Charmed OpenStack Upgrader

Canonical Group Ltd

Apr 02, 2026

CONTENTS

Charmed OpenStack Upgrader (**COU**) is an application (packaged as a snap) to upgrade a Canonical distribution of Charmed OpenStack in an automated and frictionless manner. The application detects the version of the running cloud and proposes an upgrade plan to the next available OpenStack release.

COU follows the steps defined in the charm-guide upgrades overview, and it supports the upgrades for the following OpenStack releases:

From	To
Focal/Ussuri	Focal/Victoria
Focal/Victoria	Focal/Wallaby
Focal/Wallaby	Focal/Xena
Focal/Xena	Focal/Yoga
Jammy/Yoga	Jammy/Zed
Jammy/Zed	Jammy/Antelope
Jammy/Antelope	Jammy/Bobcat
Jammy/Bobcat	Jammy/Caracal

Source code available on Github.

CHAPTER

ONE

IN THIS DOCUMENTATION

Get Started

Start here to prepare the environment and install the application

How-to guides

Step-by-step guides covering key operations and common tasks

Reference

Technical information - commands, environmental variables, and known issues

Explanation

Additional information - details of upgrade phases and scopes defined in **COU**

CHAPTER

TWO

PROJECT AND COMMUNITY

COU is a member of the Ubuntu family. It's an open source project that warmly welcomes community contributions, suggestions, fixes and constructive feedback.

- We follow the Ubuntu community Code of conduct
- Contribute to the project on GitHub (documentation contributions go under the **docs/** directory)
- GitHub is also our central hub for bug tracking and issue management

2.1 Get started

This section guides you through steps to prepare the environment and install the application.

2.1.1 Verify Access

To use **COU** you need to have access to a Charmed OpenStack cloud, which is deployed using Juju. **COU** uses Juju credentials to access the OpenStack cloud, so accessing the cloud with Juju is mandatory.

COU requires at minimum *write* permission to the target model (see User access levels for more information). To verify your current user's model-scoped access level, run the following Juju command and look for your OpenStack model.

```
juju models
```

2.1.2 Installation

Install the **COU** snap from the Snap Store:

```
sudo snap install charmed-openstack-upgrader
```

2.2 How-to guides

These How-to guides will cover a variety of operations that are possible with **COU**.

2.2.1 Plan an upgrade

The **plan** command is used to generate the upgrade plan; the result will be printed on STDOUT.

Plan for the whole cloud

To generate a plan for the entire OpenStack cloud, including both the **control-plane** and the **data-plane**, use:

```
cou plan
```

Output example

```
user@host:~$ cou plan
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
    Verify that all OpenStack applications are in idle state
    Back up MySQL databases
    Archive old database data on nova-cloud-controller
    Control Plane principal(s) upgrade plan
        Upgrade plan for 'keystone' to 'victoria'
            Upgrade software packages of 'keystone' from the current APT repositories
                Upgrade software packages on unit 'keystone/0'
            Refresh 'keystone' to the latest revision of 'ussuri/stable'
            Change charm config of 'keystone' 'action-managed-upgrade' to 'False'
            Upgrade 'keystone' to the new channel: 'victoria/stable'
            Change charm config of 'keystone' 'openstack-origin' to 'cloud:focal-victoria
↪'
            Wait for up to 2400s for model 'test_model' to reach the idle state
            Verify that the workload of 'keystone' has been upgraded on units: keystone/0
    Subordinate(s) upgrade plan
        Upgrade plan for 'keystone-ldap' to 'victoria'
            Refresh 'keystone-ldap' to the latest revision of 'ussuri/stable'
            Upgrade 'keystone-ldap' to the new channel: 'victoria/stable'
    Upgrading all applications deployed on machines with hypervisor.
        Upgrade plan for 'az-1' to 'victoria'
            Disable nova-compute scheduler from unit: 'nova-compute/0'
            Upgrade software packages of 'nova-compute' from the current APT repositories
                Upgrade software packages on unit 'nova-compute/0'
            Refresh 'nova-compute' to the latest revision of 'ussuri/stable'
            Change charm config of 'nova-compute' 'action-managed-upgrade' to 'True'
            Upgrade 'nova-compute' to the new channel: 'victoria/stable'
            Change charm config of 'nova-compute' 'source' to 'cloud:focal-victoria'
            Upgrade plan for units: nova-compute/0
                Upgrade plan for unit 'nova-compute/0'
                    Verify that unit 'nova-compute/0' has no VMs running
                    ├── Pause the unit: 'nova-compute/0'
                    ├── Upgrade the unit: 'nova-compute/0'
                    ├── Resume the unit: 'nova-compute/0'
            Enable nova-compute scheduler from unit: 'nova-compute/0'
            Wait for up to 2400s for model 'test_model' to reach the idle state
            Verify that the workload of 'nova-compute' has been upgraded on units: nova-
↪compute/0
    Remaining Data Plane principal(s) upgrade plan
        Upgrade plan for 'ceph-osd' to 'victoria'
```

(continues on next page)

(continued from previous page)

```
            Verify that all 'nova-compute' units has been upgraded
            Upgrade software packages of 'ceph-osd' from the current APT repositories
                Upgrade software packages on unit 'ceph-osd/0'
            Change charm config of 'ceph-osd' 'source' to 'cloud:focal-victoria'
            Wait for up to 300s for app 'ceph-osd' to reach the idle state
            Verify that the workload of 'ceph-osd' has been upgraded on units: ceph-osd/0
    Data Plane subordinate(s) upgrade plan
        Upgrade plan for 'ovn-chassis' to 'victoria'
            Refresh 'ovn-chassis' to the latest revision of '22.03/stable'
```

Plan for the control-plane

To generate a plan targeting only the **control-plane** applications use:

```
cou plan control-plane
```

Plan for the data-plane

To generate a plan targeting only the **data-plane** applications use:

```
cou plan data-plane
```

Note:

- It's essential to complete the upgrade of the **control-plane** components before being able to generate a plan for the **data-plane**.
- By default, if non-empty hypervisor are identified, they are going to be excluded from the planning and a warning message will be shown. See the *Plan for non-empty hypervisors* section for instructions on how to include them.

Plan for the hypervisors

To generate a plan targeting just the **hypervisors** use:

```
# plan for all empty hypervisors
cou plan hypervisors
```

It's also possible to target specific Juju **availability-zones** or **machines**:

```
# plan for hypervisors with machine ID 0 and 1 (unless they're hosting VMs)
cou plan hypervisors --machine "0, 1"

# plan for all empty hypervisors that are in zone-1
cou plan hypervisors --availability-zone=zone-1
```

Note:

- Those specific filters are mutually exclusive, meaning that it's not possible to use them together.
- Since **hypervisors** are part of the **data-plane**, they won't be upgraded unless the **control-plane** has already been upgraded.
- By default, if non-empty hypervisor are identified, they are going to be excluded from the planning and a warning message will be shown. See the *Plan for non-empty hypervisors* section for instructions on how to include them.

Plan for non-empty hypervisors

If it's necessary to plan for non-empty hypervisors, use the *–force* option. For example:

```
# plan for all data-plane applications, including hypervisors currently running instances
cou plan data-plane --force

# plan for all hypervisors, even if they are hosting running instances
cou plan hypervisors --force

# plan for hypervisors on machines 0 and 1, even if they are hosting running instances
cou plan hypervisors --machine "0, 1" --force

# plan for all hypervisors that are in zone-1, even if they are hosting running instances
cou plan hypervisors --availability-zone=zone-1 --force
```

2.2.2 Upgrade a cloud

Recommendations and guidelines for upgrading production clouds

1. Upgrades can be disruptive; perform them during a maintenance window
2. The **control-plane** must be upgraded before the **data-plane**. Simply use *cou upgrade control-plane* without subcommands to ensure the correct ordering of operations.
3. Forcing the upgrade of non-empty hypervisors will disrupt the connectivity of the VMs they are hosting; live-migrate instances away from hypervisors undergoing an upgrade if possible
4. Use the *hypervisors* subcommand to test the upgrade of a single *canary machine* before upgrading the rest of the **data-plane**.
5. If no issues are found after upgrading the *canary machine*, proceed with the upgrade.

Upgrade the whole cloud

To upgrade the entire OpenStack cloud, including both the **control-plane** and the **data-plane**, use:

```
cou upgrade
```

Upgrade the control-plane

To run an upgrade targeting only the **control-plane** applications use:

```
cou upgrade control-plane
```

Upgrade the data-plane

To run an upgrade targeting only the **data-plane** applications use:

```
cou upgrade data-plane
```

Note:

- It's essential to complete the upgrade of the **control-plane** components before being able to upgrade the **data-plane**.
- By default, if non-empty hypervisor are identified, they are going to be excluded from the upgrade and a warning message will be shown. See the *Upgrade non-empty hypervisors* section for instructions on how to include them.

Upgrade the hypervisors

To upgrade just the **hypervisors** use:

```
# upgrade for all empty hypervisors
cou upgrade hypervisors
```

It's also possible to target specific Juju **availability-zones** or **machines**:

```
# upgrade for hypervisors with machine ID 0 and 1 (unless they're hosting VMs)
cou upgrade hypervisors --machine "0, 1"

# upgrade for all empty hypervisors that are in zone-1
cou upgrade hypervisors --availability-zone=zone-1
```

Note:

- Those specific filters are mutually exclusive, meaning that it's not possible to use them together.
- Since **hypervisors** are part of the **data-plane**, they won't be upgraded unless the **control-plane** has already been upgraded.
- By default, if non-empty hypervisor are identified, they are going to be excluded from the upgrade and a warning message will be shown. See the *Upgrade non-empty hypervisors* section for instructions on how to include them.

Upgrade non-empty hypervisors

If it's necessary to upgrade non-empty hypervisors, use the *–force* option. For example:

```
# upgrade all data-plane applications, including hypervisors currently running instances
cou upgrade data-plane --force

# upgrade all hypervisors, even if they are hosting running instances
cou upgrade hypervisors --force

# upgrade hypervisors on machines 0 and 1, even if they are hosting running instances
cou upgrade hypervisors --machine "0, 1" --force

# upgrade all hypervisors that are in zone-1, even if they are hosting running instances
cou upgrade hypervisors --availability-zone=zone-1 --force
```

Note: This will disrupt connectivity for any running VM. Migrate them elsewhere before upgrading if this is undesirable.

Run interactive upgrades

By default, **COU** runs upgrade in an interactive mode, prompting the user to confirm each step.

Usage example

```
user@host:~$ cou upgrade
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211917.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
```

(continues on next page)

(continued from previous page)

```
    Verify that all OpenStack applications are in idle state
    Back up MySQL databases
    Archive old database data on nova-cloud-controller
    Control Plane principal(s) upgrade plan
    Upgrade plan for 'rabbitmq-server' to 'victoria'
        Upgrade software packages of 'rabbitmq-server' from the current APT repositories
            Upgrade software packages on unit 'rabbitmq-server/0'
            Upgrade software packages on unit 'rabbitmq-server/1'
            Upgrade software packages on unit 'rabbitmq-server/2'
        Upgrade 'rabbitmq-server' to the new channel: '3.9/stable'
        Change charm config of 'rabbitmq-server' 'source' to 'cloud:focal-victoria'
        Wait for up to 2400s for model 'test-model' to reach the idle state
        Verify that the workload of 'rabbitmq-server' has been upgraded
    ...
Would you like to start the upgrade? Continue (y/N): y
Running cloud upgrade...
Verify that all OpenStack applications are in idle state ✓
Back up MySQL databases ✓

Upgrade plan for 'rabbitmq-server' to 'victoria'
    Upgrade software packages of 'rabbitmq-server' from the current APT repositories
        Upgrade software packages on unit 'rabbitmq-server/0'
        Upgrade software packages on unit 'rabbitmq-server/1'
        Upgrade software packages on unit 'rabbitmq-server/2'
    Upgrade 'rabbitmq-server' to the new channel: '3.9/stable'
    Change charm config of 'rabbitmq-server' 'source' to 'cloud:focal-victoria'
    Wait for up to 2400s for model 'test-model' to reach the idle state
    Verify that the workload of 'rabbitmq-server' has been upgraded

Continue (y/n): y
Upgrade plan for 'rabbitmq-server' to 'victoria' ✓

Upgrade plan for 'keystone' to 'victoria'
        Upgrade software packages of 'keystone' from the current APT repositories
            Upgrade software packages on unit 'keystone/0'
            Upgrade software packages on unit 'keystone/1'
            Upgrade software packages on unit 'keystone/2'
        Upgrade 'keystone' to the new channel: 'victoria/stable'
        Change charm config of 'keystone' 'openstack-origin' to 'cloud:focal-victoria'
        Wait for up to 2400s for model 'test-model' to reach the idle state
        Verify that the workload of 'keystone' has been upgraded

Continue (y/n): y
Upgrade software packages of 'keystone' from the current APT repositories \

...  # apply each step
Upgrade completed.
```

Run non-interactive upgrades

COU provides a non-interactive mode which suppresses user prompts and automatically continues executing each planned step. This option allows **COU** to be used by scripts or during upgrade testing. A quiet mode switch is also offered, which suppresses all logs and only prints important information including the generated plan and critical messages like the completion of the upgrade.

Usage examples

Non-interactive mode:

```
user@host:~$ cou upgrade --auto-approve
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
...  # the generated plan
Running cloud upgrade...
Verify that all OpenStack applications are in idle state ✓
Back up MySQL databases ✓
Upgrade software packages of 'keystone' from the current APT repositories ✓
Upgrade 'keystone' to the new channel: 'victoria/stable' ✓
...
Upgrade completed.
```

Non-interactive and quiet mode:

```
user@host:~$ cou upgrade --auto-approve --quiet
```

```
Upgrade cloud from 'ussuri' to 'victoria'
    Verify that all OpenStack applications are in idle state
    Back up MySQL databases
    ...
Upgrade completed.
```

2.2.3 Target a different model or controller

The current active model will be used by default, but it's possible to select a different one, even on a different controller. There are two ways to choose the model you want to operate on.

Using CLI argument

```
cou plan --model <model-name>
cou plan --model <controller>:<model-name>
```

Using environment variables

Since **COU** is using python-libjuju, it's possible for some of the environment variables mentioned in the documented Juju environment variables to affect the behaviour of the program. For example, JUJU_DATA can be used to specify a different path for Juju configuration files.

```
JUJU_DATA=./my-remote-cloud-juju-data cou plan
```

2.2.4 Interrupt and resume an upgrade

Because a partially executed upgrade step can leave the cloud in an inconsistent state, **COU** ensures that upgrades can only be interrupted between steps. This approach allows upgrades to be safely stopped and resumed later on.

Abort

In interactive mode, the user must approve each step and has the option to interrupt the process at any prompt.

Usage example:

```
user@host:~$ cou upgrade
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
...
Running cloud upgrade...
Verify that all OpenStack applications are in idle state ✓
Back up MySQL databases ✓
Upgrade plan for 'keystone' to 'victoria'
    Upgrade software packages of 'keystone' from the current APT repositories
        Upgrade software packages on unit 'keystone/0'
        Upgrade software packages on unit 'keystone/1'
        Upgrade software packages on unit 'keystone/2'
    Upgrade 'keystone' to the new channel: 'victoria/stable'
    Change charm config of 'keystone' 'openstack-origin' to 'cloud:focal-victoria'
    Wait for up to 2400s for model 'test-model' to reach the idle state
    Verify that the workload of 'keystone' has been upgraded

Would you like to start the upgrade? Continue (y/N): n
```

SIGINT or SIGTERM signals

COU will exit upon receiving SIGINT or SIGTERM signals. If the upgrade is already in progress, one of two behaviours will occur. If SIGINT or SIGTERM is sent only once (e.g. by pressing *ctrl+c* once), currently running steps will be allowed to finish, but any further upgrade step will be cancelled. If **COU** receives two or more SIGINTs (e.g. by pressing *ctrl+c* multiple times) or SIGTERMs, the upgrade will be cancelled abruptly in a potentially unsafe way: currently running steps will be immediately stopped, and no further step will be executed. This is generally not recommended as the cloud may be left in an inconsistent state.

Exiting before running upgrade plan:

```
user@host:~$ cou upgrade - # ctrl+c is pressed while connecting to the controller
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connecting to 'default' model... ×
charmed-openstack-upgrader has been terminated
```

```
user@host:~$ cou upgrade # ctrl+c is pressed while the cloud is being analyzed
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connecting to 'default' model... ✓
Analyzing cloud... ×
charmed-openstack-upgrader has been terminated
```

Safe cancel:

```
user@host:~$ cou upgrade # ctrl+c is pressed once during the upgrade
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
...
Running cloud upgrade...
Canceling upgrade... (Press ctrl+c again to stop immediately) ✓
charmed-openstack-upgrader has been stopped safely
```

Unsafe cancel:

```
user@host:~$ cou upgrade # ctrl+c is pressed twice during the upgrade
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
...
Running cloud upgrade...
Canceling upgrade... (Press ctrl+c again to stop immediately) ×
charmed-openstack-upgrader has been terminated without waiting
```

2.2.5 Change model connection behaviour

There are three variables through which the connection behaviour to the Juju model can be tuned. This may be necessary if **COU** is run from behind a VPN or if the network is heavily used.

- **COU_TIMEOUT** - sets the timeout of retries for any calls by Model to libjuju. It's unit-less and the number represents the number of seconds. Defaults to 10 seconds.
- **COU_MODEL_RETRIES** - sets how many times to retry connecting to the Juju model before giving up. Defaults to 5.
- **COU_MODEL_RETRY_BACKOFF** - sets the number of seconds in between connection retry attempts (for example, a backoff of 10 with 3 retries would wait 10s, 20s, and 30s). It's unit-less and the number represents the number of seconds. Defaults to 2 seconds.

Usage example

```
COU_TIMEOUT=120 cou upgrade
```

2.2.6 Plan/Upgrade without some applications

> **Warning**
>
> Skipping the upgrade for some applications might result in the cloud being broken. You are strongly encouraged to verify if the applications you want to skip will still be compatible with the cloud after the cloud is upgraded.

> **Note**
>
> This option accepts any application name. You can provide more than one application either as comma-separated value, or by repeating the option. Duplicate names are ignored.

By default, COU plan and upgrade will generate upgrade plan and run the upgrade for all the applications supported by COU. However, it is possible that some applications in the existing deployment can be in the unsupported version for COU, and COU will not perform the upgrade unless the operator adjust the deployment until it meets the version requirement. Often time, due to technical reasons, adjusting the application version on production environment is not desired. COU offers the –skip-apps to allow the operator to skip upgrading applications that are known to be safe.

Usage examples

Plan without vault.

```
cou plan --skip-apps vault
```

Upgrade without vault.

```
cou upgrade --skip-apps vault
```

Multiple applications (comma-separated):

```
cou upgrade --skip-apps vault, gnocchi
```

Repeat the option:

```
cou upgrade --skip-apps vault --skip-apps gnocchi
```

2.2.7 Archive old data

By default, **COU** plans for and runs an archive step before proceeding to actual upgrade steps. This can be turned off with the `--no-archive` flag.

This archive step is a performance optimisation, moving data for soft deleted nova instances into a shadow table.

The archiving is run in batches. The default batch size is 1000. On some clouds, it may be desirable to reduce the batch size to reduce database load. The batch size can be configured with `archive-batch-size <size>` where `size` is a positive integer.

Usage examples

With a custom batch size:

```
user@host:~$ cou plan --archive-batch-size 200
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
    Verify that all OpenStack applications are in idle state
    Back up MySQL databases
    Archive old database data on nova-cloud-controller
    Control Plane principal(s) upgrade plan
    ...
```

Disabling the archive step:

```
user@host:~$ cou plan --no-archive
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
    Verify that all OpenStack applications are in idle state
    Back up MySQL databases
    Control Plane principal(s) upgrade plan
    ...
```

More information

- nova-cloud-controller charm actions
- nova-manage reference - see `archive_deleted_rows` subcommand
- OpenStack upgrade guide information on archiving old database data
- *Purge data on shadow table*
- *Nova data migration*

2.2.8 Change verbosity level

Using the **verbose** parameter enables the adjustment of verbosity levels. This parameter can be specified repeatedly, up to three times, to escalate the verbosity from the warning level up to the debug log level.

The default verbosity level is **warning**.

Usage examples

The info level.

```
cou upgrade -v
```

The debug level for all messages except **python-libjuju** and **websockets**.

```
cou upgrade -vv
```

The debug level for all messages including the **python-libjuju** and **websockets**.

```
cou upgrade -vvv
```

2.2.9 Purge data on shadow tables

The purge step is disable by default. When enabled, it will run before proceeding to actual upgrade steps. This can be enabled with the `--purge` flag.

This purge step is a performance optimisation, delete data from the shadow tables in nova database. The behaviour is equal to run juju action `purge` on nova-cloud-controller unit, which help to reduce the size of the database, make queries faster, backups efficiency, and follow the data retention policies.

The `--purge-before-date` flag is supported to delete the data older than the date provided. The date string format should be `YYYY-MM-DD[ HH:mm[:ss]]`. This flag is useful to retain recent data for debugging or data retention policies.

Usage examples

user@host:~$

```
:input cou plan --purge --purge-before-date 2000-01-02
```

More information

- nova-cloud-controller charm actions
- nova-manage reference - see *purge* subcommand
- *Archive old data*
- *Nova data migration*

2.2.10 Test on Juju OpenStack provider

This document explain how to setup the testing environment on top of the Juju OpenStack provider and how to run COU testing.

Bootstrap juju controller on openstack cloud

Follow the step on the official documentation to bootstrap a Juju Openstack controller

Deploy openstack with juju openstack-provider

Follow the step on STSStack Bundles to deploy the openstack cloud

```
git clone https://github.com/canonical/stsstack-bundles.git
cd openstack

# Some of the overlays may be missing here, check all the supported overlays with:
./generate-bundle.sh --list-overlays

# ussuri-focal is the lowest version that COU supports
./generate-bundle.sh --name some-juju-model-name -r ussuri -s focal --ovn --telemetry
```

(Optional) sshuttle as proxy

If your openstack environment is behind the vpn and you have the bastion server, you can use sshuttle:

```
sshuttle --ssh-cmd "ssh -i /home/myuser/.ssh/mykey" -v -r ubuntu@your-bastion-ip 10.5.0.
↪0/16
```

Install COU for testing

There are two ways to execute COU:

- Use snap
- Use Python

Using python is more useful to debugging the code base bugs. However Snap is a official way to run COU on production, and we need to make sure all the parts in snap is working fine.

(Optional) Prepare local python environment for COU

One way to execute COU is to run it in your python environment

```
virtualenv ./.venv
source .venv/bin/activate

# Install COU in editable mode
pip install -e .

# Verify cou is installed in the environment
which cou

# Execute cou in python environment
cou
```

(Optional) Install COU from local snap build

```
# Running snapcraft command in the project's root directory.
snapcraft

# Install local snap with --dangerous
sudo snap install ./LOCAL_SNAP_FILE --dangerous

# snap list command should show the COU
snap list
```

(Optional) Copy the JUJU_DATA

If you don't want to mix the configuration with your local.

```
export JUJU_DATA=./juju-data
```

Execute the COU

Run cou plan and review the steps generate by COU

```
cou plan
```

(Optional) Run upgrade to execute the upgrade steps

```
cou upgrade
```

(Optional) Tail COU log message

Use below script to tail all the log files, old and new created, in follow mode:

```
#!/bin/bash

DIRECTORY="/home/myuser/.local/share/cou/log/"
CHECK_INTERVAL=2  # Check for new files every 2 seconds
LOGFILE=".tailed_files.log"

```

(continues on next page)

(continued from previous page)

```
# Function to tail new files
tail_files() {
    for file in "$DIRECTORY"/*; do
        if [ -f "$file" ] && ! grep -q "$file" "$LOGFILE"; then
            echo "Tailing new file: $file"
            tail -F "$file" &
            echo "$file" >> "$LOGFILE"
        fi
    done
}

# Function to clean up logfile on exit
cleanup() {
    echo "Cleaning up..."
    rm -f "$LOGFILE"
    exit 0
}

# Set trap to clean up logfile on exit
trap cleanup EXIT

# Create or clear the log file
> "$LOGFILE"

# Initial tailing of existing files
tail_files

# Periodically check for new files and tail them
while true; do
    sleep "$CHECK_INTERVAL"
    tail_files
done
```

2.2.11 Watch COU logs

Below is the script to watch the COU logs when doing the development. This script watches every log file in the directory, even newly created files.

```
#!/bin/bash

# Change LOG_DIR to the target directory
DIRECTORY="$HOME/.local/share/cou/log/"
CHECK_INTERVAL=2  # Check for new files every 2 seconds
LOGFILE=".tailed_files.log"

# Function to tail new files
tail_files() {
    for file in "$DIRECTORY"/*; do
        if [ -f "$file" ] && ! grep -q "$file" "$LOGFILE"; then
            echo "Tailing new file: $file"
            tail -F "$file" &
            echo "$file" >> "$LOGFILE"
```

(continues on next page)

(continued from previous page)

```
        fi
    done
}

# Function to clean up logfile on exit
cleanup() {
    echo "Cleaning up..."
    rm -f "$LOGFILE"
    exit 0
}

# Set trap to clean up logfile on exit
trap cleanup EXIT

# Create or clear the log file
> "$LOGFILE"

# Initial tailing of existing files
tail_files

# Periodically check for new files and tail them
while true; do
    sleep "$CHECK_INTERVAL"
    tail_files
done
```

2.2.12 Backup database

By default, **COU** plans for and runs a backup step of the cloud database before proceeding to actual upgrade steps.

Find mysql backup file

The file is store under `$COU_DATA`, which is `/home/$USER/.local/share/cou` if `$USER` exists else current directory.

Plan/Upgrade without cloud database backup

To plan/upgrade without backup, this can be turned off with `--no-backup` flag.

Usage examples

Plan:

```
user@host:~$ cou plan --no-backup
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
Upgrade cloud from 'ussuri' to 'victoria'
    Verify that all OpenStack applications are in idle state
    # note that there's no backup step planned
    Archive old database data on nova-cloud-controller
    Control Plane principal(s) upgrade plan
```

(continues on next page)

(continued from previous page)

```
    Upgrade plan for 'rabbitmq-server' to 'victoria'
        Upgrade software packages of 'rabbitmq-server' from the current APT repositories
            Upgrade software packages on unit 'rabbitmq-server/0'
            Upgrade software packages on unit 'rabbitmq-server/1'
            Upgrade software packages on unit 'rabbitmq-server/2'
    ...
```

Upgrade:

```
user@host:~$ cou upgrade --no-backup
```

```
Full execution log: '/home/ubuntu/.local/share/cou/log/cou-20231215211717.log'
Connected to 'test-model' ✓
Analyzing cloud... ✓
Generating upgrade plan... ✓
...
Running cloud upgrade...
Verify that all OpenStack applications are in idle state ✓
# note that there's no backup step being executed

Upgrade plan for 'rabbitmq-server' to 'victoria'
    Upgrade software packages of 'rabbitmq-server' from the current APT repositories
        Upgrade software packages on unit 'rabbitmq-server/0'
        Upgrade software packages on unit 'rabbitmq-server/1'
        Upgrade software packages on unit 'rabbitmq-server/2'
    Upgrade 'rabbitmq-server' to the new channel: '3.9/stable'
    Change charm config of 'rabbitmq-server' 'source' to 'cloud:focal-victoria'
    Wait for up to 2400s for model 'test-model' to reach the idle state
    Verify that the workload of 'rabbitmq-server' has been upgraded

Continue (y/n): y
Upgrade plan for 'rabbitmq-server' to 'victoria' ✓

... # apply steps
Upgrade completed.
```

2.2.13 Using Landscape

In air-gapped environments, users can still leverage COU to perform OpenStack upgrades by configuring the **LANDSCAPE_MIRROR_URI** and **LANDSCAPE_APT_COMPONENT** variables to point to an internal APT mirror. COU automatically detects the currently deployed OpenStack release and derives the appropriate Ubuntu Cloud Archive pocket (e.g., focal-ussuri), ensuring compatibility between the Ubuntu release and OpenStack version. Based on this detection, COU populates the relevant charm configuration fields such as **openstack-origin** or **source** with the correct repository entry—for example, **deb http://example.com focal-ussuri main**—allowing the upgrade process to proceed without requiring external network access.

2.3 Reference

This section outlines the available commands, environmental variables, and known issues.

2.3.1 Commands

COU provides commands to specify the type of operation to perform, as well as commands to define the scope of the operation. The bash completion feature facilitates the usage of these commands, simplifying interaction with the tool.

Plan

The **plan** command will analyse the cloud and output a human-readable representation of the proposed upgrade plan. It does not require any interaction. Refer to the output below for the description of all available options.

```
user@host:~$ cou plan --help
```

```
Usage: cou plan [options]

Show the steps COU will take to upgrade the cloud to the next release.
If upgrade-group is unspecified, plan upgrade for the whole cloud.

Options:
  -h, --help            Show this help message and exit.
  --model MODEL_NAME    Set the model to operate on.
                        If not set, the currently active Juju model will be used.
  --backup, --no-backup
                        Include database backup step before cloud upgrade.
                        Default to enabling database backup.
  --force               Force the plan/upgrade of non-empty hypervisors.
  --verbose, -v         Increase logging verbosity in STDOUT.
                        Multiple 'v's yield progressively more detail (up to 3).
                        Note that by default the logfile will not include standard logs
                        from juju and websockets, as well as debug logs from all other
                        modules. To also include the debug level logs from juju and
                        websockets modules, use the maximum verbosity.
  --quiet, -q           Disable output in STDOUT.

Upgrade groups:
  {control-plane,data-plane,hypervisors}
                        Run 'cou plan <upgrade-group> -h' for more info about an upgrade␣
↪group.
    control-plane       Show the steps for upgrading the control-plane components.
    data-plane          Show the steps for upgrading all data-plane components.
                        This is possible only if control-plane has been fully upgraded,
                        otherwise an error will be thrown.
    hypervisors         Show the steps for upgrading machines with nova-compute and
                        colocated services. This is possible only if control-plane
                        has been fully upgraded, otherwise an error will be thrown.
```

By default, COU plans upgrade for the entire OpenStack cloud with *cou plan*. However, the upgrade process can be tailored to target a specific group through a sub-command for more granular control. For further details, please see the *Upgrade Groups* section.

Upgrade

The **upgrade** command will implicitly generate a plan before moving onto the actual upgrade phase. Refer to the output below for a description of all available options.

```
user@host:~$ cou upgrade --help
```

```
Usage: cou upgrade [options]

Run the cloud upgrade.
If upgrade-group is unspecified, upgrade the whole cloud.

Options:
  -h, --help            Show this help message and exit.
  --model MODEL_NAME    Set the model to operate on.
                        If not set, the currently active Juju model will be used.
  --backup, --no-backup
                        Include database backup step before cloud upgrade.
                        Default to enabling database backup.
  --force               Force the plan/upgrade of non-empty hypervisors.
  --verbose, -v         Increase logging verbosity in STDOUT.
                        Multiple 'v's yield progressively more detail (up to 3).
                        Note that by default the logfile will not include standard logs
                        from juju and websockets, as well as debug logs from all other
                        modules. To also include the debug level logs from juju and
                        websockets modules, use the maximum verbosity.
  --quiet, -q           Disable output in STDOUT.
  --auto-approve        Automatically approve and continue with each upgrade step␣
↪without prompt.

Upgrade group:
  {control-plane,data-plane,hypervisors}
                        Run 'cou upgrade <upgrade-group> -h' for more info about an␣
↪upgrade group
    control-plane       Run upgrade for the control-plane components.
    data-plane          Upgrade all data-plane components.
                        This is possible only if control-plane has been fully upgraded,
                        otherwise an error will be thrown.
    hypervisors         Upgrade machines with nova-compute and colocated services.
                        This is possible only if control-plane has been fully upgraded,
                        otherwise an error will be thrown.
```

By default COU upgrades the entire OpenStack cloud with *cou upgrade*. However, the upgrade process can be tailored to target a specific group through a sub-command for more granular control. For further details, please see the *Upgrade Groups* section.

Upgrade Groups

In COU, users can choose to selectively target only certain components in OpenStack cloud for planning and executing upgrades, grouped by their roles. The available upgrade groups are **control-plane**, **data-plane**, and **hypervisors**.

The options available for **control-plane** upgrade are:

```
user@host:~$ cou upgrade control-plane --help
```

```
Usage: cou upgrade control-plane [options]

Run upgrade for the control-plane components.

Options:
  -h, --help            Show this help message and exit.
```

(continues on next page)

(continued from previous page)

```
  --model MODEL_NAME    Set the model to operate on.
                        If not set, the currently active Juju model will be used.
  --backup, --no-backup
                        Include database backup step before cloud upgrade.
                        Default to enabling database backup.
  --force               Force the plan/upgrade of non-empty hypervisors.
  --verbose, -v         Increase logging verbosity in STDOUT.
                        Multiple 'v's yield progressively more detail (up to 3).
                        Note that by default the logfile will not include standard logs
                        from juju and websockets, as well as debug logs from all other
                        modules. To also include the debug level logs from juju and
                        websockets modules, use the maximum verbosity.
  --quiet, -q           Disable output in STDOUT.
  --auto-approve        Automatically approve and continue with each upgrade step␣
↪without prompt.
```

The available options for a **data-plane** upgrade align closely with those offered for a **control-plane** upgrade.

```
user@host:~$ cou upgrade data-plane --help
```

```
Usage: cou upgrade data-plane [options]

Upgrade all data-plane components.
This is possible only if control-plane has been fully upgraded,
otherwise an error will be thrown.

Options:
  -h, --help            Show this help message and exit.
  --model MODEL_NAME    Set the model to operate on.
                        If not set, the currently active Juju model will be used.
  --backup, --no-backup
                        Include database backup step before cloud upgrade.
                        Default to enabling database backup.
  --force               Force the plan/upgrade of non-empty hypervisors.
  --verbose, -v         Increase logging verbosity in STDOUT.
                        Multiple 'v's yield progressively more detail (up to 3).
                        Note that by default the logfile will not include standard logs
                        from juju and websockets, as well as debug logs from all other
                        modules. To also include the debug level logs from juju and
                        websockets modules, use the maximum verbosity.
  --quiet, -q           Disable output in STDOUT.
  --auto-approve        Automatically approve and continue with each upgrade step␣
↪without prompt.
```

For upgrading **hypervisors**, in addition to the common options also found in **data-plane** upgrades, users can specify either **–machine** or **–az** to narrow the upgrade to a particular subset of nodes.

```
user@host:~$ cou upgrade hypervisors --help
```

```
Usage: cou upgrade hypervisors [options]

Upgrade machines with nova-compute and colocated services.
This is possible only if control-plane has been fully upgraded,
```

(continues on next page)

(continued from previous page)

```
otherwise an error will be thrown.

Note that only principal applications colocated with nova-compute units
that support action-managed upgrades are within the scope of this command.
Other principal applications (e.g. ceph-osd) and subordinates
can be upgraded via the data-plane subcommand.

Options:
  -h, --help            Show this help message and exit.
  --model MODEL_NAME    Set the model to operate on.
                        If not set, the currently active Juju model will be used.
  --backup, --no-backup
                        Include database backup step before cloud upgrade.
                        Default to enabling database backup.
  --force               Force the plan/upgrade of non-empty hypervisors.
  --verbose, -v         Increase logging verbosity in STDOUT.
                        Multiple 'v's yield progressively more detail (up to 3).
                        Note that by default the logfile will not include standard logs
                        from juju and websockets, as well as debug logs from all other
                        modules. To also include the debug level logs from juju and
                        websockets modules, use the maximum verbosity.
  --quiet, -q           Disable output in STDOUT.
  --machine MACHINES, -m MACHINES
                        Specify machine id(s) to upgrade.
                        This option accepts a single machine id as well as a stringified
                        comma-separated list of ids, and can be repeated multiple times.
                        This option cannot be used together with [--availability-zone/--
↪az].
  --availability-zone AVAILABILITY_ZONES, --az AVAILABILITY_ZONES
                        Specify Juju availability zone(s) to upgrade.
                        This option accepts a single availability zone as well as a
                        stringified comma-separated list of AZs, and can be repeated
                        multiple times. This option cannot be used together with
                        [--machine/-m]
  --auto-approve        Automatically approve and continue with each upgrade step␣
↪without prompt.
```

2.3.2 Environment Variables

- **JUJU_DATA** - sets the path containing Juju configuration files (e.g. credentials.yaml). The default value is **~/.local/share/juju**
- **COU_TIMEOUT** - defines timeout for **COU** retry policy. The default value is 10 seconds.
- **COU_MODEL_RETRIES** - defines how many times to retry the connection to Juju model before giving up. The default value is 5 times.
- **COU_MODEL_RETRY_BACKOFF** - defines by how many seconds the wait between juju model connection retry attempts will be increased every time an attempt fails. The default value is 2 seconds.
- **COU_STANDARD_IDLE_TIMEOUT** - defines how long **COU** will wait for an application to settle to **active/idle** and declare the upgrade complete. The default value is 300 seconds.
- **COU_LONG_IDLE_TIMEOUT** - a longer version of **COU_STANDARD_IDLE_TIMEOUT** for applications that are known to need more time than usual to upgrade, such as Keystone and Octavia. The default value

is 2400 seconds.

- **LANDSCAPE_MIRROR_URI** - defines the base URI of the Landscape-managed APT mirror. When set, it is used to construct the openstack-origin value so that charms pull packages from the private repository instead of public archives.
- **LANDSCAPE_APT_COMPONENT** - specifies the APT repository component (e.g., main, universe). It is used together with **LANDSCAPE_MIRROR_URI** and the suite to fully define the **openstack-origin** entry of charms.

2.3.3 Known issues

The following issues are known to the OpenStack charms, and they are not related to the **COU** project. Some of the bug reports might contains potential workarounds. If you encounter those issues, please check the relevant bug reports for more information.

Ceph

Workload version does not changed

Ceph applications might need manual intervention after the upgrade because the payload version doesn't update automatically. You might find a message like this

```
[WARNING] Ceph mon (ceph-mon/0) sees mismatched versions in ceph daemons:
{
    "mon": {
        "ceph version 18.2.4 (e7ad5345525c7aa95470c26863873b581076945d) reef (stable)": 3
    },
    "mgr": {
        "ceph version 18.2.4 (e7ad5345525c7aa95470c26863873b581076945d) reef (stable)": 3
    },
    "osd": {
        "ceph version 17.2.7 (b12291d110049b2f35e32e0de30d70e9a4c060d2) quincy (stable)
↪": 9
    },
    "rgw": {
        "ceph version 17.2.7 (b12291d110049b2f35e32e0de30d70e9a4c060d2) quincy (stable)
↪": 3
    },
    "overall": {
        "ceph version 17.2.7 (b12291d110049b2f35e32e0de30d70e9a4c060d2) quincy (stable)
↪": 12,
        "ceph version 18.2.4 (e7ad5345525c7aa95470c26863873b581076945d) reef (stable)": 6
    }
}
```

See bug 2068151 and #401 for more details.

Ceph Dashboard

Upgrading from `quincy/stable` to `reef/stable` disables TLS

After the Ceph Dashboard charm is upgraded to `reef/stable`, the units might go to `blocked` state with a message includes something like: "Dashboard not responding". This is because the listening port was changed from `8443` to `8080`.

See bug 2111793 for more details.

Designate

Upgrade causes units in error state

Refreshing designate charm from `yoga/stable` to `zed/stable` will leave the designate units in error state. You might find a message in *juju debug* like this

```
subprocess.CalledProcessError: Command '['/var/lib/juju/agents/unit-designate-1/.venv/
↪bin/pip',
'install', '-U', '--force-reinstall', '--no-index', '--no-cache-dir', '-f', 'wheelhouse',
'pyparsing==3.0.9', 'flit-core==3.7.1', 'dnspython==2.2.1', 'pyaml==21.10.1', 'Jinja2==3.
↪0.3',
'packaging==21.3', 'tomli==1.2.3', 'netifaces==0.11.0', 'netaddr==0.7.19', 'psutil==5.9.2
↪',
'charms.openstack==0.0.1.dev1', 'pbr==5.10.0', 'charmhelpers==1.1.1.dev86', 'PyYAML==5.3.
↪1',
'charms.reactive==1.5.1']' returned non-zero exit status 1.
```

See bug 2114254 for more details.

Manila

Workload does not get upgraded

The configuration based upgrade does not work on certain version of Manila charms. You might find a message like this

```
Verify that the workload of 'manila' has been upgraded on units: manila/0, manila/1,␣
↪manila/2 ×

[ERROR] Unit(s) 'manila/0, manila/1, manila/2' did not complete the upgrade
to zed. Some local processes may still be executing; you may try re-running COU in a few
minutes.
```

See bug 2111738 for more details.

Manila Ganesha

Workload does not get upgraded

The configuration based upgrade does not work on certain versions of Manila Ganesha charms. You might find a message like this

```
Verify that the workload of 'manila-ganesha' has been upgraded on units: manila-ganesha/
↪0,
manila-ganesha/1, manila-ganesha/2 ×

[ERROR] Unit(s) 'manila-ganesha/0, manila-ganesha/1, manila-ganesha/2' did not complete␣
↪the
upgrade to zed. Some local processes may still be executing; you may try re-running COU␣
↪in a
few minutes.
```

See bug 2111738 for more details.

Workload version is reported incorrectly

Certain versions of the Manila Ganesha charm incorrectly report their workload version, which can lead to false positive COU errors resembling the following one:

```
[WARNING] Not possible to find the charm manila-ganesha in the lookup
[ERROR] 'manila-ganesha' with workload version 17.1.0 has no compatible OpenStack␣
↪release.
```

If affected, refresh the Manila Ganesha charm to its most recent version (within the same release channel) and re-run **COU**.

See bug 2060751 for details.

Service `manila-share` is masked after charm refresh

Certain versions of the Manila Ganesha charm will cause the `manila-share` service to be masked after the charm is refreshed. This will confuse the HA Cluster charm that deploys with Manila Ganesha charm, causing corosync to flap constantly which in turn causes the hacluster to go in and out of the blocked state.

See bug 2111818 for details.

Rabbitmq Server

Upgrade does not work with `enable-auto-restarts=True`

The rabbitmq-server charm must have `enable-auto-restarts=False` for **COU** to work properly due to the known charm bug.

We suggest that users should temporarily set `enable-auto-restarts=False` when performing `cou upgrade`, and rollback to original setting after the upgrade is completed.

See bug 2046381 for details.

Additional Information

Potential known upgrade bugs and non-standard procedures are listed in the OpenStack Charm Guide:

- Issues, charm procedures, and OpenStack upgrade notes

2.3.4 Security

Charmed OpenStack Upgrader orchestrates upgrade operations on a cloud by leveraging the Juju client and automating the actions a human operator would perform. It does not by itself perform any cryptographic operation, although it relies on libjuju for communicating securely with the Juju controller and performing lower level secure operations like ssh or scp.

Having a juju snap installed and configured to connect to the desired controller is a prerequisite for using Charmed OpenStack Upgrader.

Please refer to Juju's documentation for more details.

2.4 Explanation

This section provides additional information about the upgrade stages, scopes, and the implementation.

2.4.1 Analysis

This phase is responsible for gathering all essential information about the OpenStack cloud from Juju. It collects the model status, details about each application including its units and configurations, and information about machines hosting these applications. These data are stored in an Analysis object, which organises applications into control-plane and data-plane components.

Each application is represented by a generic **OpenStackApplication** class or by a custom subclass, e.g. **Keystone(OpenStackApplication)**. Each unit is represented by a *Unit* class. Each machine is represented by a *Machine* class.

2.4.2 Planning

The plan has a tree structure with six main sections:

- cloud pre-upgrade steps,
- control-plane principal applications upgrade
- control-plane subordinate applications upgrade
- data-plane hypervisors upgrade
- data-plane non-hypervisor principal applications upgrade
- data-plane subordinate applications upgrade
- cloud post-upgrade steps.

```
Upgrade cloud from current release to target release
    cloud pre-upgrade steps
        Verify cloud is in idle state
        MySQL backup
    control-plane principal applications upgrade
            application 1
                pre-upgrade steps
                upgrade steps
                post-upgrade steps
            ...
            application N
                ...
    control-plane subordinate applications upgrade
            application M
                ...
    data-plane hypervisors upgrade
            availability zone 1
                pre-upgrade steps
                upgrade steps
                     unit 1
                        unit upgrade step A
                        ...
                        unit upgrade step Z
                    ...
                     unit K
                        unit upgrade step A
                        ...
                        unit upgrade step Z
                post-upgrade steps
```

(continues on next page)

(continued from previous page)

```
            ...
            availability zone N
                ...
    data-plane non-hypervisor principal applications upgrade
            application P
                ...
    data-plane subordinate applications upgrade
            application Q
                ...
    cloud post-upgrade steps
        (if ceph exists) Ensure correctness of 'require-osd-release' option in 'ceph-mon'
...
```

The above demonstrates a complete cloud upgrade plan. However, it's also possible to target a specific subset of the cloud. For more information, please refer to *Upgrade Groups*.

The **pre-upgrade** steps prepare COU for the upgrade process, which includes verifying the states or configurations of the applications, units, or of the OpenStack cloud.

The **upgrade** steps are the main steps needed to run upgrades for each application.

The **post-upgrade** steps are responsible for making sure that the upgrade finishes successfully.

The plan can also be obtained without the need to perform a cloud upgrade using the **plan** command. See *Plan an upgrade*.

The Greek letter Psi indicates that the step and its sub-steps will run in parallel with other steps in the same level. In the example above, within the *data-plane hypervisors upgrade*, the *unit 1* step will run in parallel with the *unit K* step, but their *upgrade steps A to Z* sub-steps will run sequentially.

Different upgrade strategies are chosen for control-plane and data-plane applications when preparing the plan. For details, please refer to *Upgrade*.

2.4.3 Upgrade

This phase is responsible for applying the upgrade plan step by step (see *Upgrade a cloud*).

For the **control-plane**, COU upgrades applications in a strict sequential manner, following a predefined order (see *Upgrade groups*) and halting the process should any application encounter an upgrade failure. All applications within the **control-plane** employ an all-in-one method, upgrading all units of an application simultaneously.

COU adopts a more cautious strategy when upgrading **data-plane** applications to minimize the risk of downtime, which can occur if the unit undergoing upgrade is actively handling client requests. Between applications, similar to the **control-plane**, upgrades are executed sequentially following the predefined order (see *Upgrade groups*). But for applications that support the **openstack-upgrade** action (such as **nova-compute** and its colocated services like **cinder**), COU handles the upgrades in a paused-single-unit fashion. This method is notably more time-intensive, so to streamline the process while still maintaining cloud stability, units are grouped based on the Juju availability zones of their respective hosting machines. COU then progresses through these availability zone groups in a sequential manner, performing upgrades on units within a single zone in parallel.

Note: ceph-osd, while being a component of the **data-plane**, employs the *all-in-one* method for upgrades. because its charm is designed to maintain service availability throughout the upgrade process.

SLURP support

COU aligns with Charmed OpenStack upgrade strategy. Since Charmed OpenStack does not support SLURP, COU also does not provide support for it.

2.4.4 Upgrade Groups

In the process of planning or executing an upgrade for an OpenStack cloud, users have the capability to target a specific group of applications based on their services responsibilities.

Note: COU will upgrade **openstack-dashboard** and **octavia** at the end of the **control-plane** upgrade (before upgrading **control-plane** subordinate applications and **data-plane** services) due to the desired upgrade group splits. This is slightly different from the upstream upgrade documentation.

Control Plane

The **control-plane** includes services tasked with making decisions related to data management, routing, and processing. Services considered as **control-plane** in OpenStack and in scope of COU are (following their upgrade order):

- rabbitmq-server
- ceph-mon
- keystone
- aodh
- barbican
- ceilometer
- ceph-fs
- ceph-radosgw
- cinder
- designate
- designate-bind
- glance
- gnocchi
- heat
- manila
- manila-ganesha
- neutron-api
- neutron-gateway
- ovn-dedicated-chassis
- ovn-central
- placement
- nova-cloud-controller
- openstack-dashboard
- octavia

- additional principal applications that contribute to the formation of the OpenStack cloud (typically **mysql-innodb-cluster** and **vault**)
- **control-plane** subordinate applications

Data Plane

On the other hand, the **data-plane** is composed of services that handle the actual data transfer. Services considered as **data-plane** in OpenStack and in scope of COU are (following their upgrade order):

- nova-compute
- any **control-plane** services colocated on the same machines with the **nova-compute** application (typically **cinder**)
- ceph-osd
- **data-plane** subordinate applications

Note:

- It's essential to complete the upgrade of the **control-plane** components before proceeding to any **data-plane** components to ensure cloud functionality.
- Swift applications like **swift-proxy** and **swift-storage** are recognised as valid components of the OpenStack **data-plane**. However, they are not supported by COU. Therefore, manual upgrades are needed.

Hypervisors

Within the data-plane are **hypervisors**. In COU they represent machines hosting the hypervisor service (**nova-compute**), which facilitate the distribution of compute and memory resources among virtual machines (VMs), and other services colocated on the same nodes.

Note: Since **hypervisors** comprise a subset of **data-plane** components, it is also necessary to complete the upgrade of the **control-plane** components before proceeding to **hypervisors** upgrades.

2.4.5 Data Migration on nova-cloud-controller

This document explains the details of the database migration for nova-cloud-controller.

nova-cloud-controller Database Migration Details

When users upgrade the Juju application nova-cloud-controller, the commands `nova-manage db sync` and `nova-manage db online_data_migration` will be executed. In addition, **COU** supports running `nova-manage db archive_deleted_rows` and `nova-manage db purge` to enhance the performance of data migration.

- `nova-manage db sync` - Upgrades the main database schema to the most recent version.
- `nova-manage db online_data_migration` - Performs data migration to update all live data.
- `nova-manage db archive_deleted_rows` - Moves deleted rows from production tables to shadow tables.
- `nova-manage db purge` - Deletes rows from shadow tables.

db sync

Before the Yoga version, nova-cloud-controller used **SQLAlchemy** to handle database migrations. Starting from Yoga, it switched to **Alembic**, a migrations tool for **SQLAlchemy**.

Below is a table showing the migration versions from Ussuri to Caracal.

Version	Migration Version
Ussuri	407
Victoria	412
Wallaby	417
Xena	422
Yoga	422 - (move to Alembic) - `8f2f1571d55b_initial_version` - `16f1fbcab42b_resolve_shadow_table_diffs`
Zed	`ccb0fa1a2252_add_encryption_fields_to`
2023.1 (Antelope)	`960aac0e09ea_de_duplicate_indexes_in_instances`
2023.2 (Bobcat)	`1acf2c98e646_add_compute_id_to_instance` - `1b91788ec3a6_drop_legacy_migrate_version_table`
2024.1 (Caracal)	`13863f4e1612_create_share_mapping_table`

Details of each migration can be found at:

- unmaintained/yoga - nova/nova/db/main/legacy_migrations/versions
- unmaintained/2024.1 - nova/nova/db/main/migrations/versions

db online_data_migration

The list of online data migrations can be found at nova_online_migrations. There are only two online migration cases after Victoria:

- `pci_device_obj.PciDevice.populate_dev_uuids`, added in Victoria
- `instance_obj.populate_instance_compute_id`, added in 2023.2

db archive_deleted_rows

When an instance is deleted, an entry will still be kept in the active table but marked as **deleted**. This command will move those entries from the active table to the shadow table, and hence improve the performance of queries over the active table.

db purge

This command will remove the entries in the shadow table, and reduce the size of the database.

On COU

Generally, the data migration operation load is not too high, as observed from previous information. **COU** provides two optional steps, `archive` and `purge` to reduce the possible load during the upgrade. These two steps run the following commands on the nova-cloud-controller Juju unit respectively:

- `db archive_deleted_rows`
- `db purge`

> **Tip**
>
> Performing `archive` and `purge` before the cloud upgrade or during a maintenance window is generally **recommended**, since metadata of deleted instances remain in the active table of the database, and so the database size

can grow unbounded. A large active table can slow down database queries during cloud operation and affect the upgrade process.

Make sure to check the details of database schema migrations and online data migrations before each upgrade.

Please refer to *archive* and *purge* in **COU**.

More Information

- opendev/openstack/nova
- *Archive old data*
- *Purge data on shadow table*