
MicroOVN

Canonical Group Ltd

Apr 03, 2026

CONTENTS

1	In this documentation	3
2	Project and community	5
2.1	How-to guides	5
2.2	Tutorials	26
2.3	Explanation	27
2.4	Reference	31
2.5	Contribute to MicroOVN	38

MicroOVN is a snap-based distribution of OVN - [Open Virtual Network](#).

It allows users to deploy an OVN cluster with just a few commands. Aside from the regular OVN packages, MicroOVN comes bundled with a CLI utility (`microovn`) that facilitates operational management. In particular, it simplifies the task of adding/removing cluster members and incorporates status checking out of the box.

Besides the ease of deployment and a convenient CLI tool, another benefit of MicroOVN is in its self-contained nature: it is distributed as a [strictly confined snap](#). This means that it can be easily upgraded/downgraded/removed without affecting the host system.

MicroOVN caters to a wide range of user and environment types. It lowers the barrier of entry to OVN for people that are less familiar with it by automating much of the deployment process. It also provides a fully fledged, unrestricted OVN deployment that is suitable for both development and production environments.

IN THIS DOCUMENTATION

Tutorial

Start here: a hands-on introduction to MicroOVN for new users

How-to guides

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

Explanation

Discussion and clarification of key topics

Reference

Technical information - specifications, APIs, architecture

PROJECT AND COMMUNITY

MicroOVN is a member of the Ubuntu family. It's an open source project that warmly welcomes community projects, 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 used as our bug tracker
- To speak with us, you can find us in our [MicroCloud Discourse](#) category.
- Optionally enable [Ubuntu Pro](#) on your OVN nodes. This is a service that provides the [Livepatch Service](#) and the [Expanded Security Maintenance \(ESM\)](#) program.

2.1 How-to guides

These How-to guides will cover a variety of operations and configurations that are possible with MicroOVN. They do however assume general Linux knowledge and a basic understanding of OVN.

2.1.1 Working with TLS

Starting with snap revision 111, new deployments of MicroOVN use TLS encryption by default. A self-signed CA certificate is used to issue certificates to all OVN services that require it. They provide authentication and encryption for OVSDB communication. The CA certificate is generated during cluster initialisation (**cluster bootstrap** command).

In the current implementation, self-provisioned certificates are the only mode available. Future releases may include support for externally provided certificates.

Warning

The certificate and private key generated for the self-provisioned CA are currently stored unencrypted in the database on every cluster member. If an attacker gains access to any cluster member, they can use the CA to issue valid certificates that will be accepted by other cluster members.

Certificates CLI

MicroOVN exposes a few commands for basic interaction with TLS certificates.

List certificates

To list currently used certificates:

```
microovn certificates list
```

Example output:

```
[OVN CA]
/var/snap/microovn/common/data/pki/cacert.pem (OK: Present)

[OVN Northbound Service]
/var/snap/microovn/common/data/pki/ovnnb-cert.pem (OK: Present)
/var/snap/microovn/common/data/pki/ovnnb-privkey.pem (OK: Present)

[OVN Southbound Service]
/var/snap/microovn/common/data/pki/ovnsb-cert.pem (OK: Present)
/var/snap/microovn/common/data/pki/ovnsb-privkey.pem (OK: Present)

[OVN Northd Service]
/var/snap/microovn/common/data/pki/ovn-northd-cert.pem (OK: Present)
/var/snap/microovn/common/data/pki/ovn-northd-privkey.pem (OK: Present)

[OVN Chassis Service]
/var/snap/microovn/common/data/pki/ovn-controller-cert.pem (OK: Present)
/var/snap/microovn/common/data/pki/ovn-controller-privkey.pem (OK: Present)
```

This command does not perform any certificate validation, it only ensures that if a service is available on the node, the file that should contain a certificate is in place.

Re-issue certificates

The **certificates reissue** command is used to interact with OVN services on the local host; it does not affect peer cluster members.

Important

Services must be running in order to be affected by the **certificates reissue** command. For example, running **certificates reissue ovnnb** on a member that does not run this service is expected to fail.

To re-issue a certificate for a single service:

```
microovn certificates reissue <ovn_service_name>
```

To re-issue certificates for all services, the **all** argument is supported:

```
microovn certificates reissue all
```

Valid service names can be discovered with the **--help** option:

```
microovn certificates reissue --help
```

Manage Certificate Authority

MicroOVN stores CA certificate and private key in its database, and issues certificates for the OVN services and clients when necessary. This CA certificate can be either self-signed, automatically generated by the MicroOVN, or provided by the user. The type of certificate to be used can be defined either when the user initialises the cluster, or freely changed afterwards.

Automatic self-signed CA certificate

This is the default choice when bootstrapping/initialising a new cluster. User does not need to take any extra steps for the MicroOVN to generate the automatic CA, and use it to issue OVN TLS certificates.

Should there be a need to change/update the currently used CA certificate, the **certificates regenerate-ca** command can be used to issue a new CA and new certificates for every OVN service in the cluster:

```
microovn certificates regenerate-ca
```

This command replaces the current CA certificate and notifies all cluster members to re-issue certificates for all their services. The command's output will include evidence of successfully issued certificates for each cluster member.

Warning

A new certificate must be issued successfully for every service on every member. Any failure will result in subsequent communication errors for that service within the cluster. Any failed certificates can be tried to re-issue again with **certificates reissue <service>** on the affected node.

User-provided CA certificate

Alternative to the automatic self-signed CA certificate is for the user to provide their own CA certificate and private key. This can be done when initialising a cluster via **init**, or anytime afterwards via **certificates set-ca**. The certificate and the key are provided as a path to a file on disk. The certificate and private key can be passed via stdin when using **certificates set-ca --combined**. MicroOVN stores the contents of these files in its database, so it's safe to remove the files afterwards.

Note

With MicroOVN being a confined snap, it has limited accessibility to the host filesystem. The most reliable way to provide the certificate and the key file, is to put them into `/var/snap/microovn/common` and let the MicroOVN to read it from there, or by piping them in using the `--combined` option.

Example of replacing current CA:

```
microovn certificates set-ca --cert /var/snap/microovn/common/ca.crt --key /var/snap/
↳microovn/common/ca.key
```

Or via stdin:

```
cat /var/snap/microovn/common/ca.crt /var/snap/microovn/common/ca.key | microovn_
↳certificates set-ca --combined
```

Similar to the **certificates regenerate-ca**, this triggers reissue of all service and client certificates on the OVN cluster. The command's output will include evidence of successfully issued certificates for each cluster member.

Warning

A new certificate must be issued successfully for every service on every member. Any failure will result in subsequent communication errors for that service within the cluster. Any failed certificates can be tried to re-issue again with `certificates reissue <service>` on the affected member.

There are some limitations to the private key type used to sign the CA certificate. It has to be one of the following types:

- RSA
- ECDSA
- ED25519
- ECDH

Upgrade from plaintext to TLS

Plaintext communication is used when MicroOVN is initially deployed with a snap revision of less than 111, and there's no way to automatically convert to encrypted communication. The following manual steps are needed to upgrade from plaintext to TLS:

1. ensure that all MicroOVN snaps in the cluster are upgraded to, at least, revision 111
2. run `microovn certificates regenerate-ca` on one of the cluster members
3. run `sudo snap restart microovn.daemon` on **all** cluster members. Allow commands to complete before proceeding to the next step.
4. run `sudo snap restart microovn.ovn-northd` on **all** cluster members

Once this is done, OVN API services throughout the cluster will start listening on TLS-secured ports. However, the process is not complete yet because OVN Southbound and Northbound database clusters themselves are not capable of automatically switching to TLS communication in existing clusters.

Manually switch OVN Northbound and Southbound clusters to TLS

Both database clusters need to be manually switched over by individually removing cluster members that use `tcp` connection and reconnecting them with `ssl`. This process technically replaces every member in the original cluster, but because we are doing it gradually, cluster data remains intact.

Let's assume that we have a 3 node cluster. We'll start with switching over the OVN Northbound cluster.

Preparation: We will be running commands on multiple nodes throughout this process, it is recommended to open a separate shell on each node and keep it open with following variables exported:

```
CONTROL_SOCKET=/var/snap/microovn/common/run/ovn/ovnnb_dbctl
DB=OVN_Northbound
DB_FILE=/var/snap/microovn/common/data/central/db/ovnnb_db.db
PORT=6643
```

1. Leave cluster on the node 1:

```
ovn-appctl -t $CONTROL_SOCKET cluster/leave $DB
```

2. Make sure that member properly left the cluster by inspecting cluster status on nodes 2 and 3 and ensuring that node 1 is no longer part of the cluster:

```
ovn-appctl -t /var/snap/microovn/common/run/ovn/ovnnb_dbctl cluster/status OVN_
↔Northbound
```

3. Clean up remaining DB files on node 1:

```
snap stop microovn.ovn-ovsdb-server-nb
snap stop microovn.ovn-ovsdb-server-sb
snap stop microovn.ovn-northd
rm $DB_FILE
```

4. Rejoin the cluster with node 1, using `ssl` as protocol for local listening port. Notice that we will still use `tcp` as a protocol for remote cluster connection because no other node listens on `ssl` yet. This will get fixed automatically when other cluster members switch to `ssl`:

```
ovsdb-tool join-cluster $DB_FILE $DB ssl:<local_ip>:$PORT tcp:<node_2_ip>:$PORT
snap restart microovn.ovn-ovsdb-server-nb
snap restart microovn.ovn-ovsdb-server-sb
snap restart microovn.ovn-northd
```

5. Monitor cluster, from node 1, as it converges to stable state. Use following command to monitor cluster until it indicates three members and field `Entries not yet applied` reaches 0:

```
ovn-appctl -t $CONTROL_SOCKET cluster/status $DB
```

Now that node 1 successfully transitioned to TLS we can repeat the same steps on node 2 and then on node 3. The only difference is in **4. step** where we will use protocol `ssl` and IP of a node 1 as last arguments for `ovsdb-tool` command. To save you some searching and replacing, here are the revised commands for the **4. step** to be used on node 2 and 3:

```
ovsdb-tool join-cluster $DB_FILE $DB ssl:<local_ip>:$PORT ssl:<node_1_ip>:$PORT
snap restart microovn.ovn-ovsdb-server-nb
snap restart microovn.ovn-ovsdb-server-sb
snap restart microovn.ovn-northd
```

After all three nodes transitioned to TLS usage, you can once again inspect cluster status on any node:

```
ovn-appctl -t $CONTROL_SOCKET cluster/status $DB
```

to verify that all three cluster members are using `ssl` as their connection protocol.

This whole process needs to be repeated again for OVN Southbound cluster. Steps and commands are the same, just with different set of variables configured in the **Preparation** step:

```
CONTROL_SOCKET=/var/snap/microovn/common/run/ovn/ovnsb_dbctl
DB=OVN_Southbound
DB_FILE=/var/snap/microovn/common/data/central/db/ovnsb_db.db
PORT=6644
```

Common issues

This section contains some well known or expected issues that you can encounter.

I'm getting failed to load certificates error

If you run commands like `ovn-sbctl` and you get complaints about missing certificates while the rest of the commands seem to work fine.

Example:

```
ovn-sbctl show
```

Example output:

```
2023-06-14T15:09:31Z|00001|stream_ssl|ERR|SSL_use_certificate_file:↵  
↵error:80000002:system library::No such file or directory  
2023-06-14T15:09:31Z|00002|stream_ssl|ERR|SSL_use_PrivateKey_file: error:10080002:BI0↵  
↵routines::system lib  
2023-06-14T15:09:31Z|00003|stream_ssl|ERR|failed to load client certificates from /var/  
↵snap/microovn/common/data/pki/cacert.pem: error:0A080002:SSL routines::system lib  
Chassis microovn-0  
  hostname: microovn-0  
  Encap geneve  
    ip: "10.5.3.129"  
    options: {csum="true"}
```

This likely means that your MicroOVN snap got upgraded to a version that supports TLS, but it requires some manual upgrade steps. See section [Upgrade from plaintext to TLS](#).

2.1.2 Downscaling the cluster

Impact

Downscaling can have an adverse effect on the availability and resiliency of the cluster, especially when a member is being removed that runs an OVN central service (OVN SB, OVN NB, OVN Northd).

OVN uses the [Raft consensus algorithm](#) for cluster management, which has a fault tolerance of up to $(N-1)/2$ members. This means that fault resiliency will be lost if a three-node cluster is reduced to two nodes.

Monitoring

You can watch logs on the departing member for indications of removal failures with:

```
snap logs -f microovn.daemon
```

Any issues that arise during the removal process will need to be resolved manually.

Remove a cluster member

To remove a cluster member:

```
microovn cluster remove <member_name>
```

The value of `<member_name>` is taken from the **Name** column in the output of the `cluster list` command.

Any chassis components (`ovn-controller` and `ovs-vswitchd`) running on the member will first be stopped and disabled (prevented from starting). For a member with central components present (`microovn.central`), the Northbound and Southbound databases will be gracefully removed.

Verification

Upon removal, check the state of OVN services to ensure that the member was properly removed.

```
# Check status of OVN SB cluster
ovn-appctl -t /var/snap/microovn/common/run/central/ovnsb_db.ctl cluster/status OVN_
↳Southbound

# Check status of OVN NB cluster
ovn-appctl -t /var/snap/microovn/common/run/central/ovnnb_db.ctl cluster/status OVN_
↳Northbound

# Check registered chassis
ovn-sbctl show
```

Data preservation

MicroOVN will back up selected data directories into the timestamped location `/var/snap/microovn/common/backup_<timestamp>/`. These backups will include:

- logs
- OVN database files
- OVS database file
- issued certificates and keys

Securely decommission the cluster member

If you wish to wipe any remaining data from the machine, after you removed the node from the cluster, you can run:

```
snap remove --purge microovn
```

This will remove MicroOVN's runtime components, any data it stored on disk, and prevents creation of the snapshot of the data.

2.1.3 Accessing logs

The *MicroOVN services* provide logs as part of their normal operation.

By default they are provided through the systemd journal, and can be accessed through the use of the `journalctl` or `snap logs` commands.

This is how you can access the logs of the `microovn.chassis` service using the `snap logs` command:

```
snap logs microovn.chassis
```

and using the `journalctl` command:

```
journalctl -u snap.microovn.chassis
```

This is how you can view a live log display for the same service using the `snap logs` command:

```
snap logs -f microovn.chassis
```

and using the `journalctl` command:

```
journalctl -f -u snap.microovn.chassis
```

Log files

Inside the `/var/snap/microovn/common/logs` directory you will find files for each individual service, however these will either be empty or not contain updated information, this is intentional.

On a fresh install the files are created, as a precaution, in the event a need arises for enabling *debug logging*. When upgrading MicroOVN, existing files will be retained, but not updated.

Debug logging

The Open vSwitch (OVS) and Open Virtual Network (OVN) daemons have a rich set of debug features, one of which is the ability to specify log levels for individual modules at run time.

A list of modules can be acquired through the `ovs-appctl` and `ovn-appctl` commands.

This is how to enable debug logging for the Open vSwitch `vswitchd` module:

```
ovs-appctl vlog/set vswitchd:file:dbg
```

This is how to enable debug logging for the Open Virtual Network `reconnect` module:

```
ovn-appctl vlog/set reconnect:file:dbg
```

For more details on how to configure logging, see [ovs-appctl manpage](#).

2.1.4 Upgrade MicroOVN across major versions

MicroOVN is released in channels that signify which version of OVN it bundles (e.g. `22.03/stable` channel comes with OVN `22.03`). These channels track a specific major version, and wont upgrade to next major version on their own. To upgrade to next major version of MicroOVN, you have to change `microovn snap` channel.

In this how-to, we'll upgrade a cluster with four members, running MicroOVN `22.03`, to MicroOVN `24.03`.

Prepare cluster for upgrade

We start by ensuring that **each** of our cluster members runs MicroOVN from a channel that precedes the version to which we are upgrading, and that it has latest upgrades from this channel.

In this example we are upgrading to `24.03`, so we'll check that our cluster members run `22.03`.

```
snap info microovn
```

Example of relevant output from `snap info`:

```
<snipped preceding output>

snap-id:      1lLUDjCLf2hf4zr1ty82XqaYTwN4afUP
tracking:     22.03/stable
refresh-date: today at 10:07 UTC

<snipped remaining output>
```

Next we ensure that MicroOVN runs the latest version in the channel (again on **each** cluster member):

```
sudo snap refresh microovn
```

As a final preparation step, we'll ensure that all MicroOVN cluster members are online by running:

```
sudo microovn cluster list -f compact
```

It's sufficient to run this command on a single member. Resulting output should show status of all members as ONLINE:

NAME	ADDRESS	ROLE	FINGERPRINT	STATUS
movn1	10.75.224.44:6443	voter	0e359bed39fb0aaedcb730c707b89701abfb0a65ed5e0f9b5ff883a75c914683	ONLINE
movn2	10.75.224.233:6443	stand-by	b084c2fadd4ca66ffd8fb7e58a1f90f2bbec1fec5ec6d4091eba7e7fbbb66981	ONLINE
movn3	10.75.224.128:6443	voter	fc9efe07194030ec212a75d32e525a321eb973a0cf071c2bc8841480457a248a	ONLINE
movn4	10.75.224.11:6443	voter	fa3380a109f48e5bce60ba942cf24617d5db3b4f371dedc6ef732303ada7ed0b	ONLINE

Ensure sufficient election timer

Upgrade of OVN cluster can be computationally stressful operation, especially for nodes that run OVN central services. To prevent cluster members from missing heartbeats and causing leadership flapping, we recommend setting election timer of Northbound and Southbound databases to at least 16 seconds.

To check current values, run following commands:

```
# Get OVN Northbound cluster status
sudo ovn-appctl -t /var/snap/microovn/common/run/ovnnb_db.ctl cluster/status OVN_
↳Northbound

# Get OVN Southbound cluster status
sudo ovn-appctl -t /var/snap/microovn/common/run/ovnsb_db.ctl cluster/status OVN_
↳Southbound
```

Look for Election timer: in the output of these commands. Value of this field is expressed in milliseconds.

```
<snipped preceding output>
```

```
Last Election won: 56593 ms ago
Election timer: 16000
Log: [2, 8]
Entries not yet committed: 0
Entries not yet applied: 0
Connections:
Disconnections: 0
```

```
<snipped remaining output>
```

If the value is lower than 16000, we recommend gradually increasing it with:

```
# Command example for Northbound election timer increase
ovn-appctl -t /var/snap/microovn/common/run/ovnnb_db.ctl cluster/change-election-
↳timer OVN_Northbound <new_value>
```

(continues on next page)

(continued from previous page)

```
# Command example for Southbound election timer increase
ovn-appctl -t /var/snap/microovn/common/run/ovn/ovnsb_db.ctl cluster/change-election-
↳timer OVN_Southbound <new_value>
```

OVN won't let you increase the timer by more than twice its current value, so you will have to proceed gradually.

Upgrade single cluster member

Now we can proceed with upgrade of individual members in the cluster. The process itself is very straightforward, we just need to keep an eye on it, to ensure that it finishes as expected.

We'll start by upgrading single cluster member by running following command on it:

```
sudo snap refresh --channel=24.03/stable microovn
```

❗ Important

Above command causes restart of MicroOVN and OVN services running on this cluster member. This results in temporary data plane outage, for ports connected to OVN Chassis located on this member, while services come back up and reconfigure datapaths.

After the snap is successfully upgraded, there may be changes to either the `dqlite` schema or the `ovsdb` schema, or both. We can check the cluster status with:

```
sudo microovn cluster list -f compact
```

Systems that report `UPGRADING` have encountered a `dqlite` schema update and are awaiting all cluster members to receive the update. The systems that report `NEEDS UPGRADE` have not yet received the update and continue to function as before. Any systems that are `UPGRADING` will be unreachable by these systems.

NAME	ADDRESS	ROLE	FINGERPRINT
↳ STATUS			
movn1	10.75.224.44:6443	voter	0e359bed39fb0aaedcb730c707b89701abfb0a65ed5e0f9b5ff883a75c914683 UPGRADING
movn2	10.75.224.233:6443	stand-by	b084c2fadd4ca66ffd8fb7e58a1f90f2bbec1fec5ec6d4091eba7e7fbbb66981 NEEDS UPGRADE
movn3	10.75.224.128:6443	voter	fc9efe07194030ec212a75d32e525a321eb973a0cf071c2bc8841480457a248a NEEDS UPGRADE
movn4	10.75.224.11:6443	voter	fa3380a109f48e5bce60ba942cf24617d5db3b4f371dedc6ef732303ada7ed0b NEEDS UPGRADE

After all systems are refreshed, they should report `ONLINE` once again:

NAME	ADDRESS	ROLE	FINGERPRINT
↳ STATUS			
movn1	10.75.224.44:6443	voter	0e359bed39fb0aaedcb730c707b89701abfb0a65ed5e0f9b5ff883a75c914683 ONLINE
movn2	10.75.224.233:6443	stand-by	b084c2fadd4ca66ffd8fb7e58a1f90f2bbec1fec5ec6d4091eba7e7fbbb66981 ONLINE
movn3	10.75.224.128:6443	voter	fc9efe07194030ec212a75d32e525a321eb973a0cf071c2bc8841480457a248a ONLINE

(continues on next page)

(continued from previous page)

```
movn4 10.75.224.11:6443 voter ↵
↪ fa3380a109f48e5bce60ba942cf24617d5db3b4f371dedc6ef732303ada7ed0b ONLINE
```

If there was no `dqlite` schema update, there may still be an `ovsdb` schema update. In this case the systems may report `ONLINE` as soon as the first system is refreshed. The cluster status can be viewed with:

```
sudo microovn status
```

The output of the command above will look something like this:

```
<snipped preceding output>

OVN Database summary:
OVN Southbound: Upgrade or attention required!
Currently running schema: 20.21.0
Cluster report (expected schema versions):
  movn1: 20.33.0
  movn4: Missing API. MicroOVN needs upgrade
  movn2: Missing API. MicroOVN needs upgrade
  movn3: Missing API. MicroOVN needs upgrade

OVN Northbound: Upgrade or attention required!
Currently running schema: 6.1.0
Cluster report (expected schema versions):
  movn1: 7.3.0
  movn4: Missing API. MicroOVN needs upgrade
  movn3: Missing API. MicroOVN needs upgrade
  movn2: Missing API. MicroOVN needs upgrade
```

We can see, from the output above, that host `movn1`, as the only upgraded member so far, reports that it expects different OVN Southbound and OVN Northbound database schema version, as the cluster is currently running. This is expected and it will remain the case until all the cluster members are upgraded, at which point the schema upgrade will be triggered.

Note

As the MicroOVN version 24.03 is first to support API required to report expected schema versions, you will see placeholder messages `Missing API. MicroOVN needs upgrade` coming from hosts that run older MicroOVN versions. Going forward, the output during the future upgrades would look something like this:

```
OVN Northbound: Upgrade or attention required!
Currently running schema: 6.1.0
Cluster report (expected schema versions):
  movn1: 7.3.0
  movn4: 6.1.0
  movn3: 6.1.0
  movn2: 6.1.0
```

Note

If you run `microovn status` immediately after the snap refresh, you may encounter following, or similar, error

messages in the output:

```
OVN Database summary:
```

```
Failed to fetch OVN Southbound schema status: failed to fetch OVN Southbound cluster_
```

```
↳schema status from 'http://control.socket': Internal Server Error
```

```
Error: failed to fetch either Southbound or Northbound database status
```

It is expected, as it takes few seconds for the member to reconnect back to the cluster. The error message should go away after few seconds.

If you run `microovn status` and you encounter the following error, it means there is also a `dqlite` schema update, which can be viewed with `sudo microovn cluster list`:

```
Failed listing services: Database is waiting for an upgrade. 3 cluster members have_
```

```
↳not yet received the update
```

Continue with cluster upgrade

Same commands, from the previous section, can be run on the rest of the cluster members. You should progress one cluster member at a time and check the output of `microovn cluster status` to see if the upgrade continues as expected.

Final verification

After the last cluster member is upgraded, MicroOVN will trigger schema upgrade of OVN databases. This is an asynchronous process that can take from few seconds, to few minutes, depending on the size of the database. You can run:

```
sudo microovn status
```

and if the schema upgrade finished successfully, you'll see following output:

```
<snipped preceding output>
```

```
OVN Database summary:
```

```
OVN Southbound: OK (20.33.0)
```

```
OVN Northbound: OK (7.3.0)
```

2.1.5 Create custom OVN underlay network

The underlay network is the physical network infrastructure that provides connectivity between the nodes in an OVN deployment and is responsible for carrying encapsulated traffic between OVN components through Geneve (*Generic Network Virtualization Encapsulation*) tunnels. This allows the virtual network traffic to be transported over the physical underlay network. Now, by default, MicroOVN uses the hostname of a cluster member as a Geneve endpoint to set up the underlay network, but it is also possible to use custom Geneve endpoints for the cluster members.

Set up the underlay network

To tell MicroOVN to use the underlay network, you need to provide the IP address of the underlay network interface on each node. Let us assume that we want to create a three-node OVN cluster and that each node has a dedicated interface `eth1` with an IP address. Let says that `10.0.1.{2,3,4}` are the respective addresses on the `eth1` interface on each node. You can set the underlay network IP address in the `init` :

```
microovn init
```

Example of the interaction:

```

root@micro1:~# microovn init
Please choose the address MicroOVN will be listening on [default=10.242.68.93]:
Would you like to create a new MicroOVN cluster? (yes/no) [default=no]: yes
Please choose a name for this system [default=micro1]:
Would you like to define a custom encapsulation IP address for this member? (yes/no) ↵
↪ [default=no]: yes
Please enter the custom encapsulation IP address for this member: 10.0.1.2
Would you like to add additional servers to the cluster? (yes/no) [default=no]: yes
What's the name of the new MicroOVN server? (empty to exit): micro2
eyJzZWNyZXQiOiJmOWU1OWU0N2Q1M2E0ZjJlYTYzNWYwMzIzYTE5ZTgyMjEyMzA3ZmJmY2U5OTRiOTk3NzQ4ZTAyM2VmOGYyN2MyIiw
What's the name of the new MicroOVN server? (empty to exit): micro3
eyJzZWNyZXQiOiI5MwYzODUyZTA4ZjQyOWQxNGE2Y2JiZWl0NGNmODkyMjRjNzUzZjU1NjYzYTY3MjE5ZjZkMmVhOGM0MTdhM2YxIiw
What's the name of the new MicroOVN server? (empty to exit):

root@micro2:~# microovn init
Please choose the address MicroOVN will be listening on [default=10.242.68.13]:
Would you like to create a new MicroOVN cluster? (yes/no) [default=no]: no
Please enter your join token: ↵
↪ eyJzZWNyZXQiOiJmOWU1OWU0N2Q1M2E0ZjJlYTYzNWYwMzIzYTE5ZTgyMjEyMzA3ZmJmY2U5OTRiOTk3NzQ4ZTAyM2VmOGYyN2MyI
Would you like to define a custom encapsulation IP address for this member? (yes/no) ↵
↪ [default=no]: yes
Please enter the custom encapsulation IP address for this member: 10.0.1.3

root@micro3:~# microovn init
Please choose the address MicroOVN will be listening on [default=10.242.68.170]:
Would you like to create a new MicroOVN cluster? (yes/no) [default=no]:
Please enter your join token: ↵
↪ eyJzZWNyZXQiOiI5MwYzODUyZTA4ZjQyOWQxNGE2Y2JiZWl0NGNmODkyMjRjNzUzZjU1NjYzYTY3MjE5ZjZkMmVhOGM0MTdhM2YxI
Would you like to define a custom encapsulation IP address for this member? (yes/no) ↵
↪ [default=no]: yes
Please enter the custom encapsulation IP address for this member: 10.0.1.4

```

Now, the MicroOVN cluster is configured to use the underlay network with the IP addresses `10.0.1.{2,3,4}` on each node as tunnel endpoint for the encapsulated traffic. To verify that the underlay network is correctly configured, you can check the IP of OVN Geneve tunnel endpoint on each node:

```

root@micro1:~# ovs-vsctl get Open_vSwitch . external_ids:ovn-encap-ip
"10.0.1.2"

root@micro2:~# ovs-vsctl get Open_vSwitch . external_ids:ovn-encap-ip
"10.0.1.3"

root@micro3:~# ovs-vsctl get Open_vSwitch . external_ids:ovn-encap-ip
"10.0.1.4"

```

2.1.6 Service Control

Service control refers to the ability to specify which OVN services run on any given node in the cluster. The list of services can be found here [Services Reference](#) and they are responsible for handling core functionality.

The services of the each cluster node can be specified either during the initialisation of the node, or after the deployment via MicroOVN's CLI.

The approach of using snap CLI to enable/disable services is not recommended, because it does not update the desired

state or handle joining clusters and configuring the service properly.

Change services on a deployed cluster

MicroOVN offers `disable` and `enable` subcommands that can change which services are active on the current cluster node.

Note

This assumes you have MicroOVN installed and a clustered across three or more nodes. These nodes will be referred to as `first`, `second` and `third` respectively.

Disable a MicroOVN service

Disabling a MicroOVN service will configure it to not start automatically at boot and stop the service if it is running. run on `first`:

```
microovn disable switch
```

```
Service switch disabled
```

To validate that this has worked, we can query the status of MicroOVN and check which services are enabled. We should find that all nodes have `central`, `chassis` and `switch`, except `first` having only `central` and `chassis`. This shows the disabling worked

run on `first`:

```
microovn status
```

```
MicroOVN deployment summary:
- first (10.190.155.5)
Services: central, chassis
- second (10.190.155.174)
Services: central, chassis, switch
- third (10.190.155.55)
Services: central, chassis, switch
OVN Database summary:
OVN Northbound: OK (7.3.0)
OVN Southbound: OK (20.33.0)
```

The other nodes have also been informed of this change to the service placement and when queried will confirm that `switch` is disabled on `first` from their perspective too.

run on `second`:

```
microovn status
```

```
MicroOVN deployment summary:
- first (10.190.155.5)
Services: central, chassis
- second (10.190.155.174)
Services: central, chassis, switch
- third (10.190.155.55)
```

(continues on next page)

(continued from previous page)

```
Services: central, chassis, switch
OVN Database summary:
OVN Northbound: OK (7.3.0)
OVN Southbound: OK (20.33.0)
```

Note

We can also disable services on one node from another node using the optional `--node` argument, allowing for easier configuration of services.

```
microovn disable switch --node third
```

Enable a MicroOVN service

Enabling a MicroOVN service will configure it to start automatically at boot and if the service is not running, start it. run on first:

```
microovn enable switch
```

```
Service switch enabled
```

Note

If the switch service is enabled you may get an error, this is fine.

This will enable the switch service in MicroOVN, This can be shown through the listing of system services owned by MicroOVN. As mentioned in the disable section, these do not always translate directly to a MicroOVN service, but in this case it does.

run on first:

```
microovn status
```

```
MicroOVN deployment summary:
- first (10.190.155.5)
Services: central, chassis, switch
- second (10.190.155.174)
Services: central, chassis, switch
- third (10.190.155.55)
Services: central, chassis, switch
OVN Database summary:
OVN Northbound: OK (7.3.0)
OVN Southbound: OK (20.33.0)
```

You should be able to see here that the service is running and enabled on startup. The other nodes are also aware of this as if you query the status you will see it there and running.

run on second:

```
microovn status
```

```
MicroOVN deployment summary:
- first (10.190.155.5)
Services: central, chassis, switch
- second (10.190.155.174)
Services: central, chassis, switch
- third (10.190.155.55)
Services: central, chassis, switch
OVN Database summary:
OVN Northbound: OK (7.3.0)
OVN Southbound: OK (20.33.0)
```

Note

We can also enable services on one node from another node using the optional `--node` argument, allowing for easier configuration of services.

```
microovn enable switch --node third
```

Uses

Typically the most common use case of this will be to control the nodes the central services are running on and to increase the number of central services beyond the default of 3.

Specify services during the cluster deployment

The default selection of services for the node can be adjusted via the interactive `microovn init` command during the deployment (instead of using `bootstrap` and `join` methods). The user is asked question:

```
Please select comma-separated list services you would like to enable on this node.
↔(central/chassis/switch) or let MicroOVN automatically decide (auto) [default=auto]:
```

Here, they can either define the desired services as a comma-separated string or select `auto` option which falls back to the default behaviour. Leaving this option empty has same effect as selecting `auto`.

Note

The default behaviour for selecting services is to always enable `switch` and `chassis` services. The `central` service is enabled only if configuration option `ovn.central-ips` is not set and there are less than 3 nodes with `central` service enabled in the cluster.

2.1.7 Datapath-only Mode

This is a mode of operation in which MicroOVN cluster runs only `switch` and `chassis` service, with `central` service being provided externally by another MicroOVN cluster or by some other OVN deployment.

Important

This is not an equivalent of the OVN Interconnect feature. It does not allow connecting two full OVN deployments. Instead it is meant to help with migrations, or in hybrid deployments, where OVN central services are deployed by some other method.

Cluster setup

We will start by deploying the MicroOVN cluster without the `central` service. Our hosts that run MicroOVN are called `node-1`, `node-2` and `node-3`. Setup of the external OVN central cluster is not covered by this how-to. For the sake of this guide we assume that there are OVN Southbound and OVN Northbound services running on their default ports on IPs:

- 10.0.0.1
- 10.0.0.2
- 10.0.0.3

Note

If you already have MicroOVN deployed, see *Service Control* about how to disable `central` service on the running nodes, and *MicroOVN services* regarding implications of removing all `central` services on the running cluster.

Obtain CA certificate and private key for the MicroOVN

Because the MicroOVN enforces encrypted communication between OVN services, we need to ensure that OVN services running on the MicroOVN cluster and on the external OVN central cluster are mutually trusted.

Obtain intermediate CA certificate and private key from the CA that issued certificates for the external OVN central cluster and place it in the `/var/snap/microovn/common/` on the `node-1`, under names `ca.cert` and `ca.key`.

Bootstrap MicroOVN cluster

First we bootstrap the MicroOVN cluster on the `node-1` and generate tokens, so that `node-2` and `node-3` can join.

```
sudo microovn init
```

We will be taken through an initialisation process that will look something like this:

```
Please choose the address MicroOVN will be listening on [default=10.75.224.213]:
Would you like to create a new MicroOVN cluster? (yes/no) [default=no]: yes
Please select comma-separated list services you would like to enable on this node.
↳(central/chassis/switch) or let MicroOVN automatically decide (auto) [default=auto]:
↳switch,chassis
Please choose a name for this system [default=node-1]:
Would you like to define a custom encapsulation IP address for this member? (yes/no)
↳[default=no]:
Would you like to provide your own CA certificate and private key for issuing OVN TLS
↳certificates? (yes/no) [default=no]: yes
Please enter the path to the CA certificate file: /var/snap/microovn/common/ca.cert
Please enter the path to the CA private key file: /var/snap/microovn/common/ca.key
Would you like to add additional servers to the cluster? (yes/no) [default=no]: yes
What's the name of the new MicroOVN server? (empty to exit): node-2
```

(continues on next page)

(continued from previous page)

```
<NODE_2_TOKEN>
What's the name of the new MicroOVN server? (empty to exit): node-3
<NODE_3_TOKEN>
```

Then we join the cluster with node-2 and node-3

```
sudo microovn init
```

Similar dialogue takes us through the joining process, the most notable difference is that we will select no when asked about whether we want to create a new cluster.

```
Please choose the address MicroOVN will be listening on [default=10.75.224.175]:
Would you like to create a new MicroOVN cluster? (yes/no) [default=no]: no
Please select comma-separated list services you would like to enable on this node.
↳(central/chassis/switch) or let MicroOVN automatically decide (auto) [default=auto]:
↳chassis,switch
Please enter your join token: <NODE_TOKEN>
Would you like to define a custom encapsulation IP address for this member? (yes/no)
↳[default=no]:
```

External central configuration

Now that the MicroOVN cluster is deployed without the central services, we can configure it to connect to the external OVN central cluster. On any of the nodes, run following command:

```
sudo microovn config set ovn.central-ips "10.0.0.1,10.0.0.2,10.0.0.3"
```

Verification

To verify that the configuration was successfully applied, we can check that our chassis successfully registered with the OVN Southbound database.

```
sudo microovn.ovn-sbctl show
```

The output should look something like this:

```
Chassis node-1
  hostname: node-1.lxd
  Encap geneve
    ip: "10.75.224.213"
    options: {csum="true"}
Chassis node-3
  hostname: node-3.lxd
  Encap geneve
    ip: "10.75.224.138"
    options: {csum="true"}
Chassis node-2
  hostname: node-2.lxd
  Encap geneve
    ip: "10.75.224.175"
    options: {csum="true"}
```

This proves that our client commands (`ovn-sbctl`) are able to connect to the external OVN central and that `ovn-controller` services running on nodes in the MicroOVN cluster got registered in the external Southbound database.

2.1.8 Configure OVN BGP integration

Configuration of the OVN integration with BGP is a single-command process in the MicroOVN, for more information about what's happening under the hood, see: *Explanation: OVN integration with BGP*.

Enable BGP integration

In this example, we have a host connected to two external networks via interfaces `eth1` and `eth2`.

The only required configuration is specifying the external connection interfaces. Both VRF table number and AS number are optional:

- If VRF is not specified, MicroOVN will automatically select an available VRF table ID
- If AS number is not specified, MicroOVN will automatically generate a unique ASN

Important

Never use interface that provides actual host connectivity for the purpose of OVN BGP integration. These interfaces are meant for the OVN's traffic, they will be assigned to a OVS bridge and you will lose your connection to the host.

Basic usage with automatic configuration

To enable BGP integration with automatic VRF and ASN selection:

```
microovn enable bgp --config ext_connection=eth1,eth2
```

MicroOVN will automatically:

- Select an available VRF table ID
- Generate a unique ASN from the default range (4210000000-4294967294)

Note

The default ASN range reserves the first IDs of the full RFC 6996 private range (4200000000-4209999999) for lower tier network infrastructure components (switches, routers, top-of-rack equipment).

Advanced configuration

You can explicitly specify VRF table ID, ASN, or provide a custom ASN range, in any combination.

Specify both VRF and ASN explicitly:

```
microovn enable bgp --config ext_connection=eth1,eth2 --config vrf=10 --config_
↪ asn=4210000000
```

Provide a custom ASN range for automatic selection:

```
microovn enable bgp --config ext_connection=eth1,eth2 --config asn_range=4210000000-
↪ 4210999999
```

Important

When using a custom ASN range, provide one sufficiently large to avoid collisions between cluster members. Take into account that the ASN that would be allocated for a certain cluster member will never be reused by another, even if the first one is removed from the cluster.

You will receive positive confirmation message in the CLI and the setup is done.

Inspect the changes

If you are interested in changes that the MicroOVN made to the system, We can inspect the new interfaces that were created.

```
ip link
```

The output will show that we have a new VRF device and two veth pairs for the BGP control-plane traffic (one pair for each external interface):

```
<snipped preceding output>
13: ovnvrf10: <NOARP,MASTER,UP,LOWER_UP> mtu 65575 qdisc noqueue state UP mode DEFAULT_
↪group default qlen 1000
   link/ether 2e:25:b7:f0:f7:21 brd ff:ff:ff:ff:ff:ff
14: veth1-brg@veth1-bgp: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master_
↪ovs-system state UP mode DEFAULT group default qlen 1000
   link/ether 46:d7:9e:a5:81:fa brd ff:ff:ff:ff:ff:ff
15: veth1-bgp@veth1-brg: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master_
↪ovnvrf10 state UP mode DEFAULT group default qlen 1000
   link/ether 02:0e:da:a6:c4:28 brd ff:ff:ff:ff:ff:ff
16: veth2-brg@veth2-bgp: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master_
↪ovs-system state UP mode DEFAULT group default qlen 1000
   link/ether ba:1e:35:32:8b:36 brd ff:ff:ff:ff:ff:ff
17: veth2-bgp@veth2-brg: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master_
↪ovnvrf10 state UP mode DEFAULT group default qlen 1000
   link/ether 02:66:5b:b1:78:6e brd ff:ff:ff:ff:ff:ff
<snipped remaining output>
```

And since we requested auto-configuration of BGP daemon, we can check the BIRD configuration found at `/var/snap/microovn/common/data/bird/bird.conf`. There should be two “bgp” instances instances.

```
<snipped preceding output>
protocol bgp microovn_eth2 {
    router id 192.0.2.10;
    interface "veth2-bgp";
    vrf "ovnvrf10";
    local as 4210000000;
    neighbor range fe80::/10 external;
    dynamic name "dyn_microovn_eth2_";
<snipped remaining output>
```

and

```
<snipped preceding output>
protocol bgp microovn_eth1 {
```

(continues on next page)

(continued from previous page)

```
router id 192.0.2.10;
  interface "veth1-bgp";
  vrf "ovnvrf10";
  local as 42100000000;
  neighbor range fe80::/10 external;
  dynamic name "dyn_microovn_eth1_";
<snipped remaining output>
```

Note

Note that for then neighbour configuration, we are not using the names of actual physical interfaces (e.g. eth1), but the names of the interfaces that were created for BGP redirect (e.g. eth1-bgp)

If there are BGP neighbours already running and configured on the external networks, you can validate that they successfully established connections:

```
microovn.birdc show protocols
```

The output should contain established BGP sessions.

```
<snipped preceding output>
microovn_eth1 BGP      ---      start  15:21:14.086  Passive
microovn_eth2 BGP      ---      start  15:21:14.086  Passive
dyn_microovn_eth1_1 BGP  ---      up      15:37:34.578  Established
dyn_microovn_eth2_1 BGP  ---      up      15:38:00.689  Established
<snipped remaining output>
```

Manual BGP daemon configuration

In case that the automatic BIRD configuration provided by MicroOVN does not suit your needs, you can just use the `--manual-bgpd-config` flag when enabling BGP, for example:

```
microovn enable bgp --config ext_connection=eth1,eth2 --manual-bgpd-config
```

With this flag, MicroOVN won't configure the built-in BIRD daemon, allowing you to perform manual configuration or use entirely different BGP daemon.

Disable BGP integration

To disable BGP integration, simply run:

```
microovn disable bgp
```

This will remove all VRF tables, virtual interfaces, OVS bridges, Logical Switches and Logical Routers that were created when the integration was enabled.

MicroOVN will also backup and reset BIRD startup configuration. The current configuration file will be backed up in the same directory under name `bird.conf_<unix_timestamp>` and then replaced with the default BIRD configuration.

2.2 Tutorials

These tutorials provide step-by-step instructions for common goals. They do not assume special Linux knowledge nor any particular understanding of OVN.

2.2.1 Single-node

This tutorial shows how to install MicroOVN in the simplest way possible.

Caution

A single-node OVN cluster does not have any redundancy (service failover).

Install the software

Install MicroOVN on the designated node with the following command:

```
sudo snap install microovn
```

Initialise the cluster

```
microovn cluster bootstrap
```

Manage the cluster

You can interact with OVN using its native commands due to automatically created snap aliases, for example, to show the contents of the OVN Southbound database:

```
ovn-sbctl show
```

2.2.2 Multi-node

This tutorial shows how to install a 3-node MicroOVN cluster.

One big advantage of a multi-node cluster is that it provides redundancy (service failover). A 3-node deployment can tolerate up to one node failure.

Requirements

You will need three (virtual or physical) machines that can communicate with each other over the network. They will be known here as **node-1**, **node-2**, and **node-3**.

Install the software

Install MicroOVN on **each** of the designated nodes with the following command:

```
sudo snap install microovn
```

Initialise the cluster

On **node-1**, initialise the cluster:

```
microovn cluster bootstrap
```


Minor version upgrades

Dedicated major version channels of MicroOVN (e.g. `24.03/stable`) will automatically receive minor version upgrades whenever the minor upgrade for the OVN package becomes available in the Ubuntu repository. They may also receive updates regarding MicroOVN itself in form of features or bugfixes if it's deemed that the backport is warranted.

We try to keep the updates of dedicated stable channels to minimum. Any automatic upgrades within branch are expected to cause only minimal plane outage while services restart.

Major version upgrades

Starting with version `22.03`, OVN introduced concept of LTS releases and started to guarantee the ability to upgrade OVN deployment from one LTS release to next ([rolling upgrades](#)). Therefore, MicroOVN also provides ability to upgrade deployments from one LTS to another. It tries to take as much complexity as possible from the process, but it's still potentially disruptive operation and needs to be triggered by operator manually.

For more information on how to actually perform these upgrades, see [How-To: Major Upgrades](#)

How MicroOVN manages major upgrades

Upgrades without unnecessary downtime constitutes a challenge for distributed systems like OVN.

OVN consists of two distributed databases (`Southbound` and `Northbound`) and multiple processes (e.g. `ovn-controller` or `ovn-northd`) that rely on ability to read and understand data in these databases. Major upgrades of OVN often introduce database schema changes and applying these changes before every host in the deployment is able to understand them can cause unnecessary outage.

Thanks to the backward compatibility guarantees between LTS versions, new versions of `ovn-northd` and `ovn-controller` are able to understand old database schemas. Therefore we can hold back schema upgrades until every cluster member is ready for it. And this is what MicroOVN does. It waits until it receives positive confirmation from every node in the deployment that it's capable of understanding new database schemas, before triggering database schema upgrades for `Southbound` and `Northbound` databases.

2.3.2 OVN integration with BGP

OVN fully enabled support for integration with BGP in version `25.03`. The support includes the ability to redirect BGP control-plane traffic to a specific `Logical Switch Port` and the ability of the `ovn-controller` to advertise and learn routes via a [Linux VRF](#).

BGP control plane

To avoid implementing the BGP protocol inside the OVN, the upstream project decided to provide a way to redirect the BGP control plane traffic from a `Logical Router Port` to a `Logical Switch Port`. As a result, an off-the-shelf BGP daemon can be run inside a VRF, bound to the `Logical Switch Port`. Routing information is confined inside the VRF, leaving the routing table of the main host unaffected. And finally, the BGP daemon appears to the rest of the network as if it was bound to the `Logical Router Port`.

The last point is important for the purpose of hardware offload and it also allows us to use “BGP unnumbered” with BGP authentication. If the BGP daemon acts as if it was bound to the `Logical Router Port`, it advertises its routes with the next hop address of the `Logical Router Port`. The data-plane traffic can be then accelerated via the hardware offload without any further intervention.

Route advertisement and learning via VRF

The `ovn-controller` is capable of maintaining VRFs and using them to learn and advertise routes.

When the chassis binds a `Logical Router Port` configured to advertise or learn routes, it creates a VRF and inserts the routes to the VRF table. A BGP daemon can be configured to use the OVN's VRF table, and announce the routes

to its peers. Conversely, when the BGP daemon learns routes from its peers, it inserts them to the VRF, from which they are picked up by the `ovn-controller` and learned by the OVN.

What MicroOVN sets up

MicroOVN can simplify the BGP integration setup described in the previous sections. For more information on how to do it, see: *How-To: Configure OVN BGP integration*

To fully set up BGP redirection, MicroOVN requires following:

- one or more physical interfaces that provide connectivity to the external networks
- VRF table ID that the OVN will create and to which the internal routes will be redistributed.
- (Optional) AS number that will be used by the BGP daemon to identify itself. Note that if this is not provided, the BGP daemon won't be configured. You can choose to omit the ASN if you wish to configure the BGP daemon manually.

With this information provided, MicroOVN will then set up an OVN Logical Router that will act as a gateway to the external networks. This is going to be a “gateway router” with name `lr-<hostname>-microovn`.

Each of the provided physical interfaces will be plugged to its own OVS bridge and connected to a unique Logical Switch. The bridge will be called `br-<interface_name>` and the switch will be `ls-<hostname>-<interface_name>`. The Logical Router will then be connected to the Logical Switches via ports named `lrp-<hostname>-<interface_name>`. The router ports are not configured with any IP address and rely on IPv6 link local address to talk to the hosts on the external network.

After the external connectivity is set up, MicroOVN will create Logical Switch Port named `lsp-<hostname>-<interface_name>-bgp` in each Logical Switch. This is the logical port to which the BGP traffic will be redirected. On the system interface level, the redirected traffic is handled by a veth pair. Ends of this pair are named `v<interface_name>-brg` and `v<interface_name>-bgp`. The `-brg` end is plugged into the OVS integration bridge and bound to the above mentioned Logical Switch Port. The `-bgp` end is plugged to the VRF, where the BGP daemon can be bound to it.

MicroOVN will then configure required OVN options.

On the Logical Router that provides the external connectivity:

- `dynamic-routing-vrf-id` set to the value of the VRF table.
- `dynamic-routing` set to `true`

On each Logical Router Port plugged to the external network:

- `dynamic-routing-maintain-vrf` set to `true`
- `dynamic-routing-redistribute` set to `nat,lb`
- `dynamic-routing-port-name` set to a unique identifier that is used as a key in the `dynamic-routing-port-mapping` in the OVS database
- `routing-protocols` set to `BGP,BFD`
- `routing-protocol-redirect` set to the name of the Logical Switch Port to which the traffic will be redirected

In the local Open vSwitch database:

- External ID `dynamic-routing-port-mapping` in the `Open_vSwitch` table that maps `dynamic-routing-port-name` option on Logical Router Ports to system interfaces on which the routes are learned

Bundled BGP daemon

MicroOVN comes bundled with [BIRD Routing Daemon](#) that implements BGP and BFD protocols. Once the BGP integration is enabled, BIRD can be used to listen on the newly created interfaces in the VRF and form connections with neighbours on the external networks.

Automatic daemon configuration

MicroOVN is capable of automatically configuring BIRD's BGP and BFD services. If user provides `asn config` option when enabling BGP in MicroOVN, it will configure BIRD to listen on each "BGP redirect" system interface. This is a very opinionated configuration that uses "BGP unnumbered" mode for automatic neighbour discovery. In effect it looks something like this:

```
protocol bgp microovn_eth1 {
  router id 192.0.2.10;
  interface "veth1-bgp";
  vrf "ovnvrf10";
  local as 42100000000;
  neighbor range fe80::/10 external;
  dynamic name "dyn_microovn_eth1_";
  ipv4 {
    next hop self ebgp;
    extended next hop on;
    require extended next hop on;
    import all;
    export filter no_default_v4;
  };
  ipv6 {
    import all;
    export filter no_default_v6;
  };
  bfd {
    # We only want to use BFD for liveness and failure detection if
    # our peer has it configured.
    passive yes;
  };
};
}
```

Note

There's currently a quirk in BIRD's behaviour. When it's configured in the dynamic mode (by using `neighbor range ...`), it doesn't try to discover any neighbours on the link.

This means that if you use BIRD in dynamic mode on both ends (in the MicroOVN and on the external network), they will never connect. The solution is to either configure `neighbor` explicitly on either end, or use other routing daemons that do perform active discovery, like [FRR](#).

Example topology

Below is a diagram of an example topology. It's a single MicroOVN node "movn1", connected to two external networks via physical interfaces "eth1" and "eth2".

2.4 Reference

MicroOVN reference material is specific to the MicroOVN project. It does not cover upstream OVN/OVS topics.

2.4.1 Configuration

MicroOVN exposes configuration options that can be changed by the user via the `microovn config` subcommands. These include:

- `microovn config set` - Set or update value of the config option
- `microovn config get` - Print value the config option
- `microovn config delete` - Remove the configuration option completely
- `microovn config list` - List allowed configuration keys

Below is the list of available configuration options.

`ovn.central-ips`

Key	<code>ovn.central-ips</code>
Type	String
Scope	Cluster
Description	Comma-separated list of IP addresses of the OVN central cluster
Example	<code>10.0.0.1,10.0.0.2,10.0.0.3</code>

By default, MicroOVN assumes that it manages the whole OVN stack from the OVN Northbound database down to the Open vSwitch. This option can be used to change that. It instructs MicroOVN to connect to an external OVN central cluster (OVN Northbound and OVN Southbound databases). Setting this option causes:

- `ovn-controller` on all nodes with `chassis` service enabled to connect and register to the OVN Southbound database specified by this option
- `ovn-nbctl` and `ovn-sbctl` client commands to connect to their respective database specified by this option.

This option is applied cluster-wide. Setting it on any of the nodes will apply necessary changes across the whole MicroOVN cluster.

Prerequisites

There are few things that need to be taken into consideration when applying this configuration option.

Certificate Authority match

Since MicroOVN enforces encrypted communication between clients and services, the external OVN central cluster needs to use SSL/TLS on its database and API endpoints. The certificates used by the external cluster also need to be trusted by the clients in the MicroOVN cluster (and vice versa). In practice this means obtaining an intermediate CA certificate and key from the CA that issued certificates for the external cluster, and setting it as a CA certificate/key in the MicroOVN. See *Working with TLS* about how to set user-provided CA certificate and key in the MicroOVN.

(Optional) Disable internal OVN central cluster

While it's not strictly necessary, if the MicroOVN deployment is using an external OVN central cluster, it is usually unnecessary for any of the internal nodes to actually run the `central` service. If any of your nodes still run the `central` service, you can disable it. See *MicroOVN services* and *Service Control* about how to do it.

2.4.2 Cryptography

Transport layer security (TLS)

All network endpoints exposed by MicroOVN services are secured using multiple components of the [TLS protocol](#), including encryption, authentication and integrity. Through the use of the [Ubuntu OpenSSL](#) packages, TLS versions below 1.2 are disabled for security reasons.

There are two self-signed certificate authorities in use, one for the [MicroCluster](#) based `microovnd` daemon, another for the [OVN](#) daemons. These are initialised during the initial bootstrap of the cluster.

Keys are generated using a 384 bit [Elliptic Curve](#) algorithm often referred to as P-384.

MicroOVN's Go code uses package `crypto` from standard library to parse, generate and validate TLS certificates and associated cryptographic keys.

Both sets of daemons are by default configured to make use of TLS to encrypt on the wire communication, as well as using certificate data for authenticating and verifying remote peers, ensuring only trusted components can participate in the cluster.

User interaction

MicroOVN exposes limited actions for user to interact with TLS certificates used by the OVN services. Note that no mechanism is provided to interact with certificates used internally by MicroOVN API endpoints. For more information about how to manage OVN certificates, please see [Working with TLS](#), specifically sections:

- [Re-issue certificates](#)
- [Manage Certificate Authority](#)

OVN Certificate lifecycle

OVN service certificates that are automatically provisioned by MicroOVN have the following lifespans:

- CA certificate: 10 years
- OVN service/client certificate: 2 years

MicroOVN runs daily checks for certificate lifespan validity. When a certificate is within 10 days of expiration, it will be automatically renewed.

Note

CA certificate is automatically renewed only if it's automatically generated by the MicroOVN. User-supplied CA certificate is not automatically renewed and needs to be manually updated by the user via `certificates set-ca`

Data at rest

While MicroOVN ensures that data is transmitted securely over the network between its various endpoints, data on disk is stored unencrypted under the `/var/snap/microovn/common/data` directory. Access to this directory is restricted to the `root` user only. Potentially sensitive data in there includes:

- OVN and OVS databases
- OVN certificates and private keys

2.4.3 Configuration safety guidelines

MicroOVN is a very opinionated way to deploy OVN. It enforces TLS encryption and authentication on its network endpoints, and it tries to use sane defaults wherever possible. In general, it does not provide many ways to deviate from the standard configuration, but this section will highlight those places where it does and where the security can be improved by the user manually.

BGP integration

MicroOVN provides a way to integrate OVN natively with BGP routers on the external networks. See [Configure OVN BGP integration](#) page for more information. When the integration is enabled with the `--asn` option specified, MicroOVN will auto-configure a [BIRD 3](#) BGP service to listen on connections from the physical external network. This auto-configured BGP daemon has a very lax security settings, most importantly it:

- doesn't perform peer authentication (see [RFC 2385](#))
- doesn't employ RPKI to validate route advertisements (see [RFC 6480](#))
- doesn't apply any route filtering on learned routes
- does connect to the first peer it finds on the external link

BGP security is a very broad topic that's out of scope for this document, but the above points should cover basics when deploying BGP daemons in an environment where the peers can't be necessarily trusted.

If the user desires any of the above security features, they are advised to omit the `--asn` option when enabling the BGP integration. This will allow them to bind any external BGP daemon to the interface inside the VRF created by the MicroOVN. Then they will be able to tailor the daemon configuration to their specific security needs.

2.4.4 MicroOVN Release Process

Release Strategy

MicroOVN feature development takes place on the “main” branch.

The main branch has `snapcraft.yaml` set up to use the `base` for the next core version, and a `build-base` set to ‘`devel`’, which sources stage packages from the most recent Ubuntu development release. The test suite will automatically handle installing the `base` from the ‘`edge`’ channel when required.

Stable MicroOVN releases follow the [Ubuntu release cycle](#), and a new stable version is made shortly after each new Ubuntu LTS release.

The [stable branches](#) are named “branch-YY.MM”, where the numbers come from the corresponding upstream OVN version string, for example: “branch-24.03”.

Release Numbering

The main component of the MicroOVN snap is OVN, consequently the main component of the snap version string come from the upstream version string of the OVN binary embedded in the snap.

The binaries in the snap are sourced from the deb package in the Ubuntu version corresponding to the Ubuntu Core build base, typically the most recent Ubuntu LTS release.

Our [build pipeline](#) is configured in Launchpad, and the [MicroOVN snap recipes](#) are configured to automatically build and publish the snap for supported channels. Builds are triggered whenever relevant packages in the source Ubuntu release change, or when the relevant branch in the [MicroOVN GitHub repository](#) changes.

To allow quick identification of the snap artefact in use, an abbreviated commit hash from the `microovn` Git repository, is appended to the version string.

The full package version string for all embedded packages can be retrieved by issuing the `microovn --version` command on a system with the snap installed.

Stable Branches

We go out of our way to embed logic in the product itself, its test suite and CI pipeline to avoid manual effort on each new release.

Steps to cut a stable branch:

1. Create a PR named “Prepare for YY.MM” that contains two (or more) commits.
 - First commit
 - Set base to a stable version of core and remove any `build-base` statements.
 - Pin any parts with `source-type git` to the most recent stable version available.
 - Second commit
 - Set base back to a edge version of core (when available), and add a `build-base` statement with ‘devel’ as value.
 - Unpin any parts with `source-type git`.
2. Review and merge as separate commits.
3. Create branch `branch-YY.mm` using the first commit from step 1 as base.

Build pipeline

Steps to set up a build pipeline:

1. Go to [Launchpad MicroOVN code repository](#) and ensure required branches have been imported.
2. Create new [MicroOVN snap package recipe](#) make sure to populate fields:
 - Owner: `ubuntu-ovn-eng`.
 - Git repository and branch.
 - Processors: `amd64, arm64, ppc64el, riscv64, s390x`.
 - Automatically build when branch changes.
 - Automatically upload to store.
 - Track that corresponds with branch.
 - Risk: `edge`.

2.4.5 Security process

What is a vulnerability?

All vulnerabilities are bugs, but not every bug is a vulnerability. Vulnerabilities compromise one or more of:

- Confidentiality (personal or corporate confidential data).
- Integrity (trustworthiness and correctness).
- Availability (uptime and service).

If in doubt, please use the process for [reporting a vulnerability](#), and we will assess whether your report is in fact a security vulnerability, or if it should be reported as a bug using the normal bug process.

Reporting a vulnerability

To report a security issue, please file a [Private Security Report on Launchpad](#) (to ensure the information stays private, change the “This bug contains information that is:” from “Public” to “Private Security” when filing), or email security@ubuntu.com with a description of the issue, the steps you took to create the issue, affected versions, and, if known, mitigations for the issue.

The [Ubuntu Security disclosure and embargo policy](#) contains more information about what you can expect when you contact us and what we expect from you.

Product lifetime

The main components of MicroOVN, Open vSwitch (OVS) and Open Virtual Network (OVN), comes from the Ubuntu distribution. Releases of MicroOVN in stable *MicroOVN snap channels and upgrades* that align with Ubuntu Long Term Support (LTS) releases, receive the same level of support throughout the lifetime of the corresponding Ubuntu LTS release. Please refer to the [Ubuntu lifecycle and release cadence](#) documentation for more information.

Tracking vulnerabilities

Vulnerabilities, their status, and the state of the analysis or response will all be tracked through the [Ubuntu CVE tracker](#).

Responding to vulnerabilities

Vulnerabilities are classified by [priority](#), and the MicroOVN project guarantees response to all High and Critical severity vulnerabilities, as well as any [Known Exploited Vulnerability](#).

Security updates will be made available to consumers of stable *MicroOVN snap channels and upgrades* that align with supported Ubuntu Long Term Support (LTS) releases.

The MicroOVN snap is automatically rebuilt by Launchpad whenever there is an update to the underlying packages in the Ubuntu distribution.

Updated versions of the snap will be put through the MicroOVN functional test suites before being promoted to stable *MicroOVN snap channels and upgrades*.

Information about new builds are made available through the [Snap store](#).

Responsible disclosure

We follow the [Ubuntu Security disclosure and embargo policy](#). Please refer to the section on [reporting a vulnerability](#).

2.4.6 MicroOVN services

MicroOVN functionality is separated into distinct services that can be easily controlled via `microovn enable` and `microovn disable`.

This page presents a list of all MicroOVN services. Their descriptions are for reference only - the user is not expected to interact directly with these services.

Handling services with enable/disable

The status of all services is displayed by running:

```
microovn status
```

central service

This is responsible for the OVN Southbound and Northbound database control. The database is clustered and uses the [RAFT](#) algorithm for consensus it can handle $(n-1)/2$ failures, where n is the number of nodes.

Central is enabled on a new node whenever there are less than 3 nodes running the central services

This service controls the following *Snap services*:

- `microovn.ovn-ovsdb-server-nb`
- `microovn.ovn-ovsdb-server-sb`
- `microovn.ovn-northd`

Caution

If the last instance of the central service is disabled via the `microovn disable central --allow-disable-last-central` command, the Southbound and Northbound databases are removed completely. Subsequent call to `microovn enable central` will re-initialise new, empty, databases.

The contents of the previously removed databases can be found in the backup directory (`/var/snap/microovn/common/backup_<timestamp>`) on the last node that disabled the central service.

chassis service

This service controls the `ovn-controller` daemon, which is OVN's agent on each hypervisor and software gateway. It is a distributed component running on the side of every Open vSwitch instance. It is enabled by default.

The snap service this controls is `microovn.chassis`

switch service

This service Open vSwitch and ensures its running properly. Much like chassis it is enabled by default.

The snap service this controls is `microovn.switch`

Snap services

The status of all services is displayed by running:

```
snap services microovn
```

`microovn.chassis`

This service maps directly to the `ovn-controller` daemon.

`microovn.daemon`

The main MicroOVN service/process that manages all the other processes. It also handles communication with other MicroOVN cluster members and provides an API for the `microovn` client command.

`microovn.ovn-ovsdb-server-nb`

This service maps directly to the OVN Northbound database/service.

microovn.ovn-northd

This service maps directly to the `ovn-northd` daemon.

microovn.ovn-ovsdb-server-sb

This service maps directly to the OVN Southbound database/service.

microovn.refresh-expiring-certs

This service is a recurring process that runs once a day between `02:00` and `02:30`. It triggers TLS certification reissue for certificates that are nearing the expiration. For more information see the [certificates lifecycle](#).

microovn.switch

This services maps directly to the `ovs-vswitchd` daemon.

2.4.7 Automatic Aliases

MicroOVN is distributed by `snap`, which has automatic aliases for OVN and OVS binaries. You can view these with the `snap aliases` command:

```
snap aliases microovn
```

Command	Alias	Notes
<code>microovn.ovn-appctl</code>	<code>ovn-appctl</code>	-
<code>microovn.ovn-nbctl</code>	<code>ovn-nbctl</code>	-
<code>microovn.ovn-sbctl</code>	<code>ovn-sbctl</code>	-
<code>microovn.ovn-trace</code>	<code>ovn-trace</code>	-
<code>microovn.ovs-appctl</code>	<code>ovs-appctl</code>	-
<code>microovn.ovs-dpctl</code>	<code>ovs-dpctl</code>	-
<code>microovn.ovs-ofctl</code>	<code>ovs-ofctl</code>	-
<code>microovn.ovs-vsctl</code>	<code>ovs-vsctl</code>	-
<code>microovn.ovsdb-client</code>	<code>ovsdb-client</code>	-
<code>microovn.ovsdb-tool</code>	<code>ovsdb-tool</code>	-

Further inspection can be done by inspecting the files themselves:

```
ls $(which ovn-nbctl) -l
```

```
lrwxrwxrwx 1 root root 18 Nov 28 15:25 /snap/bin/ovn-nbctl -> microovn.ovn-nbctl
```

These aliases are not related to the MicroOVN `snap` version and are managed by the store. All installations, done through the `snap` store, should have access to these aliases. This does mean if you install a locally built version of MicroOVN, these aliases are not created for you.

Note

Netplan requires MicroOVN to have these aliases in order to consume OVS through MicroOVN, which is essential for enabling and configuring BGP. So if you are using a locally built MicroOVN, you must manually create at least the alias for `ovs-vsctl`.

2.5 Contribute to MicroOVN

As an open source project, we welcome contributions of any kind. These can range from bug reports and code reviews, to significant code or documentation features.

If you'd like to contribute, you will first need to sign the Canonical contributor agreement. This is the easiest way for you to give us permission to use your contributions. In effect, you're giving us a license, but you still own the copyright — so you retain the right to modify your code and use it in other projects.

Please review and sign the [Canonical contributor licence agreement](#).

2.5.1 Contributor guidelines

- Each commit should be a logical unit.
- Each commit should pass tests individually to allow bisecting.
- Each commit must be signed. (See [GitHub documentation about commit signing](#))
- The commit message should focus on WHY the change is necessary, we get the what and how by looking at the code.
- Include a Signed-off-by [tag](#) in the commit message. (See [Git sign-off documentation](#))
- When a patch has been created with the assistance of an AI tool, include a Assisted-by [tag](#) to disclose that fact. The author of the patch remains fully responsible for the content.
- MicroOVN uses [Launchpad](#) for tracking bugs. If you encounter any issue, or have a feature suggestion. Feel free to [open a bug report](#).
- MicroOVN makes use of the GitHub Pull Request workflow. There is no meaningful way to manage interdependencies between GitHub PRs, so we expect dependent changes proposed in a single PR reviewed and merged as separate commits.
- A proposal for change is not complete unless it contains updates to documentation and tests.

2.5.2 Next Steps

Contribute to our code

This page covers topics on how to make/build/test your changes to the MicroOVN source code.

Get the source code

MicroOVN development happens on GitHub. You can find, and contribute to, its source code in in our [GitHub repository](#).

Build and install MicroOVN from source

Build requirements

MicroOVN is distributed as a snap and the only requirements for building it are Make and snapcraft. You can install them with:

```
sudo apt install make
sudo snap install snapcraft --classic
```

Snapcraft requires LXD to build snaps. So if your system does not have LXD installed and initiated, you can check out either [LXD getting started guides](#) or go with following default setup:

```
sudo snap install lxd
lxd init --auto
```

Build MicroOVN

To build MicroOVN, go into the repository's root directory and run:

```
make
```

This will produce the `microovn.snap` file that can be then used to install MicroOVN on your system.

Adjust build parameters

`snapcraft.yaml` is by nature a very static build recipe that does not allow build-time modification without changing the file itself. To achieve some level of control over MicroOVN builds, we are using a `microovn/build-aux/environment` file that is loaded and during the build process. Environment variables defined in this file can influence properties of the final build. Currently supported variables are:

- `MICROOVN_COVERAGE` (default: `no`) - When set to `yes`, MicroOVN binaries will be built with coverage instrumentation and output coverage data into `$SNAP_COMMON/data/coverage`.

Install MicroOVN

Using the `microovn.snap` file created in the previous section, you can install MicroOVN in this way:

```
sudo snap install --dangerous ./microovn.snap
```

Note

If you are building latest MicroOVN from the `main` branch, it's possible that it's using a non-stable core snap as its base. In that case, you may get a message like this:

```
Ensure prerequisites for "microovn" are available (cannot install snap base "core24": ↵
↵no snap revision available as specified)
```

In such a case, you will need to install the required core snap manually from the `edge` risk level. For example:

```
snap install core24 --edge
```

Then repeat the installation step.

You will also need to manually connect required plugs, as `snapped` won't do it automatically for locally installed snaps.

```
for plug in firewall-control \
    hardware-observe \
    hugepages-control \
    network-control \
    network-setup-control \
    openvswitch-support \
    process-control \
    system-trace; do \
    sudo snap connect microovn:$plug;done
```

Note

Once this is done you will need to restart running MicroOVN processes as certain newly added permissions do not apply to running process.

```
snap restart microovn
```

To verify that all the required plugs are correctly connected to their slots, you can run:

```
snap connections microovn
```

An example of correctly connected connected plugs would look like this:

Interface	Plug	Slot	Notes
content	-	microovn:ovn-certificates	-
content	-	microovn:ovn-chassis	-
content	-	microovn:ovn-env	-
firewall-control	microovn:firewall-control	:firewall-control	manual
hardware-observe	microovn:hardware-observe	:hardware-observe	manual
hugepages-control	microovn:hugepages-control	:hugepages-control	manual
microovn	-	microovn:microovn	-
network	microovn:network	:network	-
network-bind	microovn:network-bind	:network-bind	-
network-control	microovn:network-control	:network-control	manual
openvswitch-support	microovn:openvswitch-support	:openvswitch-support	manual
process-control	microovn:process-control	:process-control	manual
system-trace	microovn:system-trace	:system-trace	manual

And if the plugs are not connected, the output would look like this:

Interface	Plug	Slot	Notes
content	-	microovn:ovn-certificates	-
content	-	microovn:ovn-chassis	-
content	-	microovn:ovn-env	-
firewall-control	microovn:firewall-control	-	-
hardware-observe	microovn:hardware-observe	-	-
hugepages-control	microovn:hugepages-control	-	-
microovn	-	microovn:microovn	-
network	microovn:network	:network	-
network-bind	microovn:network-bind	:network-bind	-
network-control	microovn:network-control	-	-
openvswitch-support	microovn:openvswitch-support	-	-
process-control	microovn:process-control	-	-
system-trace	microovn:system-trace	-	-

Tests

The tests mainly focus on functional validation of MicroOVN and how we build and configure OVN itself.

We expect Go unit tests for pure functions.

For impure functions, i.e. functions with side effects, if you find yourself redesigning interfaces or figuring out how to mock something to support unit tests, then stop and consider the following strategies instead:

1. Extract the logic you want to test into pure functions. When done right the side effect would be increased composability, setting you up for future code reuse.
2. Contain the remaining functions with side effects in logical units that can be thoroughly tested in the integration test suite.

MicroOVN has two types of tests, linter checks and functional tests and this page will show how to run them.

Linter checks

Go code

We make use of [golangci-lint](#) and you can find a list of enabled linters in the `microovn/.golangci.yml` configuration file.

Successfully running the tool requires build dependencies to be installed and build environment variables properly set up.

Developer ergonomics are important to us, and we want the same experience in local development environments as in our gate.

As such we have opted to run [golangci-lint](#) as part of the snap build process as it gives us consistent results in both environments and relieves the developer of the burden of manually installing build dependencies to perform the checks locally.

If you use an IDE with [golangci-lint](#) support and want to utilise it, the tool should automatically discover this configuration. You will however need to install additional build dependencies and set up environment variables to make it work. Refer to the definition of the `microovn` part in `snap/snapcraft.yaml` for more information.

Test code

The prerequisites for running linting on the test code are:

- `make`
- `shellcheck`

You can install them with:

```
sudo apt install make shellcheck
```

To perform linting, go into the repository's root directory and run:

```
make check-lint
```

Functional tests

These tests build the MicroOVN snap and use it to deploy the OVN cluster in LXD containers. This cluster is then used for running functional test suites.

Satisfy the test requirements

There is no need to run tests in dedicated VMs or in isolated environments as all functional tests run inside containers and no changes are made to the host running them.

Ensure that you have installed [Bash Automated Testing System \(BATS\)](#), a software dependency. Due to the reliance on its latest features, MicroOVN uses BATS directly from its source. If you cloned the MicroOVN repository with submodules (using `--recurse-submodules` flag), you are all set and you will have the following **non-empty** directories:

- `.bats/bats-assert/`

- `.bats/bats-core/`
- `.bats/bats-support/`

If they are empty, you can fetch the submodules with:

```
git submodule update --init --recursive
```

Run functional tests

Once you have your environment set up, running tests is just a matter of invoking the appropriate make target. To run all available test suites, use the `check-system` make target:

```
make check-system
```

To run individual test suites you can execute:

```
make tests/<name_of_the_test_suite>.bats
```

Tip

If your hardware can handle it, you can run test suites in parallel by supplying `make` with `-j` argument (e.g. `make check-system -j4`). To avoid interleaving output from these parallel test suites, you can specify the `-O` argument as well.

Control test environment

By default, functional tests will pre-build an LXD image with the MicroOVN snap already installed. This image is then used to spin up test containers, which significantly improves test times, because it avoids installing snap manually on each container. If you want to opt-out of this behaviour, and instead force tests to manually install MicroOVN snap, you can set `MICROOVN_TESTS_USE_SNAP=yes`. In that case, test containers will be based either on `ubuntu:lts` image, or whatever is specified in `MICROOVN_TEST_CONTAINER_IMAGE`. Below are few examples of how these environment variables can be combined.

```
# Default behavior, using pre-built image based on 'ubuntu:lts'
make check-system

# Using pre-built image based on non-default base image
MICROOVN_TEST_CONTAINER_IMAGE="ubuntu:jammy" make check-system

# Using default base image, but forcing tests to install microovn
# snap on each container manually
MICROOVN_TESTS_USE_SNAP=yes make check-system

# Using custom base image and forcing tests to install microovn snap
# on each container manually
MICROOVN_TEST_CONTAINER_IMAGE="ubuntu:jammy" MICROOVN_TESTS_USE_SNAP=yes make check-
↪system
```

Run tests on remote LXD server

Making use of [LXD remotes](#) to spawn containers on a remote cluster or server is supported through the use of the `LXC_REMOTE LXD` environment variable.

```
export LXC_REMOTE=microcloud
make check-system
```

Test coverage information

When MicroOVN build is configured with the code coverage support via `microovn/build-aux/environment` file (see [Adjust build parameters](#) section for more info), system tests can collect coverage data. All you need to do is export `MICROOVN_COVERAGE_ENABLED=yes` environment variable. Example .. code-block:: none

```
# Run all test suites with code coverage export MICROOVN_COVERAGE_ENABLED=yes make check-system
```

You can find collected data in the `.coverage/` directory, where it's organised in a `<test_name>/<container_name>/coverage` structure. For more information about the coverage data format and what you can do with it, see [Go Coverage Documentation](#).

Clean up

Functional test suites will attempt to clean up their containers. However, if a test crashes, or if it's forcefully killed, you may need to do some manual cleanup.

If you suspect that tests did not clean up properly, you can list all containers with:

```
lxc list
```

Any leftover containers will be named according to: `microovn-<test_suite_name>-<number>`. You can remove them with:

```
lxc delete --force <container_name>
```

Contribute to our documentation

Contributing to documentation is a great way to get started as a contributor to open-source projects, no matter your level of experience. Before you start, please review our [general guide on contributing to MicroOVN](#).

MicroOVN is growing rapidly, and we would love your help. We welcome, encourage and appreciate contributions from our user community in the form of suggestions, fixes and constructive feedback. Whether you are new to MicroOVN and want to highlight something you found confusing, or you're an expert and want to create a how-to guide to help others, we will be happy to work with you to make our documentation better for everybody.

The MicroOVN documentation is hosted in the [GitHub repository](#), alongside the rest of the codebase, and is published on [Read the Docs](#).

Diátaxis

Our documentation content, style and navigational structure follows the [Diátaxis](#) systematic framework for technical documentation authoring. This framework splits documentation pages into tutorials, how-to guides, reference material and explanatory text:

- **Tutorials** are lessons that accomplish specific tasks through *doing*. They help with familiarity and place users in the safe hands of an instructor.

- **How-to guides** are recipes, showing users how to achieve something, helping them get something done. A *How-to* has no obligation to teach.
- **Reference** material is descriptive, providing facts about functionality that is isolated from what needs to be done.
- **Explanation** is discussion, helping users gain a deeper or better understanding of MicroOVN, as well as how and why MicroOVN functions as it does.

To learn more about our Diátaxis strategy, see [Diátaxis, a new foundation for Canonical documentation](#).

Improving our documentation and applying the principles of Diátaxis are on-going tasks. There's a lot to do, and we don't want to deter anyone from contributing to our docs. If you don't know whether something should be a tutorial, how-to, reference doc or explanatory text, either ask on the forum or publish what you're thinking. Changes are easy to make, and every contribution helps.

Open Documentation Academy

A key aim of [Canonical Open Documentation Academy](#) initiative is to help lower the barrier into successful open-source software contribution, by making documentation into the gateway, and it's a great way to make your first open source documentation contributions to MicroOVN.

But even if you're an expert, we want the academy to be a place to share knowledge, a place to get involved with new developments, and somewhere you can ask for help on your own projects.

The best way to get started is with our [documentation task list](#). Take a look, bookmark it, and see our [Getting started](#) guide for next steps.

Stay in touch either through the task list, or through one of the following locations:

- Our [documentation discussion forum](#) on the Ubuntu Community Hub.
- In the [documentation Matrix room](#) for interactive chat.
- [Follow us on Fosstodon](#) for the latest updates and events.

If you'd like to ask us questions outside of our public forums, feel free to email us at docsacademy@canonical.com.

In addition to the above, we have a weekly Community Hour starting at 16:00 UTC every Friday. Everyone is welcome, and links and comments can be found on the [forum post](#).

Finally, subscribe to our [Documentation event calendar](#). We'll expand our Community Hour schedule and add other events throughout the year.

Agreements

Everyone involved with CODA needs to follow the words and spirit of the [Ubuntu Code of Conduct v2.0](#). You must also sign and agree to the Canonical CLA.

Identifying suitable task

The academy uses issue labels to give the contributor a glimpse into the task and what it requires, including the type of task, skills or level of expertise required, and even the size estimation for the task. You can find tasks of all sizes in the academy issues list.

From small tasks, such as replacing outdated terminology, checking for broken links, testing a tutorial or ensuring adherence to the [Canonical documentation style guide](#); to medium-sized tasks like, converting documentation from one format to another, or migrating the contents of a blog post into the official documentation; to more ambitious tasks, such as adding a new *How-to* guide, restructuring a group of documents, or developing new tests and automations.

Completing and closing tasks

When a task has been completed to your satisfaction, we'll ask the contributor whether they would prefer to merge their work into your project themselves, or leave this to the project.

Recognition

After successfully completing a task, we'll give credit to the contributor and share their success in our forums, on the pages themselves, and in our news updates and release notes.

Guidance for writing

Consistency of writing style in documentation is vital for a good user experience. To accommodate our audience with a huge variation in experience, we:

- write with our target audience in mind
- write inclusively and assume very little prior knowledge of the reader
- link or explain phrases, acronyms and concepts that may be unfamiliar, and if unsure, err on the side of caution
- adhere to the style guide

Language

MicroOVN documentation currently uses British (GB) English. However, Canonical recently switched to US English. It is our aim to switch to the US English as well. Until we completely switch over, the contributions should continue to use British English.

There are many small differences between UK and US English, but for the most part, it comes down to spelling. Some common differences are:

- the *ize* suffix in preference to *ise* (e.g. *capitalize* and *capitalise*)
- *our* instead of *or* (as in *color* and *colour*)
- *licence* as both a verb and noun
- *catalog* and *catalogue*
- dates take the format 1 January 2013, 1-2 January 2025 and 1 January - 2 February 2025

We use an automated spelling checker that sometimes throws errors about terms we would like it to ignore:

- If it complains about a file name or a command, enclose the word in double backticks (```) to render it as inline code.
- If the word is a valid acronym or a well-known technical term (that should not be rendered as code), add it to the spelling exception list, `docs/.custom_wordlist.txt` (terms should be added in alphabetical order).

Both methods are valid, depending on whether you want the term to be rendered as normal font, or as inline code (monospaced).

Acronyms

Acronyms should always be capitalised.

They should always be expanded the first time they appear on a page, and then can be used as acronyms after that. E.g. LSP should be shown as Logical Switch Port (LSP), and then can be referred to as LSP for the rest of the page.

Links

The first time you refer to a package or other product, you should make it a link to either that product's website, or its documentation, or its manpage.

Links should be from reputable sources (such as official upstream docs). Try not to include blog posts as references if possible. And, always verify that the links are correct and accurate.

Try to use inline links sparingly. If you have a lot of useful references you think the reader might be interested in, feel free to include a "Further reading" section at the end of the page.

Writing style

Try to be concise and to-the-point in your writing.

It's alright to be a bit lighthearted and playful in your writing, but please keep it respectful, and don't use emoji (they don't render well in documentation, and may not be deemed professional).

It's also good practice not to assume that your reader will have the same knowledge as you. If you're covering a new topic (or something complicated) then try to briefly explain, or link to supporting explanations of, the things the typical reader may not know, but needs to (refer to the Diátaxis framework to help you decide what type of documentation you are writing and the level and type of information you need to include, e.g. a tutorial may require additional context but a how-to guide can skip some foundational knowledge - it is safer to assume some prior knowledge).

Documentation source language

MicroOVN uses reStructuredText language (reST) for writing the documentation. You can read about the basic language syntax in the [reStructuredText Primer](#), or you can see our `docs/doc-cheat-sheet.rst` for some examples.

Preview your changes

You can verify that your documentation changes render as you expect them by building the whole documentation set and serve it as a web page locally. To do that, you can

```
# Change your working directory to the "docs/" if you are in the project root directory
cd docs/
# build and serve the documentation as a web page
make serve
```

This will start a local web server that serves the current version of the documentation. If the build was successful, you will see an output like this:

```
The HTML pages are in _build.
cd "_build"; python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

You can either click on the `http` link, or open your browser and manually open the `http://localhost:8000` page. From there you can navigate to the documentation page you changed, and see your changes.

Local Verification

We have a set of tests that need to pass before we can consider documentation contribution, similar to tests we expect to pass for the code. These tests will be executed automatically when you open your pull request on GitHub, but to speed up the submission and approval process, it is recommended that you run them locally before you submit your contribution. The tests are defined in the `docs/Makefile` and to run them, you can:

```
# Change your working directory to the "docs/" if you are in the project root directory
cd docs/
# Run spelling check
make spelling
# Run link validation check
make linkcheck
# Run inclusive language check
make woke
```

If all of the check pass without errors, your contribution is ready for submission.

Thank you

We would like to thank you for spending your time to help make the MicroOVN better. Every contribution, big or small, is important to us, and hopefully a step in the right direction.