Oracle Linux 9

Creating Custom Images With Image Builder

F96411-01
June 2024

Oracle Linux 9 Creating Custom Images With Image Builder,

F96411-01

Contents

B Blueprint Format

Preface

Oracle Linux 9: Creating Custom Images With Image Builder provides information about creating customized images of Oracle Linux that you can deploy on different platforms such as the cloud or bare metal systems.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also

mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Image Builder

Image builder is a tool for creating customized images of Oracle Linux that you can deploy on different platforms such as the cloud or bare metal systems. With image builder, you can generate these customized images in different formats, such as `tar` or `iso`.

Image Builder isn't automatically included in an Oracle Linux installation and requires a separate package download. After installing image builder and completing the required system configurations, you can use either the command line or the Cockpit web console to create custom Oracle Linux images.

> **Note:**
>
> For more information about installing Image Builder, see Installing Image Builder.

To use image builder, you need to understand the following concepts that are specific to the way the tool works:

- Blueprints
- Customizations
- Composer Images

Blueprints

Image builder uses blueprints to create custom images.

A blueprint is the primary reference for creating custom images with image builder. A blueprint consists of block entries that define the specifications for the images. Block entries in a blueprint contain varied information but can be divided into two kinds: entries for packages and entries for other customizations.

Each block entry has a heading. The general headings are either `[[packages]]` if the block entry contains package information, or `[[customizations]]` if the block contains customizations that are unrelated to packages.

Package information specify packages to be downloaded for the image and include the following:

- Packages: A package block entry requires the package name and version. The version format follows `dnf` version specifications. For example, the version for a major release is specified as *`n.n.n`*, such as 8.7.0. To specify the latest package version, enter an asterisk (`*`) in place of the version number. For a minor release, type *`major-nbr`*`.*`, such as 8.*.
- Package groups: A package group block entry requires only the name of the package group. The information is entered under a `[[groups]]` heading.

Definitions in a block entry follow the convention *`parameter`* `= "`*`value`*`"`. For more information about the contents of a blueprint, see Blueprint Format.

When you create images based on a blueprint, those images become associated with that blueprint in the image builder interface of the Cockpit web console.

To create or edit a blueprint, you can use any text editor. However, you must save the blueprint in Tom's Obvious Minimal Language (TOML) format.

Customizations

Customizations are blueprint entries that aren't package related.

Customizations are other image specifications that aren't part of packages. These other items might be defined users and groups, SSH keys that implement system security, and other data.

Customizations in blueprints can be varied. A customization that has a general application, such as specifying a host name for the image, are defined in a `[[customizations]]` block entry. More specific customizations would have tags appended to the heading to become better identifiers. For example, user definitions are in a `[[customizations.user]]` block, while `serviced` customizations are under the `[[customizations.services]]` heading, and so on.

Similar to package listings, customization parameters also follow the *`parameter = value`* format.

Composer Images

Composer images are images that are generated by the `composer cli-compose` command. They're the final product in image building.

The actual creation of a custom image occurs when you run the `composer-cli compose` command. Therefore, the image is called a Composer image. A Composer image is the final result of an image builder operation.

Aside from the definitions that are specified in the blueprint, a Composer image also includes logs, metadata, processes that are run to create the image, and other relevant data.

Composer images are of different types. For each type, the system automatically installs default packages. Also, the type has associated services that are enabled automatically when the image is deployed. For example, for a `tar` image, the system automatically includes the `policycoreutils` and `selinux-policy-targetd` packages. However, no extra services are enabled.

Customizations in the blueprint can specify other services that need to be enabled. However, these customizations can not override the required services for an image type that are automatically enabled when the image is deployed.

2

Preparing to Use Image Builder

To prepare for using image builder, complete the following:

- Fulfill the system requirements.
- Install the image builder component packages.
- Configure specific repositories needed by the OS image.

System Requirements

Image builder needs to run on a dedicated system with the following minimal configurations:

- 2-core processors
- 4 GiB of memory
- 20 GiB available disk space in the `/var` directory
- Access to the Internet
- Appropriate privileges for performing administrator tasks

> **Note:**
>
> A dedicated virtual machine can also serve as the environment for running image builder.

Installing Image Builder

Image builder isn't automatically included in an Oracle Linux installation. This task describes how to install and configure image builder.

1. Install the image builder packages.

   ```
   sudo dnf install -y osbuild-composer composer-cli cockpit-composer bash-completion
   ```

2. Enable the image builder service to automatically start after every system reboot.

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

3. (Optional) Enable the autocomplete feature of the `composer-cli` command by loading the configuration script.

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

About Default Image Builder Repositories

Composed images use image builder repositories to download their required packages.

Image builder doesn't use the system repositories that are defined in `/etc/yum.repos.d/` in a typical Oracle Linux installation. Instead, the repository definitions for image builder are automatically installed in `/usr/share/osbuild-composer/repositories`. In this directory, repository definitions are contained in files in JSON format, which is different from the `*.repo` files in `/etc/yum.repos.d/`.

The following extract is an example of a `*.json` repository file for the latest Oracle Linux 9 release on the x86_64 platform.

```
{
    "x86_64": [
        {
        "name": "ol9_baseos_latest",
        "baseurl": "https://yum.oracle.com/repo/OracleLinux/OL9/baseos/latest/
x86_64/",
            "gpgkey": "THE ENTIRE GPG PUBLIC KEY BLOCK",
            "check_gpg": true,
            "enabled": true
        },
        {
            ...other repositories...
        }
    ]
}
```

The repository definitions in the JSON file correspond to information in the parallel `*.repo` file in `/etc/yum.repos.d` directory. In the previous example, the JSON file is based on the `/etc/yum.repos.d/oracle-linux-ol9.repo` file.

You can override the default repositories in `/usr/share/osbuild-composer/repositories` by defining custom repositories in a different location.

Creating Custom Repositories

This task describes how to create a separate custom repository file to override the default repositories that are created when you install image builder.

1. Create directory to store the customized repositories.

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

2. Using a text editor, create a file for the version of Oracle Linux image you want to create.

   As an alternative, copy a default repository file to use as a template. For example, for an Oracle Linux 9 image, you would copy the Oracle Linux 9 json file from the default location.

   ```
   sudo cp /usr/share/osbuild-composer/repositories/OL9.json /etc/osbuild-
   composer/repositories/
   ```

3. Specify the following information, as they're available, on the file:
   - System architecture of the OS
   - `name`: name of the repository
   - `metalink`
   - `baseurl`: the yum URL of the repository
   - `mirrorlist`
   - `gpgkey`: package GPG key block
   - `check_gpg`: must always be set to true

   The following is an example of a JSON file for an Oracle Linux 9 image:

```
{
    "x86_64": [
        {
        "name": "ol9_baseos_latest",
            "metalink": "",
            "baseurl": "https://yum.oracle.com/repo/OracleLinux/OL9/baseos/
latest/x86_64/",
            "mirrorlist": "",
            "gpgkey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\n (…)",
            "check_gpg": true,
            "enabled": true,
            "metadata_expire": ""
        }
    ]
}
```

4. (Optional) Verify that the URLs in the file are correct by comparing them to the corresponding repository file in `/etc/yum.repos.d`.

   For the current example, you would use an Oracle Linux repository file for verification.

   ```
   cat /etc/yum.repos.d/oracle-linux-ol9.repo
   ```

5. Restart `osbuild-composer.service`.

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

3
Deploying Custom Image Builder Images

Deploying a custom image applies the following work flow:

1. Create a blueprint, or edit an existing blueprint.
2. Import the blueprint.
3. Create the image based on the blueprint.
4. Download the resulting image.
5. Install the software according to the image speifications.

Preparing the Blueprint

The process for deploying a custom image begins with the blueprint preparation. The following task uses command lines to create or edit a blueprint. To use the web console interface instead, see Oracle Linux: Using the Cockpit Web Console.

1. Prepare the blueprint, which can either be a newly created blueprint or a an existing one.
   - **Create a blueprint**

   a. Use any text editor to create a text file.

   b. Enter blueprint specifications based on the packages and customizations that you want to be associated with the image.

      Ensure that you provide the basic metadata information about the blueprint. For reference, see Blueprint Format.

   c. Save the file as a `toml` file, for example, `myblueprint.toml`.

   d. Push or import the blueprint into image builder.

      ```
      sudo composer-cli blueprints push blueprint
      ```

   - **Edit an existing blueprint**

   a. (Optional) List available blueprints.

      ```
      sudo composer-cli blueprints list
      ```

   b. Save or export the blueprint you want to edit.

      ```
      sudo composer-cli save blueprint
      ```

   c. Use a text editor to edit the blueprint by revising package and customization entries as required.

   d. Remove the line `packages = []` if it exists in the blueprint.

   e. Update the version by incrementing the number as appropriate.

      Ensure that the version follows the scheme in https://semver.org/.

   f. Save the changes.

   g. Push or import the blueprint into image builder.

```
sudo composer-cli blueprints push blueprint
```

2. (Optional) Display the blueprint configuration.

```
sudo composer-cli blueprints show blueprint
```

3. Verify that the blueprint's components and versions and corresponding dependencies are valid.

```
sudo composer-cli blueprints depsolve blueprint.toml
```

If image builder is unable to validate the dependencies, remove the `osbuild-composer` cache.

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

Creating the Image in Image Builder

You must have an existing blueprint to use for creating an image.

1. Create the image with the blueprint specifications.

```
sudo composer-cli compose start blueprint image-type
```

For a list of valid image types, see Image Types and Output Formats. Alternatively, type:

```
sudo composer-cli compose types
```

While the process is running in the background, the composer image's UUID is displayed.

Use the UUID to track the progress of the image building process with the following command:

```
sudo composer-cli compose info image-uuid
```

2. After the process ends, check the status of the image.

```
sudo composer-cli status image-uuid
```

The image's status should indicate `FINISHED`.

3. Download the image file.

```
sudo composer-cli compose image image-uuid
```

To download the image's metadata and logs, type:

```
sudo composer-cli compose [metadata|logs] image-uuid
```

4
Use Cases in Deploying Image Builder Images

This chapter shows cases where image builder is used to create and deploy images for specific setup and configurations.

Creating ISO Images for Deployment

You must have a valid blueprint with the specifications you require for the image. This blueprint must be pushed or imported to Image Builder. To fulfill these requirements, see Preparing the Blueprint.

Perform this procedure to create an ISO which installs the OS on a bare metal system. At the end of the procedure, an `.iso` file is created that contains the following:

- Standard Anaconda installer ISO
- Embedded Oracle Linux system tar file
- Kickstart file that installs the commit with the minimum default requirements

The installer ISO contains a preconfigured system image that you can use to install on a bare metal system.

1. (Optional) Verify that the blueprint for the ISO image is in Image Builder.

   ```
   sudo composer-cli blueprints show blueprint
   ```

2. Create the ISO image.

   ```
   sudo composer-cli compose start blueprint iso
   ```

   While the process is running in the background, the composer image's UUID is displayed.

3. After the process ends, check the status of the image.

   ```
   sudo composer-cli status image-uuid
   ```

   The image's status should indicate `FINISHED`.

4. Download the ISO image file.

   ```
   sudo composer-cli compose image image-uuid
   ```

   The ISO image file contains a `*.tar` file which is the OS image to be installed on a system.

5. (Optional) Mount the downloaded image and extract the contents.

   a. Mount the downloaded image.

      ```
      sudo mount -o ro iso-image /mnt
      ```

iso-image includes the full path and the name of the ISO image file.

The `/mnt/` mount point contains the `liveimag.tar.gz` file.

b. Extract the contents of the `*.tar` file

```
tar xvf /mnt/liveimg.tar.gz
```

6. Select the appropriate method for installing the OS.

For example, you can use the ISO image as an installer when booting a system from a location where you want to automatically load the image to a hard disk. Otherwise, you can extract the image file, as described, and use this file to manually deploy the image to a target environment (such as a cloud environment, virtual machine, and so on).

For more information about installing Oracle Linux, see Oracle Linux 9: Installing Oracle Linux.

A

Image Types and Output Formats

Image builder can generate different types of image that can be deployed on specific platforms.

Image Description	Output File Extension
Oracle Linux optical disc image	`.iso`
Oracle Cloud Infrastructure images	`.qcow2`
TAR Archive	`.tar`
QEMU QCOW2 image	`.qcow2`
Azure Disk Image	`.vhd`
Amazon Machine Image Disk	`.raw`

To list the types of image that you can build, run the following command:

```
sudo composer-cli compose types
```

B
Blueprint Format

Blueprint content consists of basic metadata information, package information, and customizations.

A blueprint contains the specifications for an image builder custom image. The general types of information that you can add to a blueprint are package related information and other customizations. Elements in a typical blueprint file include: 1) basic metadata information, 2) package listing, and 3) other customizations. All configuration entries in a blueprint use the *parameter* = "*value*" format convention.

Basic Metadata Information

The blueprint's metadata provides general information about the blueprint itself. The metadata is entered at the top of the file and consists of the blueprint name, an optional description, and a version. The version follows the semantic versioning scheme in https://semver.org/. You define the metadata only one time in the entire blueprint.

> **Important:**
>
> The metadata information is required in any blueprint.

```
name = "Blueprint name"
description = "Description of the blueprint"
version = "Version number"
```

Package Information

Package information is a general term that includes list of packages, modules, containers, and groups., each Each entry has its corresponding heading in the format `[[heading]]`.

The parameters under each heading depend on what the heading describes. For example package and module lists require the name and version of the package. Containers and groups require different information and would use other parameters. Each package, module, container, or group listing must have its own heading, as shown in the following example:

```
[[packages]]
name = "tmux"
version = "2.7"

[[packages]]
name = "python3"
version = "3.6.8"

[[groups]]
name = "graphical-admin-tools"
```

Customizations

Customizations are blueprint specifications in addition to package lists. While a customization would typically appear under the heading `[[customization]]`, more specific customizations

append a keyword to the heading for better identification. Customizations typically use multiple parameters than package information entries.

The following example shows customizations for the image's hostname, locale, time zone, and groups.

> **Note:**
>
> A custom group refers to a UNIX group, which is different from a group of packages to be downloaded. Therefore, a custom UNIX group definition has its own blueprint heading that's distinct from a package group heading.

```
[[customizations]]
hostname = "system1"

[[customizations.locale]]
languages = ["en_GB.utf8"]
keyboard = "gb"

[[customizations.timezone]]
timezone = "Europe/Dublin"
ntpservers = ["uk.pool.ntp.org"]

[[customizations.groups]]
name = "students"
```

Other customizations that you can define include the following:

- Users: Contains all the required details that apply to the specific user account, such as the user's name, home directory, the user's default shell, groups to which the user is assigned, and so on.
- SSH key: Contains the particular user's name and the public SSH key that you create for the user. This key is added to the user's `authorized_keys` file.
- Kernel: Contains arguments to append to the bootloader's command line.
- Firewall ports: Contain the list of ports that you want to open. The ports are specified by using the *`port`*`:`*`protocol`* format, for example, `22:tcp`.
- Firewall services: A separate listing that contains a list of services that you enable and disable for the image. To identify which services you can enable or disable, run the `firewall-cmd --get-services` command.
- `Systemd` services: Similar to firewall services, this entry contains a list of `systemd` services that you enable and disable for an image.

The preceding list is partial. For a complete list of blueprint entries, see https://osbuild.org/docs/user-guide/blueprint-reference/.