kolla Documentation

Release 12.7.1.dev19

OpenStack Foundation

Jun 23, 2023

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.

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` is installed.

```
python3 -m pip install kolla
```

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
- rhel (deprecated)
- 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

When building images, there are two methods of the OpenStack install. One is `binary`. Another is `source`. The `binary` means that OpenStack will be installed from apt/dnf. And the `source` means that OpenStack will be installed from upstream sources. The default method of the OpenStack install is `binary`. It can be changed to `source` using the `-t` option:

```
kolla-build -t source
```

The locations of OpenStack source code are written in `kolla-build.conf`. The source 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.

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

```
[glance-base]
type = url
location = https://tarballs.openstack.org/glance/glance-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
```

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_redhat_binary_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.

RHEL containers and RHN

To build RHEL containers, it is necessary to include registration with RHN of the container runtime operating system. To obtain a RHN username/password/pool id, contact Red Hat. Use templates header block in the overrides file, e.g.:

```
{% extends parent_template %}

{% block base_header %}
RUN subscription-manager register --user=<user-name> \
  --password=<password> && subscription-manager attach --pool <pool-id>
{% endblock %}
```

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 binary install type build (among others):

- `openstack-dashboard`
- `openstack-magnum-ui`

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 `openstack-magnum-ui`, one would do:

```
{% extends parent_template %}

# Horizon
{% set horizon_packages_remove = ['openstack-magnum-ui'] %}
```

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 %}

{% 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`.

Note: The following is available for source build types only.

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 %}
```

Many of the Dockerfiles already copy the `plugins-archive` to the image and install available plugins at build time.

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.

Note: The following is available for source build types only.

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
```

Alternatively, it is also possible to create an `additions-archive.tar` file yourself bypasssing `kolla-build.conf` in order to work with binary build type.

The template becomes now:

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

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. See examples below.

To use current RDO packages (aka Delorean or DLRN), update `rpm_setup_config` in `kolla-build.conf`:

```
rpm_setup_config = https://trunk.rdoproject.org/centos8/current/delorean.
↪repo,https://trunk.rdoproject.org/centos8/delorean-deps.repo
```

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

```
rpm_setup_config = epel.repo,delorean.repo,delorean-deps.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.

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`.

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
        }
    ]
}
```

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_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)
- **KOLLA_INSTALL_TYPE**: `install_type` used to build the image (binary, source)
- **KOLLA_INSTALL_METATYPE**: `install_metatype` used to build the image (rdo,)

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
CentOS Stream 8	quay.io/centos/centos	stream8
Debian Bullseye	debian	bullseye
RHEL8 (deprecated)	registry.access.redhat.com/ubi8	latest
Ubuntu Focal	ubuntu	20.04

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
CentOS	CentOS Storage SIG	Nautilus
Ubuntu	Ubuntu Cloud Archive	Pacific
Debian	Debian	Nautilus

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

C - Community maintained

Coverage:

- supported by the broader community (not core team) or not supported at all

N - Not Available/Unknown

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

5.1.4 x86_64 images

Table 2: x86_64 images

Image	CentOS		Ubuntu		Debian	
	Binary	Source	Binary	Source	Binary	Source
aodh	C	C	C	C	C	C
barbican	C	T	C	C	C	C
bifrost	N	T	N	C	N	N
blazar	N	C	N	C	N	C
ceilometer	C	C	C	C	C	C
chrony (deprecated)	T	T	T	T	C	T
cinder	C	T	C	T	C	C
cloudkitty	C	C	N	C	N	C
collectd	C	C	C	C	C	C
cron	T	T	T	T	C	T
cyborg	N	C	N	C	N	C
designate	C	C	C	C	C	C
dnsmasq	T	T	C	T	C	C
elasticsearch	C	C	C	C	C	C
etcd	C	T	C	T	C	C
fluentd	T	T	T	T	C	T
freezer	N	C	N	C	N	C
glance	T	T	T	T	C	T
gnocchi	C	C	C	C	C	C
grafana	C	C	C	C	C	C
hacluster	C	C	C	C	C	C
hacluster-pcs	N	N	C	C	C	C
haproxy	T	T	T	T	C	C

continues on next page

Table 2 – continued from previous page

Image	CentOS		Ubuntu		Debian	
	Binary	Source	Binary	Source	Binary	Source
heat	T	T	N	T	C	T
horizon	T	T	T	T	C	T
influxdb	C	C	C	C	C	C
ironic	T	T	C	T	C	C
iscsid	T	T	T	T	C	C
kafka	C	C	C	C	C	C
keepalived	T	T	T	T	C	C
keystone	T	T	T	T	C	T
kibana	C	C	C	C	C	C
kolla-toolbox	T	T	T	T	C	T
kuryr	N	T	N	T	N	C
logstash	C	C	C	C	C	C
magnum	C	C	C	C	C	C
manila	C	C	C	C	C	C
mariadb	T	T	T	T	C	T
masakari	N	T	C	T	C	C
memcached	T	T	T	T	C	C
mistral	C	T	N	C	C	C
monasca	N	C	N	C	N	N
multipathd	C	C	C	C	C	C
murano	C	C	C	C	C	C
neutron	T	T	T	T	C	T
neutron-mlnx-agent	C	C	N	C	C	C
nova	T	T	T	T	C	T
nova-spicehtml5proxy	N	N	T	T	C	T
octavia	C	C	N	C	C	C
openvswitch	T	T	T	T	C	T
ovn	C	C	C	C	N	N
ovsdpdk	N	N	C	C	C	C
panko	C	C	C	C	C	C
placement	T	T	T	T	C	T
prometheus	C	C	C	C	C	C
qdrouterd	C	C	N	N	N	N
rabbitmq	T	T	T	T	C	T
rally (deprecated)	C	C	N	C	C	C
redis	C	T	C	C	C	C
sahara	C	C	C	C	C	C
senlin	C	C	C	C	C	C
skydive	C	C	C	C	C	C
solum	N	C	N	C	N	C
storm	C	C	C	C	C	C
swift	C	T	C	T	C	C
tacker	C	T	N	C	N	C
telegraf	C	C	C	C	C	N
tempest (deprecated)	C	C	C	C	C	C
tgtd	N	N	C	T	C	C

continues on next page

Table 2 – continued from previous page

Image	CentOS		Ubuntu		Debian	
	Binary	Source	Binary	Source	Binary	Source
trove	C	C	C	C	N	C
vitrage	C	C	N	C	C	C
vmtp	N	C	N	C	N	C
watcher	C	C	C	C	C	C
zookeeper	C	C	C	C	C	C
zun	N	T	N	T	N	C

5.1.5 aarch64 images

Table 3: aarch64 images

Image	CentOS		Ubuntu		Debian	
	Binary	Source	Binary	Source	Binary	Source
aodh	C	C	C	C	N	C
barbican	C	C	C	C	N	C
bifrost	N	C	N	N	N	N
blazar	N	C	N	C	N	C
ceilometer	C	C	C	C	N	C
chrony (deprecated)	C	C	C	C	N	C
cinder	C	C	C	C	N	C
cloudkitty	C	C	N	C	N	C
collectd	C	C	C	C	N	C
cron	C	C	C	C	N	C
cyborg	N	C	N	C	N	C
designate	C	C	C	C	N	C
dnsmasq	C	C	C	C	N	C
elasticsearch	N	N	C	C	N	C
etcd	C	C	C	C	N	C
fluentd	C	C	C	C	N	C
freezer	N	C	N	C	N	C
glance	C	C	C	C	N	C
gnocchi	C	C	C	C	N	C
grafana	C	C	C	C	N	C
hacluster	N	N	C	C	N	C
haproxy	C	C	C	C	N	C
heat	C	C	C	C	N	C
horizon	C	C	C	C	N	C
influxdb	N	N	C	C	N	C
ironic	C	C	C	C	N	C
iscsid	C	C	C	C	N	C
kafka	C	C	C	C	N	C
keepalived	C	C	C	C	N	C
keystone	C	C	C	C	N	C
kibana	N	N	N	N	N	C
kolla-toolbox	C	C	C	C	N	C

continues on next page

Table 3 – continued from previous page

Image	CentOS		Ubuntu		Debian	
	Binary	Source	Binary	Source	Binary	Source
kuryr	N	C	N	C	N	C
logstash	C	C	C	C	N	C
magnum	C	C	C	C	N	C
manila	C	C	C	C	N	C
mariadb	C	C	C	C	N	C
masakari	N	C	C	C	N	C
memcached	C	C	C	C	N	C
mistral	C	C	C	C	N	C
monasca	N	N	N	N	N	N
multipathd	C	C	C	C	N	C
murano	C	C	C	C	N	C
neutron	C	C	C	C	N	C
neutron-mlnx-agent	C	C	N	C	N	C
nova	C	C	C	C	N	C
nova-spicehtml5proxy	N	N	C	C	N	C
octavia	C	C	N	C	N	C
openvswitch	C	C	C	C	N	C
ovn	C	C	C	C	N	N
ovsdpdk	N	N	C	C	N	C
panko	C	C	C	C	N	C
placement	C	C	N	C	N	C
prometheus	C	C	C	C	N	C
qdrouterd	C	C	C	C	N	N
rabbitmq	C	C	C	C	N	C
rally (deprecated)	C	C	N	C	N	C
redis	C	C	C	C	N	C
sahara	C	C	C	C	N	C
senlin	C	C	C	C	N	C
skydive	N	N	N	N	N	N
solum	N	C	N	C	N	C
storm	C	C	C	C	N	C
swift	C	C	C	C	N	C
tacker	C	C	N	C	N	C
telegraf	N	N	N	N	N	N
tempest (deprecated)	C	N	C	N	N	N
tgtd	C	C	C	C	N	C
trove	C	C	N	C	N	C
vitrage	C	C	N	C	N	C
vmtp	N	C	N	C	N	C
watcher	C	C	C	C	N	C
zookeeper	C	C	C	C	N	C
zun	N	C	N	C	N	C

5.1.6 ppc64le images

Note: TODO

5.1.7 Currently unbuildable images

For a list of currently unbuildable images please look into `kolla/image/build.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 On Wednesdays at 15:00 UTC in the IRC channel (meetings logs)

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

Meeting Agenda https://wiki.openstack.org/wiki/Meetings/Kolla

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

The list in alphabetical order (on first name):

Name	IRC nick	Email address
Christian Berendt	berendt	berendt@betacloud-solutions.de
Dincer Celik	osmanlicilegi	hello@dincercelik.com
Eduardo Gonzalez	egonzalez	dabarren@gmail.com
Jeffrey Zhang	Jeffrey4l	jeffrey.zhang@99cloud.net
Marcin Juszkiewicz	hrw	marcin.juszkiewicz@linaro.org
Mark Goddard	mgoddard	mark@stackhpc.com
Micha Nasiadka	mnasiadka	mnasiadka@gmail.com
Radosaw Piliszek	yoctozepto	radoslaw.piliszek@gmail.com
Surya Prakash	spsurya	singh.surya64mnnit@gmail.com
Cao Yuan	caoyuan	cao.yuan@99cloud.net
wu.chunyang	wuchunyang	wuchunyang@yovole.com

The current effective list is also available from Gerrit: https://review.opendev.org/#/admin/groups/460,members

New Feature Planning

New features are discussed via IRC or mailing list (with [kolla] prefix). Kolla project keeps blueprints in Launchpad. Specs are welcome but not strictly required.

Task Tracking

Kolla project tracks tasks in Launchpad. Note this is the same place as for bugs.

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

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? You can do so on Launchpad. Note this is the same place as for tasks.

Getting Your Patch Merged

Most changes proposed to Kolla require two +2 votes from core reviewers before +W. 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.

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 Running Kolla Build in development

The recommended way to run in development

The preferred way to run kolla-build for development is using `tox`. Run the following from inside the repository:

```
tox -e venv -- kolla-build ...
```

The alternative way to run in development

Sometimes, developers prefer to manage their venvs themselves. This is also possible. Remember to install in editable mode (`-e`). Run the following from inside the repository:

```
python3 -m venv ~/path/to/venv
source ~/path/to/venv/bin/activate
python3 -m pip install -e .
kolla-build ...
```

6.1.3 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 %}
```

(continues on next page)

(continued from previous page)

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

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

<< binary specific steps >>

<< source specific steps >>

<< 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 (CentOS, Debian, Ubuntu) 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/build.py` source file we keep a list of images which cannot be built for some distribution/architecture/build-type combinations.

```
UNBUILDABLE_IMAGES = {
    'aarch64': {
        "bifrost-base",       # someone need to get upstream working first
    },

    'ppc64le': {
        "elasticsearch",      # no binary package
    },

    'binary': {
        "bifrost-base",
        "blazar-base",
    },

    'ubuntu': {
       "qdrouterd",   # There is no qdrouterd package for ubuntu bionic
    },

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

(continues on next page)

(continued from previous page)

```
    },
}
```

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.

6.1.4 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`.

6.1.5 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. Each change should include a release note unless being a `TrivialFix` change or affecting only docs or CI. Such changes should *not* include a release note to avoid confusion. 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 `git`s 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.
    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>`__
```

6.1.6 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.

6.1.7 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.

6.1.8 Bug triage

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

6.1.9 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)

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 priorites

- 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

6.1.10 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.

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.

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. stein-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

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.

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 typically freeze around **three weeks** after the common feature freeze. During this time, no features should be merged to the master branch.

Before RC1

Prior to creating a release candidate:

- test the code and fix (at a minimum) all critical bugs
- 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/
- 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
    ```

- update dependencies for source images on master to use release candidates:
  - `./tools/version-check.py --openstack-release $SERIES`
  - this will only work when release candidates have been created for the dependent projects
  - add `--include-independent` to update projects with an independent release cycle
  - example (kolla): https://review.opendev.org/647819
- update `OPENSTACK_RELEASE` variable in `kolla/common/config.py`
  - example (kolla): https://review.opendev.org/689729
- add cycle highlights when requested by the release team
  - example (all): https://review.opendev.org/644506/

RC1

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

- create RC1 by submitting patches to the releases repository
  - example (kolla): https://review.opendev.org/650236
  - example (kolla-ansible): https://review.opendev.org/650237
- create stable branches by submitting patches to the releases repository
  - example (kolla): https://review.opendev.org/650411
  - example (kolla-ansible): https://review.opendev.org/650412

Note: Use the new-release tool for those activities.

After RC1

- approve bot-proposed patches to master and the new stable branch
- revert the patch to use release candidates of dependencies on the master branch
  - example (kolla): https://review.opendev.org/650419
- revert the patch to switch OPENSTACK_RELEASE in kolla on the master branch
  - example (kolla): https://review.opendev.org/689731
- switch to use the new release of RDO on the new stable branch (master uses the delorean development packages)
  - example (kolla): https://review.opendev.org/651601
- switch to use the newly tagged container images (the branch for development mode on the new stable branch follows automatically since Victoria)
  - example (kolla-ansible): https://review.opendev.org/711214
- update previous release variables on master
  - example (kolla-ansible): https://review.opendev.org/650854
- search for TODOs/FIXMEs/NOTEs in the codebases describing tasks to be performed during the next 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).

After OpenStack Final Release

- update dependencies for source images on master to use final releases:
  - `./tools/version-check.py --openstack-release $SERIES`
  - example (kolla): https://review.opendev.org/651605/

RC2+

Further release candidates may be created on the stable branch as necessary in a similar manner to RC1.

Final Releases

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

- create final release by submitting patches to the releases repository
  - example (kolla): TODO
  - example (kolla-ansible): TODO
- ensure static links to documentation are enabled
  - https://opendev.org/openstack/openstack-manuals/src/branch/master/www/project-data
  - example for Ussuri: https://review.opendev.org/739206

Stable Releases

Stable branch releases should be made periodically for each supported stable branch, no less than once every 45 days.

- check for new releases of dependencies
  - `tools/version_check.py`
  - example (kolla): https://review.opendev.org/652674/
- create stable releases by submitting patches to the releases repository
  - follow SemVer guidelines
  - example (kolla): https://review.opendev.org/650411
  - example (kolla-ansible): https://review.opendev.org/650412
- mark milestones on Launchpad as released
- create new milestones on Launchpad for the next stable releases

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.

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 Dockerhub
- 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.

6.1.11 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.