kolla Documentation

Release 21.1.0.dev52

OpenStack Foundation

Feb 11, 2026

CONTENTS

Kollas mission is to provide production-ready containers and deployment tools for operating OpenStack clouds.

CHAPTER

ONE

RELATED PROJECTS

This documentation is for the Kolla container images.

Kolla-ansible is a subproject of Kolla that deploys the Kolla container images using Ansible.

Kayobe is a subproject of Kolla that uses Kolla Ansible and Bifrost to deploy an OpenStack control plane to bare metal.

ansible-collection-kolla is a subproject of Kolla that provides common Ansible content for use by Kolla projects.

CHAPTER TWO

SITE NOTES

This documentation is continually updated and may not represent the state of the project at any specific prior release. To access documentation for a previous release of kolla, choose one of the OpenStack release names on the right of the title.

CHAPTER THREE

RELEASE NOTES

The release notes for the project can be found here: https://docs.openstack.org/releasenotes/kolla/

CHAPTER

FOUR

ADMINISTRATOR GUIDE

4.1 Administrator Guide

4.1.1 Building Container Images

If you are a system administrator running Kolla, this section contains information that should help you understand how to build container image or build some images using `--template-override`.

Building Container Images

Firstly, ensure `kolla` and the container engine of your choice is installed.

Currently supported container engines are `docker` and `podman`.

```
python3 -m pip install kolla
#only one of these is needed:
python3 -m pip install podman
python3 -m pip install docker
```

Then, the **`kolla-build`** command is available for building Docker images.

Building kolla images

In general, images are built like this:

```
kolla-build
```

By default, the above command would build all images based on a CentOS Stream image.

The operator can change the base distro with the `-b` option:

```
kolla-build -b ubuntu
```

There are following distros (bases) available for building images:

- centos
- debian
- ubuntu

See the *support matrix* for information on supported base image distribution versions and supported images on each distribution.

It is possible to build only a subset of images by specifying them on the command line:

```
kolla-build keystone
```

In this case, the build script builds all images whose name contains the `keystone` string, along with their parents.

Multiple names may be specified on the command line:

```
kolla-build keystone nova
```

Each string is actually a regular expression so one can do:

```
kolla-build ^nova-
```

`kolla-build` can be configured via an INI file, canonically named `kolla-build.conf` and placed in `/etc/kolla`. A custom path to it can be set via the `--config-file` argument. Most CLI arguments can be set via this config file. Remember to convert the names from hyphenated to underscored. Run `kolla-build --help` to see all available options.

The set of images to build can be defined as a profile in the `profiles` section of `kolla-build.conf`. Then, profile can be specified by `--profile` CLI argument or `profile` option in `kolla-build.conf`.

For example, since Magnum requires Heat, one could add the following profile to `profiles` section in `kolla-build.conf`:

```
[profiles]
magnum = magnum,heat
```

These images could then be built using command line:

```
kolla-build --profile magnum
```

Or putting the following line in the `DEFAULT` section in `kolla-build.conf` file:

```
[DEFAULT]
profile = magnum
```

The **kolla-build** uses `kolla` as default Docker namespace. This is controlled with the `-n` command line option. To push images to a Dockerhub repository named `mykollarepo`:

```
kolla-build -n mykollarepo --push
```

To push images to a local registry, use the `--registry` flag:

```
kolla-build --registry 172.22.2.81:4000 --push
```

Build OpenStack from source

The locations of OpenStack source code are written in `kolla-build.conf`. The sources `type` supports `url`, `git` and `local`. The `location` of the `local` source type can point to either a directory containing the source code or to a tarball of the source. The `local` source type permits to make the best use of the Docker cache. A source may be disabled by setting `enabled` to `False`.

The `kolla-build.conf` file could look like this:

```
[horizon]
type = url
location = https://tarballs.openstack.org/horizon/horizon-master.tar.gz

[keystone-base]
type = git
location = https://opendev.org/openstack/keystone
reference = stable/mitaka

[heat-base]
type = local
location = /home/kolla/src/heat

[ironic-base]
type = local
location = /tmp/ironic.tar.gz
enabled = False
```

Note

Note that the name of the section should exactly match the image name you are trying to change source location for.

If using the `local` source type, the `--locals-base` flag can be used to define a path prefix, which you can reference in the config.

```
[DEFAULTS]
locals_base = /home/kolla/src

[heat-base]
type = local
location = $locals_base/heat
```

Dockerfile customisation

The `kolla-build` tool provides a Jinja2-based mechanism which allows operators to customise the Dockerfiles used to generate Kolla images.

This offers a lot of flexibility on how images are built, for example: installing extra packages as part of the build, tweaking settings or installing plugins. Examples of these are described in more detail below.

Note

The Docker file Jinja2 template for each image is found in subdirectories of the `docker` directory included in the `kolla` package.

Using a different base image

Base image can be specified using `--base-image`:

```
kolla-build --base-image <image-identifier>
```

The `image-identifier` accepts any format that Docker accepts when referencing an image.

Generic customisation

Kolla templates are designed such that each Docker file has logical sections represented by Jinja2s named `block` section directives. These can be overridden at will by Kolla users. The following is an example of how an operator would modify the setup steps within the Horizon Dockerfile.

First, create a file to contain the customisations, for example: `template-overrides.j2`. Fill it with the following contents:

```
{% extends parent_template %}

# Horizon
{% block horizon_ubuntu_source_setup %}
RUN useradd --user-group myuser
{% endblock %}
```

Then rebuild the `horizon` image, passing the `--template-override` argument:

```
kolla-build --template-override template-overrides.j2 ^horizon$
```

Note

The above example will replace all contents of the original block. Hence, one may want to copy the original contents of the block before and modify it. Do note it makes the customisations ignore changes in Kolla upstream.

We recommend users use more specific customisation functionalities, such as removing/appending entries for packages. These other customisations are described in the following sections.

Two block series are of particular interest and are safe to override as they are empty by design. The top of each Dockerfile includes `<image_name>_header` block which can be used for early customisations, such as RHN registration described later. The bottom of each Dockerfile includes `<image_name>_footer` block which is intended for image-specific modifications. Do note to use the underscored name of the image, i.e., replace dashes with underscores. All leaf Dockerfiles, i.e. those meant for direct consumption, additionally have a `footer` block which is then guaranteed to exist once at the very end of the image recipe chain.

Packages customisation

Packages installed as part of an image build can be overridden, appended to, and deleted. Taking the Horizon example, the following packages are installed as part of a package install (among others):

- `gettext`
- `locales`

To add a package to this list, say, `iproute`, first create a file, for example, `template-overrides.j2`. In it place the following:

```
{% extends parent_template %}

# Horizon
{% set horizon_packages_append = ['iproute'] %}
```

Then rebuild the `horizon` image, passing the `--template-override` argument:

```
kolla-build --template-override template-overrides.j2 ^horizon$
```

Alternatively `template_override` can be set in `kolla-build.conf`.

The `append` suffix in the above example carries special significance. It indicates the operation taken on the package list. The following is a complete list of operations available:

override
: Replace the default packages with a custom list.

append
: Add a package to the default list.

remove
: Remove a package from the default list.

To remove a package from that list, say `locales`, one would do:

```
{% extends parent_template %}

# Horizon
{% set horizon_packages_remove = ['locales'] %}
```

An example of this is the Grafana plugins, which are mentioned in the next section.

Grafana plugins

Additional Grafana plugins can be installed by adding the plugin name to the `grafana_plugins_append` list. Plugins can also be removed by adding the plugin name to the `grafana_plugins_remove` list. Additionally the entire list can be overridden by setting the `grafana_plugins_override` variable.

```
grafana_plugins_append:
  - grafana-piechart-panel
  - vonage-status-panel
```

Patching customization

Kolla provides functionality to apply patches to Docker images during the build process. This allows users to modify existing files or add new ones as part of the image creation.

You need to define a `patches_path` in the `[DEFAULT]` section of the `/etc/kolla/kolla-build.conf` file. This directory will be used to store patches for the images.

```
[DEFAULT]
patches_path = /path/to/your/patches
```

Create a directory for each image you want to patch, following a directory structure similar to the Debian patch quilt format. Refer to quilt documentation. for more details.

- `<patches_path>/image_name/` : The directory for the specific image.
- `<patches_path>/image_name/some-patch` : Contains the patch content.
- `<patches_path>/image_name/another-patch` : Contains the patch content.
- `<patches_path>/image_name/series` : Lists the order in which the patches will be applied.

For example, if you want to patch the `nova-api` image, the structure would look like this:

```
/path/to/your/patches/nova-api/some-patch
/path/to/your/patches/nova-api/another-patch
/path/to/your/patches/nova-api/series
```

The `series` file should list the patches in the order they should be applied:

```
some-patch
another-patch
```

When the images are built using `kolla-build`, the patches defined in the `patches_path` will automatically be applied to the corresponding images.

After the patches are applied, Kolla stores information about the applied patches in `/etc/kolla/patched`. The patch files themselves are stored in the `/patches` directory within the image. This allows you to track which patches have been applied to each image for debugging or verification purposes.

Python packages build options

The block `base_pip_conf` in the `base` Dockerfile can be used to provide the PyPI build customisation options via the standard environment variables like `PIP_INDEX_URL`, `PIP_TRUSTED_HOST`, etc.

To override PYPI upper-constraints of all OpenStack images, you can define the source location of openstack-base. in `kolla-build.conf`.

Upstream repository of openstack-base (requirements) has a source of upper constraints file.

Make a fork or clone the repository then customise `upper-constraints.txt` and define the location of `openstack-base` in `kolla_build.conf`.

```
# These examples use upstream openstack-base as a demonstration
# To use custom openstack-base, make changes accordingly

# Using git source
[openstack-base]
type = git
location = https://opendev.org/openstack/requirements
reference = master

# Using URL source
```

(continues on next page)

(continued from previous page)

```
[openstack-base]
type = url
location = https://tarballs.opendev.org/openstack/requirements/requirements-
↪master.tar.gz

# Using local source
[openstack-base]
type = local
location = /home/kolla/src/requirements
```

To remove or change the version of specific Python packages in `openstack-base` upper-constraints, you can use the block `openstack_base_override_upper_constraints` in your template file, for example, `template-overrides.j2`:

```
{% block openstack_base_override_upper_constraints %}
RUN {{ macros.upper_constraints_version_change("sqlparse", "0.4.4", "0.5.0") }
↪}
RUN {{ macros.upper_constraints_remove("reno") }}
{% endblock %}
```

`kolla-toolbox` image needs different approach as it does not uses `openstack-base` as a base image. A variable `UPPER_CONSTRAINTS_FILE` is set in the Dockerfile of `kolla-toolbox`. To change variable, add the following contents to the `kolla_toolbox_pip_conf` block in your template file, for example, `template-overrides.j2`:

```
{% block kolla_toolbox_pip_conf %}
ENV UPPER_CONSTRAINTS_FILE=https://releases.openstack.org/constraints/upper/
↪master
{% endblock %}
```

Note

`UPPER_CONSTRAINTS_FILE` must be a valid URL to the file

Plugin functionality

The Dockerfile customisation mechanism is useful for adding/installing plugins to services. An example of this is Neutrons third party L2 drivers.

For example, to add the `networking-cisco` plugin to the `neutron_server` image, one may be tempted to add the following to the `template-override` file:

Warning

Do NOT do the below. Read on for why.

```
{% extends parent_template %}
```

(continues on next page)

(continued from previous page)

```
{% block neutron_server_footer %}
RUN git clone https://opendev.org/x/networking-cisco \
    && python3 -m pip --no-cache-dir install networking-cisco
{% endblock %}
```

Some readers may notice there is one problem with this, however. Assuming nothing else in the Dockerfile changes for a period of time, the above `RUN` statement will be cached by Docker, meaning new commits added to the Git repository may be missed on subsequent builds. To solve this, the `kolla-build` tool also supports cloning additional repositories at build time, which will be automatically made available to the build, within an archive named `plugins-archive`.

To use this, add a section to `kolla-build.conf` in the following format:

```
[<image-name>-plugin-<plugin-name>]
```

Where <`image-name`> is the hyphenated name of the image that the plugin should be installed into, and <`plugin-name`> is the chosen plugin identifier.

Continuing with the above example, one could add the following to `kolla-build.conf`:

```
[neutron-server-plugin-networking-cisco]
type = git
location = https://opendev.org/x/networking-cisco
reference = master
```

The build will clone the repository, resulting in the following archive structure:

```
plugins-archive.tar
|__ plugins
    |__networking-cisco
```

The template now becomes:

```
{% block neutron_server_footer %}
ADD plugins-archive /
python3 -m pip --no-cache-dir install /plugins/*
{% endblock %}
```

Some plugins are installed by default. For images with default plugins, the Dockerfiles already copy the `plugins-archive` to the image and install available plugins at build time. These default plugins may be disabled by setting `enabled` to `False` in the relevant plugin source configuration section in `kolla-build.conf`.

Neutron plugins

One example of a service with many available plugins is Neutron. The `neutron-base` image Dockerfile has plugins archive copying and installation enabled already. In the `contrib` directory of Kolla (as available in the repository, the tarball or the `share` directory of the installation target), there is a `neutron-plugins` directory with examples of Neutron plugins definitions. Some of these plugins used to be enabled by default but, due to their release characteristic, have been excluded from the default builds. Please read the included `README.rst` to learn how to apply them.

Additions functionality

The Dockerfile customisation mechanism is useful for adding/installing additions into images. An example of this is adding your jenkins job build metadata (say, formatted into a jenkins.json file) into the image.

Similarly to the plugins mechanism, the Kolla build tool also supports cloning additional repositories at build time, which will be automatically made available to the build, within an archive named `additions-archive`. The main difference between `plugins-archive` and `additions-archive` is that `plugins-archive` is automatically copied in many images and processed to install available plugins while `additions-archive` processing is left solely to the Kolla user.

To use this, add a section to `kolla-build.conf` in the following format:

```
[<image>-additions-<additions-name>]
```

Where `<image-name>` is the hyphenated name of the image that the additions should be copied into, and `<additions-name>` is the chosen additions identifier.

For example, one could add the following to `kolla-build.conf` file:

```
[neutron-server-additions-jenkins]
type = local
location = /path/to/your/jenkins/data
```

The build will copy the directory, resulting in the following archive structure:

```
additions-archive.tar
|__ additions
    |__jenkins
```

The template becomes now:

```
{% block neutron_server_footer %}
ADD additions-archive /
RUN cp /additions/jenkins/jenkins.json /jenkins.json
{% endblock %}
```

Custom docker templates

In order to unify the process of managing OpenStack-related projects, Kolla provides a way of building images for external non-built-in projects.

If the template for a non-built-in project meets Kolla template standards, an operator can provide a root directory with a template via the `--docker-dir` CLI option (can be specified multiple times).

All Kollas jinja2 macros should be available the same as for built-in projects with some notes:

- The `configure_user` macro. As the non-built-in user is unknown to Kolla, there are no default values for user ID and group ID to use. To use this macro, an operator should specify non-default user details with `<custom_user_name>-user` configuration section and include info for `uid` and `gid` at least.

Lets look into how an operator can build an image for an in-house project with Kolla using openstack/releases project.

First, create a `Dockerfile.j2` template for the project.

```
FROM {{ namespace }}/{{ image_prefix }}openstack-base:{{ tag }}

{% block labels %}
LABEL maintainer="{{ maintainer }}" name="{{ image_name }}" build-date="{{␣
↪build_date }}"
{% endblock %}

{% block releaser_header %}{% endblock %}

{% import "macros.j2" as macros with context %}

{{ macros.configure_user(name='releaser') }}

RUN ln -s releaser-source/* /releaser \
    && {{ macros.install_pip(['/releaser-source'] | customizable("pip_
↪packages")) }} \
    && mkdir -p /etc/releaser \
    && chown -R releaser: /etc/releaser \
    && chmod 750 /etc/sudoers.d \
    && touch /usr/local/bin/kolla_releaser_extend_start \
    && chmod 644 /usr/local/bin/kolla_extend_start /usr/local/bin/kolla_
↪releaser_extend_start

{% block footer %}{% endblock %}
```

Suggested directory structure:

```
custom-kolla-docker-templates
|__ releaser
    |__ Dockerfile.j2
```

Then, modify Kollas configuration so the engine can download sources and configure users.

```
[releaser]
type = git
location = https://opendev.org/openstack/releases
reference = master

[releaser-user]
uid = 53001
gid = 53001
```

Last pre-check before building a new image - ensure that the new template is visible for Kolla:

```
$ kolla-build --list-images --docker-dir custom-kolla-docker-templates "^
↪releaser$"
1 : base
2 : releaser
3 : openstack-base
```

And finally, build the `releaser` image, passing the `--docker-dir` argument:

```
kolla-build --docker-dir custom-kolla-docker-templates "^releaser$"
```

Can I use the `--template-override` option for custom templates? Yes!

Custom repos

Red Hat

Kolla allows the operator to build containers using custom repos. The repos are accepted as a list of comma separated values and can be in the form of `.repo`, `.rpm`, or a url.

If specifying a `.repo` file, each `.repo` file will need to exist in the same directory as the base Dockerfile (`kolla/docker/base`) or you need to specify a url:

```
rpm_setup_config = epel.repo,https://remote-server.com/your-repo.repo
```

Debian / Ubuntu

For Debian based images, additional apt sources may be added to the build as follows:

```
apt_sources_list = custom.list
```

Building behind a proxy

We can insert http_proxy settings into the images to fetch packages during build, and then unset them at the end to avoid having them carry through to the environment of the final images. Note, however, its not possible to drop the info completely using this method; it will still be visible in the layers of the image.

To set the proxy settings, we can add this to the templates header block:

```
ENV http_proxy=https://evil.corp.proxy:80
ENV https_proxy=https://evil.corp.proxy:80
```

To unset the proxy settings, we can add this to the templates footer block:

```
ENV http_proxy=""
ENV https_proxy=""
```

Besides this configuration options, the script will automatically read these environment variables. If the host system proxy parameters match the ones going to be used, no other input parameters will be needed. These are the variables that will be picked up from the user env:

```
HTTP_PROXY, http_proxy, HTTPS_PROXY, https_proxy, FTP_PROXY,
ftp_proxy, NO_PROXY, no_proxy
```

Also these variables could be overwritten using `--build-args`, which have precedence.

Cross-compiling

It is possible to cross-compile container images in order to, e.g., build `aarch64` images on a `x86_64` machine.

To build `ARM` images on `x86_64` platform, pass the `--base-arch` and `--platform` arguments:

```
kolla-build --platform linux/arm64 --base-arch aarch64
```

Note

To make this work on x86_64 platform you can use tools like: qemu-user-static or binfmt.

To make this work on Apple Silicon you can use Docker Desktop or Podman Desktop to build `x86_64` or native `ARM` images.

Known issues

1. Mirrors are unreliable.

   Some of the mirrors Kolla uses can be unreliable. As a result occasionally some containers fail to build. To rectify build problems, the build tool will automatically attempt three retries of a build operation if the first one fails. The retry count is modified with the `--retries` option.

OVS-DPDK Source build

CentOS currently does not provide packages for ovs with dpdk. The Ubuntu packages do not support UIO based drivers. To use the uio_pci_generic driver on Ubuntu a source build is required.

Building ovs with dpdk containers from source

Append the following to `/etc/kolla/kolla-build.conf` to select the version of ovs and dpdk to use for your source build.

```
[openvswitch-base-plugin-ovs]
type = git
location = https://github.com/openvswitch/ovs.git
reference = v2.10.0

[openvswitch-base-plugin-dpdk]
type = git
location = http://dpdk.org/git/dpdk
reference = v17.11
```

To build the container, run the following command inside a cloned kolla repository:

```
tools/build.py -t source --template-override contrib/template-override/ovs-
↪dpdk.j2 ovsdpdk
```

4.1.2 Kolla Images API

Take advantage of the Kolla API to configure containers at runtime.

Kolla Images API

Kolla offers two different ways to make changes to containers at runtime. The first is via a *configuration file* exposed to the container and processed by the init scripts, and the second is via more traditional *environment variables*.

External Config

All of the Kolla images understand a JSON-formatted configuration describing a set of actions the container needs to perform at runtime before it executes the (potentially) long running process. This configuration also specifies the command to execute to run the service.

When a container runs kolla_start, the default entry-point, it processes the configuration file using kolla_set_configs with escalated privileges, meaning it is able to set file ownership and permissions.

Format of the configuration file

The kolla_set_configs script understands the following attributes:

- **command** (required): the command the container runs once it finishes the initialization step.
- **config_files**: copies files and directories inside the container. A list of dicts, each containing the following attributes:
  - **source** (required): path to the file or directory that needs to be copied. Understands shell wildcards.
  - **dest** (required): path to where the file or directory will be copied. does not need to exist, destination is deleted if it exists.
  - **owner** (required, unless `preserve_properties` is set to true): the `user:group` to change ownership to. `user` is synonymous to `user:user`. Must be user and group names, not uid/gid.
  - **perm** (required, unless *preserve_properties* is set to true): the unix permissions to set to the target files and directories. Must be passed in the numeric octal form.
  - **preserve_properties**: copies the ownership and permissions from the original files and directory. Boolean, defaults to `false`.
  - **optional**: do not raise an error when the source file is not present on the filesystem. Boolean, defaults to `false`.
  - **merge**: merges the source directory into the target directory instead of replacing it. Boolean, defaults to `false`.
- **permissions**: change the permissions and/or ownership of files or directories inside the container. A list of dicts, each containing the following attributes:
  - **path** (required): the path to the file or directory to update.
  - **owner** (required): the `user:group` to change ownership to. `user` is synonymous to `user:user`. Must be user and group names, not uid/gid.

   - **perm**: the unix permissions to set to the target files and directories. Must be passed in the numeric octal form.
   - **recurse**: whether to apply the change recursively over the target directory. Boolean, defaults to `false`.
   - **exclude**: array of names of the directories or files to be excluded when `recurse` is set to `true`. Supports Python regular expressions. Defaults to empty array.

Here is an example configuration file:

```
{
    "command": "trove-api --config-file=/etc/trove/trove.conf",
    "config_files": [
        {
            "source": "/var/lib/kolla/config_files/trove.conf",
            "dest": "/etc/trove/trove.conf",
            "owner": "trove",
            "perm": "0600",
            "optional": false
        }
    ],
    "permissions": [
        {
            "path": "/var/log/kolla/trove",
            "owner": "trove:trove",
            "recurse": true,
            "exclude": ["/var/log/^snapshot.*"]
        }
    ]
}
```

Passing the configuration file to the container

The configuration to the container can be passed through a dedicated path: `/var/lib/kolla/config_files/config.json`. It is advised to ensure this path is mounted read-only for security reasons.

Mounting the configuration file in the container:

```
docker run -e KOLLA_CONFIG_STRATEGY=COPY_ALWAYS \
    -v /path/to/config.json:/var/lib/kolla/config_files/config.json:ro \
    kolla-image
```

Environment Variables

Variables to pass to the containers

The Kolla containers also understand some environment variables to change their behavior at runtime:

- **KOLLA_CONFIG_STRATEGY** (required): Defines how the *kolla_start script* copies the configuration file. Must be one of:

  - **COPY_ONCE**: the configuration files are copied just once, the first time the container is started. In this scenario the container is perfectly immutable.
  - **COPY_ALWAYS**: the configuration files are copied each time the container starts. If a config file changes on the host, the change is applied in the container the next time it restarts.
- **KOLLA_SKIP_EXTEND_START**: if set, bypass the `extend_start.sh` script. Not set by default.
- **KOLLA_SERVICE_NAME**: if set, shows the value of the variable on the `PS1` inside the container. Not set by default.
- **KOLLA_BOOTSTRAP**: if set, and supported by the image, runs the bootstrap code defined in the images `extend_start.sh` scripts. Not set by default.
- **KOLLA_UPGRADE**: if set, and supported by the image, runs the upgrade code defined in the images `extend_start.sh` scripts. Not set by default.
- **KOLLA_UPGRADE_CHECK**: if set, and supported by the image, runs the `<service>-status upgrade check` command, defined in the images `extend_start.sh` scripts. Currently, this is hard-coded to just `nova-status upgrade check`. Not set by default.
- **KOLLA_OSM**: if set, and supported by the image, runs the online database migration code defined in the images `extend_start.sh` scripts. Not set by default.

The containers may expose other environment variables for turning features on or off, such as the horizon container that looks for `ENABLE_XXX` variables where `XXX` is a horizon plugin name. These are generally defined in the container-specific `extend_start.sh` script, example for horizon.

Variables available in the containers

The following variables available in all images and can be evaluated in scripts:

- **KOLLA_BASE_DISTRO**: `base_distro` used to build the image (e.g. centos, ubuntu)

CHAPTER

FIVE

SUPPORT MATRIX

5.1 Kolla Images Support Matrix

This page describes the supported base container image distributions and versions, and the Kolla images supported on each of those.

5.1.1 Supported base images

The following base container images are supported:

Distribution	Default base	Default base tag
Rocky Linux	quay.io/rockylinux/rockylinux	10
Debian Trixie	debian	trixie
Ubuntu Noble	ubuntu	24.04

The remainder of this document outlines which images are supported on which of these distribution.

5.1.2 Ceph versions in Kolla images

Table 1: Ceph versions

Distro	Ceph Source	Release
Rocky Linux	CentOS Storage SIG	Squid
Ubuntu	Ubuntu	Squid
Debian	Debian	Pacific

5.1.3 Support clause definitions

T - Tested

Coverage:

- CI in `kolla-ansible` is testing that images are functional
- kolla core team is maintaining versions

U - Untested

Coverage:

- CI in `kolla-ansible` is *NOT* testing that images are functional
- Many untested services are working fine, but the kolla core team cannot guarantee that they are all functional

N - Not Available/Unknown

Not available *(e.g. not buildable)*. Please see *Currently unbuildable images*

5.1.4 Deprecations

None

5.1.5 x86_64 images

Table 2: x86_64 images

Image	Rocky Linux	Ubuntu	Debian
aodh	U	U	U
barbican	T	U	U
bifrost	T	U	U
blazar	U	U	U
ceilometer	U	U	U
cinder	T	T	U
cloudkitty	U	U	U
collectd	U	U	U
cron	T	T	T
cyborg	U	U	U
designate	U	U	U
dnsmasq	T	T	U
etcd	T	T	U
fluentd	T	T	T
glance	T	T	T
gnocchi	U	U	U
grafana	U	U	U
hacluster	U	U	U
hacluster-pcs	N	U	U
haproxy	T	T	U
heat	T	T	T
horizon	T	T	T
ironic	T	T	U
iscsid	T	T	U
keepalived	T	T	U
keystone	T	T	T
kolla-toolbox	T	T	T
kuryr	T	T	U
magnum	U	U	U
manila	U	U	U

continues on next page

Table 2 – continued from previous page

Image	Rocky Linux	Ubuntu	Debian
mariadb	T	T	T
masakari	T	T	U
memcached	T	T	U
mistral	T	U	U
multipathd	U	U	U
neutron	T	T	T
neutron-mlnx-agent	U	U	U
nova	T	T	T
nova-spicehtml5proxy	N	T	T
octavia	U	U	U
opensearch	T	T	U
openvswitch	T	T	T
ovn	U	U	U
ovsdpdk	N	U	U
placement	T	T	T
prometheus	U	U	U
rabbitmq	T	T	T
skyline	U	U	U
swift	T	T	U
tacker	T	U	U
telegraf	U	U	U
tgtd	N	T	U
trove	U	U	U
valkey	T	U	U
watcher	U	U	U
zun	T	T	U

5.1.6 aarch64 images

Table 3: aarch64 images

Image	Rocky Linux	Ubuntu	Debian
aodh	U	U	U
barbican	U	U	U
bifrost	N	N	N
blazar	U	U	U
ceilometer	U	U	U
cinder	U	U	U
cloudkitty	U	U	U
collectd	U	U	U
cron	U	U	U
cyborg	U	U	U
designate	U	U	U
dnsmasq	U	U	U
etcd	U	U	U
fluentd	U	U	U
glance	U	U	U

continues on next page

Table 3 – continued from previous page

Image	Rocky Linux	Ubuntu	Debian
gnocchi	U	U	U
grafana	U	U	U
hacluster	N	U	U
haproxy	U	U	U
heat	U	U	U
horizon	U	U	U
ironic	U	U	U
iscsid	U	U	U
keepalived	U	U	U
keystone	U	U	U
kolla-toolbox	U	U	U
kuryr	U	U	U
magnum	U	U	U
manila	U	U	U
mariadb	U	U	U
masakari	U	U	U
memcached	U	U	U
mistral	U	U	U
multipathd	U	U	U
neutron	U	U	U
neutron-mlnx-agent	U	U	U
nova	U	U	U
nova-spicehtml5proxy	N	U	U
octavia	U	U	U
openvswitch	U	U	U
opensearch	U	U	U
ovn	U	U	U
ovsdpdk	N	U	U
placement	U	U	U
prometheus	U	U	U
rabbitmq	U	U	U
skyline	U	U	U
tacker	U	U	U
telegraf	N	N	N
tgtd	U	U	U
trove	U	U	U
valkey	U	U	U
watcher	U	U	U
zun	U	U	U

5.1.7 Currently unbuildable images

For a list of currently unbuildable images please look into `kolla/image/unbuildable.py` file - `UNBUILDABLE_IMAGES` dictionary.

CHAPTER

SIX

CONTRIBUTOR GUIDE

6.1 Contributor Guide

This guide is for contributors of the Kolla project. It includes information on proposing your first patch and how to participate in the community. It also covers responsibilities of core reviewers and the Project Team Lead (PTL), and information about development processes.

We welcome everyone to join our project!

6.1.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Kolla.

Basics

The source repository for this project can be found at:

> https://opendev.org/openstack/kolla

Communication

IRC Channel
: `#openstack-kolla` (channel logs) on OFTC

Weekly Meetings
: In the IRC channel (meeting information)

Mailing list (prefix subjects with `[kolla]`)
: https://lists.openstack.org/pipermail/openstack-discuss/

Meeting Agenda
: *Meeting agenda*

Whiteboard (etherpad)
: Keeping track of CI gate status, release status, stable backports, planning and feature development status. https://etherpad.openstack.org/p/KollaWhiteBoard

Contacting the Core Team

In general it is suggested to use the above mentioned public communication channels, but if you find that you need to contact someone from the Core team directly, you can find the lists in Gerrit:

- Kolla core team
- Kayobe core team

New Feature Planning

New features are discussed on PTG, via IRC or mailing list (with [kolla] prefix).

Task Tracking

Kolla family projects track tasks and bugs in Launchpad:

- Kolla
- Kolla Ansible
- Ansible Collection Kolla
- Kayobe

If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag in the relevant project.

A more lightweight task tracking is done via etherpad - Whiteboard.

Reporting a Bug

You found an issue and want to make sure we are aware of it? Please report it in the appropriate Launchpad project:

- Kolla
- Kolla Ansible
- Ansible Collection Kolla
- Kayobe

Note these are also used for task tracking.

Getting Your Patch Merged

Most changes proposed to Kolla require two +2 votes from core reviewers before being approved and sent to the gate queue for merging. A release note is required on most changes as well. Release notes policy is described in *its own section*.

Significant changes should have documentation and testing provided with them.

To see an overview of in-progress patches, sorted into useful sections, you can view the:

- Kolla Review Dashboard
- Kolla Ansible Review Dashboard
- Ansible Collection Kolla Review Dashboard
- Kayobe Review Dashboard

Note

If you feel that your change is getting overlooked, reach out via *preferred communication methods* to let us know.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide. Kolla-specific PTL duties are listed in Kolla PTL guide.

6.1.2 Kolla design philosophy

This page aims to explain the design choices that have been made in the Kolla projects (the main Kolla project for building the images as well as all subprojects aimed at deploying them).

It is written down mostly for potential contributors but may be a good guideline for anyone using and integrating with Kolla projects as it gives an overview of what the preferred approach is. As noted, this is of great interest to contributors because it might explain why certain changes will get rejected and why some others will need a remake in order to be accepted.

This page is likely an always-in-progress, living document. Please reach out to the team via the mailing list or the IRC channel to discuss the rules noted here, see our docs on *communication* for details.

Do not own the deployed services config defaults (unless necessary)

In Kolla, we try not to own service config and its defaults. We believe the upstream services know best what works for them by default.

This rule is overridden when a certain config option needs to be orchestrated among multiple services or otherwise needs to agree with certain other assumptions made in the environment. The notable exceptions here are the basic addressing of services and Keystone credentials.

Prefer documented configuration via config overrides

as opposed to new in-Ansible variables.

This plays nicely with the config overrides capabilities of Kolla Ansible which let users easily customise the services config to their hearts contents regardless of what Kolla Ansible offers via Ansible variables.

The reasoning behind this rule is that it lowers the maintenance burden yet it does not handicap the users - they can control every last detail of their config.

One of the (not necessarily planned) side-effects of this is that users config overrides are not locked-in for Kolla Ansible and can relatively easily be reused in other configuration systems.

6.1.3 Running Kolla Build in development

The recommended way to run in development

To clone the repository and install the package in development mode, run the following commands:

```
git clone https://opendev.org/openstack/kolla.git
cd kolla
python3 -m venv ~/path/to/venv
source ~/path/to/venv/bin/activate
```

(continues on next page)

(continued from previous page)

```
python3 -m pip install --editable .
kolla-build ...
```

6.1.4 Adding a new image

Kolla follows Best practices for writing Dockerfiles where at all possible.

We use `jinja2` templating syntax to help manage the volume and complexity that comes with maintaining multiple Dockerfiles for multiple different base operating systems.

Dockerfiles should be placed under the `docker` directory. OpenStack services should inherit from the provided `openstack-base` image, and infrastructure services (for example: `fluentd`) should inherit from `base`.

Projects consisting of only one service should be placed in an image named the same as that service, for example: `horizon`. Services that consist of multiple processes generally use a base image and child images, for example: `cinder-base`, `cinder-api`, `cinder-scheduler`, `cinder-volume`, `cinder-backup`.

Jinja2 *blocks* are employed throughout the Dockerfiles to help operators customise various stages of the build (refer to *Dockerfile Customisation*)

Some of these blocks are free form. However, there is a subset that should be common to every Dockerfile. The overall structure of a Dockerfiles of an OpenStack project base image is as follows:

```
FROM {{ namespace }}/{{ image_prefix }}openstack-base:{{ tag }}
{% block labels %}
LABEL maintainer="{{ maintainer }}" name="{{ image_name }}" build-date="{{␣
↪build_date }}"
{% endblock %}

{% block << service >>_header %}{% endblock %}

{% import "macros.j2" as macros with context %}

<< common steps >>

{% block << service >>_footer %}{% endblock %}
{% block footer %}{% endblock %}
```

Note

The generic footer block `{% block footer %}{% endblock %}` should **not** be included in base images (for example: `cinder-base`).

Its probably easiest to identify the most similar service being already provided, copy its Dockerfile structure and amend it to new needs.

Distribution support

By default, every new image should support all supported distributions (Debian, Ubuntu, Rocky) and both x86-64 and aarch64 architectures. Sometimes it is not doable so we have list of *unbuildable images* for that.

Unbuildable images

In `kolla/image/unbuildable.py` source file we keep a list of images which cannot be built for some distribution/architecture/build-type combinations.

```
# Note: These are just examples, find the current list at
# https://opendev.org/openstack/kolla/src/branch/master/kolla/image/
↪unbuildable.py
UNBUILDABLE_IMAGES = {
    'aarch64': {
        "bifrost-base",     # someone need to get upstream working first
    },

    'centos': {
        "hacluster-pcs",    # Missing crmsh package
    },

    'ubuntu+aarch64': {
        "kibana",           # no binary package
    },
}
```

If your new image has some unbuildable combinations, please add it into proper place in this list. If you are not sure, write it in code review comment and check CI results of your patch.

Note

Please do not overuse this list it is meant as last hope solution.

6.1.5 Generating kolla-build.conf

Install tox and generate the build configuration. The build configuration is designed to hold advanced customizations when building images.

If you have already cloned the Kolla Git repository to the `kolla` folder, generate the `kolla-build.conf` file using the following steps.

```
python3 -m pip install tox
cd kolla/
tox -e genconfig
```

The location of the generated configuration file is `etc/kolla/kolla-build.conf`.

6.1.6 Release notes

Introduction

Kolla uses the following release notes sections:

- `features` for new features or functionality; these should ideally refer to the blueprint being implemented;
- `fixes` for fixes closing bugs; these must refer to the bug being closed;
- `upgrade` for notes relevant when upgrading from previous version; these should ideally be added only between major versions; required when the proposed change affects behaviour in a non-backwards compatible way or generally changes something impactful;
- `deprecations` to track deprecated features; relevant changes may consist of only the commit message and the release note;
- `prelude` filled in by the PTL before each release or RC.

Other release note types may be applied per common sense.

When a release note is required:

- `feature` - best included with docs change (if separate from the code)
- `user impacting` - to improve visibility of the change for users

Remember release notes are mostly for end users which, in case of Kolla, are OpenStack administrators/operators. In case of doubt, the core team will let you know what is required.

To add a release note, run the following command:

```
tox -e venv -- reno new <summary-line-with-dashes>
```

All release notes can be inspected by browsing `releasenotes/notes` directory. Further on this page we show reno templates, examples and how to make use of them.

Note

The term *release note* is often abbreviated to *reno* as it is the name of the tool that is used to manage the release notes.

To generate renos in HTML format in `releasenotes/build`, run:

```
tox -e releasenotes
```

Note this requires the release note to be tracked by `git` so you have to at least add it to the `gits` staging area.

The release notes are linted in the CI system. To lint locally, run:

```
tox -e doc8
```

The above lints all of documentation at once.

Templates and examples

All approved release notes end up being published on a dedicated site:

> https://docs.openstack.org/releasenotes/kolla/

When looking for examples, it is advised to consider browsing the page above for a similar type of change and then comparing with their source representation in `releasenotes/notes`.

The sections below give further guidelines. Please try to follow them but note they are not set in stone and sometimes a different wording might be more appropriate. In case of doubt, the core team will be happy to help.

Features

Template

```
---
features:
  - |
    Implements [some feature].
    [Can be described using multiple sentences if necessary.]
    [Limitations worth mentioning can be included as well.]
    `Blueprint [blueprint id] <https://blueprints.launchpad.net/kolla/+spec/
↪[blueprint id]>`__
```

Note

The blueprint can be mentioned even if the change implements it only partially. This can be emphasised by preceding the `Blueprint` word by `Partial`. See the example below.

Example

Implementing blueprint with id *letsencrypt-https*, we use `reno` to generate the scaffolded file:

```
tox -e venv -- reno new --from-template releasenotes/templates/feature.yml␣
↪blueprint-letsencrypt-https
```

Note

Since we dont require blueprints for simple features, it is allowed to make up a blueprint-id-friendly string (like in the example here) ad-hoc for the proposed feature. Please then skip the `blueprint-` prefix to avoid confusion.

And then fill it out with the following content:

```
---
features:
  - |
    Implements support for hassle-free integration with Let's Encrypt.
```

(continues on next page)

(continued from previous page)

```
    The support is limited to operators in the underworld.
    For more details check the TLS docs of Kolla.
    `Partial Blueprint letsencrypt-https <https://blueprints.launchpad.net/
↪kolla/+spec/letsencrypt-https>`__
```

Note

The example above shows how to introduce a limitation. The limitation may be lifted in the same release cycle and it is OK to mention it nonetheless. Release notes can be edited later as long as they have not been shipped in an existing release or release candidate.

Fixes

Template

```
---
fixes:
  - |
    Fixes [some bug].
    [Can be described using multiple sentences if necessary.]
    [Possibly also giving the previous behaviour description.]
    `LP#[bug number] <https://launchpad.net/bugs/[bug number]>`__
```

Example

Fixing bug number *1889611*, we use `reno` to generate the scaffolded file:

```
tox -e venv -- reno new --from-template releasenotes/templates/fix.yml bug-
↪1889611
```

And then fill it out with the following content:

```
---
fixes:
  - |
    Fixes ``deploy-containers`` action missing for the Masakari role.
    `LP#1889611 <https://launchpad.net/bugs/1889611>`__
```

6.1.7 Running tests

Kolla contains a suite of tests in the `tests` and `kolla/tests` directories.

Any proposed code change in gerrit is automatically rejected by the OpenStack Zuul CI system if the change causes test failures.

It is recommended for developers to run the test suite before submitting patch for review. This allows to catch errors as early as possible.

Preferred way to run the tests

The preferred way to run the unit tests is using `tox`. It executes tests in isolated environment, by creating separate virtualenv and installing dependencies from the `requirements.txt`, `test-requirements.txt` and `doc/requirements.txt` files, so the only package you install is `tox` itself:

```
pip install tox
```

See the unit testing section of the Testing wiki page for more information. Following are some simple examples.

To run the Python 3.8 tests:

```
tox -e py38
```

To run the style tests:

```
tox -e pep8
```

To run multiple tests separate items by commas:

```
tox -e py38,pep8
```

Running a subset of tests

Instead of running all tests, you can specify an individual directory, file, class or method that contains test code, for example, filter full names of tests by a string.

To run the tests located only in the `kolla/tests` directory:

```
tox -e py38 kolla.tests
```

To run the tests of a specific file say `kolla/tests/test_set_config.py`:

```
tox -e py38 test_set_config
```

To run the tests in the `ConfigFileTest` class in the `kolla/tests/test_set_config.py` file:

```
tox -e py38 test_set_config.ConfigFileTest
```

To run the `ConfigFileTest.test_delete_path_not_exists` test method in the `kolla/tests/test_set_config.py` file:

```
tox -e py38 test_set_config.ConfigFileTest.test_delete_path_not_exists
```

Coverage Report Generation

In order to get coverage report for Kolla, run the below command.

```
tox -e cover
```

Debugging unit tests

In order to break into the debugger from a unit test we need to insert a breaking point to the code:

```
import pdb; pdb.set_trace()
```

Then run **tox** with the debug environment as one of the following:

```
tox -e debug
tox -e debug test_file_name.TestClass.test_name
```

For more information see the oslotest documentation.

6.1.8 Code Reviews

All Kolla code must be reviewed and approved before it can be merged. Anyone with a Gerrit account is able to provide a review. Two labels are available to everyone:

- +1: Approve
- -1: Changes requested

It is also possible to leave comments without a label. In general, a review with comments is more valuable. Comments are especially important for a negative review. Prefer quality of reviews over quantity.

You can watch specific patches in Gerrit via *Settings -> Watched Projects*. The volume of emails is not too large if you subscribe to *New Changes* only. If you do not have much time available for reviewing, consider reviewing patches in an area that is important to you or that you understand well.

Core reviewers

Core reviewers have additional labels available to them.

- +2: Approve
- -2: Do not merge
- Workflow +1: Approve and ready for merge

Zuul requires one +2 and one workflow +1, as well as a passing check, in order for a patch to proceed to the gate. The Kolla team generally requires two +2s before a workflow +1 may be added. We also have some non-voting Zuul jobs which will not block a check, but should be investigated if they are failing.

Core reviewers may still use +1 to indicate approval if they are not confident enough about a particular patch to use +2.

The Kolla core reviewers have the same rights of access to stable branches, so always check the branch for a review, and use extra care with stable branches.

Becoming a core reviewer

There are no strict rules for becoming a core reviewer. Join the community, review some patches, and demonstrate responsibility, understanding & care. If you are interested in joining the core team, ask the PTL or another core reviewer how to get there.

6.1.9 Bug triage

The triage of Kolla bugs follows the OpenStack-wide process documented on BugTriage in the wiki. Please reference Bugs for further details.

6.1.10 PTL Guide

This is just a reference guide that a PTL may use as an aid, if they choose. It is meant to complement the official PTL guide, and is laid out in rough chronological order.

Some or all of these tasks may be delegated to other team members.

New PTL

- Update the kolla meeting chair
  - https://opendev.org/opendev/irc-meetings/src/branch/master/meetings/kolla-team-meeting.yaml
- Update the team wiki
  - https://wiki.openstack.org/wiki/Kolla#Active_Contributors
- Get acquainted with the release schedule, bearing in mind that Kolla is a cycle-trailing project
  - Example: https://releases.openstack.org/train/schedule.html

Open Infrastructure Summit

Ideally the Kolla PTL will be able to attend the summit. If not, try to arrange for another member of the core team to represent the team. Good interaction with the community at these events is crucial to encourage upstream involvement, onboard new users, collect feedback and for the perceived health of the project.

- Create a summit planning etherpad and alert about it in the kolla IRC meeting and openstack-discuss mailing list
  - Example: https://etherpad.openstack.org/p/kolla-train-summit
- Gather ideas for forum sessions
  - Example: user feedback & roadmap, design sessions
- Prepare the project update presentation. Enlist help of others
- Prepare the on-boarding session materials. Enlist help of others
- Represent and promote the project while at the summit

Project Team Gathering (PTG)

Some of the Kolla team may decide to meet in person at the Project Team Gathering (PTG). Alternatively, they may decide to host a virtual PTG at a different time if there is not a critical mass of contributors attending the PTG.

- Create PTG planning etherpad and alert about it in the kolla IRC meeting and openstack-discuss mailing list
  - Example: https://etherpad.openstack.org/p/kolla-train-ptg
- Run sessions at the PTG

- Have a discussion about priorities for the upcoming release cycle at the PTG
- Sign up for group photo at the PTG (if applicable)
- Standard PTG topics:
  - Required distribution upgrades (e.g. new Ubuntu LTS release)
  - Ansible version bump
  - Infrastructure services/packages updates (e.g. RabbitMQ/Erlang upgrades)
  - Services that need to be deprecated/removed (e.g. unmaintained projects)

After Summit & PTG

- Send session summaries to the openstack-discuss mailing list
- Update the Kolla whiteboard with decided priorities for the upcoming release cycle

Day to Day

- Subscribe to the kolla projects on Launchpad to receive all bug and blueprint updates.
- *Triage new bugs*
- Monitor the status of the CI system for all supported branches. Fix issues that break the gate
- Chair the IRC meetings
- Be available in IRC to help new and existing contributors
- Keep track of the progress of cycle priorities
- Monitor the core team membership, mentor potential cores

Release Management

- Follow the projects *release management* guide
- Use the IRC meeting and/or mailing list to communicate release schedule to the team who might not be so famailiar with it

Handing Over

- Support the new PTL in their new role. Try to remember the issues you encountered
- Update this page with any useful information you have learned

6.1.11 Release Management

This guide is intended to complement the OpenStack releases site, and the project team guides section on release management.

Team members make themselves familiar with the release schedule for the current release, for example https://releases.openstack.org/train/schedule.html.

Concepts & Aims

Release Model

As a deployment project, Kollas release model differs from many other OpenStack projects. Kolla follows the cycle-trailing release model, to allow time after the OpenStack coordinated release to wait for distribution packages and support new features. This gives us three months after the final release to prepare our final releases. Users are typically keen to try out the new release, so we should aim to release as early as possible while ensuring we have confidence in the release.

Overlapping Cycles

While the community may have the intention of releasing Kolla projects shortly after the OpenStack coordinated release, there are typically issues that prevent us from doing so, some of which may be outside of our control. Because of this, it is normal for there to be a period where the community is working on two releases - stabilising one for general availability, while developing features for another.

Date Notation

The OpenStack release schedule uses an `R-$N` notation to describe the timing of milestones and deadlines, where `$N` is the number of weeks until the coordinated OpenStack release (**not** the Kolla general release). For a typical 26 week release schedule, `R-26` is the first week, and `R-0` is the week of the coordinated release. We use that notation here, extended to include the period following a release as `R+$N`.

Early Cycle Stability

Early in the OpenStack release cycle, as projects make larger changes, it is common for the master branch to become less stable than normal. This can have a negative impact Kolla community, who may be trying to complete the previous release, or develop features for the current release. For this reason, from the Xena cycle, we will continue to build and deploy the previous OpenStack release for several weeks into the development cycle.

Feature Freeze

As with projects following the common release model, Kolla uses a feature freeze period to allow the code to stabilise prior to release. There is no official feature freeze date for the cycle-trailing model, but we aim to freeze **three weeks** after the common feature freeze. During this time, no features should be merged to the master branch, until the feature freeze is lifted 3 weeks later.

Release Schedule

While we dont wish to repeat the OpenStack release documentation, we will point out the high level schedule, and draw attention to areas where our process is different.

Launchpad Admin

We track series (e.g. Stein) and milestones (e.g. 10.0.1) on Launchpad, and target bugs and blueprints to these. Populating these in advance is necessary. This needs to be done for each of the following projects:

- https://launchpad.net/kolla
- https://launchpad.net/kolla-ansible
- https://launchpad.net/ansible-collection-kolla

- https://launchpad.net/kayobe

At the beginning of a cycle, ensure a named series exists for the cycle in each project. If not, create one via the project landing page (e.g. https://launchpad.net/kolla) - in the Series and milestones section click in Register a series. Once the series has been created, create the necessary milestones, including the final release. Series can be marked as Active Development or Current Stable Release as necessary.

Milestones

At each of the various release milestones, pay attention to what other projects are doing.

R-23: Development begins

Feature freeze ends on the master branch of Kolla projects. We continue to build and deploy the previous release of OpenStack projects, as described in *Early Cycle Stability*.

- [all] Communicate end of feature freeze via IRC meeting and openstack-discuss mailing list.
- [kayobe] Switch `openstack_release` and `override_checkout` in Kayobe master branch to use the master branch of dependencies.

  Note

  The IPA image still needs to use the previous release in order to be compatible with Ironic.

  – example: https://review.opendev.org/c/openstack/kayobe/+/791764
- [all] Search for TODOs/FIXMEs/NOTEs in the codebases describing tasks to be performed during the new release cycle
  – may include deprecations, code removal, etc.
  – these usually reference either the new cycle or the previous cycle; new cycle may be referenced using only the first letter (for example: V for Victoria).

R-17: Switch source images to current release

- [kolla-ansible] Set `previous_release` variables to the previous release.
  – example: https://review.opendev.org/c/openstack/kolla-ansible/+/761835
- [kolla] Switch source images to use master branches.
  – This patch should include Depends-On in the commit message to the Kolla Ansible patch, to avoid skipping a release in upgrade tests
  – example: https://review.opendev.org/c/openstack/kolla/+/761742
- [kayobe] Set `previous_release` variables to the previous release.
  – example: https://review.opendev.org/c/openstack/kayobe/+/763375

R-8: Switch images to current release

Note

Debian does not provide repositories for the in-development release until much later in the cycle.

- [kolla] Switch Ubuntu images to use the current in-development release Ubuntu Cloud Archive (UCA) repository
  - example: https://review.opendev.org/c/openstack/kolla/+/782308

R-5: Cycle highlights deadline

- [all] Add cycle highlights when requested by the release team. They should be added to the deliverable file for the Kolla project, but also cover Kolla Ansible and Kayobe.
  - example: https://review.opendev.org/c/openstack/releases/+/779482
- [all] Check for new versions of infrastructure components
  - ansible (incl. kolla-toolbox)
  - ceph client libraries
  - fluentd (td-agent)
  - grafana
  - mariadb
  - opensearch
  - openvswitch/OVN
  - prometheus (incl. exporters)
  - rabbitmq/erlang

R-2: Feature freeze

Feature freeze for Kolla deliverables begins. Feature freeze exceptions may be granted within reason where two cores agree to review the code.

R-1: Prepare Kolla & Kolla Ansible for RC1 & stable branch creation

As defined by the cycle-trailing release model, a stable branch is created at the point of an RC1 release candidate.

Prior to creating an RC1 release candidate:

- [all] Test the code and fix (at a minimum) all critical bugs
- [all] The release notes for each project should be tidied up
  - this command is useful to list release notes added this cycle:
    - `git diff --name-only origin/stable/<previous release> -- releasenotes/`

> **Note**
>
> Release notes for backported changes (i.e. already present in the previous, stable branch) will not show in the output.

  - example (kolla): https://review.opendev.org/648677/
  - example (kolla-ansible): https://review.opendev.org/648685/
  - example (kayobe): https://review.opendev.org/c/openstack/kayobe/+/788432

- [kolla][kolla-ansible][ansible-collection-kolla] Mark bugs on Launchpad with the correct milestone
  - this command is useful to check for commits that fixed bugs:
    * `git log origin/stable/<previous release>..origin/master | grep -i Closes-Bug`
- [kolla] Update `OPENSTACK_RELEASE` variable in `kolla/common/config.py` to the name of the current in-development release
  - example: https://review.opendev.org/c/openstack/kolla/+/785500
- [kolla] Update versions of independently released projects on master:
  - `./tools/version-check.py --openstack-release $SERIES --include-independent`
  - example: TODO

R-0: Kolla & Kolla Ansible RC1 & stable branch creation

RC1 is the first release candidate, and also marks the point at which the stable branch is cut.

> **Note**
>
> Use the new-release tool for these activities.

- [kolla][kolla-ansible][ansible-collection-kolla] Create RC1 and stable branches by submitting patches to the releases repository
  - example: https://review.opendev.org/c/openstack/releases/+/786824
- [kolla][kolla-ansible][ansible-collection-kolla] Approve bot-proposed patches to master and the new stable branch
- [kolla][kolla-ansible] Ensure static links to documentation are enabled
  - https://opendev.org/openstack/openstack-manuals/src/branch/master/www/project-data
  - example: https://review.opendev.org/c/openstack/openstack-manuals/+/739206/

R-0: Prepare Kayobe for RC1 & stable branch creation

As defined by the cycle-trailing release model, a stable branch is created at the point of an RC1 release candidate.

Some of these tasks depend on the existence of Kolla and Kolla Ansible stable branches.

Prior to creating an RC1 release candidate:

- [kayobe] Synchronise with Kolla Ansible feature flags
  - Clone the Kolla Ansible repository, and run the Kayobe `tools/kolla-feature-flags.sh` script:

    ```
    tools/kolla-feature-flags.sh <path to kolla-ansible source>
    ```

  - Copy the output of the script, and replace the `kolla_feature_flags` list in `ansible/roles/kolla-ansible/vars/main.yml`.

    The `kolla.yml` configuration file should be updated to match:

    ```
    tools/feature-flags.py
    ```

  - Copy the output of the script, and replace the list of `kolla_enable_*` flags in `etc/kayobe/kolla.yml`.
  - example: https://review.opendev.org/c/openstack/kayobe/+/787775
- [kayobe] Synchronise with Kolla Ansible inventory

  Clone the Kolla Ansible repository, and copy across any relevant changes. The Kayobe inventory is based on the `ansible/inventory/multinode` inventory, but split into 3 parts - top-level, components and services.

  - The top level inventory template is `ansible/roles/kolla-ansible/templates/overcloud-top-level.j2`. It is heavily templated, and does not typically need to be changed. Look out for changes in the `multinode` inventory before the `[baremetal]` group.
  - The components inventory template is `ansible/roles/kolla-ansible/templates/overcloud-components.j2`.

    This includes groups in the `multinode` inventory from the `[baremetal]` group down to the following text:

    ```
    # Additional control implemented here. These groups allow you to␣
    ↪control which
    # services run on which hosts at a per-service level.
    ```

  - The services inventory template is `ansible/roles/kolla-ansible/templates/overcloud-services.j2`.

    This includes groups in the `multinode` inventory from the following text to the end of the file:

    ```
    # Additional control implemented here. These groups allow you to␣
    ↪control which
    # services run on which hosts at a per-service level.
    ```

  There are some small changes in this section which should be maintained.

  - example: https://review.opendev.org/c/openstack/kayobe/+/787775

- [kayobe] Update dependencies to upcoming release

  Prior to the release, we update the dependencies and upper constraints on the master branch to use the upcoming release. This is now quite easy to do, following the introduction of the `openstack_release` variable.

  - example: https://review.opendev.org/c/openstack/kayobe/+/787923

- [kayobe] Synchronise kayobe-config

  Ensure that configuration defaults in `kayobe-config` are in sync with those under `etc/kayobe` in `kayobe`. This can be done via:

  ```
  rsync -a --delete kayobe/etc/kayobe/ kayobe-config/etc/kayobe
  ```

  Commit the changes and submit for review.

  - example: https://review.opendev.org/c/openstack/kayobe-config/+/787924

- [kayobe] Synchronise kayobe-config-dev

  Ensure that configuration defaults in `kayobe-config-dev` are in sync with those in `kayobe-config`. This requires a little more care, since some configuration options have been changed from the defaults. Choose a method to suit you and be careful not to lose any configuration.

  Commit the changes and submit for review.

  - example: https://review.opendev.org/c/openstack/kayobe-config-dev/+/788426

R+1: Kayobe RC1 & stable branch creation

RC1 is the first release candidate, and also marks the point at which the stable branch is cut.

Note

Use the new-release tool for these activities.

- [kayobe] Create RC1 and stable branches by submitting patches to the releases repository

  - example: https://review.opendev.org/c/openstack/releases/+/788982

- [kayobe] Approve bot-proposed patches to master and the new stable branch

R+0 to R+13: Finalise stable branch

Several tasks are required to finalise the stable branch for release.

- [kolla-ansible][kayobe] Switch to use the new branch of `ansible-collection-kolla` in `requirements.yml`.

  Note

  This needs to be done on the stable branch.

- [kolla-ansible] Switch to use the newly tagged container images (the branch for development mode on the new stable branch follows automatically since Victoria)

  > **Note**
  >
  > This needs to be done on the stable branch.

  > **Note**
  >
  > This requires the images to have been published to quay.io with the new tag.

  - example: https://review.opendev.org/c/openstack/kolla-ansible/+/788292
- [kolla] Switch Debian images to use the Debian OpenStack repository for the new release

  > **Note**
  >
  > This needs to be done on the master branch and stable branch.

  - example: https://review.opendev.org/c/openstack/kolla/+/788304

R+0 to R+13: Further release candidates and final release

Once the stable branches are finalised, further release candidates may be created as necessary in a similar manner to RC1.

A release candidate may be promoted to a final release if it has no critical bugs against it.

- [all] Create final release by submitting patches to the releases repository
  - example: https://review.opendev.org/c/openstack/releases/+/769328
- [all] Update openstack-manuals project-data for kolla + kolla-ansible
  - example: https://review.opendev.org/c/openstack/openstack-manuals/+/934349

After final release, projects enter the *Stable Branch Lifecycle* with a status of Maintained.

R+13 marks the 3 month deadline for the release of cycle-trailing projects.

Stable Branch Lifecycle

The lifecycle of stable branches in OpenStack is described in the project team guide. The current status of each branch is published on the releases site.

Maintained

Releases should be made periodically for each maintained stable branch, no less than once every 45 days. We try to make one release per month by having a recurring topic for that in the first Kolla meeting each month.

- Create stable releases by submitting patches to the releases repository
  - follow SemVer guidelines, for simplicity consider always making minor version bumps
  - you can use the tooling from the requirements team to prepare the patches:

```
git checkout -b kolla-stable-monthly
for project in ansible-collection-kolla kayobe kolla kolla-ansible;␣
↪do
    for rel in zed antelope bobcat; do
        tox -e venv -- new-release $rel $project feature
    done
done
git commit -am "Tag monthly kolla stable releases"
git review -f
```

- example release patch: https://review.opendev.org/c/openstack/releases/+/860521

Extended Maintenance (EM)

When a branch is entering EM, projects will make final releases. The release team will propose tagging the Kolla deliverables as EM, but this should only be done once all other dependent projects have made their final release, and final Kolla releases have been made including those dependencies.

After a branch enters EM, we typically do the following:

- stop backporting fixes to the branch by default. Important fixes or those requested by community members may be merged if deemed appropriate
- stop publishing images to Quay.io
- stop actively maintaining CI

End of Life (EOL)

Once a branch has been unmaintained (failing CI, no patches merged) for 6 months, it may be moved to EOL. Since this is done at different times for different projects, send an email to openstack-discuss to keep the community informed.

6.1.12 Weekly Kolla team meeting

The Kolla project meets every Wednesday in #openstack-kolla (OFTC).

More info on how to join, including the meeting time: https://meetings.opendev.org/#Kolla_Team_Meeting

We track CI, release, and feature status in the whiteboard.

Agenda

The regular agenda for the weekly meeting is as follows:

```
* Roll-call
* Agenda
* Announcements
* Review action items from the last meeting
* CI status
* Release tasks
* Regular stable releases (first meeting in a month)
* Current cycle planning
```

(continues on next page)

(continued from previous page)

```
* Additional agenda (from whiteboard)
* Open discussion
```

Anyone who would like to discuss an additional topic may post it on the whiteboard under Weekly meetings additional agenda.

Chairing a meeting

Copy/Paste into IRC to kick the meeting off:

```
#startmeeting kolla
```

Then, once the bot has caught up and everyone is settled:

```
#topic rollcall
```

Once folks have checked in, run the agenda by the group present:

```
#topic agenda
copy and paste agenda from above
```

Go through topics from agenda:

```
#topic something
```

And in the end finish the meeting:

```
#endmeeting
```

Copy/Paste for IRC:

```
https://bugs.launchpad.net/kolla
#link https://bugs.launchpad.net/kolla
```

Previous meetings

IRC logs

6.1.13 Continuous Integration

To make sure that changes do not break Kolla we use Continuous Integration (CI in short) on Opendev Zuul platform.

Distribution, architecture, build type coverage

There are several builds running on CI. We cover each supported distribution on x86-64 architecture and Debian/source builds on AArch64.

Allowed to fail

During Wallaby cycle we added support for allowed to fail images.

The `allowed-to-fail` option in `kolla-build.conf` file (generated by `tests/playbooks/run.yml` lists images which are allowed to fail during CI build without bringing whole build down.

Main use will be situation when we need to wait for other projects to fix problems blocking build of image.

Note

This is meant to be used on CI in emergency situation.

6.1.14 Versions of used components

Kolla project images cover several distributions on multiple architectures. Not all packages come from distribution repositories. Table below tries to provide information about those which come from 3rdparty sources.

For each component used we list version used at branch release and provide information about package sources.

Note

When table mentions CentOS it means both CentOS Stream 10 and Rocky Linux 10.

Name	Version	Package source information
Grafana	9.x	Grafana install guide
Kibana	7.x	Kibana install guide
Logstash	7.x	Logstash install guide
MariaDB	10.11 (LTS)	MariaDB Community downloads
Galera	26.4 (LTS)	MariaDB Community downloads
OpenSearch	3.x	OpenSearch install guide
ProxySQL	3.0.x	ProxySQL repository
Rabbitmq	4.1.x	• CentOS/Rocky: Team RabbitMQ Cloudsmith repo (RPM) • Debian/Ubuntu: Team RabbitMQ Cloudsmith repo (Deb)
Erlang	27.X	• CentOS/Rocky aarch64: openstack-kolla COPR • CentOS/Rocky x86-64: Team RabbitMQ Cloudsmith repo (RPM) • Debian/Ubuntu: Team RabbitMQ Modern Erlang PPA
Fluentd	6.x (LTS)	Fluentd install guide
Telegraf	1.24.x	InfluxDB upstream repo