"
size = 1073741824
```

You can also define the mount point size by using units.



NOTE

You can only define the mount point size by using units for the package version provided for RHEL 8.6 and RHEL 9.0 distributions onward.

For example:

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

or

[[customizations.filesystem]]
mountpoint = "/boot"
size = "1 GiB"
```

Create customized directories and files for your image under the /etc directory

To create customized files and directories in your image, use the **[[customizations.files]]** and the **[[customizations.directories]]** blueprint customizations. Currently, 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**.

Create a custom directory blueprint customization

With the **[[customizations.directories]]** blueprint customization, you can create customized directories in the **/etc** directory of your image.

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.

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.

To customize a directory configuration in your blueprint, create a file with the following content, for example:

```
[[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 a custom file blueprint customization

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. If you are not sure about the customizations, use the appropriate blueprint customization.

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**

To customize a file in your blueprint, create a file with the following content, for example:

```
[[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.

Additional resources

- Blueprint import fails after adding filesystem customization "size" .
- Blueprint reference.

4.8. PACKAGES INSTALLED BY IMAGE BUILDER

When you create a system image using image builder, the system installs a set of base packages.

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, gdisk, insights-client, kernel, langpacks-en, net-tools, NetworkManager, redhat-release, redhat-release-eula, rng-tools, rsync, selinux-policy-targeted, tar, yum-utils**
openstack	**@core, langpacks-en**
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 type	Default Packages
edge-commit	**attr, audit, basesystem, bash, bash-completion, chrony, clevis, clevis-dracut, clevis-luks, container-selinux, coreutils,criu, cryptsetup, curl, dnsmasq, dosfstools, dracut-config-generic, dracut-network, e2fsprogs, firewalld, fuse-overlayfs, fwupd, glibc, glibc-minimal-langpack, gnupg2, greenboot, gzip, hostname, ima-evm-utils, iproute, iptables, iputils, keyutils, less, lvm2, NetworkManager, NetworkManager-wifi, NetworkManager-wwan, nss-altfiles, openssh-clients, openssh-server, passwd, pinentry, platform-python, podman, policycoreutils, policycoreutils-python-utils, polkit, procps-ng, redhat-release, rootfiles, rpm, rpm-ostree, rsync, selinux-policy-targeted, setools-console, setup, shadow-utils, shadow-utils, skopeo, slirp4netns, sudo, systemd, tar, tmux, traceroute, usbguard, util-linux, vim-minimal, wpa_supplicant, xz**
edge-container	**dnf, dosfstools, e2fsprogs, glibc, lorax-templates-generic, lorax-templates-rhel, lvm2, policycoreutils, python36, python3-iniparse, qemu-img, selinux-policy-targeted, systemd, tar, xfsprogs, xz**

Image type	Default Packages
edge-installer	**aajohan-comfortaa-fonts, abattis-cantarell-fonts, alsa-firmware, alsa-tools-firmware, anaconda, anaconda-install-env-deps, anaconda-widgets, audit, bind-utils, bitmap-fangsongti-fonts, bzip2, cryptsetup, dbus-x11, dejavu-sans-fonts, dejavu-sans-mono-fonts, device-mapper-persistent-data, dnf, dump, ethtool, fcoe-utils, ftp, gdb-gdbserver, gdisk, gfs2-utils, glibc-all-langpacks, google-noto-sans-cjk-ttc-fonts, gsettings-desktop-schemas, hdparm, hexedit, initscripts, ipmitool, iwl3945-firmware, iwl4965-firmware, iwl6000g2a-firmware, iwl6000g2b-firmware, jomolhari-fonts, kacst-farsi-fonts, kacst-qurn-fonts, kbd, kbd-misc, kdump-anaconda-addon, khmeros-base-fonts, libblockdev-lvm-dbus, libertas-sd8686-firmware, libertas-sd8787-firmware, libertas-usb8388-firmware, libertas-usb8388-olpc-firmware, libibverbs, libreport-plugin-bugzilla, libreport-plugin-reportuploader, libreport-rhel-anaconda-bugzilla, librsvg2, linux-firmware, lklug-fonts, lldpad, lohit-assamese-fonts, lohit-bengali-fonts, lohit-devanagari-fonts, lohit-gujarati-fonts, lohit-gurmukhi-fonts, lohit-kannada-fonts, lohit-odia-fonts, lohit-tamil-fonts, lohit-telugu-fonts, lsof, madan-fonts, metacity, mtr, mt-st, net-tools, nmap-ncat, nm-connection-editor, nss-tools, openssh-server, oscap-anaconda-addon, pciutils, perl-interpreter, pigz, python3-pyatspi, rdma-core, redhat-release-eula, rpm-ostree, rsync, rsyslog, sg3_utils, sil-abyssinica-fonts, sil-padauk-fonts, sil-scheherazade-fonts, smartmontools, smc-meera-fonts, spice-vdagent, strace, system-storage-manager, thai-scalable-waree-fonts, tigervnc-server-minimal, tigervnc-server-module, udisks2, udisks2-iscsi, usbutils, vim-minimal, volume_key, wget, xfsdump, xorg-x11-drivers,xorg-x11-fonts-misc,xorg-x11-server-utils,xorg-x11-server-Xorg, xorg-x11-xauth**

Image type	Default Packages
edge-simplified-installer	**attr, basesystem, binutils, bsdtar, clevis-dracut, clevis-luks, cloud-utils-growpart, coreos-installer, coreos-installer-dracut, coreutils, device-mapper-multipath, dnsmasq, dosfstools, dracut-live, e2fsprogs, fcoe-utils, fdo-init, gzip, ima-evm-utils, iproute, iptables, iputils, iscsi-initiator-utils, keyutils, lldpad, lvm2, passwd, policycoreutils, policycoreutils-python-utils, procps-ng, rootfiles, setools-console, sudo, traceroute, util-linux**
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**
edge-raw-image	**dnf, dosfstools, e2fsprogs, glibc, lorax-templates-generic, lorax-templates-rhel, lvm2, policycoreutils, python36, python3-iniparse, qemu-img, selinux-policy-targeted, systemd, tar, xfsprogs, xz**
gce	**@core, langpacks-en, acpid, dhcp-client, dnf-automatic, net-tools, python3, rng-tools, tar, vim**

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
```

Additional resources

- Image builder description

4.9. 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. CREATING SYSTEM IMAGES USING THE IMAGE BUILDER WEB CONSOLE INTERFACE

Image builder is a tool for creating custom system images. To control image builder and create your custom system images, you can use the web console interface. Note that the command-line interface is the currently preferred alternative, because it offers more features.

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

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

Prerequisites

- You must have root access to the system.
- You installed image builder.

Procedure

1. Open **https://localhost:9090/** in a web browser on the system where image builder is installed. For more information about how to remotely access image builder, see *Managing systems using the RHEL web console* document.
2. Log in to the web console as the root user.
3. To display the image builder controls, click the **image builder** icon, in the upper-left corner of the window.
   The image builder dashabord opens, listing existing blueprints, if any.

Additional resources

- Creating system images with image builder command-line interface

5.2. CREATING AN IMAGE BUILDER 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.

NOTE

These blueprint customizations are available for Red Hat Enterprise Linux 9.2 or later versions and Red Hat Enterprise Linux 8.8 or later versions.

Prerequisites

- You have opened the image builder app from the web console in a browser. See Accessing the 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. Click **Next**.

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

   a. On the **Available packages** search, enter the package name and click the **>** button to move it to the Chosen packages field. Search and include as many packages as you want. 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**, and so on.

ii. For the **Minimum size partition** field of the file system, enter the desired 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 the 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 timezone settings:

a. On the **Timezone** field, enter the timezone you want to add to your system image. For example, add the following timezone format: "US/Eastern".
If you do not set a timezone, 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 image builder view opens, listing existing blueprints.

5.3. IMPORTING A BLUEPRINT IN THE 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 image builder app from thr web console in a browser.

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

Procedure

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

Verification

- 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.
- 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 using image builder in the web console interface .

5.4. EXPORTING A BLUEPRINT FROM THE 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 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. To save the **Exported blueprint**, click the **Export** button.
   a. Optionally, click the **Copy** button to copy the blueprint.

Verification

1. Open the exported blueprint to inspect and review it.

5.5. CREATING A SYSTEM IMAGE USING 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 image builder app from web console in a browser.
- You created a blueprint.

Procedure

1. Access image builder dashboard.
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. Navigate to the **Images** tab and click **Create Image**. The **Create image** wizard opens.
5. 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.
6. Click **Next**.
7. 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.
- Download the logs of the image to inspect the elements and verify if any issue is found.

CHAPTER 6. CREATING A BOOT ISO INSTALLER IMAGE WITH IMAGE BUILDER

You can use 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.

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**. 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.

6.1. CREATING A BOOT ISO INSTALLER IMAGE USING THE IMAGE BUILDER IN THE COMMAND-LINE INTERFACE

You can create a customized boot ISO installer image by using the image builder command-line interface.

Prerequisites

- You have created a blueprint for the image with a user included and pushed it back into image builder. See Blueprint customization for users.

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.

2. Wait until the compose is finished. Note that this might take several minutes.
   To check the status of the compose:

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

   A finished compose shows a status value of **FINISHED**. Identify the compose in the list by its UUID.

3. After the compose is finished, download the created image file:

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

Replace *UUID* with the UUID value obtained in the previous steps.

As a result, 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.

Verification

1. Navigate to the folder where you downloaded the image file.
2. Locate the **.iso** image you downloaded.
3. Mount the ISO.

```
$ mount -o ro path_to_ISO /mnt
```

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

4. Extract **.tar** file content.

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

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.

Additional resources

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

6.2. CREATING A BOOT ISO INSTALLER IMAGE USING IMAGE BUILDER IN THE GUI

You can build a customized boot ISO installer image using image builder GUI.

Prerequisites

- You have created a blueprint for your image. See Creating an image builder blueprint in the web console interface.

Procedure

1. Open the image builder interface of the RHEL web console in a browser. See Creating an image builder blueprint using the command-line interface.
2. Locate the blueprint that you want to use to build your image. To do so, enter the blueprint name or a part of it into the search box at upper left, and click **Enter**.
3. On the right side of the blueprint, click the corresponding **Create Image** button.
The **Create image** dialog wizard opens.
4. On the **Create image** dialog wizard:

a. In the **Image Type** list, select **"RHEL Installer (.iso)"**.

b. Click **Create**.

Image builder adds the compose of a RHEL ISO image to the queue.

NOTE

The image build process takes a few minutes to complete.

After the process is complete, you can see the image build complete status. 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
```

You can use the resulting ISO image file on a hard disk or to boot in a virtual machine, for example, in an HTTP Boot or a USB installation.

Additional resources

- Creating an image builder blueprint in the web console interface .
- Creating system images with image builder command-line interface .
- Creating a bootable installation medium for RHEL .

6.3. INSTALLING AN IMAGE BUILDER ISO IMAGE TO A BARE METAL SYSTEM

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

Prerequisites

- You created the bootable ISO image by using image builder. See Creating a boot ISO installer image using the image builder in the command-line interface.
- 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 recommended 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
- Customizing your installation
- Creating a bootable USB device on Linux

CHAPTER 7. CREATING PRE-HARDENED IMAGES WITH IMAGE BUILDER OPENSCAP INTEGRATION

Image builder on-premise supports OpenSCAP integration to produce pre-hardened RHEL images. With image builder on-premise integrated with OpenSCAP, you can produce pre-hardened RHEL images. You can set up a blueprint, choose from a set of predefined security profiles, add a set of packages or add-on files, and 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.

WARNING

Image builder does not include support for FIPS boot mode. As a result, OpenSCAP profiles that require FIPS mode to be enabled, such as DISA STIG, are not currently supported.

7.1. DIFFERENCES BETWEEN KICKSTART AND PRE-HARDENED IMAGES

For the traditional image creation using a Kickstart file, you have to choose which packages you must install and ensure that the system is not affected by a vulnerability. With image builder OpenSCAP integration, you can build security hardened images. During the image build process an OSBuild **oscap remediation stage** runs the **OpenSCAP** tool in the chroot, on the filesystem tree. The **OpenSCAP** tool runs the standard evaluation for the profile you choose and applies the remediations to the image. With this, you can build a more completely hardened image, if you compare it to running the remediation on a live system.

7.2. INSTALLING OPENSCAP

Install the OpenSCAP tool to have access to SCAP tools to help you to create standard security checklists for your systems.

Procedure

1. Install OpenSCAP on your system:

```
# *dnf install openscap-scanner*
```

2. Install **scap-security-guide** package:

```
# *dnf install scap-security-guide*
```

After the installation is completed you can start using the **oscap** command line tool. The SCAP content will be installed in the **/usr/share/xml/scap/ssg/content/** directory.

7.3. THE OPENSCAP BLUEPRINT CUSTOMIZATION

With the OpenSCAP support of blueprint customization, you can create blueprints and then use them to build your own pre-hardened images. To create a pre-hardened image you can customize the mount points and configure the file system layout according to the selected security profile. During the image build, OpenSCAP applies a first-boot remediation.

After you select the OpenSCAP profile, the OpenSCAP blueprint customization configures the image to trigger the remediation during the image build with the selected profile.

To use the OpenSCAP blueprint customization in your image blueprints, enter the following information:

- The datastream path to the **datastream** remediation instructions. The datastream path is located in the /usr/share/xml/scap/ssg/content/ directory.
- The **profile_id** of the required security profile. The **profile_id** field accepts both the long and short forms, for example: **cis** or **xccdf_org.ssgproject.content_profile_cis**. See SCAP Security Guide profiles supported in RHEL 9 for more details.
  The following is a blueprint with OpenSCAP customization example:

```
[customizations]
[customizations.openscap]
datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml"
profile_id = "xccdf_org.ssgproject.content_profile_cis"
```

  The most common SCAP file type is an SCAP source datastream. To show details about the SCAP source datastream from the **scap-security-guide** package, enter the command:

```
$ oscap info /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

  The **oscap** tool runs on the image tree to perform an offline scan of a file system that is mounted at an arbitrary path. You can use it for scanning of custom objects that are not supported by **oscap-docker** or **oscap-vm**, such as containers in formats other than Docker. **oscap-chroot** mimics the usage and options of the **oscap** tool.

  Image builder generate the necessary configurations for the **osbuild** stage based on your blueprint customizations. Additionally, image builder adds two packages to the image:

- **openscap-scanner** - the **OpenSCAP** tool.
- **scap-security-guide** - package which contains the remediation 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 9 .

7.4. CREATING A PRE-HARDENED IMAGE WITH IMAGE BUILDER

With the OpenSCAP and image builder integration, you can create pre-hardened images.

Procedure

1. Create a blueprint in the TOML format, with the following content:

```
name = "blueprint_name"
description = "blueprint_description"
version = "0.0.1"
modules = []
groups = []
distro = ""

[customizations]
[[customizations.user]]
name = "scap-security-guide"
description = "Admin account"
password = secure_password_hash
key = ssh-key
home = "/home/scap-security-guide"
group = ["wheel"]


[[customizations.filesystem]]
mountpoint = "/tmp"
size = 13107200
[customizations.openscap]
datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml "
profile_id = "cis"
```

2. Start the build of a OpenSCAP image:

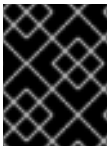
```
# composer-cli compose start blueprint_name qcow2
```

Where *blueprint_name* is the blueprint name.

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 in a VM, you can perform a configuration compliance scan to verify that the image is aligned to the selected security profile.



IMPORTANT

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

1. Connect to the image using SSH.
2. Run the **oscap** scanner.

```
# scap-workbench
```

3. Select the version of the system you want to scan. Click **Load content**.
4. Select the profile you want to scan and click **Scan**. OpenSCAP checks all the requirements for the system.
5. After the scan finishes, click **Show Report**.

Additional resources

- Scanning the system for configuration compliance and vulnerabilities .
- OpenSCAP available security profiles .

CHAPTER 8. PREPARING AND DEPLOYING KVM GUEST IMAGES USING IMAGE BUILDER

To create a purpose-built image for use on Red Hat Virtualization, you can use image builder to compose a **KVM** guest image. The KVM guest images are only supported by **rhel-guest-image** in Red Hat Virtualization.

When you create a disk image by using RHEL image builder, even if you specify your own partition layout, it creates the partitioning table in the GUID Partition Table (GPT) layout, with a dual-bootable image: BIOS mode or UEFI mode. It then creates a BIOS boot partition, along with the UEFI partition.

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

1. Creating a KVM guest Image **.qcow2** image using image builder.
2. Creating a virtual machine from the KVM guest image.

8.1. CREATING CUSTOMIZED KVM GUEST IMAGES USING IMAGE BUILDER

You can create a customized **.qcow2** KVM guest image by using image builder.

Prerequisites

- You must have root or wheel group user access to the system.
- The **cockpit-composer** package is installed.
- On a RHEL system, you have opened the image builder dashboard of the web console.
- You have created a blueprint. See Creating an image builder 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. A pop-up 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

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

Additional resources

- Creating an image builder blueprint in the web console interface .

8.2. CREATING A VIRTUAL MACHINE FROM A KVM GUEST IMAGE

With image builder, you can build a **.qcow2** image, and use a KVM guest image to create a VM. The KVM guest images created using image builder already have **cloud-init** installed and enabled.

Prerequisites

- You created a **.qcow2** image using image builder. See Creating an image builder blueprint in the web console interface.
- The **qemu-kvm** package is installed on your system. You can check if the **/dev/kvm** folder is available on your system.
- You have the **libvirt** and **virt-install** packages installed on your system.
- The **genisoimage** utility is installed on your system.

Procedure

1. Move the KVM Guest Image you created using 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
```

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

5. Use the **genisoimage** command 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 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-9-x86_64-kvm.qcow2,device=disk,bus=virtio,format=qcow2 \
    --disk cloud-init.iso,device=cdrom \
    --os-variant rhel 9 \
    --virt-type kvm \
    --graphics none \
    --import
```

Where,

- --graphics none - means it is a headless RHEL 9 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 9 (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 VM:

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 using the CLI.

Additional resources

- Enabling virtualization.
- Configuring and managing cloud-init for RHEL 9 .

- cloud-init significant directories and files.

CHAPTER 9. PUSHING A CONTAINER TO A REGISTRY AND EMBEDDING IT INTO AN IMAGE

With support for container customization in the blueprints, you can create a container and embed it directly into the image you create.

9.1. BLUEPRINT CUSTOMIZATION TO EMBED A CONTAINER INTO AN IMAGE

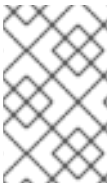
To embed a container from registry.access.redhat.com registry, you must add a container customization to your blueprint. For example:

```
[[containers]]
source = "registry.access.redhat.com/ubi9/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** - Optional boolean field. The **tls-verify** boolean field controls the transport layer security. The default value is **true**.
  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** like 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.

9.2. THE CONTAINER REGISTRY CREDENTIALS

The **osbuild-worker** service is responsible for the communication with the container registry. To enable that, you can 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.

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.

The following is an example:

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

Additional resources

- **containers-auth.json** man page.

9.3. PUSHING A CONTAINER ARTIFACT DIRECTLY TO A CONTAINER REGISTRY

You can push container artifacts, such as RHEL for Edge container images directly to a container registry after you build it, using the image builder CLI. For that you must set up an **upload provider** and optionally, credentials, and then you can build the container image, passing the registry and the repository to **composer-cli** as arguments. After the image is ready, it is available in the container registry that you set up.

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.

```
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:

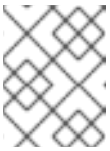
```
# composer-cli blueprints push blueprint.toml
```

4. Build the container image:

```
# 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.
6. Access 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.
```

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

Additional resources

- Quay.io - Working with tags.

9.4. 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.
- You have access to registry.access.redhat.com.
- You have a container **manifest ID**.
- You have the **qemu-kvm** and **qemu-img** packages installed. To install it, run the command:

```
# yum install qemu-kvm qemu-img
```

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-90"
```

```
[[packages]]
name = "podman"
version = "*"

[[containers]]
source = "registry.access.redhat.com/ubi9:8080/osbuild/container/container-image@sha256:manifest-ID-from-Repository-tag: tag-version"
name =  "source-name"
tls-verify = true
```

2. Push the blueprint:

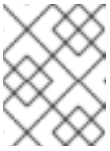
```
# composer-cli blueprints push blueprint-image.toml
```

3. Build the container image:

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

Where:

- *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.

After the image build status is "FINISHED", you can use the **qcow2** image you created in a VM.

Verification

1. Pull the **.qcow2** image from the **composer-cli** to your local file system:

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

2. Start the **qcow2** image in a VM. See Creating a virtual machine from a KVM guest image .
3. 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**.
4. Run the container image and open a shell prompt inside the container:

```
# podman run -it registry.access.redhat.com/ubi9:8080/osbuild/repository /bin/bash/
```

Where:

- **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.

5. 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.
- Running Skopeo in a container .

CHAPTER 10. PREPARING AND UPLOADING CLOUD IMAGES USING IMAGE BUILDER

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 image builder using the respective 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. See Pushing images to AWS Cloud AMI and Pushing VHD images to Microsoft Azure cloud.

10.1. PREPARING TO 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
# dnf install python3-pip
```

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

```
# pip3 install awscli
```

3. Run the following command to 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 use the following command to 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:

```
$ printf '{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Service":
```

```
"vmie.amazonaws.com" }, "Action": "sts:AssumeRole", "Condition": { "StringEquals":{
"sts:Externalid": "vmimport" } } } ] }' > trust-policy.json
$ printf '{ "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 > role-policy.json
$ aws iam create-role --role-name vmimport --assume-role-policy-document file://trust-
policy.json
$ 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

10.2. UPLOADING AN AMI IMAGE TO AWS USING THE CLI

You can use image builder to build **ami** images and push them directly to Amazon AWS Cloud service provider 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.

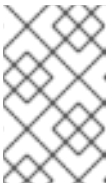
2. Save the file as *CONFIGURATION-FILE*.toml and close the text editor.

3. Start the compose:

```
# 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 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 and upload it to AWS:

```
# 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

10.3. PUSHING IMAGES TO AWS CLOUD AMI

You can 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 image builder interface of the RHEL web console in a browser.
- You have create a blueprint. See Creating an image builder 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. Click the **blueprint name**.

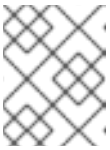
2. Select the tab **Images**.

3. Click **Create Image** to create your customized image.
   A pop-up window opens.

   a. From the **Type** drop-down menu list, select **Amazon Machine Image Disk (.raw)**.

   b. Check the **Upload to AWS** check box 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, you can 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 small 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. Click Service→EC2 on the menu and choose the correct region in the AWS console. The image must have the **Available** status, to indicate that it is uploaded.

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

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

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

9. 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.

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

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

12. Click **Connect**. A pop-up window appears with instructions on how to connect 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 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 using SSH.

Verification

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

Additional resources

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

10.4. PREPARING TO UPLOAD MICROSOFT AZURE VHD IMAGES

You can use image builder to prepare a VHD image that can be uploaded to **Microsoft Azure** cloud.

Prerequisites

- You must have a usable Microsoft Azure resource group and storage account.
- You have python2 installed because the **AZ CLI** tool depends specifically on python 2.7.

Procedure

1. Import the Microsoft repository key:

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

2. Create a local **azure-cli** repository information:

```
# sh -c 'echo -e "[azure-cli]\nname=Azure
CLI\nbaseurl=https://packages.microsoft.com/yumrepos/azure-
cli\nenabled=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/yum.repos.d/azure-cli.repo'
```

3. Install the Microsoft Azure CLI:

```
# dnfdownloader azure-cli
# rpm -ivh --nodeps azure-cli-2.0.64-1.el7.x86_64.rpm
```

NOTE

The downloaded version of the Microsoft Azure CLI package may vary depending on the current available version.

4. Run the 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 https://microsoft.com/devicelogin 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 page https://microsoft.com/devicelogin and enter the device code to authenticate.

5. List the keys for the storage account in Microsoft Azure:

```
$ GROUP=resource-group-name
$ ACCOUNT=storage-account-name
$ az storage account keys list --resource-group $GROUP --account-name $ACCOUNT
```

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

NOTE

You can list the available resources using the following command:

```
$ az resource list
```

6. Make note of value **key1** in the output of the previous command, and assign it to an environment variable:

```
$ KEY1=value
```

7. Create a storage container:

```
$ CONTAINER=storage-account-name
$ az storage container create --account-name $ACCOUNT \
--account-key $KEY1 --name $CONTAINER
```

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

Additional resources

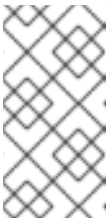
- Microsoft Azure CLI.

10.5. UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

After you have created your customized VHD image, you can upload it to the Microsoft Azure cloud.

Prerequisites

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



NOTE

When creating a **.vhd** image using the CLI, 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.

Procedure

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

```
$ VHD=25ccb8dd-3872-477f-9e3d-c2970cd4bbaf-disk.vhd
$ az storage blob upload --account-name $ACCOUNT --container-name $CONTAINER --file
$VHD --name $VHD --type page
...
```

2. After the upload to the Microsoft Azure Blob storage completes, create a Microsoft Azure image from it:

```
$ az image create --resource-group $GROUP --name $VHD --os-type linux --location eastus
--source https://$ACCOUNT.blob.core.windows.net/$CONTAINER/$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 $GROUP --location eastus --name $VHD --image $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**.

Additional Resources

- Composing an image for the **.vhd** format fails.

10.6. PUSHING VHD IMAGES TO MICROSOFT AZURE CLOUD

You can create **.vhd** images using image builder. Then, you can push the output **.vhd** images to a Blob Storage of the Microsoft Azure Cloud service provider.

Prerequisites

- You have root access to the system.
- You have access to the image builder interface of the RHEL web console.
- You created a blueprint. See Creating an 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 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** check box 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 image build and upload processes start.

Next Steps

1. To access the image you pushed into **Microsoft Azure Cloud**, access the Microsoft Azure portal.
2. In the search bar, type "storage account" and click **Storage accounts** from the list.
3. On the search bar, type "Images" and select the first entry under **Services**. You are redirected to the **Image dashboard**.
4. On the navigation panel, click **Containers**.
5. Find the container you created. Inside the container is the ***your_image_name*.vhd** file you created and pushed.

Verification

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

1. In the search bar, type images account and click **Images** from the list.
2. Click **+Create**.
3. From the dropdown list, choose the resource group you used earlier.
4. Enter a name for the image.
5. For the **OS type**, select **Linux**.
6. For the **VM generation**, select **Gen 2**.
7. Under **Storage Blob**, click **Browse** and click through the storage accounts and container until you reach your VHD file.
8. Click **Select** at the end of the page.
9. Choose an Account Type, for example, **Standard SSD**.
10. Click **Review + Create** and then **Create**. Wait a few moments for the image creation.

To launch the VM, follow the steps:

1. Click **Go to resource**.
2. Click **+ Create VM** from the menu bar on the header.
3. Enter a name for your virtual machine.
4. Complete the **Size** and **Administrator account** sections.

5. 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.

6. Open a terminal to create an SSH connection to connect to VM.

Additional resources

- Microsoft Azure Storage Documentation .
- Create a Microsoft Azure Storage account .
- Open a case on Red Hat Customer Portal .
- Help + support.
- Contacting Red Hat.

10.7. UPLOADING VMDK IMAGES AND CREATING A RHEL VIRTUAL MACHINE IN VSPHERE

Upload a .vmdk image to VMware vSphere using the **govc import.vmdk** CLI tool.

NOTE

Uploading the image via the UI is not supported.

Prerequisites

- You created a blueprint with username and password customizations.
- You created an **.vmdk** image by using image builder and downloaded it to your host system.
- You installed the **govc import.vmdk** CLI tool.
- You configured the **govc import.vmdk** CLI tool client.
  - you must set the following values in the environment:

```
GOVC_URL
GOVC_DATACENTER
GOVC_FOLDER
GOVC_DATASTORE
GOVC_RESOURCE_POOL
GOVC_NETWORK
```

Procedure

1. Navigate to the directory where you downloaded your **.vmdk** image.
2. Launch the image on vSphere by following the steps:
   a. Import the **.vmdk** image in to vSphere:

```
$ govc import.vmdk ./composer-api.vmdk foldername
```

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="foldername/composer-api.vmdk" \
-disk.controller=scsi -on=false \
 vmname
```

c. Power-on the VM:

```
govc vm.power -on vmname
```

d. Retrieve the VM IP address:

```
HOST=$(govc vm.ip vmname)
```

e. Use SSH to log in to the VM, using the username and password you specified in your blueprint:

```
$ ssh admin@HOST
```

NOTE

If you copied the **.vmdk** image from your local host to the destination using the **govc datastore.upload** command, using the image is not supported. There is no option to use the **import.vmdk** command in the vSphere GUI and as a consequence, the vSphere GUI does not support the direct upload, as a consequence, the **.vmdk** image is not directly usable from the vSphere GUI.

10.8. UPLOADING IMAGES TO GCP WITH IMAGE BUILDER

With 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.

10.8.1. Uploading a gce image to GCP using the CLI

Follow the procedure to set up a configuration file with credentials to upload your **gce** image to GCP.

Prerequisites

- You have a user or service account Google credentials to upload images to GCP. 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

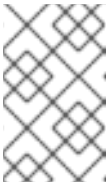
1. Using a text editor, create a **gcp-config.toml** configuration file with the following content:

```
provider = "gcp"

[settings]
bucket = "GCP_BUCKET"
region = "GCP_STORAGE_REGION"
object = "OBJECT_KEY"
credentials = "GCP_CREDENTIALS"
```

Where:

- **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** is optional if you use a different mechanism to authenticate with GCP. For more details on different ways to authenticate with GCP, see Authentication with GCP.

2. 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
```

Note: 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.

10.8.2. Authenticating with GCP

You can use several different types of credentials with image builder to authenticate with GCP. If image builder configuration is set to authenticate with GCP 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 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 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.

10.8.2.1. Specifying credentials with the composer-cli command

You can specify GCP authentication credentials in the provided upload target configuration **gcp-config.toml**. Use a **Base64**-encoded scheme of the Google account credentials JSON file to save time.

Procedure

- In the provided upload target configuration **gcp-config.toml**, set the credentials:

```
provider = "gcp"

[settings]
provider = "gcp"

[settings]
...
credentials = "GCP_CREDENTIALS"
```

- To get the encoded content of the Google account credentials file with the path stored in **GOOGLE_APPLICATION_CREDENTIALS** environment variable, run the following command:

```
$ base64 -w 0 "${GOOGLE_APPLICATION_CREDENTIALS}"
```

10.8.2.2. 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"
```

10.9. PUSHING VMDK IMAGES TO VSPHERE USING THE GUI IMAGE BUILDER TOOL

You can build VMware images by using the GUI image builder tool and push the images directly to your vSphere instance, to avoid having to download the image file and push it manually. To create **.vmdk** images using image builder directly to vSphere instances service provider, follow the steps:

Prerequisites

- You have **root** or **wheel** group user access to the system.
- You have opened the https://localhost:9090/ interface of the RHEL web console in a browser.
- You have created a blueprint. See Creating an image builder blueprint in the web console interface.
- 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 2GB.
   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:
   a. **Image name**: a name for the image to be uploaded.
   b. **Host**: The URL of your VMware vSphere where the image will be uploaded.
   c. **Cluster**: The name of the cluster where the image will be uploaded.
   d. **Data center**: The name of the data center where the image will be uploaded.

e. **Data store**:The name of the Data store where the image will be uploaded.

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.

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.

NOTE

The image build and upload processes take a few minutes to complete.

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. You can create a new VM from the image you uploaded:
   a. Select the image you uploaded.
   b. Right-click the selected image.
   c. Click **New Virtual Machine**.
   A **New Virtual Machine** window opens.

   In the **New Virtual Machine** window, provide the following details:

      i. Select **New Virtual Machine**.
      ii. Select a name and a folder for your VM.
      iii. Select a computer resource: choose a destination computer resource for this operation.
      iv. Select storage: For example, select NFS-Node1
      v. Select compatibility: The image should be BIOS only.
      vi. Select a guest operating system: For example, select *Linux* and *Red Hat Fedora (64-bit)*.
      vii. **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:
      viii. Ready to complete: Review the details and click **Finish** to create the image.

   d. Navigate to the **VMs** tab.

i. From the list, select the VM you created.

ii. Click the **Start** button from the panel. A new window appears, showing the VM image loading.

iii. Log in with the credentials you created for the blueprint.

iv. You can verify if the packages you added to the blueprint are installed. For example:

```
$ rpm -qa | grep firefox
```

Additional resources

- Installing the vSphere Client.

10.10. PUSHING CUSTOMIZED IMAGES TO OCI

With image builder, you can build customized images and 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 **wheel** 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 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** check box 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**.

Image builder adds the compose of a RHEL **.qcow2** image to the queue.

Verification

1. Access the OCI dashboard → 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.
- Generating SSH keys.

10.11. UPLOADING QCOW2 IMAGES TO OPENSTACK

With the image builder tool, you can create customized **.qcow2** images that are suitable for uploading to OpenStack cloud deployments, and starting instances there.

WARNING

Do not mistake the generic **QCOW2** image type output format you create by using image builder with the OpenStack image type, which is also in the QCOW2 format, but contains further changes specific to OpenStack.

Prerequisites

- You have created a blueprint.
- created a **QCOW2** image by using image builder. See

Procedure

1. Start the compose of a **QCOW2** image.

```
# composer-cli compose start blueprint_name openstack
```

2. Check the status of the building.

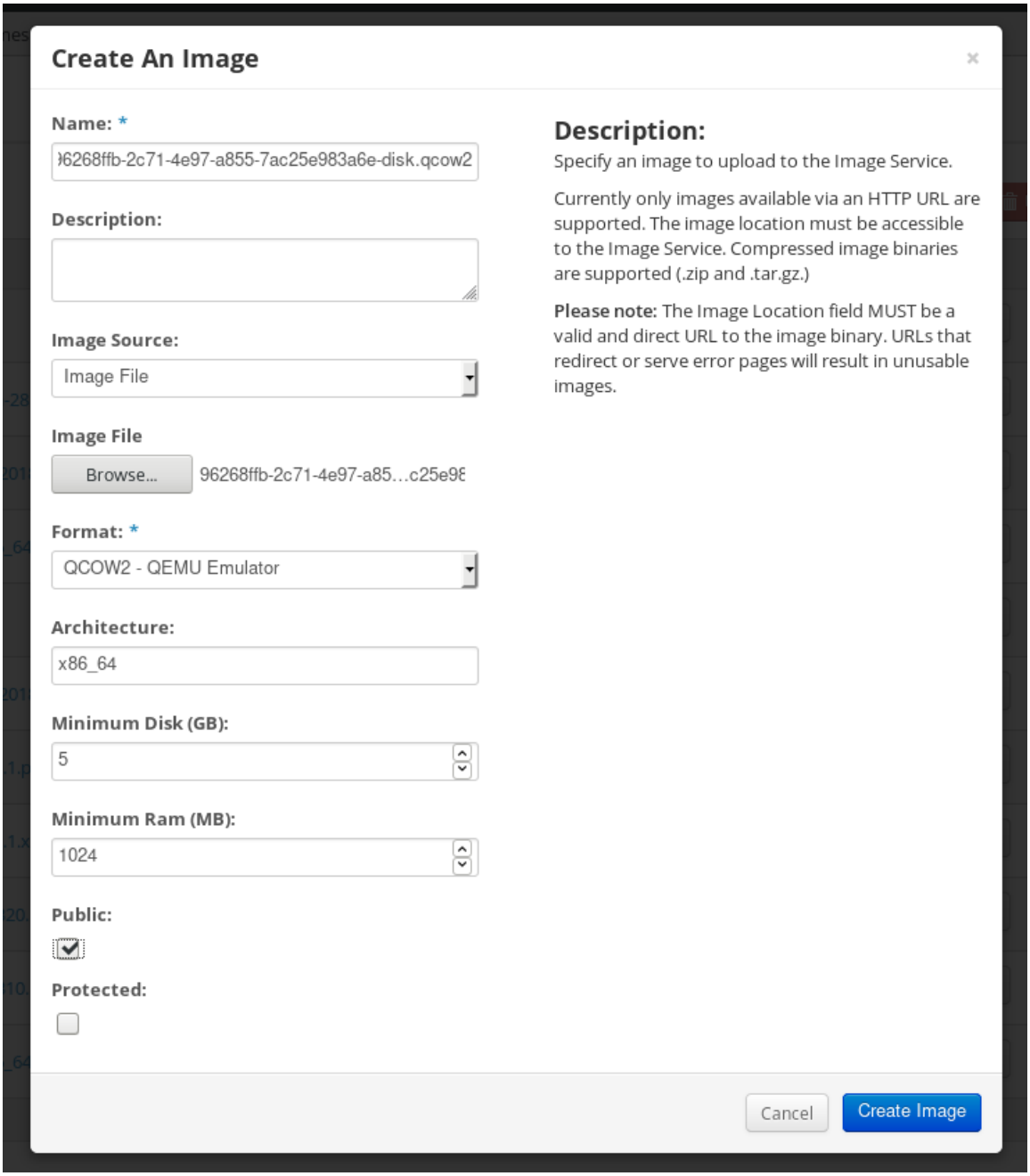
```
# composer-cli compose status
```

After the image build finishes, you can download the image.

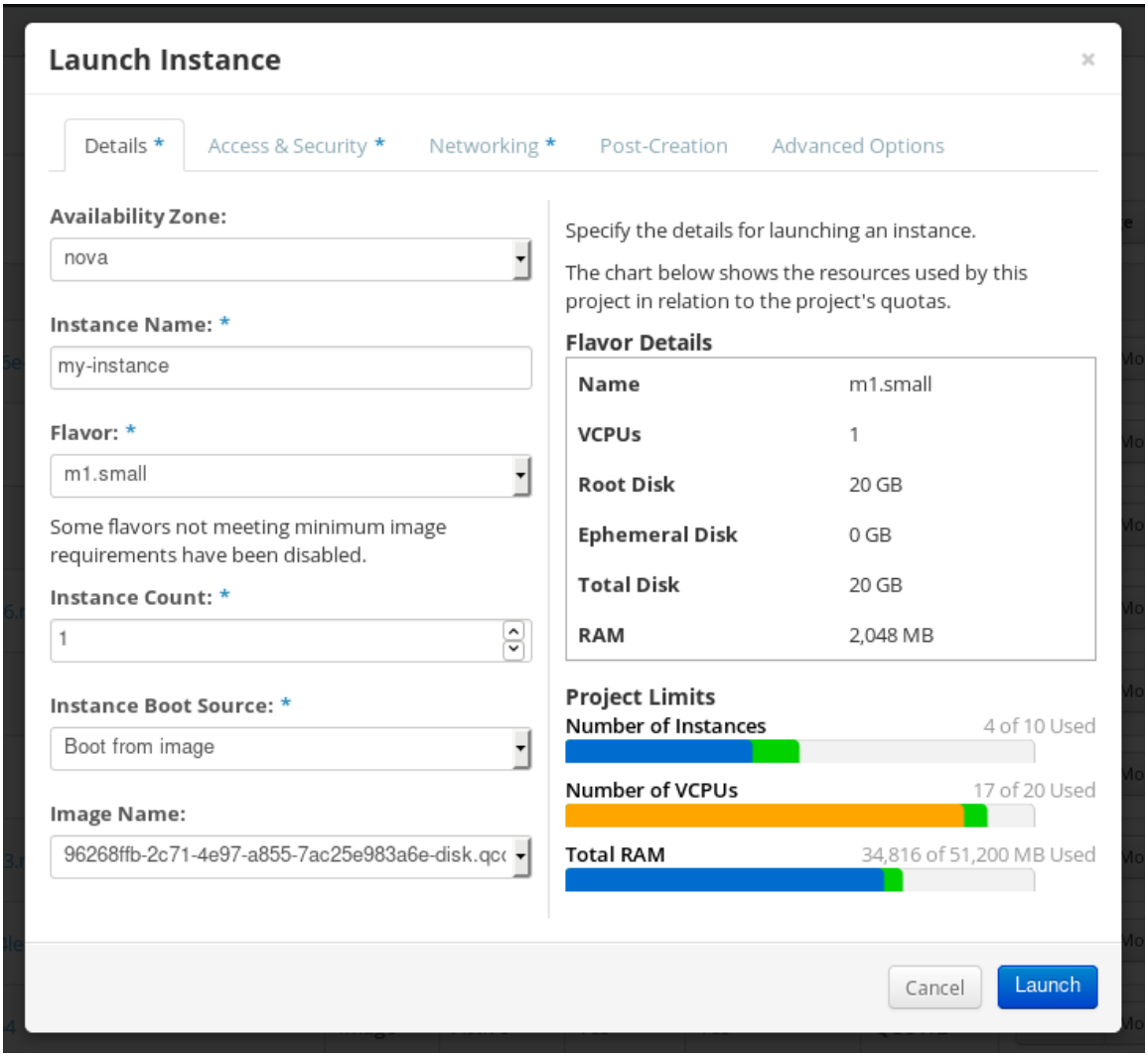
3. Download the **QCOW2** image:

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

4. Access the OpenStack dashboard and click **+Create Image**.
5. On the left menu, select the **Admin** tab.
   a. From the **System Panel**, click **Image**.
   The **Create An Image** wizard opens.
6. 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**.



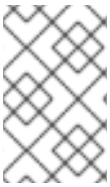
7. On the left menu, select the **Project** tab.
   a. From the **Compute** menu, select **Instances**.
   b. Click the **Launch Instance** button.
      The **Launch Instance** wizard opens.
   c. On the **Details** page, enter a name for the instance. Click **Next**.
   d. On the **Source** page, select 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**.



8. 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**.

10.12. PREPARING TO UPLOAD CUSTOMIZED RHEL IMAGES TO ALIBABA

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

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 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 http://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. See link: Detection items section.

Additional resources

- Image Compliance Tool.

10.13. UPLOADING CUSTOMIZED RHEL IMAGES TO ALIBABA

You can upload the customized **AMI** image you created by using image builder to the Object Storage Service (OSS).

Prerequisites

- Your system is set up for uploading Alibaba images. See Preparing for uploading images to Alibaba.
- You have created an **ami** image by using 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 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.
7. Click **Open**.

As a result, the customized **AMI** image is uploaded to the OSS Console.

Additional resources

- Upload an object.
- Creating an instance from custom images.
- Importing images.

10.14. IMPORTING IMAGES TO ALIBABA

To import a customized Alibaba RHEL image that you created by using image builder to the Elastic Cloud Console (ECS), follow the steps:

Prerequisites

- Your system is set up for uploading Alibaba images. See Preparing for uploading images to Alibaba.
- You have created an **ami** image by using 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). See Uploading images to Alibaba.

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. Optionally, 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.

10.15. CREATING AN INSTANCE OF A CUSTOMIZED RHEL IMAGE USING ALIBABA

You can create instances of a customized RHEL image using **Alibaba ECS Console**.

Prerequisites

- You have activated OSS and uploaded your custom image.
- You have successfully imported your image to ECS Console. See Importing images to Alibaba .

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.