Skyline Console Developer Documentation

Release 1.0.0.0rc2.dev31

Skyline Console contributors

Sep 29, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

Skyline Console is one part of OpenStack Modern Dashboard, which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.

CHAPTER

TWO

USING SKYLINE CONSOLE

How to use Skyline Console in your own projects.

2.1 Installation Guide

This section describes how to install and configure Skyline.

2.1.1 System Requirements

System Requirements

Supported Operating Systems

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

- Ubuntu Focal (20.04)

2.1.2 Installing Guide

Skyline Installation Guide for Ubuntu

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

Source Install Ubuntu

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

For more information about skyline-apiserver installation, refer to the OpenStack Skyline APIServer Guide.

Prerequisites

1. Install system dependencies

```
sudo apt update
sudo apt install -y git python3-pip nginx make ssl-cert
sudo apt-get install libgtk2.0-0 libgtk-3-0 libgbm-dev libnotify-dev␣
↪libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth xvfb
```

2. Install nvm (version control system for nodejs)

```
wget -P /root/ --tries=10 --retry-connrefused --waitretry=60 --no-dns-
↪cache --no-cache  https://raw.githubusercontent.com/nvm-sh/nvm/master/
↪install.sh
bash /root/install.sh
. /root/.nvm/nvm.sh
```

3. Install nodejs

```
nvm install --lts=Erbium
nvm alias default lts/erbium
nvm use default
```

4. Install yarn

```
npm install -g yarn
```

Install and configure components

We will install the Skyline Console service from source code.

1. Git clone the repository from OpenDev (GitHub)

```
cd ${HOME}
git clone https://opendev.org/openstack/skyline-console.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-sonsole.git/: server certificate verification failed. CAfile: none CRLfile: none

---

2. Install skyline-console

```
cd ${HOME}/skyline-console
make package
sudo pip3 install --force-reinstall dist/skyline_console-*.whl
```

3. Ensure that skyline folders have been created

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

Note: Ensure that skyline.yaml file is available in /etc/skyline folder. For more information about skyline.yml, see OpenStack Skyline Settings.

4. Generate nginx configuration file

```
skyline-nginx-generator -o /etc/nginx/nginx.conf
sudo sed -i "s/server .* fail_timeout=0;/server 0.0.0.0:28000 fail_
↪timeout=0;/g" /etc/nginx/nginx.conf
```

Note: We need to change the `upstream skyline` value in `/etc/nginx/nginx.conf` to `0.0.0.0:28000`. Default value is `unix:/var/lib/skyline/skyline.sock`.

Finalize installation

Start nginx service

```
sudo systemctl start nginx.service
sudo systemctl enable nginx.service
```

Docker Install Ubuntu

This section describes how to install and configure Skyline 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 Skyline 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 Skyline service from docker image.

1. Pull Skyline service image from Docker Hub:

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

2. Ensure that some folders of skyline have been created

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

3. Configure `/etc/skyline/skyline.yaml` file

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

```
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'
+ 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

```
$ 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 you can visit the skyline UI `https://xxxxx:9999`.

2.2 Configuration Guide

2.2.1 Skyline Console Settings Reference

- Prepare a usable backend
  - Prepare an accessible backend, for example: `https://172.20.154.250`
  - Modify the corresponding configuration in `config/webpack.dev.js`:

```
if (API === 'mock' || API === 'dev') {
  devServer.proxy = {
    '/api': {
      target: 'https://172.20.154.250',
      changeOrigin: true,
      secure: false,
    },
  };
}
```

- Configure access host and port
  - Modify `devServer.host` and `devServer.port`
  - Modify the corresponding configuration in `config/webpack.dev.js`

```
const devServer = {
  host: '0.0.0.0',
  // host: 'localhost',
  port: 8088,
  contentBase: root('dist'),
  historyApiFallback: true,
  compress: true,
  hot: true,
  inline: true,
  disableHostCheck: true,
```

(continues on next page)

(continued from previous page)

```
  // progress: true
};
```

- Execute in the project root directory, which is the same level as `package.json`

```
yarn run dev
```

- Use the `host` and `port` configured in `config/webpack.dev.js` to access, such as `http://localhost:8088`
- The front-end real-time update environment used for development is done.

For more information about skyline configuration settings, see OpenStack Skyline Settings.

2.3 User Documentation

2.3.1 Openstack Dashboard

Console page

Home tab

Home tab shows user information and details of quota about Compute, Storage, Network, etc. And it provides a button to quickly jump to Instances, Volumes, Networks and Routers page.

Compute tab

- *Instances*: View, launch, delete, create a snapshot from, attach or detach interface to, attach or detach volume to, associate floating ip to, manage security group of, stop, pause, lock, shelve, suspend, reboot or soft reboot instances, modify instance tags, or connect to them through VNC.
- *Instance snapshots*: View, edit, delete instance snapshots, launch instances or create volumes from them.
- *Flavors*: View flavors.
- *Server Groups*: View, create or delete server groups.
- *Images*: View, create, edit, delete images, launch instances or create volumes from them.
- *Key Pairs*: View, create, edit, import, and delete key pairs.

Storage tab

- *Volumes*: View, create, edit, and delete volumes. Create volume snapshot, create volume backup, clone volume, extend volume or change volume type, attach or detach them to instance.
- *Volume Backups*: View, create, edit, and delete volume backups. Also, create volumes from them and restore backup.
- *Volume Snapshots*: View, create, edit, delete volume snapshots. And create volumes from them.

Network tab

- *Networks*: View, create, edit and delete networks.
- *Ports*: View, create, edit, delete ports and manage security group for ports.
- *Routers*: View, create, edit, delete and manage routers.
- *Floating IPs*: Allocate IP addresses or release them.
- *Topology*: View the network topology.
- *Security Groups*: View, create, edit, and delete security groups and security group rules.

User page

User center tab

User center shows the details of user, including Username, Email, Phone, Real Name and User ID.

Application credentials tab

Application credentials provide a way to delegate a user authorization to an application without sharing the user password authentication. This is a useful security measure, especially for situations where the user identification is provided by an external source, such as LDAP or a single-sign-on service. Instead of storing user passwords in config files, a user creates an application credential for a specific project, with all or a subset of the role assignments they have on that project, and then stores the application credential identifier and secret in the config file.

Administrator page

Home tab

Home tab shows basic platform information (the numbers of projects, users, nodes), virtual resource usage (CPU usages, memory usages), compute and network services status.

Compute tab

- *Instances*: View, stop, suspend, cold or live migrate, soft or hard reboot, and delete instances that belong to all projects. Also, view the log for an instance or access an instance through VNC.
- *Instance snapshots*: View, edit, delete instance snapshots.
- *Flavors*: View, create, edit, manage metadata for, and delete flavors.
- *Server Groups*: View, create or delete server groups.
- *Images*: View, create, edit, manage metadata for, and delete images.
- *Hypervisors*: View the hypervisor summary. Also, view and manage compute nodes.
- *Host Aggregates*: View, create, manage metadata for, edit and delete host aggregates. View the list of availability zones.

Storage tab

- *Volumes*: View, update status for, migrate and delete volumes.
- *Volume Backups*: View and delete restore backup.
- *Volume Snapshots*: View and delete volume snapshots.
- *Volume Types*: View, create, edit, encrypt, manage access for and delete volume types.
- *Storage Backends*: View storage backends.

Network tab

- *Networks*: View, create and delete networks.
- *Ports*: View and delete ports.
- *Routers*: View and delete routers.
- *Floating IPs*: Allocate IP addresses or release them.
- *Security Groups*: View and delete security groups.

Identity tab

- *Domains*: View, create, edit, enable, disable and delete domains.
- *Projects*: View, create, edit, enable, disable and delete projects. Also, manage users or user groups of projects, modify tags for them.
- *Users*: View, create, edit, enable, disable, delete users. And edit System Permission of users, update user password.
- *User Groups*: View, create, edit and delete user groups.
- *Roles*: View, create, edit and delete roles.

Global setting tab

- *System Info*: Use the following tabs to view the service information:
  - *Services*: View a list of the services.
  - *Compute Services*: View a list of all Compute services and enable or disable them.
  - *Network Agents*: View the network agents and enable or disable them.
  - *Block Storage Services*: View a list of all Block Storage services and enable or disable them.
  - *Orchestration Services*: View a list of all Orchestration services.
- *System Config*: View, edit and reset system config.
- *Metadata Definitions*: View, edit and delete system metadata definitions.

2.3.2 Create and manage networks (Network tab)

The OpenStack Networking service provides a scalable system for managing the network connectivity within an OpenStack cloud deployment. It handles the creation and management of a virtual networking infrastructure, including networks, switches, subnets, and routers. Advanced services such as firewalls or virtual private network (VPN) can also be used.

Networking in OpenStack is complex. This section provides the basic instructions for creating a network and a router. For detailed information about managing networks, refer to the OpenStack Networking Guide.

create a network

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Network* tab and click *Networks* category.
4. Click *Create Network*.
5. In the *Create Network* dialog box, specify the following values.

   Network Name: Specify a name to identify the network.

   Description: A human-readable description for the resource.

   Shared: Share the network with other projects. Non admin users are not allowed to set shared option.

   Available Zone: Select a availability zone for the network.

   Port Security Enabled: Select the port security status of the network.

   Create Subnet: Select this check box to create a subnet.

   You do not have to specify a subnet when you create a network, but if you do not specify a subnet, the network can not be attached to an instance.

   Subnet Name: Specify a name for the subnet.

   CIDR: Specify the IP address for the subnet.

IP Version: Select IPv4 or IPv6.

Gateway IP: Specify an IP address for a specific gateway. This parameter is optional.

Disable Gateway: Select this check box to disable a gateway IP address.

DHCP: Select this check box to enable DHCP.

Allocation Pools: Specify IP address pools.

DNS: Specify the DNS server.

Host Routes: Specify the IP address of host routes.

6. Click *OK*.

   The dashboard shows the network on the *Networks* tab.

create a router

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Network* tab and click *Routers* category.
4. Click *Create Router*.
5. In the *Create Router* dialog box, specify the following values.

   Name: Specify a name to identify the router.

   Open External Gateway: Select this check box to specify external gateway.

   External Gateway: Specify external gateway for the router.

   Click *OK*, and the new router is now displayed in the *Routers* tab.
6. To connect a private network to the newly created router, perform the following steps:

   A) On the *Routers* tab, select *More* of the router, click *connect Subnet*.

   C) In the *Connect Subnet* dialog box, select a *Network* and *Subnet*.
7. Click *OK*.

You have successfully created the router. You can view the new topology from the *Topology* tab.

create a port

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Network* tab and click *Ports* category.
4. Click *Create Virtual Adapter*.
5. In the *Create Virtual Adapter* dialog box, specify the following values.

   Name: Specify name to identify the port.

   Owned Network: Select a network attached to the port.

Owned Subnet: Select a subnet attached to the port.

If you specify both a subnet ID and an IP address, OpenStack tries to allocate the IP address on that subnet to the port.

If you specify only a subnet ID, OpenStack allocates an available IP from that subnet to the port.

Port Security: Select this check box to specify security group.

Security Group: Select a security groups applied to the port.

6. Click *OK*.

   The new port is now displayed in the *Ports* list.

create a fip

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Network* tab and click *Floating IPs* category.
4. Click *Allocate IP*.
5. In the *Allocate IP* dialog box, specify the following values.

   Network: Specify a network associated with the floating IP.

   Description: A human-readable description for the resource.

   Batch Allocate: Select this check box to specify the number of batch creation.

   Count: Specify the number of batch creation.

6. Click *OK*.

   The dashboard shows the floating ip on the *Floating IPs* tab.

create a security group

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Network* tab and click *Security Groups* category.
4. Click *Create Security Group*.
5. In the *Create Security Group* dialog box, specify *Name* and *Description*, click *OK* and the new security group is now displayed in the *Security Groups* list.

2.3.3 Launch and manage instances (Compute tab)

The OpenStack Compute service provides a way to provision compute instances (aka virtual servers). It supports creating virtual machines, baremetal servers (through the use of ironic), and has limited support for system containers. For detailed information, refer to the OpenStack Nova Guide.

Create a key pair

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Key Pairs* category.
4. Click *Create Keypair*.
5. In the *Create Volume* dialog box, select one of *Create Type* options:
   - *Create Keypair*: If you choose this option, enter a *Name*.
   - *Import Keypair*: If you choose this option, a new field for *Public Key* displays. Enter the *Name* of your key pair, copy the public key into the *Public Key* box.
6. Click *OK*.

   The Dashboard lists the key pair on the *Key Pairs* tab.

Launch an instance

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Instances* category.
4. Click *Create Instance*.
5. On the *Create Instance* page, enter the instance values.

   Available Zone: The availability zone from which to launch the server.

   Specification: Select a flavor for your server instance.

   Start Source: Select one of the following options:

   - Image: If choose this option, a new field for *Operating System* displays. You can select the image from the list. And enter the size of the volume used as *System Disk* of the instance.

     Note: click the *Deleted with the instance* option to delete the volume on deleting the instance.

   - Instance Snapshot: Using this option, you can boot from a volume snapshot and create a new volume by choosing *Instance Snapshot* from a list.
   - Bootable Volume: If you choose this option, a new field for *Bootable Volume* displays. You can select the volume from the list.

   Data Disk: The disks mounted on the instance.

6. Click *Next: Network Config*.

   You can choose *Networks*, *Ports* or a mix of both for the instance network config.

   Networks: Add a network to the instance. If you specify the networks, *Virtual LAN* and *Security Group* are required fields.

   Virtual LAN: Specify a subnet of the network and assign fixed IP address automatically or manually for the instance.

   Security Group: Security groups are a kind of cloud firewall that define which incoming network traffic is forwarded to instances.

   Ports: Activate the ports that you want to assign to the instance.

   Note: The port executes its own security group rules by default.

7. Click *Next: System Config*.

   Name: The server name.

   Login Type: Select one of the following options:

   - Keypair: If you choose this option, a new field for *Keypair* displays. The key pair allows you to SSH into your newly created instance. You can select an existing key pair, import a key pair, or generate a new key pair.
   - Password: Enter the *Login Password* and confirm it. And you can login to the instance by using password.

8. Click *Next: Confirm Config* and confirm your choice.

   The instance are created and you can wait for a few seconds to follow the changes of the instance list data or manually refresh the data to get the final display result.

Create an instance snapshot

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Instances* category.
4. Select an instance to create a snapshot from it.
5. In the *Action* column, select *Backups & Snapshots* and click *Create Snapshot*.
6. In the *Create Instance Snapshot* dialog box, enter a snapshot name.
7. Click *OK*.

   The dashboard shows the new instance snapshot in *Instance Snapshots* tab.

Control the state of an instance

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Instances* category.
4. Select the instance for which you want to change the state.
5. In the *Action* column of the instance, click *Instance Status* and select the status.

Allocate a floating IP address to an instance

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Instances* category.
4. In the *Action* column, select *Related Resources* and click *Associate Floating IP*.
5. In the *Associate Floating IP* dialog box, select *Instance IP* and *Floating Ip Address*.
6. Click *OK*.

Note: To disassociate an IP address from an instance, click the *Disassociate Floating Ip* button.

Upload an image

Images are used to create virtual machine instances within the cloud. For information about creating image files, see the OpenStack Glance Guide.

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Compute* tab and click *Images* category.
4. Click *Create Image*.
5. On the *Create Image* page, enter the following values:

   Name: The name of the image.

   Upload Type: Select one of the following options:

   - Upload File: If choose this option, click *Click to Upload* to upload the binary image data file.
   - File URL: If choose this option, enter the *File URL*.

   Format: Select the image format (for example, QCOW2) for the image.

   OS: Select the image operating system (for example, CentOS).

   OS Version: The image operating system version.

   OS Admin: The administrator name of image operating system. in general, administrator for Windows, root for Linux.

Min System Disk (GiB): Amount of disk space in GB that is required to boot the image.

Min Memory (GiB): Amount of Memory in GB that is required to boot the image.

Protected: Image protection for deletion.

Usage Type: Select usage type (for example, Common Server) for the image.

Description: A human-readable description for the resource.

6. Click *Confirm*.

   The image is queued to be uploaded. It might take some time before the status changes from Queued to Active.

2.3.4 Create and manage volumes (Storage tab)

A volume is a detachable block storage device similar to a USB hard drive. You can attach a volume to a running instance or detach a volume and attach it to another instance at any time. You can also create a snapshot from or delete a volume. Only administrative users can create volume types.

OpenStack Block Storage enables you to add extra block-level storage to your OpenStack Compute instances. For detailed information, refer to the OpenStack Cinder Guide.

Create a volume

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. Click *Create Volume*.
5. On the *Create Volume* page, specify the following values.

   Available Zone: Select a availability zone for the volume.

   Data Source Type: Select one of the following options:

   - Blank Volume: If you choose this option, a new field for *Volume Type* displays. You can select the volume type from the list. You can create an empty volume. An empty volume does not contain a file system or a partition table.
   - Image: If you choose this option, a new field for *Operating System* displays. You can select the image from the list.
   - Volume Snapshot: If you choose this option, a new field for *Volume Snapshot* displays. You can select the snapshot from the list.

   Volume Type: Specify a volume type to choose an appropriate storage back end.

   Capacity (GiB): Specify the size of the volume, in gibibytes (GiB).

   Name: Specify a name to identify the volume.

6. Click *Confirm*.

   You have successfully created the volume. You can view the volume from the *Volumes* tab.

Attach a volume to an instance

After you create one or more volumes, you can attach them to instances. You can attach a volume to one instance at a time.

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. Select the volume to add to an instance.
5. In the *Action* column, select *Instance Related* and click *Attach*.
6. In the *Attach* dialog box, select an instance.
7. Click *OK*.

   The dashboard shows the instance to which the volume is now attached and the device name.

You can view the status of a volume in the Volumes tab of the dashboard. The volume is either Available or In-Use.

Now you can log in to the instance and mount, format, and use the disk.

Detach a volume from an instance

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. In the *Action* column of the volume, select *Instance Related* and click *Detach*.
5. In the *Detach* dialog box, select an instance.
6. Click *OK*.

   A message indicates whether the action was successful.

Create a snapshot from a volume

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. Select a volume from which to create a snapshot.
5. In the *Action* column, select *Data Protection* and click *Create Snapshot*.
6. In the *Create Volume Snapshot* dialog box, enter a snapshot name.
7. Click *OK*.

   The dashboard shows the new volume snapshot in *Volume Snapshots* tab.

Edit a volume

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. In the *Action* column of the volume, click *Edit*.
5. In the *Edit* dialog box, update the name and description of the volume.
6. Click *OK*.

Note: You can extend a volume by using the *Extend Volume* option available in the *More* dropdown list and entering the new value for volume size.

Delete a volume

When you delete an instance, the data in its attached volumes is not deleted.

1. Log in to the dashboard.
2. Select the appropriate project from the Switch Project menu at the top left.
3. On the *Console* page, open the *Storage* tab and click *Volumes* category.
4. Select the check boxes for the volumes that you want to delete.
5. Click *Delete* and confirm your choice.

Note: If you select the *cascading deletion* check box, when the volume has snapshots, the associated snapshot will be automatically deleted first, and then the volume will be deleted, thereby improving the success rate of deleting the volume.

A message indicates whether the action was successful.

2.3.5 Supported Browsers

Skyline is primarily tested and supported on the latest version of Chrome.

2.4 Administration Guide

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

2.4.1 Manage instances (Compute tab)

As an administrative user, you can manage compute nodes. You can create and delete flavors, and you can create and manage host aggregates. For more information, refer to the *Launch and manage instances (Compute tab)*.

Manage compute hosts

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Compute* tab and click *Hypervisors* category.
3. Click *Compute Hosts* tab.

   On the *Compute Hosts* page, you can *Enable* or *Disable* the compute hosts.

Create a flavor

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Compute* tab and click *Flavors* category.
3. Click *Create Flavor*.
4. On the *Params Setting* page, specify the following values.

   Architecture: Select one of the following options:

   - X86 Architecture: If choose this option, you can select *General Purpose*, *Compute Optimized*, *Memory Optimized* or *High Clock Speed* as flavor *Type*.
   - Heterogeneous Computing: If choose this option, you can select *Compute Optimized Type with GPU* or *Visualization Compute Optimized Type with GPU* as flavor *Type*.

   Name: Enter the flavor name.

   CPU(Core): Enter the number of virtual CPUs to use.

   Ram Size (GiB): Enter the amount of RAM to use, in gigabytes.

   NUMA Nodes: Enter the number of Non-Uniform Memory Access nodes.
5. Click *Next: Access Type Setting*.
6. On the *Access Type Setting* page, specify the following values.

   Access Type: Select one of the following options:

   - *Public‘*: Select this option to make the flavor publicly visible.
   - *Access Control*: If choose this option, you can select projects from *Access Control* list to determine which projects are visible to the flavor.
7. Click *Confirm*.

   The new flavor is now displayed in the *Flavors* list.

Delete a flavor

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Compute* tab and click *Flavors* category.
3. Select the check boxes for the flavors that you want to delete.
4. Click *Delete* and confirm your choice.

   A message indicates whether the action was successful.

Create a host aggregate

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Compute* tab and click *Host Aggregates* category.
3. Click *Host Aggregate* tab.
4. Click *Create Host Aggregate*.
5. In the *Create Host Aggregate* dialog box, specify the following values.
   - *Name*: The host aggregate name.
   - *Create new AZ*:
     - If set this option to *Yes*, specify *New Availability Zone* name to create new availability zone.
     - If set this option to *No*, select a *Availability Zone* of the host aggregate.
6. Click *OK*.

Manage host aggregates

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Compute* tab and click *Host Aggregates* category.
3. Click *Host Aggregate* tab.
4. In the *Action* column of the host aggregate, click *Manage Host*.
5. In the *Manage Host* dialog box, add hosts to the aggregate or remove hosts from it.
6. Click *OK*.

2.4.2 Manage volumes (Storage tab)

As an administrative user, you can manage volumes and volume types for users in various projects. You can create and delete volume types, and you can view and delete volumes. For more information, refer to the *Create and manage volumes (Storage tab)*.

Create a volume type

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Storage* tab and click *Volume Types* category.
3. Click *Create Volume Type*.
4. In the *Create Volume Type* dialog box, specify the following values.

   Name: Specify a name to identify the volume type.

   Description: A human-readable description for the resource.

   Shared: Shared volume can be mounted on multiple instances.

   Public: Select this check box to make the volume type publicly visible.

   Note: If you do not choose this check box, a new field for *Access Control* displays. You can select projects from the list to determine which projects are visible to the volume type.

   ---

5. Click *OK*.

   You have successfully created the volume type. You can view the volume type from the *Volume Types* tab.

Delete a volume type

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Storage* tab and click *Volume Types* category.
3. Select the check boxes for the volume types that you want to delete.
4. Click *Delete* and confirm your choice.

   A message indicates whether the action was successful.

2.4.3 Manage projects, users and roles (Identity tab)

OpenStack administrators can create projects, create accounts for new users and create roles.

A project is the base unit of resource ownership. Resources are owned by a specific project. A project is owned by a specific domain. A role is a personality that a user assumes to perform a specific set of operations. A role includes a set of rights and privileges. A user is an individual consumer that is owned by a domain. A role explicitly associates a user with projects or domains. A user with no assigned roles has no access to OpenStack resources.

OpenStack Identity Service is the module in the OpenStack framework that manages the authentication, service rules and service token functions. For detailed information, refer to the OpenStack Keystone Guide.

Create a role

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Roles* category.
3. Click *Create Role*.
4. In the *Create Role* dialog box, enter the role *Name* and *Description*.
5. Click *OK*.

   The new role is now displayed in the *Roles* list.

Edit a role

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Roles* category.
3. In the *Action* column of the role, click *Edit*.
4. In the *Edit* dialog box, update *Name* and *Description* of the role.
5. Click *OK*.

   A message indicates whether the action was successful.

Delete a role

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Roles* category.
3. Select the check boxes for the roles that you want to delete.
4. Click *Delete* and confirm your choice.

   A message indicates whether the action was successful.

Add a new project

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Projects* category.
3. Click *Create Project*.
4. In the *Create Project* dialog box, enter the Project *Name*, *Description*, *Status* and *Affiliated Domain*.
5. Click *OK*.

   The new project is now displayed in the *Projects* list.

Delete a project

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Projects* category.
3. Select the check boxes for the projects that you want to delete.
4. Click *Delete* and confirm your choice.

   A message indicates whether the action was successful.

Update a project

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Projects* category.
3. In the *Action* column of the project, click *Edit*.
4. In the *Edit* dialog box, update *Name* and *Description* of the project.
5. Click *OK*.

   A message indicates whether the action was successful.

Note: You can enable or disable the project by using the *Enable* or *Forbidden* options available in the *More* dropdown list.

Add a new user

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Users* category.
3. On the *Create User* page, enter the user *User Name*, *Password*, *Confirm Password*, *Email*, *Phone*, *Real Name* and *Status*.

   If you choose *Advanced Options*, new fields for *Select Project* and *Select User Group* display. You can assign role to user on project. You can also add user to group.
4. Click *Confirm*.

   The new user is now displayed in the *Users* list.

Delete a user

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Users* category.
3. Select the check boxes for the users that you want to delete.
4. Click *Delete* and confirm your choice.

   A message indicates whether the action was successful.

Update a user

1. Log into the OpenStack Dashboard as the Admin user.
2. On the *Administrator* page, open the *Identity* tab and click *Users* category.
3. In the *Action* column of the user, click *Edit*.
4. In the *Edit* dialog box, update *User Name*, *Description*, *Email*, *Phone* and *Real Name* of the user.
5. Click *OK*.

   A message indicates whether the action was successful.

   Note: You can enable or disable the user by using the *Enable* or *Forbidden* options available in the *More* dropdown list.

For more information about dashboard, see *Openstack Dashboard*.

To deploy skyline, see *Skyline Installation Guide for Ubuntu*.

To configure skyline, see OpenStack Skyline Settings.

CHAPTER

THREE

CONTRIBUTOR DOCS

3.1 Contributor Documentation

In this section you will find information on how to contribute to skyline-console. 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-console project, which is responsible for the following OpenStack deliverables:

skyline-console

The OpenStack Modern Dashboard - front-end.
code: https://opendev.org/openstack/skyline-console
docs: https://docs.openstack.org/skyline-console/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 Console 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-console 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 Console project Active Technical Contributors (ATCs) elect a Project Team Lead who is responsible for running midcycles, and skyline-console 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-console 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 Console project Project Team Lead (PTL) is listed in the Skyline Console project reference maintained by the OpenStack Technical Committee.

All common PTL duties are enumerated in the PTL guide.

New Feature Planning

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

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

Getting Your Patch Merged

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

The Skyline Console 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 Console project specific reviewing guidelines in the *Code Reviews* section of the Skyline Console 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 Console 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.

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 Console 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 Console 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-console 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-console* 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 Console

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 Consoles 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`.

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 Consoles Documentation Page at https://docs.openstack.org/skyline-console/latest. When

a new release is cut a snapshot of that documentation will be kept at `https://docs.openstack.org/skyline-console/<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 Console 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 development environment that can be used in developing skyline-console on Linux. These instructions assume youre already familiar with git. Refer to GettingTheCode for additional information.

Following these instructions will allow you to run the skyline-console unit tests.

Linux Systems

Install system dependencies

- *Ubuntu/Debian*

```
sudo apt-get install libgtk2.0-0 libgtk-3-0 libgbm-dev libnotify-
↪dev libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth xvfb
```

- *CentOS*

```
yum install -y xorg-x11-server-Xvfb gtk2-devel gtk3-devel␣
↪libnotify-devel GConf2 nss libXScrnSaver alsa-lib
```

Getting the code

Grab the code:

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

Setup Your Local Development Env

- Install nvm (version control system for nodejs)

```
wget -P /root/ --tries=10 --retry-connrefused --waitretry=60 --no-dns-
↪cache --no-cache  https://raw.githubusercontent.com/nvm-sh/nvm/master/
↪install.sh
bash /root/install.sh
. /root/.nvm/nvm.sh
```

- Install nodejs

```
nvm install --lts=Erbium
nvm alias default lts/erbium
nvm use default
```

- Verify nodejs and npm versions

```
node -v
# v12.*.*
npm -v
# 6.*.*
```

- Install yarn

```
npm install -g yarn
```

- Install the project dependency under the root directory, with `package.json` in the same place.

```
yarn install
```

  After those steps, please just wait until the installation is complete.

You can also use the following commands:

- `yarn run mock`: Use the mock interface of rap2
- `yarn run dev`: To use the actual interface, please change the http://pre.xxx.com in line 47 into the real address in file `webpack.dev.js`.
- `yarn run build`: Build packages and then you can hand over the contents of the generated *dist* directory to the back end.

For more information about configuration, see *Skyline Console Settings Reference*.

Running tests

- e2e tests

```
yarn run test:e2e
```

- unit tests

```
yarn run test:unit
```

Contributing Your Work

Once your work is complete you may wish to contribute it to the project. skyline-console 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 Console follows the same Review guidelines outlined by the OpenStack community. This page provides additional information that is helpful for reviewers of patches to Skyline Console.

Gerrit

Skyline Console 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 Console 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 Console has a modern technology stack and ecology, is easier for developers to maintain and operate by users, and has higher concurrency performance. And it focus on functional design and user experience. Embrace modern browser technology and ecology: React, Ant Design and Mobx. Use React component to process rendering, the page display process is fast and smooth, bringing users a better UI and UE experience.

Unit Tests

Skyline Console 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-console 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 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 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 JavaScript and working of Skyline Console. 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.
- **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 e2e 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.

3.2 Development Guide

This section describes how to develop the Skyline Console.

3.2.1 Ready To Work

For more information about installation, refer to the *Source Install Ubuntu*

Preparation before development

- Node environment
  - Requirement in package.json: `"node": ">=10.22.0"`
  - Verify nodejs version

```
node -v
```

- Yarn
  - Install yarn

```
npm install -g yarn
```

- Install dependencies
  - Execute in the project root directory, which is the same level as `package.json`, and wait patiently for the installation to complete

```
yarn install
```

- Prepare a usable backend
  - Prepare an accessible backend, for example: `https://172.20.154.250`
  - Modify the corresponding configuration in `config/webpack.dev.js`:

```
if (API === 'mock' || API === 'dev') {
  devServer.proxy = {
    '/api': {
      target: 'https://172.20.154.250',
      changeOrigin: true,
      secure: false,
    },
  };
}
```

- Configure access host and port
  - Modify *devServer.host* and *devServer.port*
  - Modify the corresponding configuration in `config/webpack.dev.js`

```
const devServer = {
  host: '0.0.0.0',
  // host: 'localhost',
  port: 8088,
  contentBase: root('dist'),
  historyApiFallback: true,
  compress: true,
  hot: true,
  inline: true,
  disableHostCheck: true,
  // progress: true
};
```

- Completed
  - Execute in the project root directory, which is the same level as `package.json`

```
yarn run dev
```

  - Use the *host* and *port* configured in `config/webpack.dev.js` to access, such as `http://localhost:8088`
  - The front-end real-time update environment used for development is done.

Front-end package used in production environment

Have the required `nodejs` and `yarn`

Execute in the project root directory, which is the same level as `package.json`

```
yarn run build
```

The packaged files are in the `dist` directory and handed over to the deployment personnel.

Front-end package used for testing

Have the required `nodejs` and `yarn`

Execute in the project root directory, which is the same level as `package.json`

```
yarn run build:test
```

The packaged files are in the `dist` directory

Note: This test package is designed to measure code coverage

It is recommended to use nginx to complete the E2E test with code coverage

---

3.2.2 Catalog Introduction

Introduction to the first-level directory

- *Gruntfile.js*: Used to collect i18n
- *LICENSE*: This project uses Apache License
- *Makefile*
- *README.rst*: A brief description of the front-end startup, please refer to the docs for details
- *config*: webpack configuration, which contains webpack configuration in public, development environment, test environment, and build environment
- *cypress.json*: E2E test configuration file
- *docker*: Contains the docker configuration used in the development environment, generation environment, and test environment
- *docs*: Documentation introduction, including Chinese, English, development documentation, testing documentation
- *jest.config.js*: Unit test configuration file
- *jsconfig.json*: javascript code configuration file
- *package.json*: Configuration files such as installation packages and commands
- *yarn.lock*: The version lock file of the package
- *.babelrc*: Babel configuration file

- *.dockerignore*: File configuration ignored by docker
- *.eslintignore*: File configuration ignored by eslint
- *.eslint*: Eslint configuration
- *.gitignore*: File configuration ignored by git
- *.gitreview*: Gitreview configuration
- *.prettierignore*: File configuration ignored by prettier
- *.prettierrc*: Prettier configuration
- *src*: **The folder where the development code is located**
- *test*: **The folder where the test code is located, contains e2e test code and basic code for unit testing**
- *tools*: Other tools folder, containing git tools

Catalog Introduction-Image Version

```
.
 Gruntfile.js (Used to collect i18n)
 LICENSE
 Makefile
 README.rst
 config
ăă  theme.js
ăă  webpack.common.js
ăă  webpack.dev.js  (Webpack configuration used during development)
ăă  webpack.e2e.js  (The webpack configuration used during e2e testing can␣
↪generate a package for testing coverage)
ăă  webpack.prod.js (Webpack packaging configuration used by the generation␣
↪environment)
 cypress.json    (E2E configuration)
 docker
ăă  dev.dockerfile
ăă  nginx.conf
ăă  prod.dockerfile
ăă  test.dockerfile
 docs    (Documents)
 jest.config.js  (Unit testing configuration)
 jsconfig.json
 package.json
 src
ăă  api (Api summary, not used yet)
ăă  asset
ăă ăă  image   (Images placement)
ăă ăă  template
ăă ăă      index.html
ăă  components  (Public components)
ăă  containers
```

(continues on next page)

(continued from previous page)

```
ăă ăă  Action
ăă ăă ăă  ConfirmAction   (Confirmed action base class)
ăă ăă ăă  FormAction   (Single page action base class)
ăă ăă ăă  ModalAction   (Pop-up action base class)
ăă ăă ăă  StepAction   (Multi-step single-page action, for example: create a␣
↪cloud host)
ăă ăă ăă  index.jsx
ăă ăă  BaseDetail  (Detail page base class with detailed information)
ăă ăă  List    (The base class of the list page, for example: cloud host)
ăă ăă  TabDetail   (The base class of the detail page with tab switching, for␣
↪example: instance details)
ăă ăă  TabList (List page with tab switch)
ăă  core
ăă ăă  App.jsx
ăă ăă  i18n.js
ăă ăă  index.jsx   (Entry)
ăă ăă  routes.js   (Routing configuration by module)
ăă  layouts
ăă ăă  Base    (Layout used after login)
ăă ăă  Blank    (Blank layout)
ăă ăă  User    (Layout used for login)
ăă ăă  admin-menu.jsx  (Menu configuration used by the management platform)
ăă ăă  menu.jsx    (Menu configuration used by the console)
ăă  locales (Translation)
ăă ăă  en.json
ăă ăă  index.js
ăă ăă  zh.json
ăă  pages (The page-directory structure is assigned according to: menu item-
↪secondary menu, where the pages of the secondary menu are placed in the␣
↪containers folder)
ăă ăă  base
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  404 (404 page)
ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă  AdminOverview   (Management platform home page)
ăă ăă ăă ăă ăă  components
ăă ăă ăă ăă ăă ăă  ComputeService.jsx
ăă ăă ăă ăă ăă ăă  NetworkService.jsx
ăă ăă ăă ăă ăă ăă  PlatformInfo.jsx
ăă ăă ăă ăă ăă ăă  ResourceOverview.jsx
ăă ăă ăă ăă ăă ăă  VirtualResource.jsx
ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă  style.less
ăă ăă ăă ăă  Overview   (Console home page)
ăă ăă ăă ăă      components
ăă ăă ăă ăă     ăă  ProjectInfo.jsx
ăă ăă ăă ăă     ăă  QuotaOverview.jsx
ăă ăă ăă ăă     ăă  ResourceStatistic.jsx
```

(continues on next page)

(continued from previous page)

```
ăă ăă ăă ăă      index.jsx
ăă ăă ăă ăă      style.less
ăă ăă ăă  routes  (Routing configuration)
ăă ăă ăă      index.js
ăă ăă  compute
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  BareMetalNode   (Bare metal configuration)
ăă ăă ăă ăă  Flavor  (Instance type)
ăă ăă ăă ăă  HostAggregate   (Host Aggregate)
ăă ăă ăă ăă ăă  Aggregate   (Host Aggregate)
ăă ăă ăă ăă ăă  AvailabilityZone    (Availability zone)
ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă  Hypervisors (Hypervisors management)
ăă ăă ăă ăă ăă  ComputeHost (Compute host)
ăă ăă ăă ăă ăă  Hypervisor  (Hypervisor manager)
ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă  Image   (Image)
ăă ăă ăă ăă  Instance    (Instance)
ăă ăă ăă ăă ăă  Detail  (Detail page)
ăă ăă ăă ăă ăă ăă  BaseDetail  (Base info)
ăă ăă ăă ăă ăă ăă  SecurityGroup   (Security group)
ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă  actions (Actions)
ăă ăă ăă ăă ăă ăă  AssociateFip.jsx    (Associate fip ip)
ăă ăă ăă ăă ăă ăă  AttachInterface.jsx (Attach interface)
ăă ăă ăă ăă ăă ăă  AttachIsoVolume.jsx (Attach iso volume)
ăă ăă ăă ăă ăă ăă  AttachVolume.jsx (Attach volume)
ăă ăă ăă ăă ăă ăă  ChangePassword.jsx  (Change password)
ăă ăă ăă ăă ăă ăă  Console.jsx (Console)
ăă ăă ăă ăă ăă ăă  CreateImage.jsx (Create Image)
ăă ăă ăă ăă ăă ăă  CreateIronic    (Create ironic-Step-by-step Form)
ăă ăă ăă ăă ăă ăă ăă  BaseStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  ConfirmStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  NetworkStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  SystemStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  index.less
ăă ăă ăă ăă ăă ăă  CreateSnapshot.jsx  (Create snapshot)
ăă ăă ăă ăă ăă ăă  Delete.jsx  (Delete instance)
ăă ăă ăă ăă ăă ăă  DeleteIronic.jsx    (Delete ironic)
ăă ăă ăă ăă ăă ăă  DetachInterface.jsx (Detach interface)
ăă ăă ăă ăă ăă ăă  DetachIsoVolume.jsx (Detach iso volume)
ăă ăă ăă ăă ăă ăă  DetachVolume.jsx    (Detach volume)
ăă ăă ăă ăă ăă ăă  DisassociateFip.jsx (Disassociate fip iP)
```

(continues on next page)

(continued from previous page)

```
ăă ăă ăă ăă ăă ăă  Edit.jsx    (Edit instance)
ăă ăă ăă ăă ăă ăă  ExtendRootVolume.jsx    (Expand the root disk)
ăă ăă ăă ăă ăă ăă  LiveMigrate.jsx (Live migrate)
ăă ăă ăă ăă ăă ăă  Lock.jsx    (Lock instance)
ăă ăă ăă ăă ăă ăă  ManageSecurityGroup.jsx (Manage security group)
ăă ăă ăă ăă ăă ăă  Migrate.jsx (Migrate)
ăă ăă ăă ăă ăă ăă  Pause.jsx   (Pause instance)
ăă ăă ăă ăă ăă ăă  Reboot.jsx  (Reboot instance)
ăă ăă ăă ăă ăă ăă  Rebuild.jsx (Rebuild instance)
ăă ăă ăă ăă ăă ăă  RebuildSelect.jsx   (Select the image to rebuild the␣
↪instance)
ăă ăă ăă ăă ăă ăă  Resize.jsx  (Change configuration)
ăă ăă ăă ăă ăă ăă  ResizeOnline.jsx    (Modify configuration online)
ăă ăă ăă ăă ăă ăă  Resume.jsx  (Resume instance)
ăă ăă ăă ăă ăă ăă  Shelve.jsx  (Shelve instance)
ăă ăă ăă ăă ăă ăă  SoftDelete.jsx  (Soft delete instance)
ăă ăă ăă ăă ăă ăă  SoftReboot.jsx  (Soft reboot instance)
ăă ăă ăă ăă ăă ăă  Start.jsx   (Start instance)
ăă ăă ăă ăă ăă ăă  StepCreate  (Create a instance-step by step creation)
ăă ăă ăă ăă ăă ăă ăă  BaseStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  ConfirmStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  NetworkStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  SystemStep
ăă ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă ăă  index.less
ăă ăă ăă ăă ăă ăă  Stop.jsx    (Stop instance)
ăă ăă ăă ăă ăă ăă  Suspend.jsx (Suspend instance)
ăă ăă ăă ăă ăă ăă  Unlock.jsx  (Unlock instance)
ăă ăă ăă ăă ăă ăă  Unpause.jsx (Unpause instance)
ăă ăă ăă ăă ăă ăă  Unshelve.jsx    (Unshelve instance)
ăă ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă ăă  index.less
ăă ăă ăă ăă ăă  components  (Component)
ăă ăă ăă ăă ăă ăă  FlavorSelectTable.jsx
ăă ăă ăă ăă ăă ăă  index.less
ăă ăă ăă ăă ăă  index.jsx
ăă ăă ăă ăă ăă  index.less
ăă ăă ăă ăă  Keypair (Key pair)
ăă ăă ăă ăă  ServerGroup (Instance group)
ăă ăă ăă  routes  (Routing configuration under the compute menu)
ăă ăă ăă      index.js
ăă ăă  configuration   (Platform configuration)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  Metadata    (Metadata definition)
```

(continues on next page)

(continued from previous page)

```
ăă ăă ăă ăă  Setting (System configuration)
ăă ăă ăă ăă  SystemInfo  (System info)
ăă ăă ăă  routes  (Routing configuration under the platform configuration␣
↪menu)
ăă ăă ăă      index.js
ăă ăă  heat    (Resource orchestration)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  Stack   (Stack)
ăă ăă ăă  routes  (Routing configuration under the resource arrangement menu)
ăă ăă ăă      index.js
ăă ăă  identity    (Identity management)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  Domain  (Domain)
ăă ăă ăă ăă  Project (Project)
ăă ăă ăă ăă  Role    (Role)
ăă ăă ăă ăă  User    (User)
ăă ăă ăă ăă  UserGroup   (User group)
ăă ăă ăă  routes  (Routing configuration)
ăă ăă ăă      index.js
ăă ăă  management  (Operation and maintenance management)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  RecycleBin  (Recycle bin)
ăă ăă ăă  routes  (Routing configuration)
ăă ăă ăă      index.js
ăă ăă  network (Network)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  FloatingIp  (Floating ip)
ăă ăă ăă ăă  LoadBalancers   (Load balancing)
ăă ăă ăă ăă  Network (Network)
ăă ăă ăă ăă  QoSPolicy   (Qos policy)
ăă ăă ăă ăă  Router  (Routing)
ăă ăă ăă ăă  SecurityGroup   (Security group)
ăă ăă ăă ăă  Topology    (Network topology)
ăă ăă ăă ăă  VPN (VPN)
ăă ăă ăă ăă  VirtualAdapter  (Virtual Adapter)
ăă ăă ăă  routes  (Routing configuration)
ăă ăă ăă      index.js
ăă ăă  storage (Storage)
ăă ăă ăă  App.jsx
ăă ăă ăă  containers
ăă ăă ăă ăă  Backup  (Backup)
ăă ăă ăă ăă  Snapshot    (Volume snapshot)
ăă ăă ăă ăă  Storage (Storage backend)
ăă ăă ăă ăă  Volume  (Volume)
ăă ăă ăă ăă  VolumeType  (Volume type)
```

(continues on next page)

(continued from previous page)

```
ăă ăă ăă ăă      QosSpec (QoS)
ăă ăă ăă ăă      VolumeType  (Volume type)
ăă ăă ăă ăă      index.jsx
ăă ăă ăă  routes  ()
ăă ăă ăă      index.js
ăă ăă  user    (Login page)
ăă ăă      App.jsx
ăă ăă      containers
ăă ăă     ăă  ChangePassword  (Change password-according to system␣
↪configuration)
ăă ăă     ăă ăă  index.jsx
ăă ăă     ăă ăă  index.less
ăă ăă     ăă  Login   (Login)
ăă ăă     ăă      index.jsx
ăă ăă     ăă      index.less
ăă ăă      routes  (Routing configuration)
ăă ăă          index.js
ăă  resources   (Store the public functions and status of each resource used␣
↪by itself)
ăă  stores  (Data processing, divide folders by resource type)
ăă ăă  base-list.js    (Base class for list data)
ăă ăă  base.js (Base class for data manipulation)
ăă ăă  cinder
ăă ăă  glance
ăă ăă  heat
ăă ăă  ironic
ăă ăă  keystone
ăă ăă  neutron
ăă ăă  nova
ăă ăă  octavia
ăă ăă  overview-admin.js
ăă ăă  project.js
ăă ăă  root.js
ăă ăă  skyline
ăă  styles  (Public styles)
ăă ăă  base.less
ăă ăă  main.less
ăă ăă  reset.less
ăă ăă  variables.less
ăă  utils   (Public functions)
ăă      RouterConfig.jsx
ăă      constants.js
ăă      cookie.js
ăă      file.js
ăă      file.spec.js
ăă      index.js
ăă      index.test.js   (Unit testing)
ăă      local-storage.js
ăă      local-storage.spec.js   (Unit testing)
```

(continues on next page)

(continued from previous page)

```
ăă      request.js
ăă      table.jsx
ăă      time.js
ăă      time.spec.js
ăă      translate.js
ăă      translate.spec.js
ăă      validate.js
ăă      yaml.js
ăă      yaml.spec.js
 test
ăă  e2e (E2E testing)
ăă  unit (Unit testing)
 tools
ăă  git_config
ăă      commit_message.txt
 yarn.lock
```

3.3 Tests Guide

This section describes how to test the Skyline Console.

3.3.1 Ready To Work

We provide two test methods

- E2E test - Focus on function point testing - Can provide code coverage data - User `Cypress` frame - Test results are saved in a static page for easy preview
- Unit test - Focus on basic function testing - User `Jest` frame

E2E test

1. Set up E2E test environment

   Note: For more information about installation, refer to the *Source Install Ubuntu*

   E2E test environment has been successfully built in Centos and wsl2 of Windows

   1. node environment
      - requirement in package.json: `"node": ">=10.22.0"`
      - verify nodejs version

      ```
      node -v
      ```

   2. install yarn

```
npm install -g yarn
```

3. Install dependencies

   - Execute in the project root directory, which is the same level as `package.json`, and wait patiently for the installation to complete

```
yarn install
```

4. Install system dependencies

   - *Ubuntu/Debian*

```
sudo apt-get install libgtk2.0-0 libgtk-3-0 libgbm-dev libnotify-dev␣
↪libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth xvfb
```

   - *CentOS*

```
yum install -y xorg-x11-server-Xvfb gtk2-devel gtk3-devel libnotify-
↪devel GConf2 nss libXScrnSaver alsa-lib
```

5. Adjust the access path, account and other information

   E2E configuration files are stored in `test/e2e/config/config.yaml`, Configured in it:

   - *baseUrl*, test access path
   - *env*, environment variable
     - *switchToAdminProject*, Switch to the `admin` project after logging in
     - *username*, User name to access the console, a user with console operation permissions is required
     - *password*, Password to access the console
     - *usernameAdmin*, The user name to access the management platform, a user with the operation authority of the management platform is required
     - *passwordAdmin*, Password to access the management platform
   - *testFiles*, Test files list

   The configuration change can be completed by directly modifying the corresponding value in `config.yaml`

   You can also complete configuration changes through `local_config.yaml`

   - Copy `test/e2e/config/config.yaml` to `test/e2e/config/local_config.yaml`
   - Modify the corresponding variables in `local_config.yaml`
   - For the value of the variable, the priority is: `local_config.yaml` > `config.yaml`

2. Command line run E2E

```
yarn run test:e2e
```

3. GUI running E2E

```
yarn run test:e2e:open
```

4. E2E test results

   After the test run is over, visit `test/e2e/report/merge-report.html` to view

5. E2E Code coverage test results

   After the test run is over, visit `coverage/lcov-report/index.html` to view

Note: Code coverage, the front-end package corresponding to `baseUrl` that needs E2E access, is `dist` package with a detectable code coverage version

---

```
yarn run build:test
```

The file packaged in the above way is a front-end package with testable code coverage

Below, the nginx configuration for front-end access to the front-end package with code coverage function is given

```
server {
  listen 0.0.0.0:8088 default_server;

  root /path/to/skyline-console/dist;
  index index.html;
  server_name _;
  location / {
    try_files $uri $uri/ /index.html;
  }

  location /api {
    proxy_pass http://<backend_address>;
  }
}
```

Unit test

1. Command line run unit tests

```
yarn run test:unit
```

2. Unit test results

   You can view the running results directly in the command line console

3.3.2 Catalog Introduction

```
test
 e2e (E2E code storage location)
    config
       config.yaml (Part of the configuration when E2E running, mainly␣
↪configures the test case file list, login account and other information)
       local_config.yaml   (Part of the configuration when E2E running,␣
↪mainly configures the test case file list, login account and other␣
↪information, which is gitignore and has a higher priority than config.yaml)
    fixtures    (Store upload files, read files, etc. required during␣
↪operation)
       keypair (Test file read by key)
       metadata.json   (Test metadata read file)
       stack-content.yaml  (Files read by the test stack)
       stack-params.yaml   (Files read by the test stack)
    integration (Store unit test)
       pages   (Adjust the directory according to the webpage menu structure)
           compute (compute)
              aggregate.spec.js   (aggregate)
              baremetal.spec.js   (baremetal)
              flavor.spec.js  (instance flavor)
              hypervisor.spec.js  (hypervisor)
              image.spec.js   (image)
              instance.spec.js    (instance)
              ironic.spec.js  (ironic)
              keypair.spec.js (keypair)
              server-group.spec.js    (server group)
           configuration   (Platform configuration)
              metadata.spec.js   (metadata)
              system.spec.js  (system info)
           error.spec.js   (error page)
           heat    (heat)
              stack.spec.js   (stack)
           identity    (identity)
              domain.spec.js  (Domain)
              project.spec.js (Project)
              role.spec.js    (Role)
              user-group.spec.js  (User group)
              user.spec.js    (User)
           login.spec.js   (Login)
           management  (Operation management)
              recycle-bin.spec.js (Recycle)
           network (Network)
              floatingip.spec.js  (Floating ip)
              lb.spec.js  (Loadbalance)
              network.spec.js (Network)
              qos-policy.spec.js  (Qos policy)
              router.spec.js  (Router)
              security-group.spec.js  (Security group)
```

(continues on next page)

(continued from previous page)

```
            topology.spec.js    (Network topology)
            port.spec.js (Virtual Adapter)
            vpn.spec.js (VPN)
         storage (Storage)
             backup.spec.js  (Backup)
             qos.spec.js (QoS)
             snapshot.spec.js    (Volume snapshot)
             storage.spec.js (Storage)
             volume-type.spec.js (Volume type)
             volume.spec.js  (Volume)
   plugins (Cypress plugins)
      index.js    (Configured to read the configuration file, configured to␣
↪use the code coverage function)
   report  (Store E2E test report)
      merge-report.html   (The final test report that records the execution␣
↪of each use case)
      merge-report.json   (Summary of test results in the results directory)
   results (Store test result files)
   screenshots (Store a snapshot of the test error)
   support (When writing a test case, double-wrapped function)
      commands.js (Store login, logout and other operation functions)
      common.js   (Store base functions)
      constants.js    (Store the route of each resource)
      detail-commands.js  (Store the functions related to the resource␣
↪detail page, based on the framework, the operation of the detail page is␣
↪consistent)
      form-commands.js  (Stores form-related functions, based on the␣
↪framework, consistent with the operation of form items)
      index.js
      resource-commands.js    (Store functions related to resource␣
↪operations, such as creating instance, creating router, deleting resources,␣
↪etc.)
      table-commands.js   (Store the functions related to the resource list␣
↪based on the framework, and it has consistency in the operation of the lis)
   utils   (Store the read function for the configuration file)
       index.js
unit    (Unit test)
   local-storage-mock.js   ( Storage mock function in local)
   locales (Translation files used when testing internationalization)
      en-US.js
      zh-CN.js
   setup-tests.js  (setup uni test)
   svg-mock.js (Mock of image loading)
```

E2E test code, stored in the `test/e2e` directory

- Other global configurations of E2E are stored in `cypress.json`

The basic code of the unit test is stored in the `test/unit` directory

- Other global configuration of unit test, stored in `jest.config.js`

- The test code of the unit test is usually placed in the same directory as the file to be tested, and has a suffix of `test.js` or `spec.js`
  - case: `src/utils/index.js` and `src/utils/index.test.js`
  - case: `src/utils/local-storage.js` and `src/utils/local-storage.spec.js`

3.3.3 How To Edit E2E Case

For specific introduction and usage of Cypress, please refer to Official document.

Here we mainly give the E2E use cases corresponding to the resources in the front-end page of Skyline-console, and use function defined in `test/e2e/support`

The following is an introduction, taking the instance use case `test/e2e/integration/pages/compute/instance.spec.js` as an example

Generally, when testing the corresponding functions of a resource, follow the following order

1. Prepare relevant variables in text

- Required parameters when creating a resource, such as: name, password
- Required parameters when editing resources, such as: new name
- When creating an associated resource, the name of the associated resource, such as: network name, router name, volume name

```
const uuid = Cypress._.random(0, 1e6);
const name = `e2e-instance-${uuid}`;
const newname = `${name}-1`;
const password = 'passW0rd_1';
const volumeName = `e2e-instance-attach-volume-${uuid}`;
const networkName = `e2e-network-for-instance-${uuid}`;
const routerName = `e2e-router-for-instance-${uuid}`;
```

2. Login before operation

- If you are operating console resources, please use *cy.login*
- If you are operating administrator resource, please use *cy.loginAdmin*
- Generally, the variable *listUrl* is used in the *login* and *loginAdmin* functions, that is, directly access the page where the resource is located after logging in

```
beforeEach(() => {
  cy.login(listUrl);
});
```

3. Create associated resources

Create associated resources, use the resource creation function provided in `resource-commands.js`, take the test instance as an example

- Create a network for testing to create a instance, attach interface

```
cy.createNetwork({ name: networkName });
```

- Create router *cy.createRouter* to ensure that the floating IP is reachable when testing the associated floating IP
  - The router created in the following way will open the external network gateway and bind the subnet of the *networkName* network

```
cy.createRouter({ name: routerName, network: networkName });
```

- Create floating ip *cy.createFip*, Used to test associate floating ip

```
cy.createFip();
```

- Create volume *cy.createVolume* (Used to test attach volume)

```
cy.createVolume(volumeName);
```

4. Write cases

- Write cases for creating resources
- Write use cases for accessing resource details
- Write use cases corresponding to all operations of resources separately

  Generally, the use case of the *edit* operation is written later, and then the use case of the *delete* operation is written, so that you can test whether the editing is effective

5. delete associated resources

To delete associated resources, use the resource-deleting function provided in `resource-commands.js`, this is to make the resources in the test account as clean as possible after the test case is executed

- Delete Floating IP

```
cy.deleteAll('fip');
```

- Delete Router *routerName*

```
cy.deleteRouter(routerName, networkName);
```

- Delete Network *networkName*

```
cy.deleteAll('network', networkName);
```

- Delete Volume *volumeName*

```
cy.deleteAll('volume', volumeName);
```

- Delete all available volume

```
cy.deleteAllAvailableVolume();
```

CHAPTER

FOUR

RELEASE NOTES

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

CHAPTER

FIVE

INFORMATION

5.1 Glossary

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