Red Hat Enterprise Linux 8

Composing a customized RHEL system image

Creating customized system images with Image builder on Red Hat Enterprise Linux 8

Last Updated: 2022-11-29

Red Hat Enterprise Linux 8 Composing a customized RHEL system image

Creating customized system images with Image builder on Red Hat Enterprise Linux 8

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 Bugzilla (account required)

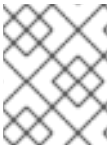
1. Log in to the Bugzilla website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. IMAGE BUILDER DESCRIPTION

To deploy a system on a cloud platform, create a system image. To create RHEL images, RHEL 8 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
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 the 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.

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**

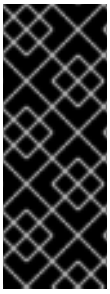
   ```
   # yum install osbuild-composer composer-cli cockpit-composer bash-completion
   ```

   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 image builder service 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
```

2.2. REVERTING TO LORAX-COMPOSER IMAGE BUILDER BACKEND

The **osbuild-composer** backend, though much more extensible, does not currently achieve feature parity with the previous **lorax-composer** backend.

To revert to the previous backend, follow the steps:

Prerequisites

- You have installed the **osbuild-composer** package

Procedure

1. Remove the osbuild-composer backend.

```
# yum remove osbuild-composer
# yum remove weldr-client
```

2. In the /**etc/yum.conf file**, add an exclude entry for **osbuild-composer** package.

```
# cat /etc/yum.conf
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
skip_if_unavailable=False
exclude=osbuild-composer weldr-client
```

3. Install the **lorax-composer** package.

```
# yum install lorax-composer composer-cli
```

4. Enable and start the **lorax-composer** service to start after each reboot.

```
# systemctl enable --now lorax-composer.socket
# systemctl start lorax-composer
```

Additional resources

- Create a Case at Red Hat Support .

CHAPTER 3. MANAGING IMAGE BUILDER REPOSITORIES

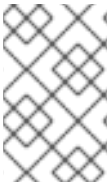
3.1. IMAGE BUILDER DEFAULT SYSTEM REPOSITORIES

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 YUM repository format known from the files in **/etc/yum.repos.d/**. Instead, they are simple JSON files.

3.2. OVERRIDING A SYSTEM REPOSITORY

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



NOTE

Prior to RHEL 8.5 release, the name of the repository overrides is **rhel-8.json**. Starting from RHEL 8.5, the names also respect the minor version: **rhel-84.json**, **rhel-85.json**, and so on.

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 8, use /etc/osbuild-composer/repositories/rhel-85.json
```

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

```
{
   "<ARCH>": [
      {
         "name": "baseos",
         "metalink": "",
         "baseurl": "http://mirror.example.com/composes/released/RHEL-
8/8.0/BaseOS/x86_64/os/",
         "mirrorlist": "",
```

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

Specify only one of the following attributes:

- **metalink** - offers increased availability and error correction on a repository link.
- **mirrorlist** - a list of URLs that point to package repositories.
- **baseurl** - a link to the repository that contains the packages required for the installation. 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-8.json** with your RHEL version, for example, **rhel-8.json**.
        In the following command, replace **rhel-version.json** with your RHEL version, for example: rhel-8.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-8.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

- latest RPMs version available in repository not visible for **osbuild-composer**.

3.3. 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-8/8.0/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-8/8.0/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 use 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 WITH 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, use the command-line interface which is currently the preferred method to use Image Builder.

4.1. IMAGE BUILDER COMMAND-LINE INTERFACE

Image Builder command-line interface is currently the preferred method to use Image Builder. It offers more functionality than the web console interface. To use this interface, run the **composer-cli** command with 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, 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 WITH COMMAND-LINE INTERFACE

This procedure describes how to create a new Image Builder blueprint using the command-line interface.

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

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

- For a specific version, use the exact version number such as **8.6.0**.
- For latest available version, use the asterisk *****.
- For the latest minor version, use formats such as **8.***.

3. You can customize your blueprints in a number of ways. For this example, Simultaneous Multi Threading (SMT) can be disabled by performing the steps below. For additional customizations available, see Supported Image Customizations.

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

4. Save the file 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.

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

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

7. To display the blueprint configuration you have just added, run the command:

```
# composer-cli blueprints show
```

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

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

NOTE

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

Verification

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

To edit an existing Image Builder blueprint in the command-line interface, follow the steps.

Procedure

1. Save (export) the blueprint to a local text file:

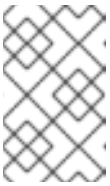
```
# composer-cli blueprints save BLUEPRINT-NAME
```

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

```
packages = []
```

   b. Increase the version number. 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
```

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

This procedure shows how to build a custom image using the Image Builder command-line interface.

Prerequisites

- You have a blueprint prepared for the image.

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 image. For possible values, see 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 a status value **FINISHED**. Identify the compose in the list by 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 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
```

Additional resources

- The *composer-cli*(1) manual page provides a full list of the available subcommands and options:

  ```
  $ man composer-cli
  ```

- The **composer-cli** command provides help on the subcommands and options:

```
# composer-cli help
```

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 blueprint metadata

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

Replace *BLUEPRINT-NAME* and *LONG FORM DESCRIPTION TEXT* with a name and description for your blueprint.

Replace *VERSION* with a version number according to the Semantic Versioning scheme.

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

The entry *modules* describe the package names and matching version glob to be installed into the image.

The entry *group* describes a group of packages to be installed into the image. Groups categorize their packages in:

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

Groups to include in the image

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

Replace *group-name* with the name of the group, such as **anaconda-tools**, **widget**, **wheel** or **users**.

Packages to include in the image

```
[[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 a version to use. This field supports **dnf** version specifications:

- For a specific version, use the exact version number such as **8.30**.

- For latest available version, use the asterisk *****.
- For the latest minor version, use format such as **8.***.

Repeat this block for every package to include.

4.7. SUPPORTED IMAGE CUSTOMIZATIONS

You can use several image customizations within blueprints. To make use of these options, you must initially configure the customizations in the blueprint and import (push) it to Image Builder.

NOTE

These customizations are not supported within the web console.

Select a package group

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

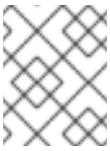
Replace "*package_group_name*" with the name of the group. For example, "@server with gui".

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
```



NOTE

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. The following command installs the **python3** package.

```
# yum 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.

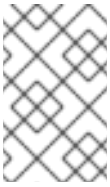
Group specifications for the resulting system image

```
[[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

This option 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 in this section.

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 the language and 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.

Set the firewall for the resulting system image

```
[customizations.firewall]
port = ["PORTS"]
```

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

Customize the firewall services

Review 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"]
```

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

You can optionally customize the firewall services for the system image that you plan to create.

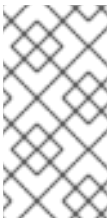
NOTE

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.



NOTE

Each time a build starts, it clones the repository. 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

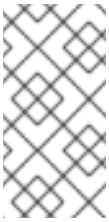
You can specify a custom filesystem configuration in your blueprints and therefore create images with a specific disk layout, instead of the default layout configuration. By using the non-default layout configuration in your blueprints, you can benefit from:

- security benchmark compliance
- protection against out-of-disk errors
- performance
- consistency with existing setups
  Customize the filesystem configuration in your blueprint:

```
[[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 early 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, you can create images with a customized file system partition on LVM and resize those partitions at runtime. For that, 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

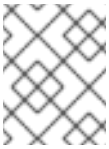
From 8.6 onward, for the **osbuild-composer-46.1-1.el8** RPM and later version, the physical partitions are no longer available and filesystem customizations creates logical volumes.

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

The **MINIMUM-PARTITION-SIZE** 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 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"
```

Additional resources

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

4.8. INSTALLED PACKAGES

When you create a system image using Image Builder, by default, the system installs a set of base packages. The base list of packages are the members of the **comps core** group. By default, Image Builder uses the **core yum** group.

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, yum, 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
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
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 type	Default Packages
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, you must make sure that the packages in the components you added do not conflict with any other package components, otherwise the system fails to solve dependencies. As a consequence, you are not able to create your customized image.

Additional resources

- Image Builder description

4.9. ENABLED SERVICES

When you configure the custom image, the services enabled are the defaults services for the RHEL release you are running **osbuild-composer** from, additionally the services enabled for specific image types.

For example, the **.ami** image type enables the services **sshd**, **chronyd** and **cloud-init** and without these services, the custom image does not boot.

Table 4.2. Enabled services to support image type creation

Image type	Enabled Services
ami	sshd, cloud-init, cloud-init-local, cloud-config, cloud-final
openstack	sshd, cloud-init, cloud-init-local, cloud-config, cloud-final

Image type	Enabled Services
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, for image types with services enabled by default, the customization does not override this feature.

Additional resources

- Supported Image Customizations

CHAPTER 5. CREATING SYSTEM IMAGES WITH 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 IMAGE BUILDER GUI IN THE RHEL WEB CONSOLE

The **cockpit-composer** plug-in for the RHEL web console enables users to manage Image Builder blueprints and composes with a graphical interface. The preferred method for controlling Image Builder is the command-line interface.

Prerequisites

- You must have root access to the system.

Procedure

1. Open **https://localhost:9090/** in a web browser on the system where Image Builder is installed. For more information on how to remotely access Image Builder, see *Managing systems using the RHEL web console* document.
2. Log in to the web console with credentials for a user account with sufficient privileges on the system.
3. To display the Image Builder controls, click the **Image Builder** icon, in the upper-left corner of the window.
   The Image Builder view opens, listing existing blueprints.

Additional resources

- Creating system images with Image Builder command-line interface

5.2. CREATING AN IMAGE BUILDER BLUEPRINT IN THE WEB CONSOLE INTERFACE

To describe the customized system image, create a blueprint first.

Prerequisites

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

Procedure

1. Click **Create Blueprint** in the top-right corner.
   A dialog wizard with fields for the blueprint name and description opens.
2. Enter the name of the blueprint and, optionally, its description.
3. Click **Create**.

The Image Builder view opens, listing existing blueprints.

5.3. CREATING A SYSTEM IMAGE WITH IMAGE BUILDER IN THE WEB CONSOLE INTERFACE

You can create a system image from the 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. Click **Back to blueprints** to show the blueprints table.
2. On the blueprint table, find the blueprint you want to build an image.
    a. Optionally, you can find the blueprint using the search box. Enter the blueprint name.
3. On the right side of the chosen blueprint, click **Create Image**. The **Create image** dialog wizard opens.
4. On the **Image output** page, complete the following steps:
    a. From the **Image output type** list, select the image type you want.
        i. You can upload some images to their target cloud environment, such as **Amazon Web Service** and **Oracle Cloud Infrastructure**. For that, check the **Upload to Target cloud** box .
        ii. You are prompted to add credentials for the cloud environment on the next page.
5. From the **Image Size** field, enter the image size. The minimum size depends on the image type. Click **Next**.
6. On the **Upload to *Targeted_Cloud*** page, complete the following steps:
NOTE:This page is not visible if you did not check the box to upload your image to the cloud environment.
    a. On the **Authentication** page, enter the information related to your target cloud account ID and click **Next**.
    b. On the **Destination page**, enter the information related to your target cloud account type and click **Next**.
7. On the **Customizations** page, complete the following steps:
    a. On the **System** page, enter the Hostname. If you do not enter a hostname, the OS determines a hostname for your system.
    b. On the **Users** page, click **Add user**:
        i. Mandatory: Enter a Username.
        ii. Enter a password.
        iii. Enter an SSH key.

iv. Check the box if you want to make the user a Server administrator. Click **Next**.

8. On the **Package** page, complete the following steps:

   a. On the **Available packages** search field, enter the package name you want to add to your system image.

   NOTE

   Searching for the package can take some time to complete.

   b. Click the > arrow to add the selected package or packages. Click **Next**.

9. On the **Review** page, review the details about the image creation. Click **Save blueprint** to save the customizations you added to your blueprint. Click **Create image**.
   The image build starts and takes up to 20 minutes to complete.

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 has the commit and 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 embeds a pre-configured system image that you can install directly to a bare metal server.

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

This procedure shows how to build a custom boot ISO installer image using the Image Builder command-line interface.

Prerequisites

- You 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 may 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 resulting image file:

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

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

As a result, Image Builder builds a **.tar** file that contains the ISO Installer image.

Verification

1. Navigate to the folder where you downloaded the image file.
2. Locate the **.tar** image you downloaded.
3. Extract the **.tar** content.

You can use the resulting ISO image file on a hard drive 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 WITH IMAGE BUILDER IN THE GUI

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

Prerequisites

- You created a blueprint for your image.

Procedure

1. Open the Image Builder interface of the RHEL web console in a browser.
2. Locate the blueprint that you want to build your image by entering its name or a part of it into the search box at top left, and click **Enter**.
3. On the right side of the blueprint, click the **Create Image** button that belongs to the blueprint. The **Create image** dialog wizard opens.
4. On the **Create image** dialog wizard, from the **Image Type** list:
   a. Select the **"RHEL Installer (.iso)"** image type.
   b. Click **Create**.

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

NOTE

The image build process take 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 your image button.

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.

You can use the resulting ISO image file on a hard disk drive 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 THE 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.
- You have downloaded the bootable ISO image.
- You installed the **dd** tool.
- You have a USB flash drive with enough capacity. The recommended minimum is 8 GB.

NOTE

The ISO size might vary depending on the configuration specified in the blueprint, and especially depends on the packages you have added to the blueprint.

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. Connect the USB flash drive to the 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.

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. 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. You can find it 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 8 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 8 .

7.3. 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_pass
key = ssh-key
home = "home/user"
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**.
   You can see from the results that the system is secure.

Additional resources

- OpenSCAP.
- OpenSCAP available security profiles .

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

Customers can manually create images from an ISO or have a purpose-built image created using Image Builder. This procedure describes steps to create a purpose-built image using Image Builder. This is limited to **rhel-guest-image** support to Red Hat Virtualization (RHV).

Creating a customized KVM guest image following 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 WITH IMAGE BUILDER

This describes steps to create a **.qcow2** KVM guest image 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 Cockpit UI.

Procedure

1. Click **Create blueprint** to create a blueprint. See Creating an Image Builder blueprint in the web console interface.
2. Select the components and packages that you want as part of the KVM guest image you are creating.
3. Click **Commit** to commit the changes you made to the blueprint. A small pop-up on the superior right side informs you of the saving progress and then the result of the changes you commited.
4. Click the blueprint name link on the left banner.
5. Select the tab **Images**.
6. Click **Create Image** to create your customized image. A pop-up window opens.
7. From the **Type** drop-down menu list, select the `QEMU Image(.qcow2)' image.
8. Set the size that you want the image to be when instantiated and click **Create**.
9. 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 steps

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 **libvirt** installed on your system.
- You have **virt-install** 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 and rename the image name to **rhel-8.6-x86_64-kvm.qcow2**.
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-8.6-x86_64-kvm.qcow2,device=disk,bus=virtio,format=qcow2 \
    --disk cloud-init.iso,device=cdrom \
    --os-variant rhel8.6 \
    --virt-type kvm \
    --graphics none \
    --import
```

Where,

- --graphics none - means it is a headless RHEL 8.4 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 8.6 (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 8 .
- 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 depsolving 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 WITH IMAGE BUILDER

Image Builder can create custom system images ready for use in clouds of various providers. 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. From Red Hat Enterprise Linux 8.3, the ability to push customized image clouds through the **Image Builder** application in the RHEL web console is available for a subset of the service providers that we support, such as **AWS** and **Azure** clouds. See Pushing images to AWS Cloud AMI and Pushing VHD images to Azure cloud.

10.1. PREPARING FOR UPLOADING AWS AMI IMAGES

This describes steps to configure a system for uploading AWS AMI 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:

```
# yum install python3
# yum 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. Then, to grant permission to access the S3 bucket, create a **vmimport** S3 Role in 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 IN 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**, **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 choose 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

The ability to push the output image that you create to **AWS Cloud AMI** is available this time. This describes steps to push **.ami** images you create using Image Builder to Amazon AWS Cloud 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 must have an Access Key ID configured in the AWS IAM account manager.
- You must have a writable S3 bucket prepared.

Procedure

1. Click **Create blueprint** to create a blueprint. See Creating an Image Builder blueprint in the web console interface.
2. Select the components and packages that you want as part of the image you are creating.

3. Click **Commit** to commit the changes you made to the blueprint.
   A small pop-up on the superior right side informs you of the saving progress and then the result of the changes you commited.

4. Click **blueprint name** link on the left banner.

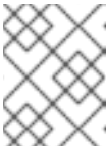
5. Select the tab **Images**.

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

   a. From the **"Type"** drop-down menu list, select the `"Amazon Machine Image Disk (.ami)"' image.

   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. We are using 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 .

7. A small pop-up on the superior right side 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.

8. 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.

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

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

11. Review your instance launch details. You can edit each section if you need to make any change. Click **Launch**

12. Before you launch the instance, you must 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. Alternatively, you can use **Image Builder** to add a user to the image with a preset public key. See Creating a user account with SSH key for more details.

    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.

13. Then, you can click **Launch Instance** to launch your instance.
You can check the status of the instance, it shows as **"Initializing"**.

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

15. Click **Connect**. A popup window appears with instructions on how to connect using SSH.

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

    b. In the location you store your private key, make sure 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 steps

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 FOR UPLOADING AZURE VHD IMAGES

This describes steps to upload an VHD image to Microsoft Azure cloud.

Prerequisites

- You must have a usable Microsoft Azure resource group and storage account.

Procedure

1. Install python2:

```
# yum install python2
```

NOTE

python2 package must be installed because since the AZ CLI depends specifically on python 2.7

2. Import the Microsoft repository key:

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

3. 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'
```

4. Install the Azure CLI:

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

NOTE

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

5. Run the Azure CLI:

```
$ az login
```

The terminal shows the message 'Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"' and open a browser where you can login.

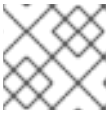
NOTE

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

6. List the keys for the storage account in 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 the Azure resource group and *storage-account-name* with name of the Azure storage account.



NOTE

You can list the available resources using the command:

```
$ az resource list
```

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

```
$ KEY1=value
```

8. 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

- Azure CLI

10.5. UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

This describes steps to upload an VHD image to Microsoft Azure.

Prerequisites

- Your system must be set up for uploading Azure VHD images.
- You must have an Azure VHD image created by Image Builder. Use the **vhd** output type in CLI or **Azure Disk Image (.vhd)** in GUI when creating the image.

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 completes, create an 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 ...
```

3. 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 ...
```

4. 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. UPLOADING VMDK IMAGES TO VSPHERE

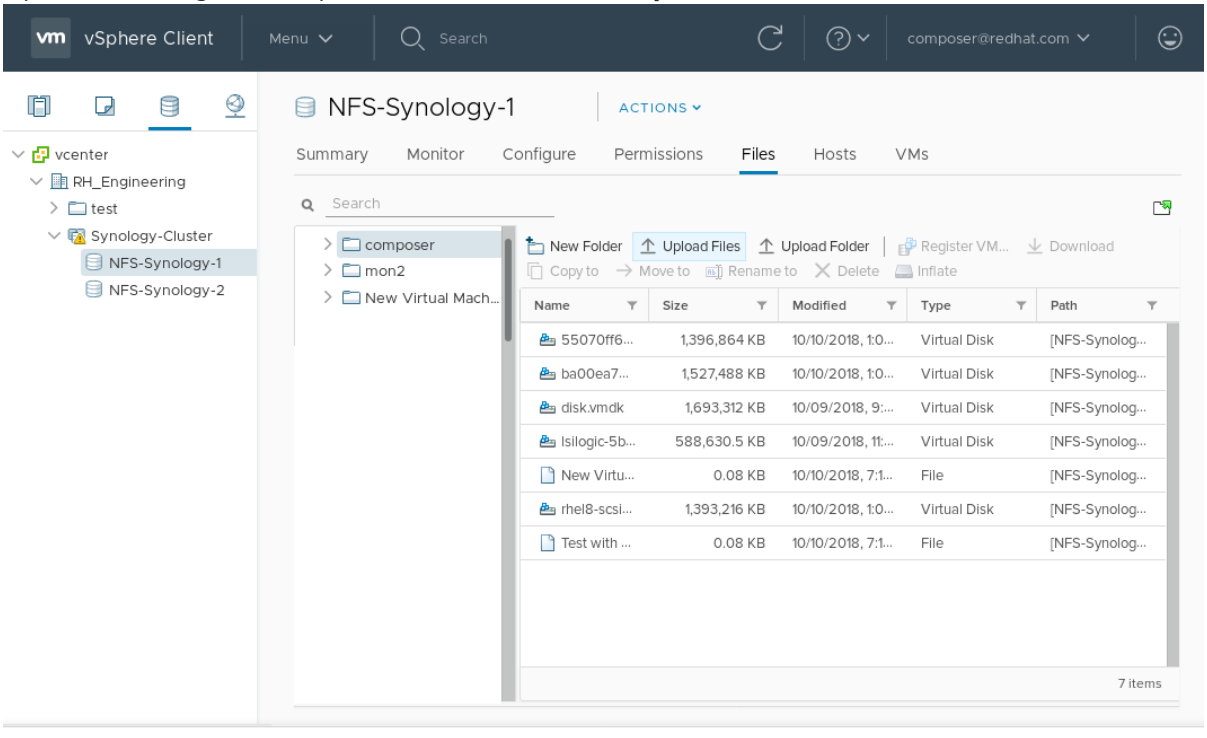
Image Builder can generate images suitable for uploading to a VMware ESXi or vSphere system. This describes steps to upload an VMDK image to VMware vSphere.

Prerequisites

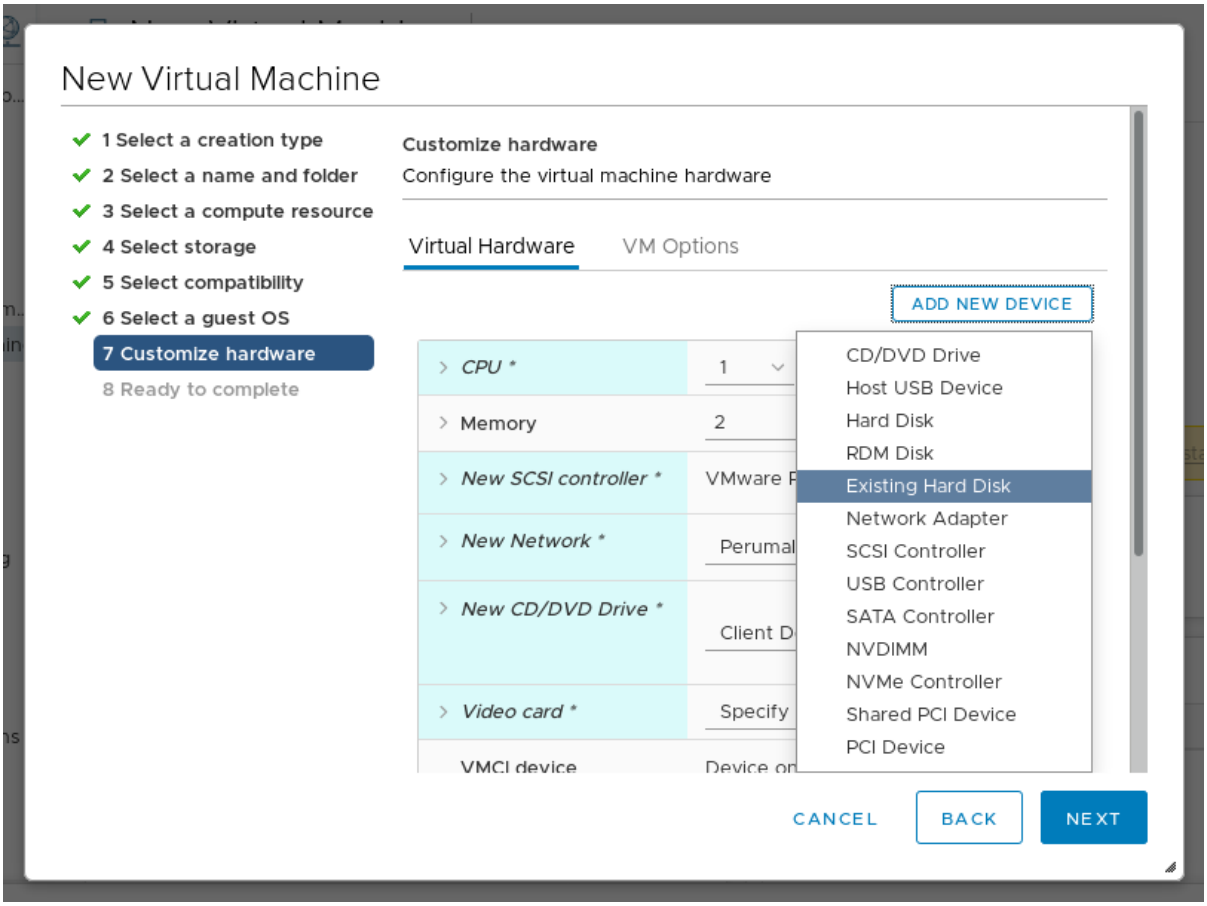
- You must have an VMDK image created by Image Builder. Use the **vmdk** output type in CLI or **VMware Virtual Machine Disk (.vmdk)**in GUI when creating the image.

Procedure

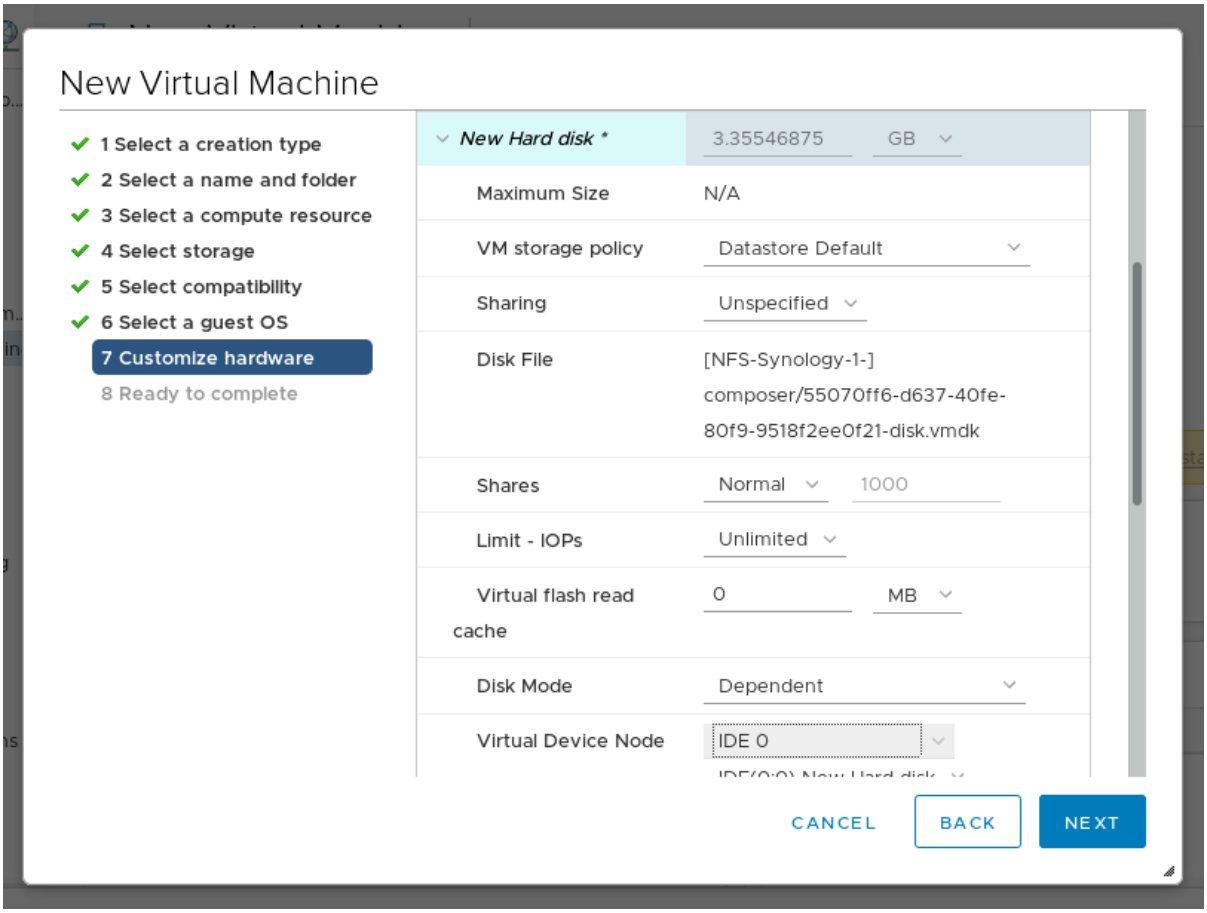
1. Upload the image into vSphere via HTTP. Click on **Upload Files** in the vCenter:



2. When you create a VM, on the **Device Configuration**, delete the default **New Hard Disk** and use the drop-down to select an **Existing Hard Disk** disk image:



3. Make sure you use an **IDE** device as the **Virtual Device Node** for the disk you create. The default value **SCSI** results in an unbootable virtual machine.



10.7. 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.

10.7.1. 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.

10.7.2. 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.

10.7.2.1. 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}"
```

10.7.2.2. 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 ways imports 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"
```

10.8. PUSHING VMWARE IMAGES TO VSPHERE

You can build VMware images and push them directly to your vSphere instance, to avoid having to download the image file and push it manually. This describes steps to push **.vmdk** images you create using Image Builder directly to vSphere instances service provider.

Prerequisites

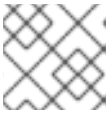
- You have **root** or **wheel** group user access to the system.
- You have opened the Image Builder interface of the RHEL web console in a browser.
- You have a vSphere Account.

Procedure

1. Click **Create blueprint**.
   See Creating an Image Builder blueprint in the web console interface .
2. Select the components and packages that you want as part of the image you are creating.
3. Click **Commit** to commit the changes you made to the blueprint.
   A pop-up on the upper right side informs you of the saving progress and then the result of the changes you commited.
4. Click **blueprint name** link on the left banner.
5. Select the **Customizations** tab to create a user account for the blueprint.
   See Creating a user account for a blueprint .

6. Select the **Images** tab .
7. Click **Create Image** to create your customized image.
   The Image type window opens.
8. 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**.
9. In the **Upload to VMware** window, under **Authentication**, enter the following details:
   a. Username: username of the vSphere account.
   b. Password: pasword of the vSphere account.
10. 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 datacenter where the image will be uploaded.
    e. **Data store**:The name of the Data store where the image will be uploaded.
    f. Click **Next**.
11. In the **Review** window, review the details about 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. Follow the steps for that:

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 Virtual Machine from the image you uploaded. For that:

a. Select the image you uploaded.

b. Click the right button on 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 a creation type: You can choose to create a New Virtual Machine.

ii. Select a name and a folder: For example, Virtual Machine name: *vSphere Virtual Machine* and location of your choice inside vSphere Client.

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 OS: For example, select *Linux* and _Red Hat Fedora (64-bit).

vii. **Customize hardware**: When you create 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.

10.9. PUSHING VHD IMAGES TO MICROSOFT AZURE CLOUD

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

Prerequisites

- You must have root access to the system.
- You have opened the Image Builder interface of the RHEL web console in a browser.

- You must have a Microsoft Storage Account created.
- You must have a writable Blob Storage prepared.

Procedure

1. Click **Create blueprint** to create a blueprint. See Creating an Image Builder blueprint in the web console interface.
2. Select the components and packages that you want as part of the image you are creating.
3. Click **Commit** to commit the changes you made to the blueprint.
   A small pop-up on the upper right side informs you of the saving progress and then the result of the changes you commited.
4. Click **blueprint name** link on the left banner.
5. Select the tab **Images**.
6. Click **Create Image** to create your customized image.
   A pop-up window opens.
   a. From the **"Type"** drop-down menu list, select the **Azure Disk Image (.vhd)** image.
   b. Check the **"Upload to Azure"** check box to upload your image to the Microsoft Azure Cloud and click **Next**.
   c. To authenticate your access to Azure, type your "Storage account" and "Storage access key" in the corresponding fields. Click **Next**.
   You can find your Microsoft Storage account details in the Settings→Access Key menu list.
   d. Type a **"Image name"** to be used for the image file that will be uploaded and the Blob "Storage container" in which the image file you want to push the image into. Click **Next**.
   e. Review the information you provided and click **Finish**.
   Optionally, you can click **Back** to modify any incorrect detail.
7. A small pop-up on the upper right side displays when the image creation process starts with the message: "Image creation has been added to the queue".
   After the image process creation is complete, click the blueprint you created an image from.
   You can see the **"Image build complete"** status for the image you created within the **Images** tab.
8. To access the image you pushed into **Microsoft Azure Cloud**, access Microsoft Azure Portal.
9. On the search bar, type **Images** and select the first entry under **Services**. You are redirected to the **Image dashboard**.
10. Click **+Add**. You are redirected to the **Create an Image** dashboard.
    Insert the below details:
    a. **Name**: Choose a name for your new image.
    b. **Resource Group**: Select a **resource group**.
    c. **Location**: Select the **location** that matches the regions assigned to your storage account. Otherwise you will not be able to select a blob.

d. **OS Type**: Set the OS type to **Linux**.

e. **VM Generation**: Keep the VM generation set on **Gen 1**.

f. **Storage Blob**: Click **Browse** on the right of **Storage blob input**. Use the dialog to find the image you uploaded earlier.
Keep the remaining fields as in the default choice.

11. Click **Create** to create the image. After the image is created, you can see the message **"Successfully created image"** in the upper right corner.

12. Click **Refresh** to see your new image and open your newly created image.

13. Click **+ Create VM**. You are redirected to the **Create a virtual machine** dashboard.

14. In the **Basic** tab, under **Project Details, your *Subscription** and the **Resource Group** are already pre-set.
If you want to create a new resource Group

a. Click **Create new**.
A pop-up prompts you to create the **Resource Group Name** container.

b. Insert a name and click **OK**.
If you want to keep the **Resource Group** that is already pre-set.

15. Under **Instance Details**, insert:

a. **Virtual machine name**

b. **Region**

c. **Image**: The image you created is pre-selected by default.

d. **Size**: Choose a VM size that better suits your needs.
Keep the remaining fields as in the default choice.

16. Under **Administrator account**, enter the below details:

a. **Username**: the name of the account administrator.

b. **SSH public key source**: from the drop-down menu, select **Generate new key pair**.
You can either use the key pair you already have or you can create a new key pair.
Alternatively, you can use **Image Builder** to add a user to the image with a preset public key. See Creating a user account with SSH key for more details.

c. **Key pair name**: insert a name for the key pair.

17. Under **Inbound port rules**, select:

a. **Public inbound ports**: **Allow selected ports**.

b. **Select inbound ports**: Use the default set **SSH (22)**.

18. Click **Review + Create**. You are redirected to the **Review + create** tab and receive a confirmation that the validation passed.

19. Review the details and click **Create**.
Optionally, you can click **Previous** to fix previous options selected.

20. A pop-up **generates new key pair** window opens. Click **Download private key and create resources**.
    Save the key file as "*yourKey*.pem".

21. After the deployment is complete, click **Go to resource**.

22. You are redirected to a new window with your VM details. Select the public IP address on the top right side of the page and copy it to your clipboard.

Now, to create an SSH connection with the VM to connect to the Virtual Machine.

1. Open a terminal.

2. At your prompt, open an SSH connection to your virtual machine. Replace the IP address with the one from your VM, and replace the path to the .pem with the path to where the key file was downloaded.

```
# ssh -i ./Downloads/yourKey.pem azureuser@10.111.12.123
```

3. You are required to confirm if you want to continue to connect. Type yes to continue.

As a result, the output image you pushed to the Microsoft Azure Storage Blob is ready to be provisioned.

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.

10.10. UPLOADING QCOW2 IMAGE TO OPENSTACK

Image Builder can generate images suitable for uploading to OpenStack cloud deployments, and starting instances there. This describes steps to upload an QCOW2 image to OpenStack.

Prerequisites

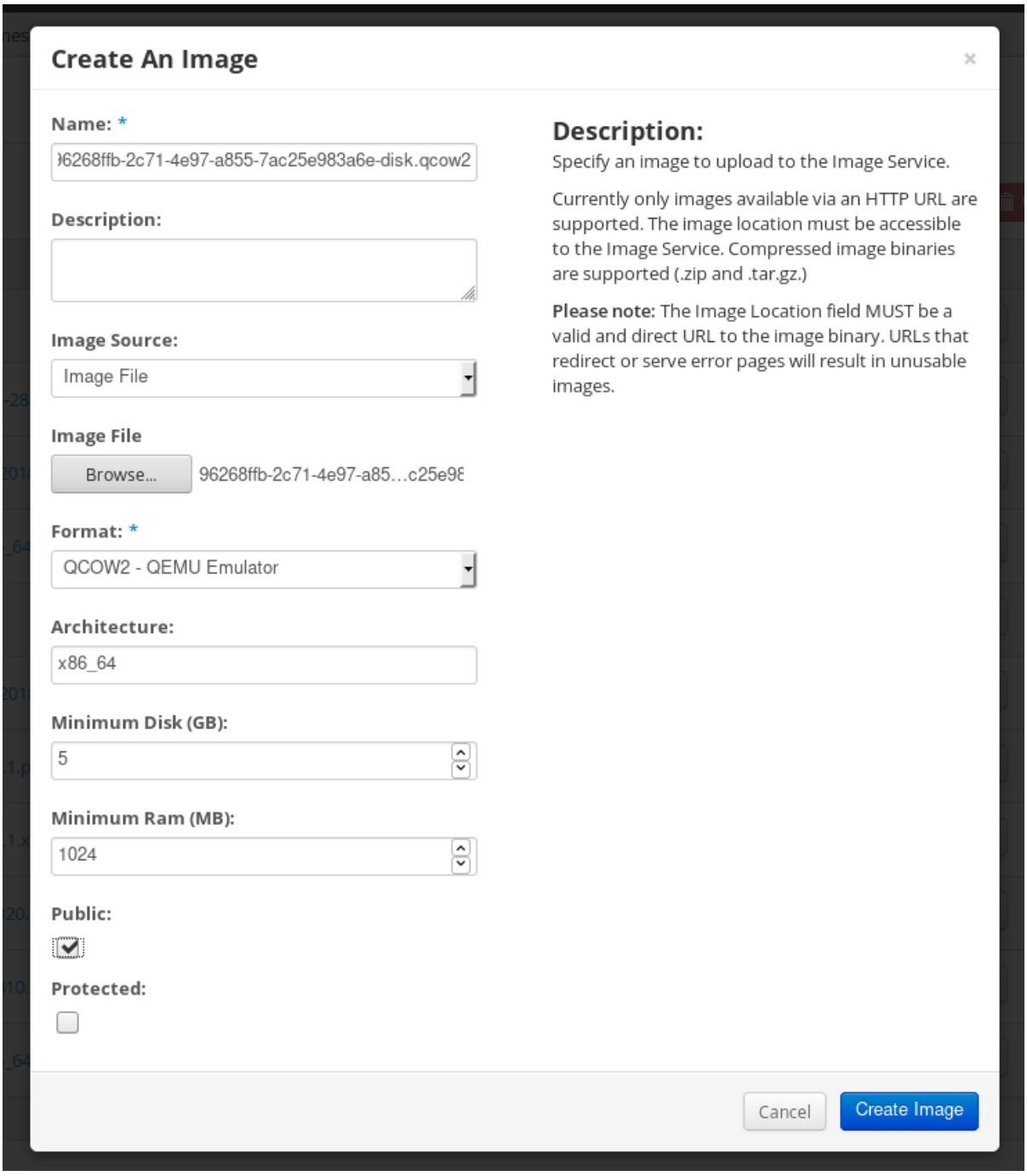
- You must have an OpenStack-specific image created by Image Builder. Use the **openstack** output type in CLI or **OpenStack Image (.qcow2)** in GUI when creating the image.

WARNING

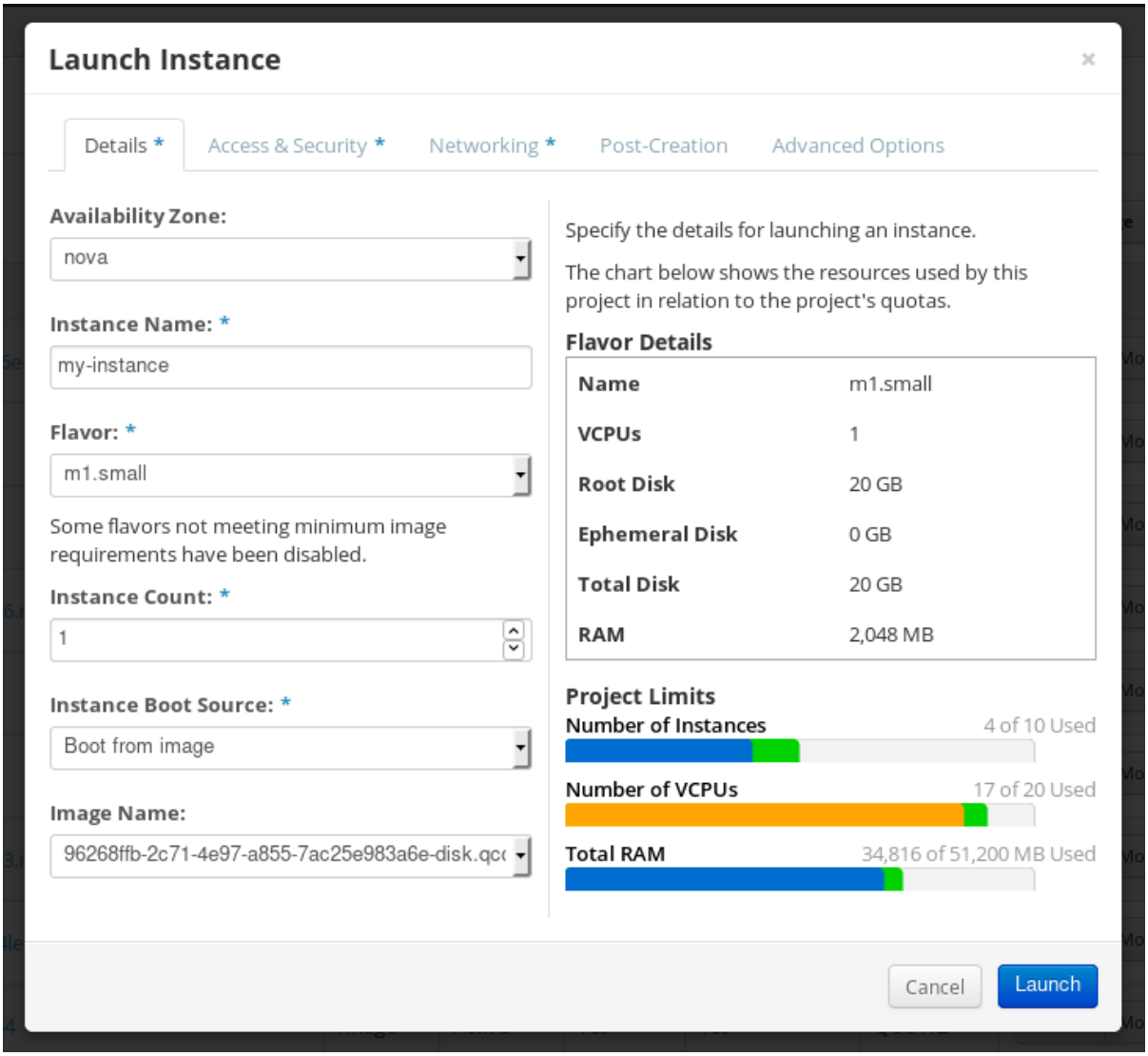
Image Builder also offers a generic QCOW2 image type output format as **qcow2** or **QEMU QCOW2 Image (.qcow2)**. Do not mistake it with the OpenStack image type which is also in the QCOW2 format, but contains further changes specific to OpenStack.

Procedure

1. Upload the image to OpenStack and start an instance from it. Use the **Images** interface to do this:



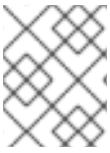
2. Start an instance with that image:



3. You can run the instance using any mechanism (CLI or OpenStack web UI) from the snapshot. Use your private key via SSH to access the resulting instance. Log in as **cloud-user**.

10.11. PREPARING FOR UPLOADING IMAGES TO ALIBABA

This section describes steps to verify custom images that you can deploy on Alibaba Cloud. The images will need a specific configuration to boot successfully, because Alibaba Cloud requests the custom images to meet certain requirements before you use it. For this, it is recommended that you use the Alibaba **image_check tool**.



NOTE

The custom image verification is an optional task. Image Builder generates images that conform to Alibaba's requirements.

Prerequisites

- You must have an Alibaba image created by Image Builder.

Procedure

1. Connect to the system containing the image you want to check it by 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.

5. If any of the **Detection Items** fail, follow the instructions to correct it. For more information, see link: Detection items section.

Additional resources

- Image Compliance Tool

10.12. UPLOADING IMAGES TO ALIBABA

This section describes how to upload an Alibaba image to Object Storage Service (OSS).

Prerequisites

- Your system is set up for uploading Alibaba images.
- You must have an Alibaba image created by Image Builder. Use the **ami** output type on RHEL 7 or Alibaba on RHEL 8 when creating the image.
- 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. On the left side Bucket menu, select the bucket to which you want to upload an image.
3. On the right upper menu, click the **Files** tab.
4. Click **Upload**. A window dialog opens on the right side. Choose the following information:
   - **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. Choose the image you want to upload.
7. Click **Open**.

As a result, the custom image is uploaded to OSS Console.

Additional resources

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

10.13. IMPORTING IMAGES TO ALIBABA

This section describes how to import an Alibaba image to Elastic Cloud Console (ECS).

Prerequisites

- You have uploaded the image to Object Storage Service (OSS).

Procedure

1. Log in to the ECS console.
    i. On the left side menu, click **Images**.
    ii. On the right upper side, click **Import Image**. A window dialog 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, select Files section and then click the **Details** link on the right for the appropriate image. A window will appear on the right side of the screen,

showing image details. The OSS object address is in the URL box.

2. Click **OK**.

NOTE

The importing process time can vary depending on the image size.

As a result, the custom image is imported to ECS Console. You can create an instance from the custom image.

Additional resources

- Notes for importing images
- Creating an instance from custom images
- Upload an object

10.14. CREATING AN INSTANCE OF A CUSTOM IMAGE USING ALIBABA

You can create instances of the custom image using Alibaba ECS Console.

Prerequisites

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

Procedure

1. Log in to the ECS console.
2. On the left side menu, choose **Instances**.
3. In the top corner, click **Create Instance**. You are redirected to a new window.
4. Fill in 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.

Additional resources

- Creating an instance by using a custom image
- Create an instance by using the wizard