Skyline APIServer Developer Documentation

Release 1.0.0.0rc2.dev5

Skyline APIServer contributors

Sep 30, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

OpenStack Skyline APIServer is the back-end server of Skyline. It provides RESTful APIs to Skyline Console.

Skyline is an OpenStack dashboard optimized by UI and UE, support OpenStack Train+. It has a modern technology stack and ecology, is easier for developers to maintain and operate by users, and has higher concurrency performance.

Skylines mascot is the nine-color deer. The nine-color deer comes from Dunhuang mural the nine-color king deer, whose moral is Buddhist cause-effect and gratefulness, which is consistent with 99clouds philosophy of embracing and feedback community since its inception. We also hope Skyline can keep light, elegant and powerful as the nine-color deer, to provide a better dashboard for the openstack community and users.

CHAPTER

TWO

USING SKYLINE APISERVER

How to use Skyline APIServer in your own projects.

2.1 Installation Guide

This section describes how to install and configure the skyline-apiserver.

2.1.1 System Requirements

System Requirements

Supported Operating Systems

Skyline APIServers source install supports the following host Operating Systems (OS):

- Ubuntu Focal (20.04)

2.1.2 Installing Guide

Skyline APIServer Installation Guide for Ubuntu

This section will guide you through the installation of the Skyline APIServer on Ubuntu 20.04 LTS.

Source Install Ubuntu

This section describes how to install and configure the Skyline APIServer service. Before you begin, you must have a ready OpenStack environment. At least it includes `keystone, glance, nova and neutron service`.

Prerequisites

Before you install and configure the Skyline APIServer service, you must create a database.

1. To create the database, complete these steps:
   1. Use the database access client to connect to the database server as the `root` user:

```
# mysql
```

   2. Create the `skyline` database:

```
MariaDB [(none)]> CREATE DATABASE skyline DEFAULT CHARACTER SET \
  utf8 DEFAULT COLLATE utf8_general_ci;
```

   3. Grant proper access to the `skyline` database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON skyline.* TO 'skyline'@
↪'localhost' \
  IDENTIFIED BY 'SKYLINE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON skyline.* TO 'skyline'@'%'␣
↪\
  IDENTIFIED BY 'SKYLINE_DBPASS';
```

      Replace `SKYLINE_DBPASS` with a suitable password.

   4. Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the service credentials, complete these steps:
   1. Create a `skyline` user:

```
$ openstack user create --domain default --password-prompt skyline

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 1qaz2wsx3edc4rfv5tgb6yhn7ujm8ikl |
| name                | skyline                          |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

   2. Add the `admin` role to the `skyline` user:

```
$ openstack role add --project service --user skyline admin
```

Note: This command provides no output.

Install and configure components

We will install the Skyline APIServer service from source code.

1. Git clone the repository from OpenDev (GitHub)

```
$ sudo apt update
$ sudo apt install -y git
$ cd ${HOME}
$ git clone https://opendev.org/openstack/skyline-apiserver.git
```

Note: If you meet the following error, you need to run command `sudo apt install -y ca-certificates`:

fatal: unable to access https://opendev.org/openstack/skyline-apiserver.git/: server certificate verification failed. CAfile: none CRLfile: none

2. Install skyline-apiserver from source

```
$ sudo apt install -y python3-pip
$ sudo pip3 install skyline-apiserver/
```

3. Ensure that some folders of skyline-apiserver have been created

```
$ sudo mkdir -p /etc/skyline /var/log/skyline
```

4. Copy the configuration file to the configuration folder `/etc/skyline`

```
$ sudo cp ${HOME}/skyline-apiserver/etc/gunicorn.py /etc/skyline/gunicorn.
↪py
$ sudo sed -i "s/^bind = *.*/bind = ['0.0.0.0:28000']/g" /etc/skyline/
↪gunicorn.py
$ sudo cp ${HOME}/skyline-apiserver/etc/skyline.yaml.sample /etc/skyline/
↪skyline.yaml
```

Note: We need to change the `bind` value in `/etc/skyline/gunicorn.py` to `0.0.0.0:28000`. Default value is `unix:/var/lib/skyline/skyline.sock`.

Note: Change the related configuration in `/etc/skyline/skyline.yaml`. Detailed introduction of the configuration can be found in *Settings Reference*.

```
default:
  database_url: mysql://skyline:SKYLINE_DBPASS@DB_SERVER:3306/skyline
  debug: true
```

(continues on next page)

(continued from previous page)

```
  log_dir: /var/log
openstack:
  keystone_url: http://KEYSTONE_SERVER:5000/v3/
  system_user_password: SKYLINE_SERVICE_PASSWORD
```

Replace SKYLINE_DBPASS, DB_SERVER, KEYSTONE_SERVER and SKYLINE_SERVICE_PASSWORD with a correct value.

5. Populate the Skyline APIServer database

```
$ cd ${HOME}/skyline-apiserver/
$ make db_sync
```

Finalize installation

1. Set start service config /etc/systemd/system/skyline-apiserver.service

```
[Unit]
Description=Skyline APIServer

[Service]
Type=simple
ExecStart=/usr/local/bin/gunicorn -c /etc/skyline/gunicorn.py skyline_
↪apiserver.main:app
LimitNOFILE=32768

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable skyline-apiserver
$ sudo systemctl start skyline-apiserver
```

Docker Install Ubuntu

This section describes how to install and configure the Skyline APIServer service. Before you begin, you must have a ready OpenStack environment. At least it includes keystone, glance, nova and neutron service.

Note: You have install the docker service on the host machine. You can follow the docker installation.

Prerequisites

Before you install and configure the Skyline APIServer service, you must create a database.

1. To create the database, complete these steps:
   1. Use the database access client to connect to the database server as the `root` user:

      ```
      # mysql
      ```

   2. Create the `skyline` database:

      ```
      MariaDB [(none)]> CREATE DATABASE skyline DEFAULT CHARACTER SET \
        utf8 DEFAULT COLLATE utf8_general_ci;
      ```

   3. Grant proper access to the `skyline` database:

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON skyline.* TO 'skyline'@
      ↪'localhost' \
        IDENTIFIED BY 'SKYLINE_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON skyline.* TO 'skyline'@'%'␣
      ↪\
        IDENTIFIED BY 'SKYLINE_DBPASS';
      ```

      Replace `SKYLINE_DBPASS` with a suitable password.

   4. Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   1. Create a `skyline` user:

      ```
      $ openstack user create --domain default --password-prompt skyline

      User Password:
      Repeat User Password:
      +---------------------+----------------------------------+
      | Field               | Value                            |
      +---------------------+----------------------------------+
      | domain_id           | default                          |
      | enabled             | True                             |
      | id                  | 1qaz2wsx3edc4rfv5tgb6yhn7ujm8ikl |
      | name                | skyline                          |
      | options             | {}                               |
      | password_expires_at | None                             |
      +---------------------+----------------------------------+
      ```

   2. Add the `admin` role to the `skyline` user:

      ```
      $ openstack role add --project service --user skyline admin
      ```

Note: This command provides no output.

Install and configure components

We will install the Skyline APIServer service from docker image.

1. Pull the Skyline APIServer service image from Docker Hub:

```
$ sudo docker pull 99cloud/skyline:latest
```

2. Ensure that some folders of skyline-apiserver have been created

```
$ sudo mkdir -p /etc/skyline /var/log/skyline /var/lib/skyline /var/log/
↪nginx
```

3. Set all value from *Settings Reference* into the configuration file `/etc/skyline/skyline.yaml`

Note: Change the related configuration in `/etc/skyline/skyline.yaml`. Detailed introduction of the configuration can be found in *Settings Reference*.

```
default:
  database_url: mysql://skyline:SKYLINE_DBPASS@DB_SERVER:3306/skyline
  debug: true
  log_dir: /var/log
openstack:
  keystone_url: http://KEYSTONE_SERVER:5000/v3/
  system_user_password: SKYLINE_SERVICE_PASSWORD
```

Replace `SKYLINE_DBPASS`, `DB_SERVER`, `KEYSTONE_SERVER` and `SKYLINE_SERVICE_PASSWORD` with a correct value.

Finalize installation

1. Run bootstrap server

```
$ sudo docker run -d --name skyline_bootstrap \
  -e KOLLA_BOOTSTRAP="" \
  -v /etc/skyline/skyline.yaml:/etc/skyline/skyline.yaml \
  -v /var/log:/var/log \
  --net=host 99cloud/skyline:latest
```

```
If you see the following message, it means that the bootstrap server is␣
↪successful:

+ echo '/usr/local/bin/gunicorn -c /etc/skyline/gunicorn.py skyline_
↪apiserver.main:app'
```

(continues on next page)

(continued from previous page)

```
+ mapfile -t CMD
++ xargs -n 1
++ tail /run_command
+ [[ -n 0 ]]
+ cd /skyline-apiserver/
+ make db_sync
alembic -c skyline_apiserver/db/alembic/alembic.ini upgrade head
2022-08-19 07:49:16.004 | INFO     | alembic.runtime.migration:__init__
↪:204 - Context impl MySQLImpl.
2022-08-19 07:49:16.005 | INFO     | alembic.runtime.migration:__init__
↪:207 - Will assume non-transactional DDL.
+ exit 0
```

2. Cleanup bootstrap server

```
$ sudo docker rm -f skyline_bootstrap
```

3. Run skyline-apiserver

```
$ sudo docker run -d --name skyline --restart=always \
  -v /etc/skyline/skyline.yaml:/etc/skyline/skyline.yaml \
  -v /var/log:/var/log \
  --net=host 99cloud/skyline:latest
```

Note: The skyline image is both include skyline-apiserver and skyline-console. And the skyline-apiserver is bound as socket file `/var/lib/skyline/skyline.sock`.

So you can not access the skyline-apiserver openapi swagger. But now you can visit the skyline UI `https://xxxxx:9999`.

Note: If you need to modify skyline port, add `-e LISTEN_ADDRESS=<ip:port>` in run command. Default port is 9999.

Verify Skyline APIServer operation

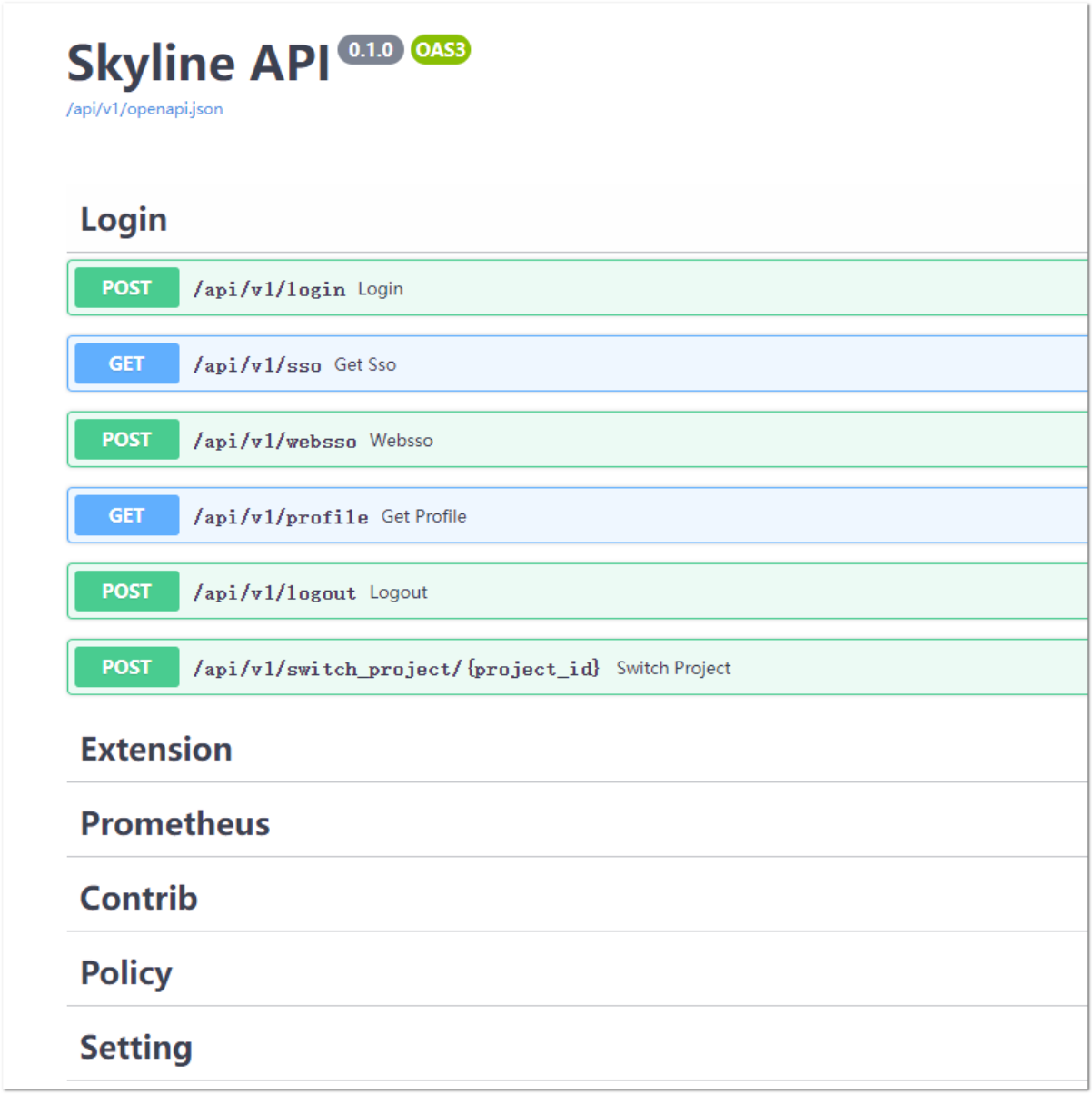
Verify operation of the Skyline APIServer service

Note: Only available when you use *Source Install Ubuntu*.

Note: Visit the OpenAPI swagger of Skyline APIServer.

---

1. Open a web browser and navigate to the Skyline APIServer OpenAPI swagger `http://xxx:28000/docs`:



2.2 Configuration Guide

2.2.1 Settings Reference

Skyline APIServer use `tox -e genconfig` to generate a sample configuration file `skyline.yaml.sample` in `etc` directory.

```
default:
  access_token_expire: 3600
  access_token_renew: 1800
  cors_allow_origins: []
  database_url: sqlite:////tmp/skyline.db
  debug: false
  log_dir: ./log
  prometheus_basic_auth_password: ''
```

(continues on next page)

(continued from previous page)

```
  prometheus_basic_auth_user: ''
  prometheus_enable_basic_auth: false
  prometheus_endpoint: http://localhost:9091
  secret_key: aCtmgbcUqYUy_HNVg5BDXCaeJgJQzHJXwqbXr0Nmb2o
  session_name: session
  ssl_enabled: true
openstack:
  base_domains:
  - heat_user_domain
  default_region: RegionOne
  enforce_new_defaults: true
  extension_mapping:
    floating-ip-port-forwarding: neutron_port_forwarding
    fwaas_v2: neutron_firewall
    qos: neutron_qos
    vpnaas: neutron_vpn
  interface_type: public
  keystone_url: http://localhost:5000/v3/
  nginx_prefix: /api/openstack
  reclaim_instance_interval: 604800
  service_mapping:
    baremetal: ironic
    compute: nova
    container: zun
    container-infra: magnum
    database: trove
    identity: keystone
    image: glance
    key-manager: barbican
    load-balancer: octavia
    network: neutron
    object-store: swift
    orchestration: heat
    placement: placement
    sharev2: manilav2
    volumev3: cinder
  sso_enabled: false
  sso_protocols:
  - openid
  sso_region: RegionOne
  system_admin_roles:
  - admin
  - system_admin
  system_project: service
  system_project_domain: Default
  system_reader_roles:
  - system_reader
  system_user_domain: Default
  system_user_name: skyline
```

(continues on next page)

(continued from previous page)

```
  system_user_password: ''
setting:
  base_settings:
  - flavor_families
  - gpu_models
  - usb_models
  flavor_families:
  - architecture: x86_architecture
    categories:
    - name: general_purpose
      properties: []
    - name: compute_optimized
      properties: []
    - name: memory_optimized
      properties: []
    - name: high_clock_speed
      properties: []
  - architecture: heterogeneous_computing
    categories:
    - name: compute_optimized_type_with_gpu
      properties: []
    - name: visualization_compute_optimized_type_with_gpu
      properties: []
  gpu_models:
  - nvidia_t4
  usb_models:
  - usb_c
```

CHAPTER

THREE

CONTRIBUTOR DOCS

3.1 Contributor Guide

In this section you will find information on how to contribute to skyline-apiserver. Content includes architectural overviews, tips and tricks for setting up a development environment.

3.1.1 Getting Started

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 the skyline-apiserver project, which is responsible for the following OpenStack deliverables:

skyline-apiserver

The OpenStack Modern Dashboard - back-end.
code: https://opendev.org/openstack/skyline-apiserver
docs: https://docs.openstack.org/skyline-apiserver/latest/
Launchpad: https://launchpad.net/skyline-apiserver

Communication

IRC We use IRC *a lot*. You will, too. You can find infomation about what IRC network OpenStack uses for communication (and tips for using IRC) in the Setup IRC section of the main *OpenStack Contributor Guide*.

People working on the Skyline APIServer project may be found in the `#openstack-skyline` IRC channel during working hours in their timezone. The channel is logged, so if you ask a question when no one is around, you can check the log to see if its been answered: http://eavesdrop.openstack.org/irclogs/%23openstack-skyline/

weekly meeting

> **Note:** Now we have not weekly meeting, we will have it in the future.

mailing list We use the openstack-discuss@lists.openstack.org mailing list for asynchronous discussions or to communicate with other OpenStack teams. Use the prefix `[skyline]` in your subject line (its a high-volume list, so most people use email filters).

More information about the mailing list, including how to subscribe and read the archives, can be found at: http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

Contacting the Core Team

The skyline-core team is an active group of contributors who are responsible for directing and maintaining the skyline-apiserver project. As a new contributor, your interaction with this group will be mostly through code reviews, because only members of skyline-core can approve a code change to be merged into the code repository.

You can learn more about the role of core reviewers in the OpenStack governance documentation: https://docs.openstack.org/contributors/common/governance.html#core-reviewer

The membership list of skyline-core is maintained in gerrit: https://review.opendev.org/admin/groups/1fe65032c39f1d459327b010730627a904d7b793,members

Project Team Lead

For each development cycle, Skyline APIServer project Active Technical Contributors (ATCs) elect a Project Team Lead who is responsible for running midcycles, and skyline-apiserver sessions at the Project Team Gathering for that cycle (and who is also ultimately responsible for everything else the project does).

- You automatically become an ATC by making a commit to one of the skyline-apiserver deliverables. Other people who havent made a commit, but have contributed to the project in other ways (for example, making good bug reports) may be recognized as extra-ATCs and obtain voting privileges. If you are such a person, contact the current PTL before the Extra-ATC freeze indicated on the current development cycle schedule (which you can find from the OpenStack Releases homepage .

The current Skyline APIServer project Project Team Lead (PTL) is listed in the Skyline APIServer project reference maintained by the OpenStack Technical Committee.

All common PTL duties are enumerated in the PTL guide.

New Feature Planning

The Skyline APIServer project uses blueprints to track new features. Heres a quick rundown of what they are and how the Skyline APIServer project uses them.

blueprints

Exist in Launchpad, where they can be targeted to release milestones.
You file one at https://blueprints.launchpad.net/skyline-apiserver

Examples of changes that can be covered by a blueprint only are:

- adding a new api

Feel free to ask in `#openstack-skyline` if you have an idea you want to develop and youre not sure whether it requires a blueprint *and* a spec or simply a blueprint.

The Skyline APIServer project observes the following deadlines. For the current development cycle, the dates of each (and a more detailed description) may be found on the release schedule, which you can find from: https://releases.openstack.org/

- bp freeze (all bps must be approved by this date)
- new feature status checkpoint

Task Tracking

We track our tasks in Launchpad. See the top of the page for the URL of Skyline APIServer project deliverable.

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 Bugs section.

When you start working on a bug, make sure you assign it to yourself. Otherwise someone else may also start working on it, and we dont want to duplicate efforts. Also, if you find a bug in the code and want to post a fix, make sure you file a bug (and assign it to yourself!) just in case someone else comes across the problem in the meantime.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so in the Launchpad space for the affected deliverable:

- skyline-apiserver: https://bugs.launchpad.net/skyline-apiserver

Getting Your Patch Merged

Before your patch can be merged, it must be *reviewed* and *approved*.

The Skyline APIServer project policy is that a patch must have two +2s before it can be merged. (Exceptions are documentation changes, which require only a single +2, for which the PTL may require more than two +2s, depending on the complexity of the proposal.) Only members of the skyline-core team can vote +2 (or -2) on a patch, or approve it.

Note: Although your contribution will require reviews by members of skyline-core, these arent the only people whose reviews matter. Anyone with a gerrit account can post reviews, so you can ask other developers you know to review your code and you can review theirs. (A good way to learn your way around the codebase is to review other peoples patches.)

If youre thinking, Im new at this, how can I possibly provide a helpful review?, take a look at How to Review Changes the OpenStack Way.

There are also some Skyline APIServer project specific reviewing guidelines in the *Code Reviews* section of the Skyline APIServer Contributor Guide.

In addition, some changes may require a release note. Any patch that changes functionality, adds functionality, or addresses a significant bug should have a release note. You can find more information about how to write a release note in the *Release notes* section of the Skyline APIServer Contributors Guide.

Note: Keep in mind that the best way to make sure your patches are reviewed in a timely manner is to review other peoples patches. Were engaged in a cooperative enterprise here.

If your patch has a -1 from Zuul, you should fix it right away, because people are unlikely to review a patch that is failing the CI system.

- If its a pep8 issue, the job leaves sufficient information for you to fix the problems yourself.
- If you are failing unit or functional tests, you should look at the failures carefully. These tests guard against regressions, so if your patch causing failures, you need to figure out exactly what is going on.
- The unit, functional, and pep8 tests can all be run locally before you submit your patch for review. By doing so, you can help conserve gate resources.

How long it may take for your review to get attention will depend on the current project priorities. For example, the feature freeze is at the third milestone of each development cycle, so feature patches have the highest priority just before M-3. These dates are clearly noted on the release schedule for the current release, which you can find from https://releases.openstack.org/

You can see whos been doing what with Skyline APIServer recently in Stackalytics: https://www.stackalytics.io/report/activity?module=skyline-group

Backporting a Fix

From time to time, you may find a bug thats been fixed in master, and youd like to have that fix in the release youre currently using (for example, Wallaby). What you want to do is propose a **backport** of the fix.

Note: The Skyline APIServer project observes the OpenStack Stable Branch Policy. Thus, not every change in master is backportable to the stable branches. In particular, features are *never* backportable. A really complicated bugfix may not be backportable if what it fixes is low-occurrence and theres a high risk that it may cause a regression elsewhere in the software.

How can you tell? Ask in the `#openstack-skyline` channel on IRC.

Since we use git for source code version control, backporting is done by *cherry-picking* a change that has already been merged into one branch into another branch. The gerrit web interface makes it really easy to do this. In fact, maybe *too* easy. Here are some guidelines:

- Before you cherry-pick a change, make sure it has already **merged** to master. If the change hasnt merged yet, it may require further revision, and the commit youve cherry-picked wont be the correct commit to backport.
- Backports must be done in *reverse chronological order*. Since OpenStack releases are named alphabetically, this means reverse alphabetical order: `stable/yoga`, `stable/xena`, etc.

- The cherry-pick must have **merged** into the closest most recent branch before it will be considered for a branch, that is, a cherry-pick to `stable/xena` will **not** be considered until it has merged into `stable/yoga` first.
  - This is because sometimes a backport requires revision along the way. For example, different OpenStack releases support different versions of Python. So if a fix uses a language feature introduced in Python 3.8, it will merge just fine into current master (during zed development), but it will not pass unit tests in `stable/yoga` (which supports Python 3.6). Likewise, if you already cherry-picked the patch from master directly to `stable/xena`, it wont pass tests there either (because xena also supports Python 3.6).

    So its better to follow the policy and wait until the patch is merged into `stable/yoga` *before* you propose a backport to `stable/xena`.
- You can propose backports directly from git instead of using the gerrit web interface, but if you do, you must include the fact that its a cherry-pick in the commit message. Gerrit does this automatically for you *if you cherry-pick from a merged commit* (which is the only kind of commit you should cherry-pick from in Gerrit); git will do it for you if you use the `-x` flag when you do a manual cherry-pick.

  This will keep the history of this backport intact as it goes from branch to branch. We want this information to be in the commit message and to be accurate, because if the fix causes a regression (which is always possible), it will be helpful to the poor sucker who has to fix it to know where this code came from without digging through a bunch of git history.

If you have questions about any of this, or if you have a bug to fix that is only present in one of the stable branches, ask for advice in `#openstack-skyline` on IRC.

Backport CI Testing

Like all code changes, backports should undergo continuous integration testing. This is done automatically by Zuul for changes that affect the main skyline-apiserver code.

This shouldnt be a big deal because presumably youve done local testing with your backend to ensure that the code works as expected in a stable branch; were simply asking that this be documented on the backport.

Skyline Project Releases

The Skyline project follows the OpenStack 6 month development cycle, at the end of which a new stable branch is created from master, and master becomes the development branch for the next development cycle.

Because many OpenStack consumers dont move as quickly as OpenStack development, we backport appropriate bugfixes from master into the stable branches and create new releases for consumers to use for a while. See the Stable Branches section of the OpenStack Project Team Guide for details about the timelines.

What follows is information about the Skyline project and its releases.

Where Stuff Is

The Skyline Project Deliverables

https://governance.openstack.org/tc/reference/projects/skyline.html#deliverables

The Code Repositories

- https://opendev.org/openstack/skyline-apiserver
- https://opendev.org/openstack/skyline-console

All Skyline Project Releases

https://releases.openstack.org/teams/skyline.html

How Stuff Works

Releases from Master

Releases from **master** for *skyline-apiserver* follow the cycle-with-rc release model.

- The cycle-with-rc model describes projects that produce a single release at the end of the cycle, with one or more release candidates (RC) close to the end of the cycle and optional development milestone betas published on a per-project need.

For more information about the release models and deliverable types: https://releases.openstack.org/reference/release_models.html

Branching

All Skyline project deliverables follow the OpenStack stable branch policy. Briefly,

- The stable branches are intended to be a safe source of fixes for high impact bugs and security issues which have been fixed on master since a given release.
- Stable branches are cut from the last release of a given deliverable, at the end of the common 6-month development cycle.

While anyone may propose a release, releases must be approved by the OpenStack Release Managers.

Contributing Documentation to Skyline APIServer

This page provides guidance on how to provide documentation for those who may not have previously been active writing documentation for OpenStack.

Documentation Content

To keep the documentation consistent across projects, and to maintain quality, please follow the OpenStack Writing style guide.

Using RST

OpenStack documentation uses reStructuredText to write documentation. The files end with a `.rst` extension. The `.rst` files are then processed by Sphinx to build HTML based on the RST files.

Note: Files that are to be included using the `.. include::` directive in an RST file should use the `.inc` extension. If you instead use the `.rst` this will result in the RST file being processed twice during the build and cause Sphinx to generate a warning during the build.

---

reStructuredText is a powerful language for generating web pages. The documentation team has put together an RST conventions page with information and links related to RST.

Building Skyline APIServers Documentation

To build documentation the following command should be used:

```
tox -e docs
```

When building documentation it is important to also run docs.

Note: The tox documentation jobs (docs, releasenotes) are set up to treat Sphinx warnings as errors. This is because many Sphinx warnings result in improperly formatted pages being generated, so we prefer to fix those right now, instead of waiting for someone to report a docs bug.

During the documentation build a number of things happen:

- All of the RST files under `doc/source` are processed and built.
  - The openstackdocs theme is applied to all of the files so that they will look consistent with all the other OpenStack documentation.
  - The resulting HTML is put into `doc/build/html`.
- All of Skyline APIServers `.py` files are processed and the docstrings are used to generate the files under `doc/source/contributor/api`

After the build completes the results may be accessed via a web browser in the `doc/build/html` directory structure.

Review and Release Process

Documentation changes go through the same review process as all other changes.

Note: Reviewers can see the resulting web page output by clicking on `openstack-tox-docs` in the Zuul check table on the review, and then look for Artifacts > Docs preview site.

This is also true for the `build-openstack-releasenotes` check jobs.

Once a patch is approved it is immediately released to the docs.openstack.org website and can be seen under Skyline APIServers Documentation Page at https://docs.openstack.org/skyline-apiserver/latest. When a new release is cut a snapshot of that documentation will be kept at `https://docs.openstack.org/skyline-apiserver/<release>`. Changes from master can be backported to previous branches if necessary.

Finding something to contribute

If you are reading the documentation and notice something incorrect or undocumented, you can directly submit a patch following the advice set out below.

There are also documentation bugs that other people have noticed that you could address:

- https://bugs.launchpad.net/skyline-apiserver/+bugs?field.tag=doc

---

Note: If you dont see a bug listed, you can also try the tag docs or documentation. We tend to use doc as the appropriate tag, but occasionally a bug gets tagged with a variant.

3.1.2 Writing Release Notes

Please follow the format, it will make everyones life easier.

Release notes

The release notes for a patch should be included in the patch.

If the following applies to the patch, a release note is required:

- Upgrades
  - The deployer needs to take an action when upgrading
  - A new config option is added that the deployer should consider changing from the default
  - A configuration option is deprecated or removed
- Features
  - A new feature is implemented
  - Feature is deprecated or removed
  - Current behavior is changed

- Bugs
  - A security bug is fixed
  - A long-standing or important bug is fixed
- APIs
  - REST API changes

Reviewing release note content

Release notes are user facing. We expect operators to read them (and other people interested in seeing whats in a new release may read them, too). This makes a release note different from a commit message, which is aimed at other developers.

Keep this in mind as you review a release note. Also, since its user facing, something you would think of as a nit in a code comment (for example, bad punctuation or a misspelled word) is not really a nit in a release noteits something that needs to be corrected. This also applies to the format of the release note, which should follow the standards set out later in this document.

In summary, dont feel bad about giving a -1 for a nit in a release note. We dont want to have to go back and fix typos later, especially for a bugfix thats likely to be backported, which would require squashing the typo fix into the backport patch (which is something thats easy to forget). Thus we really want to get release notes right the first time.

Fixing a release note

Of course, even with careful writing and reviewing, a mistake can slip through that isnt noticed until after a release. If that happens, the patch to correct a release note must be proposed *directly to the stable branch in which the release note was introduced*. (Yes, this is completely different from how we handle bugs.)

This is because of how reno scans release notes and determines what release they go with. See Updating Stable Branch Release Notes in the *reno User Guide* for more information.

Bugs

For bug fixes, release notes must include the bug number in Launchpad with a link to it as a RST link.

Note the use of the past tense (Fixed) instead of the present tense (Fix). This is because although you are fixing the bug right now in the present, operators will be reading the release notes in the future (at the time of the release), at which time your bug fix will be a thing of the past.

Additionally, keep in mind that when your release note is published, it is mixed in with all the other release notes and wont obviously be connected to your patch. Thus, in order for it to make sense, you may need to repeat information that you already have in your commit message. Thats OK.

Creating the note

Skyline APIServer uses reno to generate release notes. Please read the docs for details. In summary, use

```
$ tox -e venv -- reno new <bug-,bp-,whatever>
```

Then edit the sample file that was created and push it with your change.

To see the results:

```
$ git commit  # Commit the change because reno scans git log.

$ tox -e releasenotes
```

Then look at the generated release notes files in releasenotes/build/html in your favorite browser.

3.1.3 Programming HowTos and Tutorials

Setting Up a Development Environment

This page describes how to setup a working Python development environment that can be used in developing skyline-apiserver on Ubuntu. These instructions assume youre already familiar with git. Refer to GettingTheCode for additional information.

Following these instructions will allow you to run the skyline-apiserver unit tests. Running skyline-apiserver is currently only supported on Linux(recommend Ubuntu 20.04).

Virtual environments

Skyline-apiserver development uses virtualenv to track and manage Python dependencies while in development and testing. This allows you to install all of the Python package dependencies in a virtual environment or virtualenv (a special subdirectory of your skyline-apiserver directory), instead of installing the packages at the system level.

Note: Virtualenv is useful for running the unit tests, but is not typically used for full integration testing or production usage.

Linux Systems

Install the prerequisite packages.

On Ubuntu20.04-64:

```
sudo apt-get install libssl-dev python3-pip libmysqlclient-dev libpq-dev␣
↪libffi-dev
```

To get a full python3 development environment, the two python3 packages need to be added to the list above:

```
python3-dev python3-pip
```

Getting the code

Grab the code:

```
git clone https://opendev.org/openstack/skyline-apiserver.git
cd skyline-apiserver
```

Running unit 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` and `test-requirements.txt` files, so the only package you install is `tox` itself:

```
sudo pip install tox
```

Run the unit tests by doing:

```
tox -e py38
```

Setup Your Local Development Env

1. Installing dependency packages

   ```
   tox -e venv
   ```

2. Set skyline.yaml config file

   ```
   cp etc/skyline.yaml.sample etc/skyline.yaml
   export OS_CONFIG_DIR=$(pwd)/etc
   ```

   Maybe you should change the params with your real environment as followed:

   ```
   - database_url
   - keystone_url
   - default_region
   - interface_type
   - system_project_domain
   - system_project
   - system_user_domain
   - system_user_name
   - system_user_password
   ```

3. Init skyline database

```
source .tox/venv/bin/activate
make db_sync
deactivate
```

4. Run skyline-apiserver

```
$ source .tox/venv/bin/activate
$ uvicorn --reload --reload-dir skyline_apiserver --port 28000 --log-
↪level debug skyline_apiserver.main:app

INFO:     Uvicorn running on http://127.0.0.1:28000 (Press CTRL+C to quit)
INFO:     Started reloader process [154033] using statreload
INFO:     Started server process [154037]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

You can now access the online API documentation: `http://127.0.0.1:28000/docs`.

Or, you can launch debugger with `.vscode/lauch.json` with vscode.

Contributing Your Work

Once your work is complete you may wish to contribute it to the project. Skyline-apiserver uses the Gerrit code review system. For information on how to submit your branch to Gerrit, see GerritWorkflow.

3.1.4 Other Resources

Code Reviews

Skyline APIServer follows the same Review guidelines outlined by the OpenStack community. This page provides additional information that is helpful for reviewers of patches to Skyline APIServer.

Gerrit

Skyline APIServer uses the Gerrit tool to review proposed code changes. The review site is https://review.opendev.org

Gerrit is a complete replacement for Github pull requests. *All Github pull requests to the Skyline APIServer repository will be ignored.*

See Quick Reference for information on quick reference for developers. See Getting Started for information on how to get started using Gerrit. See Development Workflow for more detailed information on how to work with Gerrit.

The Great Change

Skyline APIServer only needs to support Python 3 runtimes (in particular, 3.8). Our biggest interaction with the stable branches is backporting bugfixes, where in the ideal case, were just doing a simple cherry-pick of a commit from master to the stable branches. You can see that theres some tension here.

With that in mind, here are some guidelines for reviewers and developers that the Skyline APIServer community has agreed on during this phase where we want to write pure Python 3 but still must support Python 2 code.

Python 2 to Python 3 transition guidelines

- New features can use Python-3-only language constructs, but bugfixes likely to be backported should be more conservative and write for Python 2 compatibilty.

Unit Tests

Skyline APIServer requires unit tests with all patches that introduce a new branch or function in the code. Changes that do not come with a unit test change should be considered closely and usually returned to the submitter with a request for the addition of unit test.

CI Job rechecks

CI job runs may result in false negatives for a considerable number of causes:

- Network failures.
- Not enough resources on the job runner.
- Storage timeouts caused by the array running nightly maintenance jobs.
- External service failure: pypi, package repositories, etc.
- Non skyline-apiserver components spurious bugs.

And the list goes on and on.

When we detect one of these cases the normal procedure is to run a recheck writing a comment with `recheck` for core Zuul jobs.

These false negative have periods of time where they spike, for example when there are spurious failures, and a lot of rechecks are necessary until a valid result is posted by the CI job. And its in these periods of time where people acquire the tendency to blindly issue rechecks without looking at the errors reported by the jobs.

When these blind checks happen on real patch failures or with external services that are going to be out for a while, they lead to wasted resources as well as longer result times for patches in other projects.

The Skyline APIServer community has noticed this tendency and wants to fix it, so now it is strongly encouraged to avoid issuing naked rechecks and instead issue them with additional information to indicate that we have looked at the failure and confirmed it is unrelated to the patch.

Efficient Review Guidelines

This section will guide you through the best practices you can follow to do quality code reviews:

- **Failing Gate**: You can check for jobs like pep8, py38, functional etc that are generic to all the patches and look for possible failures in linting, unit test, functional test etc and provide feedback on fixing it. Usually its the authors responsibility to do a local run of tox and ensure they dont fail upstream but if something is failing on gate and the author is not be aware about how to fix it then we can provide valuable guidance on it.
- **Documentation**: Check whether the patch proposed requires documentation or not and ensure the proper documentation is added. If the proper documentation is added then the next step is to check the status of docs job if its failing or passing. If it passes, you can check how it looks in HTML as follows: Go to `openstack-tox-docs job` link -> `View Log` -> `docs` and go to the appropriate section for which the documentation is added. Rendering: We do have a job for checking failures related to document changes proposed (openstack-tox-docs) but we need to be aware that even if a document change passes all the syntactical rules, it still might not be logically correct i.e. after rendering it could be possible that the bullet points are not under the desired section or the spacing and indentation is not as desired. It is always good to check the final document after rendering in the docs job which might yield possible logical errors.
- **Readability**: Readability is a big factor as remembering the logic of every code path is not feasible and contributors change from time to time. We should adapt to writing readable code which is easy to follow and can be understood by anyone having knowledge about Python constructs and working of Skyline APIServer. Sometimes it happens that a logic can only be written in a complex way, in that case, its always good practice to add a comment describing the functionality. So, if a logic proposed is not readable, do ask/suggest a more readable version of it and if thats not feasible then asking for a comment that would explain it is also a valid review point.
- **Type Annotations**: There has been an ongoing effort to implement type annotations all across Skyline APIServer with the help of mypy tooling. Certain areas of code already adapt to mypy coding style and its good practice that new code merging into Skyline APIServer should also adapt to it. We, as reviewers, should ensure that new code proposed should include mypy constructs.
- **Downvoting reason**: It often happens that the reviewer adds a bunch of comments some of which they would like to be addressed (blocking) and some of them are good to have but not a hard requirement (non-blocking). Its a good practice for the reviewer to mention for which comments is the -1 valid so to make sure they are always addressed.
- **Testing**: Always check if the patch adds the associated unit, functional and tempest tests depending on the change.
- **Commit Message**: There are few things that we should make sure the commit message includes:

  1) Make sure the author clearly explains in the commit message why the code changes are necessary and how exactly the code changes fix the issue.

  2) It should have the appropriate tags (Eg: Closes-Bug, Related-Bug, Blueprint, Depends-On etc). For detailed information refer to external references in commit message.

  3) It should follow the guidelines of commit message length i.e. 50 characters for the summary line and 72 characters for the description. More information can be found at Summary of Git commit message structure.

  4) Sometimes it happens that the author updates the code but forgets to update the commit message leaving the commit describing the old changes. Verify that the commit message is updated as per

code changes.

- **Release Notes**: There are different cases where a releasenote is required like fixing a bug, adding a feature, changing areas affecting upgrade etc. You can refer to the Release notes section in our contributor docs for more information.
- **Ways of reviewing**: There are various ways you can go about reviewing a patch, following are some of the standard ways you can follow to provide valuable feedback on the patch:

  1) Testing it in local environment: The easiest way to check the correctness of a code change proposed is to reproduce the issue (steps should be in launchpad bug) and try the same steps after applying the patch to your environment and see if the provided code changes fix the issue. You can also go a little further to think of possible corner cases where an end user might possibly face issues again and provide the same feedback to cover those cases in the original change proposed.

  2) Optimization: If youre not aware about the code path the patch is fixing, you can still go ahead and provide valuable feedback about the python code if that can be optimized to improve maintainability or performance.

CHAPTER

FOUR

RELEASE NOTES

See https://docs.openstack.org/releasenotes/skyline-apiserver

4.1 Additional reference

Contents:

4.1.1 Glossary

This glossary offers a list of terms and definitions to define a vocabulary for Skyline APIServer concepts.

Schemas Provide a description of the data that is expected to be returned or request.