Oracle Linux 10

Creating Custom Images With Image Builder

G14601-01
June 2025

Oracle Linux 10 Creating Custom Images With Image Builder,

G14601-01

Contents

B Blueprint Format

Preface

Oracle Linux 10: 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 custom images of Oracle Linux for deployment on different platforms, such as in the cloud or on bare metal systems. Images can be created based on the host Oracle Linux version, or on different versions of Oracle Linux provided in the Image Builder repository. Image Builder can generate these images in several 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:

- Blueprints
- Customizations
- Composer Images

Blueprints

Image Builder uses blueprints to create custom images.

A blueprint provides the specification for a custom image that Image Builder uses to create the image. A typical blueprint specification contains general metadata, a list of packages to install, and other customizations to apply.

Blueprints are defined in Tom's Obvious Minimal Language (TOML) format and can be created or edited in any text editor.

Blueprint definitions follow the convention `parameter = "value"`, with package and customization definitions being grouped under `[[packages]]` or `[[customizations]]`, as appropriate.

For more information about the contents of a blueprint, see Blueprint Format.

When you have created a blueprint file, you must register the blueprint with Image Builder before you can use that blueprint to create an image. See Creating a Blueprint.

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.

Customizations

Customizations are blueprint entries that aren't package-related.

Customizations are other image specifications that aren't related to the installation of packages. These include predefined users and groups, SSH keys that implement system security, and other requirements.

Customizations in blueprints are denoted depending on their scope. A customization that has a general application, such as specifying a host name for the image, is defined in a `[customizations]` block entry. More specific customizations append tags to the heading to better identify the customization. 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.

For more information about specifying customizations in a blueprint, see Blueprint Format.

Composer Images

Composer images are images that are generated by the `composer-cli compose` command.

The actual creation of a custom image occurs when you run the `composer-cli compose` command against a blueprint. Therefore, the image is called a composer image. A composer image is the final result of the image building process.

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

Composer images can be produced in different output formats. For example: `oci`, `qcow`, and `tar`. 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't override the required services for an image type that are automatically enabled when the image is deployed.

For more information about the different image types, see: Image Types and Output Formats.

2

Preparing to Use Image Builder

Before 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 requires a dedicated system with the following minimum configuration:

- 2-core processor
- 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

Describes how to install 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.

   Enter the following command to install the required packages:

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

   > **Note:**
   >
   > DNF uses `composer-cli` in the preceding command as a virtual provider to install `weldr-client`, which is the actual package that contains the `composer-cli` application.

2. Enable the Image Builder service.

   Ensure that the Image Builder service starts automatically when the system boots:

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

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

3. (Optional) Enable autocompletion for the `composer-cli` command.

   Load the configuration script:

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

4. (Optional) Add the user to the `weldr` group.

   To run `composer-cli` commands without root privileges, add the user to the `weldr` group:

   ```
   sudo usermod -a -G weldr username
   newgrp weldr
   ```

About Default Image Builder Repositories

Composed images use Image Builder repositories to download their required packages.

The Image Builder repository contains definitions for building images based on different major versions of Oracle Linux. You can therefore use Image Builder to build an image for a version of Oracle Linux that's different to the version running on the Image Builder host.

Note:

The image that Image Builder creates for a release of Oracle Linux includes all the latest packages for that release. You can't build an image for a specific update level.

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/`.

Each file corresponds to the latest update of a major version of Oracle Linux. For example, the following extract is from the `ol-10.json` repository file for the latest Oracle Linux 10 release on the x86_64 platform.

```
{
    "x86_64": [
        {
            "name": "baseos",
            "baseurl": "https://yum$ociregion.$ocidomain/repo/OracleLinux/
OL10/baseos/latest/x86_64",
            "check_gpg": false,
            "rhsm": false
        },
```

```
        {
            ...other repositories...
        }
    ]
}
```

The repository definitions in the JSON file correspond to information in the parallel `*.repo` file in the `/etc/yum.repos.d` directory. In the previous example, the JSON file is based on the `/etc/yum.repos.d/oracle-linux-ol10.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

Learn how to create a custom repository.

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 a 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.

   Or, you can copy a default repository file to use as a template. For example, for an Oracle Linux 10 image, you would copy the Oracle Linux 10 JSON file from the default location.

   ```
   sudo cp /usr/share/osbuild-composer/repositories/ol-10.json /etc/osbuild-
   composer/repositories/
   ```

3. Following the repository JSON specification, include the following information for each repository in the file that you created:

   - System architecture of the OS.
   - `name`: name of the repository.
   - `baseurl`: the URL of the yum repository.
   - `check_gpg`: must always be set to false.
   - `rhsm`: must always be set to false.

   The following is an example of a JSON file for an Oracle Linux 10 image:

   ```
   {
       "x86_64": [
           {
               "name": "baseos",
               "baseurl": "https://yum$ociregion.$ocidomain/repo/
   OracleLinux/OL10/baseos/latest/x86_64",
               "check_gpg": false,
               "rhsm": false
           },
       ]
   }
   ```

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 10 repository file for verification.

   ```
   cat /etc/yum.repos.d/oracle-linux-ol10.repo
   ```

5. Restart `osbuild-composer.service`.

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

Using Third-Party Custom Repositories

Use custom repositories from third-party sources.

You can use `composer-cli` to load custom repositories created by a third-party.

1. Create the repository source file.

   Create a TOML file that references the URL of the custom repository. For example:

   ```
   id = "custom_repo"
   name = "Custom Repository"
   type = "yum-baseurl"
   url = "https://yum.example.com/repos/packages"
   check_gpg = false
   check_ssl = false
   ```

   The value of the `type` field must be one of `yum-baseurl`, `yum-mirrorlist`, or `yum-metalink`.

2. Add the third-party source repository to Image Builder.

   Run the following command to add the repository source file:

   ```
   sudo composer-cli sources add filename.toml
   ```

3. Check that the new repository is available to Image Builder.

   Run the following command to list the available repositories:

   ```
   sudo composer-cli sources list
   ```

3

Deploying Custom Image Builder Images

Deploying a custom image requires the following steps:

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 specifications.

Creating a Blueprint

Learn how to create a blueprint.

The following procedure describes how to create a blueprint configuration using the CLI. Or, you can create a blueprint using the Cockpit web console interface, see Oracle Linux: Using the Cockpit Web Console for more details.

1. Create a blueprint configuration as follows:
   a. Using any text editor, create a text file.
   b. In the text file, provide the blueprint specifications for the base distro, the packages you want installed, and any other image customizations as required.

      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, `myblueprintfile.toml`.
   d. Push or import the blueprint into Image Builder, for example:

      ```
      sudo composer-cli blueprints push myblueprintfile.toml
      ```
2. (Optional) View the blueprint configuration, using the name you specified in the blueprint metadata *name* entry:

   ```
   sudo composer-cli blueprints show myblueprint
   ```
3. Verify that the blueprint's components and versions and corresponding dependencies are valid.

   ```
   sudo composer-cli blueprints depsolve myblueprint
   ```

If Image Builder is unable to validate the dependencies, delete the `osbuild-composer` cache.

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

```
sudo systemctl restart osbuild-composer
```

Editing a Blueprint

Learn how to edit an existing blueprint.

The following procedure describes how to edit an existing blueprint configuration using the CLI. Or, you can edit a blueprint using the Cockpit web console interface, see Oracle Linux: Using the Cockpit Web Console for more details.

1. Edit an existing blueprint as follows:
   a. (Optional) List the existing blueprints.

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

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

      ```
      sudo composer-cli blueprints save myblueprint
      ```

      The blueprint specification file is saved as the *name* of the blueprint defined in the metadata, with the `.toml` extension, in the current directory.
   c. Using a text editor, edit the blueprint specification file 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 specification file into Image Builder.

      ```
      sudo composer-cli blueprints push myblueprintfile.toml
      ```

2. (Optional) View the revised blueprint configuration.

   ```
   sudo composer-cli blueprints show myblueprint
   ```

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

   ```
   sudo composer-cli blueprints depsolve myblueprint
   ```

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

Learn how to create an image based on a blueprint.

You must have an existing blueprint to use to create an image.

The following task describes how to create a custom image based on a specified blueprint and how to download the resulting composer image.

1. Create the image with the blueprint specifications.

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

   For a list of valid image types, see Image Types and Output Formats. Or, enter the following command:

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

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

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

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

   The image's status indicates `FINISHED` when Image Builder has completed the build process.

   Tip:

   The output of the `sudo composer-cli compose info` command is verbose. Pipe the output of the command to `head` to see only the current status:

   ```
   sudo composer-cli compose info image-uuid | head -1
   ```

3. Download the image file.

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

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

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

4

Use Cases in Deploying Image Builder Images

Examples of using Image Builder to create and deploy different image types.

This chapter shows cases where Image Builder is used to create and deploy images for specific setups and configurations.

Creating ISO Images for Deployment

Learn how to use the Image Builder CLI to create an ISO image.

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 Creating a Blueprint or Editing a Blueprint.

Perform this procedure to create an ISO image 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 myblueprint
   ```

2. Create the ISO image.

   ```
   sudo composer-cli compose start myblueprint image-installer
   ```

   While the process is running in the background, the composer image's UUID is displayed.
3. Check the status of the image.

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

   Repeat this check until the status indicates `FINISHED` when Image Builder has completed the build process.
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 10, see Oracle Linux 10: 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	**Command Image Type**	**Output File Extension**
Oracle Linux optical disc image	`image-installer`	`.iso`
Oracle Cloud Infrastructure images	`oci`	`.qcow2`
TAR Archive	`tar`	`.tar`
QEMU QCOW2 image	`qcow`	`.qcow2`
Azure Disk Image	`vhd`	`.vhd`
Amazon Machine Image Disk	`ami`	`.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 provides the specification for an Image Builder custom image. Elements in a typical blueprint file include: 1) basic metadata, 2) package information, and 3) customizations. All configuration entries in a blueprint are defined using the `*parameter* = "*value*"` format.

Metadata and package information are defined at the root of the blueprint.

Basic Metadata

The blueprint's metadata provides general information about the blueprint itself. The metadata is entered at the top of the file and consists of:

- *name*: The name of the blueprint (required).
- *description*: A description for the blueprint (optional).
- *version*: A version number for the blueprint (required). The version follows the semantic versioning scheme in https://semver.org/.
- *distro*: The required distro in the Image Builder repository, which the image is based on (optional). If the *distro* value is empty, the image is based on the host image distro.

```
name = "myblueprint"
        description = "My Blueprint"
        version = "0.0.2"
        distro = "ol-10"
```

Package Information

Package information is a general term that identifies a list of packages, modules, containers, and groups. Each entry has a 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. 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-number*.*`, such as 8.*.

Containers and groups require different information and use different parameters. Each package, module, container, or group listing must have its own heading, as shown in the following example:

```
[[packages]]
name = "tmux"
version = "3.5"

[[packages]]
name = "python3"
version = "3.9.18"
```

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

- Packages: Are defined in a `[[packages]]` block. Each entry requires the package name and version.
- Package groups: Are defined in a `[[groups]]` block. Each entry requires only the name of the package group.

> **Note:**
>
> A package group is a set of related packages defined as such in a repository. Therefore, a package group has its own blueprint heading `[[groups]]`, to distinguish it from an individual package.

Customizations

Customizations include any other specifications for the image that aren't associated with packages. For example: users, groups, and keys.

Customizations are defined under the heading `[customizations]`, with more specific customizations appending a keyword to the heading for better identification, for example `[[customizations.locale]]`. Customizations typically include more parameters than package information entries.

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

```
[customizations]
hostname = "system1"

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

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

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

Other custom entries 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/.