DevStack Docs

OpenStack DevStack Team

Jul 22, 2025

CONTENTS

DevStack is a series of extensible scripts used to quickly bring up a complete OpenStack environment based on the latest versions of everything from git master. It is used interactively as a development environment and as the basis for much of the OpenStack projects functional testing.

The source is available at https://opendev.org/openstack/devstack.

Warning

DevStack will make substantial changes to your system during installation. Only run DevStack on servers or virtual machines that are dedicated to this purpose.

CHAPTER

ONE

QUICK START

1.1 Install Linux

Start with a clean and minimal install of a Linux system. DevStack attempts to support the two latest LTS releases of Ubuntu, Rocky Linux 9 and openEuler.

If you do not have a preference, Ubuntu 24.04 (Noble) is the most tested, and will probably go the smoothest.

1.2 Add Stack User (optional)

DevStack should be run as a non-root user with sudo enabled (standard logins to cloud images such as ubuntu or cloud-user are usually fine).

If you are not using a cloud image, you can create a separate *stack* user to run DevStack with

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
```

Ensure home directory for the `stack` user has executable permission for all, as RHEL based distros create it with `700` and Ubuntu 21.04+ with `750` which can cause issues during deployment.

```
$ sudo chmod +x /opt/stack
```

Since this user will be making many changes to your system, it should have sudo privileges:

```
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
$ sudo -u stack -i
```

1.3 Download DevStack

```
$ git clone https://opendev.org/openstack/devstack
$ cd devstack
```

The `devstack` repo contains a script that installs OpenStack and templates for configuration files.

1.4 Create a local.conf

Create a `local.conf` file with four passwords preset at the root of the devstack git repo.

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

This is the minimum required config to get started with DevStack.

Note

There is a sample `local.conf` file under the *samples* directory in the devstack repository.

Warning

Only use alphanumeric characters in your passwords, as some services fail to work when using special characters.

1.5 Start the install

```
$ ./stack.sh
```

This will take 15 - 30 minutes, largely depending on the speed of your internet connection. Many git trees and packages will be installed during this process.

1.6 Profit!

You now have a working DevStack! Congrats!

Your devstack will have installed `keystone`, `glance`, `nova`, `placement`, `cinder`, `neutron`, and `horizon`. Floating IPs will be available, guests have access to the external world.

You can access horizon to experience the web interface to OpenStack, and manage vms, networks, volumes, and images from there.

You can `source openrc` in your shell, and then use the `openstack` command line tool to manage your devstack.

You can *create a VM and SSH into it*.

You can `cd /opt/stack/tempest` and run tempest tests that have been configured to work with your devstack.

You can *make code changes to OpenStack and validate them*.

1.7 Going further

Learn more about our *configuration system* to customize devstack for your needs. Including making adjustments to the default *networking*.

Read *guides* for specific setups people have (note: guides are point in time contributions, and may not always be kept up to date to the latest devstack).

Enable *devstack plugins* to support additional services, features, and configuration not present in base devstack.

Use devstack in your CI with *Ansible roles* and *Jobs* for Zuul V3. Migrate your devstack Zuul V2 jobs to Zuul V3 with this full migration *how-to*.

Get *the big picture* of what we are trying to do with devstack, and help us by *contributing to the project*.

If you are a new contributor to devstack please refer: *So You Want to Contribute*

1.7.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Devstack.

Communication

- IRC channel `#openstack-qa` at OFTC.
- Mailing list (prefix subjects with `[qa][devstack]` for faster responses) http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

Contacting the Core Team

Please refer to the Devstack Core Team contacts.

New Feature Planning

If you want to propose a new feature please read Feature Proposal Process Devstack features are tracked on Launchpad BP.

Task Tracking

We track our tasks in Launchpad.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Launchpad. More info about Launchpad usage can be found on OpenStack docs page

Getting Your Patch Merged

All changes proposed to the Devstack require two `Code-Review +2` votes from Devstack core reviewers before one of the core reviewers can approve the patch by giving `Workflow +1` vote. There are 2 exceptions, approving patches to unblock the gate and patches that do not relate to the Devstacks core logic, like for example old job cleanups, can be approved by single core reviewers.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

The Release Process for QA is documented in QA Release Process.

CHAPTER

TWO

CONTENTS

2.1 Configuration

2.1.1 local.conf

DevStack configuration is modified via the file `local.conf`. It is a modified INI format file that introduces a meta-section header to carry additional information regarding the configuration files to be changed.

A sample is provided in `devstack/samples`

The new header is similar to a normal INI section header but with double brackets (`[[ ... ]]`) and two internal fields separated by a pipe (`|`). Note that there are no spaces between the double brackets and the internal fields. Likewise, there are no spaces between the pipe and the internal fields:

```
'[[' <phase> '|' <config-file-name> ']]'
```

where `<phase>` is one of a set of phase names defined by `stack.sh` and `<config-file-name>` is the configuration filename. The filename is evaled in the `stack.sh` context so all environment variables are available and may be used. Using the project config file variables in the header is strongly suggested (see the `NOVA_CONF` example below). If the path of the config file does not exist it is skipped.

The defined phases are:

- **local** - extracts `localrc` from `local.conf` before `stackrc` is sourced
- **post-config** - runs after the layer 2 services are configured and before they are started
- **extra** - runs after services are started and before any files in `extra.d` are executed
- **post-extra** - runs after files in `extra.d` are executed
- **test-config** - runs after tempest (and plugins) are configured

The file is processed strictly in sequence; meta-sections may be specified more than once but if any settings are duplicated the last to appear in the file will be used.

```
[[post-config|$NOVA_CONF]]
[DEFAULT]
use_syslog = True

[osapi_v3]
enabled = False
```

A specific meta-section `local|localrc` is used to provide a default `localrc` file (actually `.localrc.auto`). This allows all custom settings for DevStack to be contained in a single file. If `localrc` exists it will be used instead to preserve backward-compatibility.

```
[[local|localrc]]
IPV4_ADDRS_SAFE_TO_USE=10.254.1.0/24
ADMIN_PASSWORD=speciale
LOGFILE=$DEST/logs/stack.sh.log
```

Note that `Q_PLUGIN_CONF_FILE` is unique in that it is assumed to *NOT* start with a `/` (slash) character. A slash will need to be added:

```
[[post-config|/$Q_PLUGIN_CONF_FILE]]
```

Also note that the `localrc` section is sourced as a shell script fragment and MUST conform to the shell requirements, specifically no whitespace around = (equals).

2.1.2 openrc

`openrc` configures login credentials suitable for use with the OpenStack command-line tools. `openrc` sources `stackrc` at the beginning (which in turn sources the `localrc` section of `local.conf`) in order to pick up `HOST_IP` and/or `SERVICE_HOST` to use in the endpoints. The values shown below are the default values.

OS_PROJECT_NAME (OS_TENANT_NAME)

Keystone has standardized the term *project* as the entity that owns resources. In some places references still exist to the previous term *tenant* for this use. Also, *project_name* is preferred to *project_id*. OS_TENANT_NAME remains supported for compatibility with older tools.

```
OS_PROJECT_NAME=demo
```

OS_USERNAME

In addition to the owning entity (project), OpenStack calls the entity performing the action *user*.

```
OS_USERNAME=demo
```

OS_PASSWORD

Keystones default authentication requires a password be provided. The usual cautions about putting passwords in environment variables apply, for most DevStack uses this may be an acceptable trade-off.

```
OS_PASSWORD=secret
```

HOST_IP, SERVICE_HOST

Set API endpoint host using `HOST_IP`. `SERVICE_HOST` may also be used to specify the endpoint, which is convenient for some `local.conf` configurations. Typically, `HOST_IP` is set in the `localrc` section.

```
HOST_IP=127.0.0.1
SERVICE_HOST=$HOST_IP
```

OS_AUTH_URL

Authenticating against an OpenStack cloud using Keystone returns a *Token* and *Service Catalog*. The catalog contains the endpoints for all services the user/tenant has access to - including Nova, Glance, Keystone and Swift.

```
OS_AUTH_URL=http://$SERVICE_HOST:5000/v3.0
```

KEYSTONECLIENT_DEBUG, NOVACLIENT_DEBUG

Set command-line client log level to `DEBUG`. These are commented out by default.

```
# export KEYSTONECLIENT_DEBUG=1
# export NOVACLIENT_DEBUG=1
```

2.1.3 Minimal Configuration

While `stack.sh` is happy to run without a `localrc` section in `local.conf`, devlife is better when there are a few minimal variables set. This is an example of a minimal configuration that touches the values that most often need to be set.

- no logging
- pre-set the passwords to prevent interactive prompts
- move network ranges away from the local network (`IPV4_ADDRS_SAFE_TO_USE` and `FLOATING_RANGE`, commented out below)
- set the host IP if detection is unreliable (`HOST_IP`, commented out below)

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
#IPV4_ADDRS_SAFE_TO_USE=172.31.1.0/24
#FLOATING_RANGE=192.168.20.0/25
#HOST_IP=10.3.4.5
```

If the `*_PASSWORD` variables are not set here you will be prompted to enter values for them by `stack.sh`.

Warning

Only use alphanumeric characters in your passwords, as some services fail to work when using special characters.

The network ranges must not overlap with any networks in use on the host. Overlap is not uncommon as

RFC-1918 private ranges are commonly used for both the local networking and Novas fixed and floating ranges.

`HOST_IP` is normally detected on the first run of `stack.sh` but often is indeterminate on later runs due to the IP being moved from an Ethernet interface to a bridge on the host. Setting it here also makes it available for `openrc` to set `OS_AUTH_URL`. `HOST_IP` is not set by default.

`HOST_IPV6` is normally detected on the first run of `stack.sh` but will not be set if there is no IPv6 address on the default Ethernet interface. Setting it here also makes it available for `openrc` to set `OS_AUTH_URL`. `HOST_IPV6` is not set by default.

For architecture specific configurations which differ from the x86 default here, see *arch-configuration*.

2.1.4 Historical Notes

Historically DevStack obtained all local configuration and customizations from a `localrc` file. In Oct 2013 the `local.conf` configuration method was introduced (in review 46768) to simplify this process.

2.1.5 Configuration Notes

Service Repos

The Git repositories used to check out the source for each service are controlled by a pair of variables set for each service. `*_REPO` points to the repository and `*_BRANCH` selects which branch to check out. These may be overridden in `local.conf` to pull source from a different repo for testing, such as a Gerrit branch proposal. `GIT_BASE` points to the primary repository server.

```
NOVA_REPO=$GIT_BASE/openstack/nova.git
NOVA_BRANCH=master
```

To pull a branch directly from Gerrit, get the repo and branch from the Gerrit review page:

```
git fetch https://review.opendev.org/openstack/nova \
    refs/changes/50/5050/1 && git checkout FETCH_HEAD
```

The repo is the stanza following `fetch` and the branch is the stanza following that:

```
NOVA_REPO=https://review.opendev.org/openstack/nova
NOVA_BRANCH=refs/changes/50/5050/1
```

Installation Directory

The DevStack install directory is set by the `DEST` variable. By default it is `/opt/stack`.

By setting it early in the `localrc` section you can reference it in later variables. It can be useful to set it even though it is not changed from the default value.

```
DEST=/opt/stack
```

Logging

Enable Logging

By default `stack.sh` output is only written to the console where it runs. It can be sent to a file in addition to the console by setting `LOGFILE` to the fully-qualified name of the destination log file. A timestamp will be appended to the given filename for each run of `stack.sh`.

```
LOGFILE=$DEST/logs/stack.sh.log
```

Old log files are cleaned automatically if `LOGDAYS` is set to the number of days of old log files to keep.

```
LOGDAYS=2
```

Some coloring is used during the DevStack runs to make it easier to see what is going on. This can be disabled with:

```
LOG_COLOR=False
```

When using the logfile, by default logs are sent to the console and the file. You can set `VERBOSE` to `false` if you only wish the logs to be sent to the file (this may avoid having double-logging in some cases where you are capturing the script output and the log files). If `VERBOSE` is `true` you can additionally set `VERBOSE_NO_TIMESTAMP` to avoid timestamps being added to each output line sent to the console. This can be useful in some situations where the console output is being captured by a runner or framework (e.g. Ansible) that adds its own timestamps. Note that the log lines sent to the `LOGFILE` will still be prefixed with a timestamp.

Logging the Service Output

By default, services run under `systemd` and are natively logging to the systemd journal.

To query the logs use the `journalctl` command, such as:

```
sudo journalctl --unit devstack@*
```

More examples can be found in *Querying Logs*.

Example Logging Configuration

For example, non-interactive installs probably wish to save output to a file, keep service logs and disable color in the stored files.

```
[[local|localrc]]
DEST=/opt/stack/
LOGFILE=$DEST/stack.sh.log
LOG_COLOR=False
```

Database Backend

Multiple database backends are available. The available databases are defined in the lib/databases directory. `mysql` is the default database, choose a different one by putting the following in the `localrc` section:

```
disable_service mysql
enable_service postgresql
```

`mysql` is the default database.

RPC Backend

Support for a RabbitMQ RPC backend is included. Additional RPC backends may be available via external plugins. Enabling or disabling RabbitMQ is handled via the usual service functions and `ENABLED_SERVICES`.

Example disabling RabbitMQ in `local.conf`:

```
disable_service rabbit
```

Apache Frontend

The Apache web server is enabled for services that support via WSGI. Today this means HTTPD and uWSGI but historically this meant HTTPD + mod_wsgi. This historical legacy is captured by the naming of many variables, which include `MOD_WSGI` rather than `UWSGI`.

Some services support alternative deployment strategies (e.g. eventlet). You can enable these `ENABLE_HTTPD_MOD_WSGI_SERVICES` to `False` in your `local.conf`. In addition, each service that can be run under HTTPD + mod_wsgi also has an override toggle available that can be set in your `local.conf`. These are, however, slowly being removed as services have adopted standardized deployment mechanisms and more generally moved away from eventlet.

Example (Swift):

```
SWIFT_USE_MOD_WSGI="True"
```

Example (Heat):

```
HEAT_USE_MOD_WSGI="True"
```

Libraries from Git

By default devstack installs OpenStack server components from git, however it installs client libraries from released versions on pypi. This is appropriate if you are working on server development, but if you want to see how an unreleased version of the client affects the system you can have devstack install it from upstream, or from local git trees by specifying it in `LIBS_FROM_GIT`. Multiple libraries can be specified as a comma separated list.

```
LIBS_FROM_GIT=python-keystoneclient,oslo.config
```

Setting the variable to `ALL` will activate the download for all libraries.

Virtual Environments

Enable the use of Python virtual environments by setting `USE_VENV` to `True`. This will enable the creation of venvs for each project that is defined in the `PROJECT_VENV` array.

Each entry in the `PROJECT_VENV` array contains the directory name of a venv to be used for the project. The array index is the project name. Multiple projects can use the same venv if desired.

```
PROJECT_VENV["glance"]=${GLANCE_DIR}.venv
```

`ADDITIONAL_VENV_PACKAGES` is a comma-separated list of additional packages to be installed into each venv. Often projects will not have certain packages listed in its `requirements.txt` file because they

are optional requirements, i.e. only needed for certain configurations. By default, the enabled databases will have their Python bindings added when they are enabled.

```
ADDITIONAL_VENV_PACKAGES="python-foo, python-bar"
```

A clean install every time

By default `stack.sh` only clones the project repos if they do not exist in `$DEST`. `stack.sh` will freshen each repo on each run if `RECLONE` is set to `yes`. This avoids having to manually remove repos in order to get the current branch from `$GIT_BASE`.

```
RECLONE=yes
```

Upgrade packages installed by pip

By default `stack.sh` only installs Python packages if no version is currently installed or the current version does not match a specified requirement. If `PIP_UPGRADE` is set to `True` then existing required Python packages will be upgraded to the most recent version that matches requirements.

```
PIP_UPGRADE=True
```

Guest Images

Images provided in URLS via the comma-separated `IMAGE_URLS` variable will be downloaded and uploaded to glance by DevStack.

Default guest-images are predefined for each type of hypervisor and their testing-requirements in `stack.sh`. Setting `DOWNLOAD_DEFAULT_IMAGES=False` will prevent DevStack downloading these default images; in that case, you will want to populate `IMAGE_URLS` with sufficient images to satisfy testing-requirements.

```
DOWNLOAD_DEFAULT_IMAGES=False
IMAGE_URLS="http://foo.bar.com/image.qcow,"
IMAGE_URLS+="http://foo.bar.com/image2.qcow"
```

Instance Type

`DEFAULT_INSTANCE_TYPE` can be used to configure the default instance type. When this parameter is not specified, Devstack creates additional micro & nano flavors for really small instances to run Tempest tests.

For guests with larger memory requirements, `DEFAULT_INSTANCE_TYPE` should be specified in the configuration file so Tempest selects the default flavors instead.

KVM on Power with QEMU 2.4 requires 512 MB to load the firmware - QEMU 2.4 - PowerPC so users running instances on ppc64/ppc64le can choose one of the default created flavors as follows:

```
DEFAULT_INSTANCE_TYPE=m1.tiny
```

IP Version

IP_VERSION can be used to configure Neutron to create either an IPv4, IPv6, or dual-stack self-service project data-network by with either IP_VERSION=4, IP_VERSION=6, or IP_VERSION=4+6 respectively.

```
IP_VERSION=4+6
```

The following optional variables can be used to alter the default IPv6 behavior:

```
IPV6_RA_MODE=slaac
IPV6_ADDRESS_MODE=slaac
IPV6_ADDRS_SAFE_TO_USE=fd$IPV6_GLOBAL_ID::/56
IPV6_PRIVATE_NETWORK_GATEWAY=fd$IPV6_GLOBAL_ID::1
```

Note: IPV6_ADDRS_SAFE_TO_USE and IPV6_PRIVATE_NETWORK_GATEWAY can be configured with any valid IPv6 prefix. The default values make use of an auto-generated IPV6_GLOBAL_ID to comply with RFC4193.

Service IP Version

DevStack can enable service operation over either IPv4 or IPv6 by setting SERVICE_IP_VERSION to either SERVICE_IP_VERSION=4 or SERVICE_IP_VERSION=6 respectively.

When set to 4 devstack services will open listen sockets on **0.0.0.0** and service endpoints will be registered using HOST_IP as the address.

When set to 6 devstack services will open listen sockets on :: and service endpoints will be registered using HOST_IPV6 as the address.

The default value for this setting is 4. Dual-mode support, for example 4+6 is not currently supported. HOST_IPV6 can optionally be used to alter the default IPv6 address:

```
HOST_IPV6=${some_local_ipv6_address}
```

Tunnel IP Version

DevStack can enable tunnel operation over either IPv4 or IPv6 by setting TUNNEL_IP_VERSION to either TUNNEL_IP_VERSION=4 or TUNNEL_IP_VERSION=6 respectively.

When set to 4 Neutron will use an IPv4 address for tunnel endpoints, for example, HOST_IP.

When set to 6 Neutron will use an IPv6 address for tunnel endpoints, for example, HOST_IPV6.

The default value for this setting is 4. Dual-mode support, for example 4+6 is not supported, as this value must match the address family of the local tunnel endpoint IP(v6) address.

The value of TUNNEL_IP_VERSION has a direct relationship to the setting of TUNNEL_ENDPOINT_IP, which will default to HOST_IP when set to 4, and HOST_IPV6 when set to 6.

Multi-node setup

See the *multi-node lab guide*

Projects

Neutron

See the *neutron configuration guide* for details on configuration of Neutron

Swift

Swift is disabled by default. When enabled, it is configured with only one replica to avoid being IO/memory intensive on a small VM.

If you would like to enable Swift you can add this to your `localrc` section:

```
enable_service s-proxy s-object s-container s-account
```

If you want a minimal Swift install with only Swift and Keystone you can have this instead in your `localrc` section:

```
disable_all_services
enable_service key mysql s-proxy s-object s-container s-account
```

If you only want to do some testing of a real normal swift cluster with multiple replicas you can do so by customizing the variable `SWIFT_REPLICAS` in your `localrc` section (usually to 3).

You can manually override the ring building to use specific storage nodes, for example when you want to test a multinode environment. In this case you have to set a space-separated list of IPs in `SWIFT_STORAGE_IPS` in your `localrc` section that should be used as Swift storage nodes. Please note that this does not create a multinode setup, it is only used when adding nodes to the Swift rings.

```
SWIFT_STORAGE_IPS="192.168.1.10 192.168.1.11 192.168.1.12"
```

Swift S3

If you are enabling `s3api` in `ENABLED_SERVICES` DevStack will install the s3api middleware emulation. Swift will be configured to act as a S3 endpoint for Keystone so effectively replacing the `nova-objectstore`.

Only Swift proxy server is launched in the systemd system all other services are started in background and managed by `swift-init` tool.

Tempest

If tempest has been successfully configured, a basic set of smoke tests can be run as follows:

```
$ cd /opt/stack/tempest
$ tox -e smoke
```

By default tempest is downloaded and the config file is generated, but the tempest package is not installed in the systems global site-packages (the package install includes installing dependences). So tempest wont run outside of tox. If you would like to install it add the following to your `localrc` section:

```
INSTALL_TEMPEST=True
```

Cinder

The logical volume group used to hold the Cinder-managed volumes is set by `VOLUME_GROUP_NAME`, the logical volume name prefix is set with `VOLUME_NAME_PREFIX` and the size of the volume backing file is set with `VOLUME_BACKING_FILE_SIZE`.

```
VOLUME_GROUP_NAME="stack-volumes"
VOLUME_NAME_PREFIX="volume-"
VOLUME_BACKING_FILE_SIZE=24G
```

When running highly concurrent tests, the default per-project quotas for volumes, backups, or snapshots may be too small. These can be adjusted by setting `CINDER_QUOTA_VOLUMES`, `CINDER_QUOTA_BACKUPS`, or `CINDER_QUOTA_SNAPSHOTS` to the desired value. (The default for each is 10.)

DevStacks Cinder LVM configuration module currently supports both iSCSI and NVMe connections, and we can choose which one to use with options `CINDER_TARGET_HELPER`, `CINDER_TARGET_PROTOCOL`, `CINDER_TARGET_PREFIX`, and `CINDER_TARGET_PORT`.

Defaults use iSCSI with the LIO target manager:

```
CINDER_TARGET_HELPER="lioadm"
CINDER_TARGET_PROTOCOL="iscsi"
CINDER_TARGET_PREFIX="iqn.2010-10.org.openstack:"
CINDER_TARGET_PORT=3260
```

Additionally there are 3 supported transport protocols for NVMe, `nvmet_rdma`, `nvmet_tcp`, and `nvmet_fc`, and when the `nvmet` target is selected the protocol, prefix, and port defaults will change to more sensible defaults for NVMe:

```
CINDER_TARGET_HELPER="nvmet"
CINDER_TARGET_PROTOCOL="nvmet_rdma"
CINDER_TARGET_PREFIX="nvme-subsystem-1"
CINDER_TARGET_PORT=4420
```

When selecting the RDMA transport protocol DevStack will create on Cinder nodes a Software RoCE device on top of the `HOST_IP_IFACE` and if it is not defined then on top of the interface with IP address `HOST_IP` or `HOST_IPV6`.

This Soft-RoCE device will always be created on the Nova compute side since we cannot tell beforehand whether there will be an RDMA connection or not.

Keystone

Multi-Region Setup

We want to setup two devstack (RegionOne and RegionTwo) with shared keystone (same users and services) and horizon. Keystone and Horizon will be located in RegionOne. Full spec is available at: https://wiki.openstack.org/wiki/Heat/Blueprints/Multi_Region_Support_for_Heat.

In RegionOne:

```
REGION_NAME=RegionOne
```

In RegionTwo:

```
disable_service horizon
KEYSTONE_SERVICE_HOST=<KEYSTONE_IP_ADDRESS_FROM_REGION_ONE>
REGION_NAME=RegionTwo
KEYSTONE_REGION_NAME=RegionOne
```

In the devstack for RegionOne, we set REGION_NAME as RegionOne, so region of the services started in this devstack are registered as RegionOne. In devstack for RegionTwo, similarly, we set REGION_NAME as RegionTwo since we want services started in this devstack to be registered in RegionTwo. But Keystone service is started and registered in RegionOne, not RegionTwo, so we use KEYSTONE_REGION_NAME to specify the region of Keystone service. KEYSTONE_REGION_NAME has a default value the same as REGION_NAME thus we omit it in the configuration of RegionOne.

Glance

The default image size quota of 1GiB may be too small if larger images are to be used. Change the default at setup time with:

```
GLANCE_LIMIT_IMAGE_SIZE_TOTAL=5000
```

or at runtime via:

```
openstack --os-cloud devstack-system-admin registered limit set \
  --service glance --default-limit 5000 --region RegionOne image_size_total
```

Architectures

The upstream CI runs exclusively on nodes with x86 architectures, but OpenStack supports even more architectures. Some of them need to configure Devstack in a certain way.

KVM on s390x (IBM z Systems)

KVM on s390x (IBM z Systems) is supported since the *Kilo* release. For an all-in-one setup, these minimal settings in the `local.conf` file are needed:

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD

DOWNLOAD_DEFAULT_IMAGES=False
IMAGE_URLS="https://cloud-images.ubuntu.com/xenial/current/xenial-server-
↪cloudimg-s390x-disk1.img"

# Provide a custom etcd3 binary download URL and ints sha256.
# The binary must be located under '/<etcd version>/etcd-<etcd-version>-linux-
↪s390x.tar.gz'
# on this URL.
# Build instructions for etcd3: https://github.com/linux-on-ibm-z/docs/wiki/
↪Building-etcd
ETCD_DOWNLOAD_URL=<your-etcd-download-url>
```

(continues on next page)

(continued from previous page)

```
ETCD_SHA256=<your-etcd3-sha256>

enable_service n-sproxy
disable_service n-novnc

[[post-config|$NOVA_CONF]]

[serial_console]
base_url=ws://$HOST_IP:6083/  # optional
```

Reasoning:

- The default image of Devstack is x86 only, so we deactivate the download with `DOWNLOAD_DEFAULT_IMAGES`. The referenced guest image in the code above (`IMAGE_URLS`) serves as an example. The list of possible s390x guest images is not limited to that.
- This platform doesnt support a graphical console like VNC or SPICE. The technical reason is the missing framebuffer on the platform. This means we rely on the substitute feature *serial console* which needs the proxy service `n-sproxy`. We also disable VNCs proxy `n-novnc` for that reason . The configuration in the `post-config` section is only needed if you want to use the *serial console* outside of the all-in-one setup.
- A link to an etcd3 binary and its sha256 needs to be provided as the binary for s390x is not hosted on github like it is for other architectures. For more details see https://bugs.launchpad.net/devstack/+bug/1693192. Etcd3 can easily be built along https://github.com/linux-on-ibm-z/docs/wiki/Building-etcd.

Note

To run *Tempest* against this *Devstack* all-in-one, youll need to use a guest image which is smaller than 1GB when uncompressed. The example image from above is bigger than that!

2.2 System-wide debugging

A lot can go wrong during a devstack run, and there are a few inbuilt tools to help you.

2.2.1 dstat

Enable the `dstat` service to produce performance logs during the devstack run. These will be logged to the journal and also as a CSV file.

2.2.2 memory_tracker

The `memory_tracker` service periodically monitors RAM usage and provides consumption output when available memory is seen to be falling (i.e. processes are consuming memory). It also provides output showing locked (unswappable) memory.

2.2.3 file_tracker

The `file_tracker` service periodically monitors the number of open files in the system.

2.2.4 tcpdump

Enable the `tcpdump` service to run a background tcpdump. You must set the `TCPDUMP_ARGS` variable to something suitable (there is no default). For example, to trace iSCSI communication during a job in the OpenStack gate and copy the result into the log output, you might use:

```
job:
  name: devstack-job
  parent: devstack
  vars:
    devstack_services:
      tcpdump: true
    devstack_localrc:
      TCPDUMP_ARGS: "-i any tcp port 3260"
    zuul_copy_output:
      '{{ devstack_log_dir }}/tcpdump.pcap': logs
```

2.3 Developing with Devstack

Now that you have your nifty DevStack up and running, what can you do with it?

2.3.1 Inspecting Services

By default most services in DevStack are running as *systemd* units named *devstack@$servicename.service*. You can see running services with.

```
sudo systemctl status "devstack@*"
```

To learn more about the basics of systemd, see *Using Systemd in DevStack*

2.3.2 Patching a Service

If you want to make a quick change to a running service the easiest way to do that is to change the code directly in /opt/stack/$service and then restart the affected daemons.

```
sudo systemctl restart devstack@n-cpu.service
```

If your change impacts more than one daemon you can restart by wildcard as well.

```
sudo systemctl restart "devstack@n-*"
```

Warning

All changes you are making are in checked out git trees that DevStack thinks it has full control over. Uncommitted work, or work committed to the master branch, may be overwritten during subsequent DevStack runs.

2.3.3 Testing a Patch Series

When testing a larger set of patches, or patches that will impact more than one service within a project, it is often less confusing to use custom git locations, and make all your changes in a dedicated git tree.

In your `local.conf` you can add `**_REPO`, `**_BRANCH` for most projects to use a custom git tree instead of the default upstream ones.

For instance:

```
[[local|localrc]]
NOVA_REPO=/home/sdague/nova
NOVA_BRANCH=fold_disk_config
```

Will use a custom git tree and branch when doing any devstack operations, such as `stack.sh`.

When testing complicated changes committing to these trees, then doing `./unstack.sh && ./stack.sh` is often a valuable way to iterate. This does take longer per iteration than direct patching, as the whole devstack needs to rebuild.

You can use this same approach to test patches that are up for review in gerrit by using the ref name that gerrit assigns to each change.

```
[[local|localrc]]
NOVA_BRANCH=refs/changes/10/353710/1
```

2.3.4 Testing Changes to Libraries

When testing changes to libraries consumed by OpenStack services (such as oslo or any of the python-fooclient libraries) things are a little more complicated. By default we only test with released versions of these libraries that are on pypi.

You must first override this with the setting `LIBS_FROM_GIT`. This will enable your DevStack with the git version of that library instead of the released version.

After that point you can also specify `**_REPO`, `**_BRANCH` to use your changes instead of just upstream master.

```
[[local|localrc]]
LIBS_FROM_GIT=oslo.policy
OSLOPOLICY_REPO=/home/sdague/oslo.policy
OSLOPOLICY_BRANCH=better_exception
```

As libraries are not installed *editable* by pip, after you make any local changes you will need to:

- cd to top of library path
- sudo pip install -U .
- restart all services you want to use the new library

You can do that with wildcards such as

```
sudo systemctl restart "devstack@n-*"
```

which will restart all nova services.

2.4 FAQ

2.4.1 General Questions

Can I use DevStack for production?

DevStack is targeted at developers and CI systems to use the raw upstream code. It makes many choices that are not appropriate for production systems.

Your best choice is probably to choose a distribution of OpenStack.

Can I use DevStack as a development environment?

Sure, you can. That said, there are a couple of things you should note before doing so:

- DevStack makes a lot of configuration changes to your system and should not be run in your main development environment.
- All the repositories that DevStack clones when deploying are considered volatile by default and

thus are subject to hard resets. This is necessary to keep you in sync with the latest upstream, which is what you want in a CI situation, but it can result in branches being overwritten and files being removed.

The corollary of this is that if you are working on a specific project, using the DevStack project repository (defaulted to `/opt/stack/<project>`) as the single master repository for storing all your work is not recommended. This behavior can be overridden by setting the `RECLONE` config option to `no`. Alternatively, you can avoid running `stack.sh` to redeploy by restarting services manually. In any case, you should generally ensure work in progress is pushed to Gerrit or otherwise backed up before running `stack.sh`.

- If you use DevStack within a VM, you may wish to mount a local OpenStack directory, such as `~/src/openstack`, inside the VM and configure DevStack to use this as the clone location using the `{PROJECT}_REPO` config variables. For example, assuming youre using Vagrant and sharing your home directory, you should place the following in `local.conf`:

```
NEUTRON_REPO=/home/vagrant/src/neutron
NOVA_REPO=/home/vagrant/src/nova
KEYSTONE_REPO=/home/vagrant/src/keystone
GLANCE_REPO=/home/vagrant/src/glance
SWIFT_REPO=/home/vagrant/src/swift
HORIZON_REPO=/home/vagrant/src/horizon
CINDER_REPO=/home/vagrant/src/cinder
HEAT_REPO=/home/vagrant/src/heat
TEMPEST_REPO=/home/vagrant/src/tempest
HEATCLIENT_REPO=/home/vagrant/src/python-heatclient
GLANCECLIENT_REPO=/home/vagrant/src/python-glanceclient
NOVACLIENT_REPO=/home/vagrant/src/python-novaclient
NEUTRONCLIENT_REPO=/home/vagrant/src/python-neutronclient
OPENSTACKCLIENT_REPO=/home/vagrant/src/python-openstackclient
HEAT_CFNTOOLS_REPO=/home/vagrant/src/heat-cfntools
HEAT_TEMPLATES_REPO=/home/vagrant/src/heat-templates
NEUTRON_FWAAS_REPO=/home/vagrant/src/neutron-fwaas
# ...
```

Why a shell script, why not chef/puppet/

The script is meant to be read by humans (as well as ran by computers); it is the primary documentation after all. Using a recipe system requires everyone to agree and understand chef or puppet.

Id like to help!

That isnt a question, but please do! The source for DevStack is at opendev.org and bug reports go to LaunchPad. Contributions follow the usual process as described in the developer guide. This Sphinx documentation is housed in the doc directory.

Why not use packages?

Unlike packages, DevStack leaves your cloud ready to develop - checkouts of the code and services running locally under systemd, making it easy to hack on and test new patches. However, many people are doing the hard work of packaging and recipes for production deployments.

Why isnt $MY_FAVORITE_DISTRO supported?

DevStack is meant for developers and those who want to see how OpenStack really works. DevStack is known to run on the distro/release combinations listed in `README.md`. DevStack is only supported on releases other than those documented in `README.md` on a best-effort basis.

Are there any differences between Ubuntu and CentOS/Fedora support?

Both should work well and are tested by DevStack CI.

Why cant I use another shell?

DevStack now uses some specific bash-ism that require Bash 4, such as associative arrays. Simple compatibility patches have been accepted in the past when they are not complex, at this point no additional compatibility patches will be considered except for shells matching the array functionality as it is very ingrained in the repo and project management.

Can I test on OS/X?

Some people have success with bash 4 installed via homebrew to keep running tests on OS/X.

Can I at least source `openrc` with `zsh`?

People have reported success with a special function to run `openrc` through bash for this

```
function sourceopenrc {
    pushd ~/devstack >/dev/null
    eval $(bash -c ". openrc $1 $2 >/dev/null;env|sed -n '/OS_/ { s/^/export /
↪;p}'")
    popd >/dev/null
}
```

2.4.2 Operation and Configuration

Can DevStack handle a multi-node installation?

Yes, see *multinode lab guide*

How can I document the environment that DevStack is using?

DevStack includes a script (`tools/info.sh`) that gathers the versions of the relevant installed apt packages, pip packages and git repos. This is a good way to verify what Python modules are installed.

How do I turn off a service that is enabled by default?

Services can be turned off by adding `disable_service xxx` to `local.conf` (using `c-vol` in this example):

```
disable_service c-vol
```

Is enabling a service that defaults to off done with the reverse of the above?

Of course!

```
enable_service q-svc
```

How do I run a specific OpenStack release?

DevStack master tracks the upstream master of all the projects. If you would like to run a stable branch of OpenStack, you should use the corresponding stable branch of DevStack as well. For instance the `stable/ocata` version of DevStack will already default to all the projects running at `stable/ocata` levels.

Note: its also possible to manually adjust the `*_BRANCH` variables further if you would like to test specific milestones, or even custom out of tree branches. This is done with entries like the following in your `local.conf`

```
[[local|localrc]]
GLANCE_BRANCH=11.0.0.0rc1
NOVA_BRANCH=12.0.0.0.rc1
```

Upstream DevStack is only tested with master and stable branches. Setting custom BRANCH definitions is not guaranteed to produce working results.

What can I do about RabbitMQ not wanting to start on my fresh new VM?

This is often caused by `erlang` not being happy with the hostname resolving to a reachable IP address. Make sure your hostname resolves to a working IP address; setting it to 127.0.0.1 in `/etc/hosts` is often good enough for a single-node installation. And in an extreme case, use `clean.sh` to eradicate it and try again.

Why are my configuration changes ignored?

You may have run into the package prerequisite installation timeout. `tools/install_prereqs.sh` has a timer that skips the package installation checks if it was run within the last `PREREQ_RERUN_HOURS` hours (default is 2). To override this, set `FORCE_PREREQ=1` and the package checks will never be skipped.

2.4.3 Miscellaneous

`tools/fixup_stuff.sh` is broken and shouldnt fix just one version of packages.

Stuff in there is to correct problems in an environment that need to be fixed elsewhere or may/will be fixed in a future release. In the case of `httplib2` and `prettytable` specific problems with specific versions are being worked around. If later releases have those problems than well add them to the script. Knowing about the broken future releases is valuable rather than polling to see if it has been fixed.

2.5 Guides

Warning

The guides are point in time contributions, and may not always be up to date with the latest work in devstack.

Walk through various setups used by stackers

2.5.1 All-In-One Single VM

Use the cloud to build the cloud! Use your cloud to launch new versions of OpenStack in about 5 minutes. If you break it, start over! The VMs launched in the cloud will be slow as they are running in QEMU (emulation), but their primary use is testing OpenStack development and operation.

Prerequisites Cloud & Image

Virtual Machine

DevStack should run in any virtual machine running a supported Linux release. It will perform best with 4GB or more of RAM.

OpenStack Deployment & cloud-init

If the cloud service has an image with `cloud-init` pre-installed, use it. You can get one from Ubuntus Daily Build site if necessary. This will enable you to launch VMs with userdata that installs everything at boot time. The userdata script below will install and run DevStack with a minimal configuration. The use of `cloud-init` is outside the scope of this document, refer to the `cloud-init` docs for more information.

If you are directly using a hypervisor like Xen, kvm or VirtualBox you can manually kick off the script below as a non-root user in a bare-bones server installation.

Installation shake and bake

Launching With Cloud-Init

This cloud config grabs the latest version of DevStack via git, creates a minimal `local.conf` file and kicks off `stack.sh`. It should be passed as the user-data file when booting the VM.

```
#cloud-config

users:
  - default
  - name: stack
    lock_passwd: False
    sudo: ["ALL=(ALL) NOPASSWD:ALL\nDefaults:stack !requiretty"]
    shell: /bin/bash

write_files:
  - content: |
        #!/bin/sh
        DEBIAN_FRONTEND=noninteractive sudo apt-get -qqy update || sudo dnf␣
↪update -qy
        DEBIAN_FRONTEND=noninteractive sudo apt-get install -qqy git || sudo␣
↪dnf install -qy git
        sudo chown stack:stack /home/stack
        cd /home/stack
        git clone https://opendev.org/openstack/devstack
        cd devstack
```

(continues on next page)

(continued from previous page)

```
        echo '[[local|localrc]]' > local.conf
        echo ADMIN_PASSWORD=password >> local.conf
        echo DATABASE_PASSWORD=password >> local.conf
        echo RABBIT_PASSWORD=password >> local.conf
        echo SERVICE_PASSWORD=password >> local.conf
        ./stack.sh
    path: /home/stack/start.sh
    permissions: 0755

runcmd:
  - su -l stack ./start.sh
```

As DevStack will refuse to run as root, this configures `cloud-init` to create a non-root user and run the `start.sh` script as that user.

If you are using cloud-init and you have not *enabled custom logging* of the stack output, then the stack output can be found in `/var/log/cloud-init-output.log` by default.

Launching By Hand

Using a hypervisor directly, launch the VM and either manually perform the steps in the embedded shell script above or copy it into the VM.

Using OpenStack

At this point you should be able to access the dashboard. Launch VMs and if you give them floating IPs, access those VMs from other machines on your network.

One interesting use case is for developers working on a VM on their laptop. Once `stack.sh` has completed once, all of the pre-requisite packages are installed in the VM and the source trees checked out. Setting `OFFLINE=True` in `local.conf` enables `stack.sh` to run multiple times without an Internet connection. DevStack, making hacking at the lake possible since 2012!

2.5.2 All-In-One Single Machine

Things are about to get real! Using OpenStack in containers or VMs is nice for kicking the tires, but doesnt compare to the feeling you get with hardware.

Prerequisites Linux & Network

Minimal Install

You need to have a system with a fresh install of Linux. You can download the Minimal CD for Ubuntu releases since DevStack will download & install all the additional dependencies. The netinstall ISO is available for Fedora and CentOS/RHEL. You may be tempted to use a desktop distro on a laptop, it will probably work but you may need to tell Network Manager to keep its fingers off the interface(s) that OpenStack uses for bridging.

Network Configuration

Determine the network configuration on the interface used to integrate your OpenStack cloud with your existing network. For example, if the IPs given out on your network by DHCP are 192.168.1.X - where X is between 100 and 200 you will be able to use IPs 201-254 for **floating ips**.

To make things easier later change your host to use a static IP instead of DHCP (i.e. 192.168.1.201).

Installation shake and bake

Add your user

We need to add a user to install DevStack. (if you created a user during install you can skip this step and just give the user sudo privileges below)

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
```

Ensure home directory for the `stack` user has executable permission for all, as RHEL based distros create it with `700` and Ubuntu 21.04+ with `750` which can cause issues during deployment.

```
$ sudo chmod +x /opt/stack
```

Since this user will be making many changes to your system, it will need to have sudo privileges:

```
$ apt-get install sudo -y || dnf install -y sudo
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
```

> **Note**
>
> On some systems you may need to use `sudo visudo`.

From here on you should use the user you created. **Logout** and **login** as that user:

```
$ sudo su stack && cd ~
```

Download DevStack

Well grab the latest version of DevStack via https:

```
$ sudo apt-get install git -y || sudo dnf install -y git
$ git clone https://opendev.org/openstack/devstack
$ cd devstack
```

Run DevStack

Now to configure `stack.sh`. DevStack includes a sample in `devstack/samples/local.conf`. Create `local.conf` as shown below to do the following:

- Set `FLOATING_RANGE` to a range not used on the local network, i.e. 192.168.1.224/27. This configures IP addresses ending in 225-254 to be used as floating IPs.
- Set `FIXED_RANGE` to configure the internal address space used by the instances.

- Set the administrative password. This password is used for the **admin** and **demo** accounts set up as OpenStack users.
- Set the MySQL administrative password. The default here is a random hex string which is inconvenient if you need to look at the database directly for anything.
- Set the RabbitMQ password.
- Set the service password. This is used by the OpenStack services (Nova, Glance, etc) to authenticate with Keystone.

Warning

Only use alphanumeric characters in your passwords, as some services fail to work when using special characters.

`local.conf` should look something like this:

```
[[local|localrc]]
FLOATING_RANGE=192.168.1.224/27
FIXED_RANGE=10.11.12.0/24
ADMIN_PASSWORD=supersecret
DATABASE_PASSWORD=iheartdatabases
RABBIT_PASSWORD=flopsymopsy
SERVICE_PASSWORD=iheartksl
```

Note

There is a sample `local.conf` file under the *samples* directory in the devstack repository.

Run DevStack:

```
$ ./stack.sh
```

A seemingly endless stream of activity ensues. When complete you will see a summary of `stack.sh`s work, including the relevant URLs, accounts and passwords to poke at your shiny new OpenStack.

Using OpenStack

At this point you should be able to access the dashboard from other computers on the local network. In this example that would be http://192.168.1.201/ for the dashboard (aka Horizon). Launch VMs and if you give them floating IPs and security group access those VMs will be accessible from other machines on your network.

2.5.3 All-In-One Single LXC Container

This guide walks you through the process of deploying OpenStack using devstack in an LXC container instead of a VM.

The primary benefits to running devstack inside a container instead of a VM is faster performance and lower memory overhead while still providing a suitable level of isolation. This can be particularly useful when you want to simulate running OpenStack on multiple nodes.

Warning

Containers do not provide the same level of isolation as a virtual machine.

Note

Not all OpenStack features support running inside of a container. See *Limitations* section below for details. *OpenStack in a VM* is recommended for beginners.

Prerequisites

This guide is written for Ubuntu 14.04 but should be adaptable for any modern Linux distribution.

Install the LXC package:

```
sudo apt-get install lxc
```

You can verify support for containerization features in your currently running kernel using the `lxc-checkconfig` command.

Container Setup

Configuration

For a successful run of `stack.sh` and to permit use of KVM to run the VMs you launch inside your container, we need to use the following additional configuration options. Place the following in a file called `devstack-lxc.conf`:

```
# Permit access to /dev/loop*
lxc.cgroup.devices.allow = b 7:* rwm

# Setup access to /dev/net/tun and /dev/kvm
lxc.mount.entry = /dev/net/tun dev/net/tun none bind,create=file 0 0
lxc.mount.entry = /dev/kvm dev/kvm none bind,create=file 0 0

# Networking
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = lxcbr0
```

Create Container

The configuration and rootfs for LXC containers are created using the `lxc-create` command.

We will name our container `devstack` and use the `ubuntu` template which will use `debootstrap` to build a Ubuntu rootfs. It will default to the same release and architecture as the host system. We also install the additional packages `bsdmainutils` and `git` as well need them to run devstack:

```
sudo lxc-create -n devstack -t ubuntu -f devstack-lxc.conf -- --
↪packages=bsdmainutils,git
```

The first time it builds the rootfs will take a few minutes to download, unpack, and configure all the necessary packages for a minimal installation of Ubuntu. LXC will cache this and subsequent containers will only take seconds to create.

Note

To speed up the initial rootfs creation, you can specify a mirror to download the Ubuntu packages from by appending `--mirror=` and then the URL of a Ubuntu mirror. To see other other template options, you can run `lxc-create -t ubuntu -h`.

Start Container

To start the container, run:

```
sudo lxc-start -n devstack
```

A moment later you should be presented with the login prompt for your container. You can login using the username `ubuntu` and password `ubuntu`.

You can also ssh into your container. On your host, run `sudo lxc-info -n devstack` to get the IP address (e.g. `ssh ubuntu@$(sudo lxc-info -n devstack | awk '/IP/ { print $2 }')`).

Run Devstack

You should now be logged into your container and almost ready to run devstack. The commands in this section should all be run inside your container.

Tip

You can greatly reduce the runtime of your initial devstack setup by ensuring you have your apt sources.list configured to use a fast mirror. Check and update `/etc/apt/sources.list` if necessary and then run `apt-get update`.

1. Download DevStack

   ```
   git clone https://opendev.org/openstack/devstack
   ```

2. Configure

   Refer to *Minimal Configuration* if you wish to configure the behaviour of devstack.

3. Start the install

   ```
   cd devstack
   ./stack.sh
   ```

Cleanup

To stop the container:

```
lxc-stop -n devstack
```

To delete the container:

```
lxc-destroy -n devstack
```

Limitations

Not all OpenStack features may function correctly or at all when ran from within a container.

Cinder

Unable to create LVM backed volume

In our configuration, we have not whitelisted access to device-mapper or LVM devices. Doing so will permit your container to have access and control of LVM on the host system. To enable, add the following to your `devstack-lxc.conf` before running `lxc-create`:

```
lxc.cgroup.devices.allow = c 10:236 rwm
lxc.cgroup.devices.allow = b 252:* rwm
```

Additionally youll need to set `udev_rules = 0` in the `activation` section of `/etc/lvm/lvm.conf` unless you mount devtmpfs in your container.

Unable to attach volume to instance

It is not possible to attach cinder volumes to nova instances due to parts of the Linux iSCSI implementation not being network namespace aware. This can be worked around by using network pass-through instead of a separate network namespace but such a setup significantly reduces the isolation of the container (e.g. a `halt` command issued in the container will cause the host system to shutdown).

2.5.4 Multi-Node Lab

Here is OpenStack in a realistic test configuration with multiple physical servers.

Prerequisites Linux & Network

Minimal Install

You need to have a system with a fresh install of Linux. You can download the Minimal CD for Ubuntu releases since DevStack will download & install all the additional dependencies. The netinstall ISO is available for Fedora and CentOS/RHEL.

Install a couple of packages to bootstrap configuration:

```
apt-get install -y git sudo || dnf install -y git sudo
```

Network Configuration

The first iteration of the lab uses OpenStacks FlatDHCP network controller so only a single network will be required. It should be on its own subnet without DHCP; the host IPs and floating IP pool(s) will come out of this block. This example uses the following:

- Gateway: 192.168.42.1

- Physical nodes: 192.168.42.11-192.168.42.99
- Floating IPs: 192.168.42.128-192.168.42.254

Configure each node with a static IP. For Ubuntu edit `/etc/network/interfaces`:

```
auto eth0
iface eth0 inet static
    address 192.168.42.11
    netmask 255.255.255.0
    gateway 192.168.42.1
```

For Fedora and CentOS/RHEL edit `/etc/sysconfig/network-scripts/ifcfg-eth0`:

```
BOOTPROTO=static
IPADDR=192.168.42.11
NETMASK=255.255.255.0
GATEWAY=192.168.42.1
```

Installation shake and bake

Add the DevStack User

OpenStack runs as a non-root user that has sudo access to root. There is nothing special about the name, well use `stack` here. Every node must use the same name and preferably uid. If you created a user during the OS install you can use it and give it sudo privileges below. Otherwise create the stack user:

```
useradd -s /bin/bash -d /opt/stack -m stack
```

Ensure home directory for the `stack` user has executable permission for all, as RHEL based distros create it with `700` and Ubuntu 21.04+ with `750` which can cause issues during deployment.

```
chmod +x /opt/stack
```

This user will be making many changes to your system during installation and operation so it needs to have sudo privileges to root without a password:

```
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
```

From here on use the `stack` user. **Logout** and **login** as the `stack` user.

Set Up Ssh

Set up the stack user on each node with an ssh key for access:

```
mkdir ~/.ssh; chmod 700 ~/.ssh
echo "ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQCyYjfgyPazTvGpd8OaAvtU2utL8W6gWC4JdRS1J95GhNNfQd657yO6s1AH5
↪xNUC2reEXSGC7ezy+sGO1kj9Limv5vrvNHvF1+wts0Cmyx61D2nQw35/Qz8BvpdJANL7VwP/cFI/
↪p3yhvx2lsnjFE3hN8xRB2LtLUopUSVdBwACOVUmH2G+2BWMJDjVINd2DPqRIA4Zhy09KJ3O1Joabr0XpQL0yt/
↪I9x8BVHdAx6l9U0tMg9dj5+tAjZvMAFfye3PJcYwwsfJoFxC8w/
↪SLtqlFX7Ehw++8RtvomvuipLdmWCy+T9hIkl+gHYE4cS3OIqXH7f49jdJf jesse@spacey.
↪local" > ~/.ssh/authorized_keys
```

Download DevStack

Grab the latest version of DevStack:

```
git clone https://opendev.org/openstack/devstack
cd devstack
```

Up to this point all of the steps apply to each node in the cluster. From here on there are some differences between the cluster controller (aka head node) and the compute nodes.

Configure Cluster Controller

The cluster controller runs all OpenStack services. Configure the cluster controllers DevStack in `local.conf`:

```
[[local|localrc]]
HOST_IP=192.168.42.11
FIXED_RANGE=10.4.128.0/20
FLOATING_RANGE=192.168.42.128/25
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=labstack
DATABASE_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
```

In the multi-node configuration the first 10 or so IPs in the private subnet are usually reserved. Add this to `local.sh` to have it run after every `stack.sh` run:

```
for i in `seq 2 10`; do /opt/stack/nova/bin/nova-manage fixed reserve 10.4.
↪128.$i; done
```

Fire up OpenStack:

```
./stack.sh
```

A stream of activity ensues. When complete you will see a summary of `stack.sh`s work, including the relevant URLs, accounts and passwords to poke at your shiny new OpenStack. The most recent log file is available in `stack.sh.log`.

Configure Compute Nodes

The compute nodes only run the OpenStack worker services. For additional machines, create a `local.conf` with:

```
[[local|localrc]]
HOST_IP=192.168.42.12 # change this per compute node
FIXED_RANGE=10.4.128.0/20
FLOATING_RANGE=192.168.42.128/25
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=labstack
DATABASE_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
```

(continues on next page)

(continued from previous page)

```
SERVICE_PASSWORD=supersecret
DATABASE_TYPE=mysql
SERVICE_HOST=192.168.42.11
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
ENABLED_SERVICES=n-cpu,c-vol,placement-client,ovn-controller,ovs-vswitchd,
↪ovsdb-server,q-ovn-metadata-agent
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://$SERVICE_HOST:6080/vnc_lite.html"
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN
```

Fire up OpenStack:

```
./stack.sh
```

A stream of activity ensues. When complete you will see a summary of `stack.sh`s work, including the relevant URLs, accounts and passwords to poke at your shiny new OpenStack. The most recent log file is available in `stack.sh.log`.

Starting in the Ocata release, Nova requires a Cells v2 deployment. Compute node services must be mapped to a cell before they can be used.

After each compute node is stacked, verify it shows up in the `nova service-list --binary nova-compute` output. The compute service is registered in the cell database asynchronously so this may require polling.

Once the compute node services shows up, run the `./tools/discover_hosts.sh` script from the control node to map compute hosts to the single cell.

The compute service running on the primary control node will be discovered automatically when the control node is stacked so this really only needs to be performed for subnodes.

Configure Tempest Node to run the Tempest tests

If there is a need to execute Tempest tests against different Cluster Controller node then it can be done by re-using the `local.conf` file from the Cluster Controller node but with not enabled Controller services in `ENABLED_SERVICES` variable. This variable needs to contain only `tempest` as a configured service. Then variable `SERVICES_FOR_TEMPEST` must be configured to contain those services that were enabled on the Cluster Controller node in the `ENABLED_SERVICES` variable. For example the `local.conf` file could look as follows:

```
[[local|localrc]]
HOST_IP=192.168.42.12 # change this per compute node
FIXED_RANGE=10.4.128.0/20
FLOATING_RANGE=192.168.42.128/25
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=labstack
DATABASE_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
```

(continues on next page)

(continued from previous page)

```
DATABASE_TYPE=mysql
SERVICE_HOST=192.168.42.11
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://$SERVICE_HOST:6080/vnc_lite.html"
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN
ENABLED_SERVICES=tempest
SERVICES_FOR_TEMPEST=keystone,nova,neutron,glance
```

Then just execute the devstack:

```
./stack.sh
```

Cleaning Up After DevStack

Shutting down OpenStack is now as simple as running the included `unstack.sh` script:

```
./unstack.sh
```

A more aggressive cleanup can be performed using `clean.sh`. It removes certain troublesome packages and attempts to leave the system in a state where changing the database or queue manager can be reliably performed.

```
./clean.sh
```

Sometimes running instances are not cleaned up. DevStack attempts to do this when it runs but there are times it needs to still be done by hand:

```
sudo rm -rf /etc/libvirt/qemu/inst*
sudo virsh list | grep inst | awk '{print $1}' | xargs -n1 virsh destroy
```

Going further

Additional Users

DevStack creates two OpenStack users (`admin` and `demo`) and two projects (also `admin` and `demo`). `admin` is exactly what it sounds like, a privileged administrative account that is a member of both the `admin` and `demo` projects. `demo` is a normal user account that is only a member of the `demo` project. Creating additional OpenStack users can be done through the dashboard, sometimes it is easier to do them in bulk from a script, especially since they get blown away every time `stack.sh` runs. The following steps are ripe for scripting:

```
# Get admin creds
. openrc admin admin

# List existing projects
openstack project list
```

(continues on next page)

(continued from previous page)

```
# List existing users
openstack user list

# Add a user and project
NAME=bob
PASSWORD=BigSecret
PROJECT=$NAME
openstack project create $PROJECT
openstack user create $NAME --password=$PASSWORD --project $PROJECT
openstack role add Member --user $NAME --project $PROJECT
# The Member role is created by stack.sh
# openstack role assignment list
```

Swift

Swift, OpenStack Object Storage, requires a significant amount of resources and is disabled by default in DevStack. The support in DevStack is geared toward a minimal installation but can be used for testing. To implement a true multi-node test of swift, additional steps will be required. Enabling it is as simple as enabling the `swift` service in `local.conf`:

```
enable_service s-proxy s-object s-container s-account
```

Swift, OpenStack Object Storage, will put its data files in `SWIFT_DATA_DIR` (default `/opt/stack/data/swift`). The size of the data partition created (really a loop-mounted file) is set by `SWIFT_LOOPBACK_DISK_SIZE`. The Swift config files are located in `SWIFT_CONF_DIR` (default `/etc/swift`). All of these settings can be overridden in (wait for it) `local.conf`.

Volumes

DevStack will automatically use an existing LVM volume group named `stack-volumes` to store cloud-created volumes. If `stack-volumes` doesnt exist, DevStack will set up a loop-mounted file to contain it. If the default size is insufficient for the number and size of volumes required, it can be overridden by setting `VOLUME_BACKING_FILE_SIZE` in `local.conf` (sizes given in `truncate` compatible format, e.g. `24G`).

`stack-volumes` can be pre-created on any physical volume supported by Linuxs LVM. The name of the volume group can be changed by setting `VOLUME_GROUP_NAME` in `localrc`. `stack.sh` deletes all logical volumes in `VOLUME_GROUP_NAME` that begin with `VOLUME_NAME_PREFIX` as part of cleaning up from previous runs. It is recommended to not use the root volume group as `VOLUME_GROUP_NAME`.

The details of creating the volume group depends on the server hardware involved but looks something like this:

```
pvcreate /dev/sdc
vgcreate stack-volumes /dev/sdc
```

Syslog

DevStack is capable of using `rsyslog` to aggregate logging across the cluster. It is off by default; to turn it on set `SYSLOG=True` in `local.conf`. `SYSLOG_HOST` defaults to `HOST_IP`; on the compute nodes it must be set to the IP of the cluster controller to send syslog output there. In the example above, add this to the compute node `local.conf`:

```
SYSLOG_HOST=192.168.42.11
```

Using Alternate Repositories/Branches

The git repositories for all of the OpenStack services are defined in `stackrc`. Since this file is a part of the DevStack package changes to it will probably be overwritten as updates are applied. Every setting in `stackrc` can be redefined in `local.conf`.

To change the repository or branch that a particular OpenStack service is created from, simply change the value of `*_REPO` or `*_BRANCH` corresponding to that service.

After making changes to the repository or branch, if `RECLONE` is not set in `localrc` it may be necessary to remove the corresponding directory from `/opt/stack` to force git to re-clone the repository.

For example, to pull nova, OpenStack Compute, from a proposed release candidate in the primary nova repository:

```
NOVA_BRANCH=rc-proposed
```

To pull glance, OpenStack Image service, from an experimental fork:

```
GLANCE_BRANCH=try-something-big
GLANCE_REPO=https://github.com/mcuser/glance.git
```

Notes stuff you might need to know

Set MySQL Password

If you forgot to set the root password you can do this:

```
mysqladmin -u root -pnova password 'supersecret'
```

Live Migration

In order for live migration to work with the default live migration URI:

```
[libvirt]
live_migration_uri = qemu+ssh://stack@%s/system
```

SSH keys need to be exchanged between each compute node:

1. The SOURCE root users public RSA key (likely in /root/.ssh/id_rsa.pub) needs to be in the DESTINATION stack users authorized_keys file (~stack/.ssh/authorized_keys). This can be accomplished by manually copying the contents from the file on the SOURCE to the DESTINATION. If you have a password configured for the stack user, then you can use the following command to accomplish the same thing:

```
ssh-copy-id -i /root/.ssh/id_rsa.pub stack@DESTINATION
```

2. The DESTINATION hosts public ECDSA key (/etc/ssh/ssh_host_ecdsa_key.pub) needs to be in the SOURCE root users known_hosts file (/root/.ssh/known_hosts). This can be accomplished by running the following on the SOURCE machine (hostname must be used):

```
ssh-keyscan -H DEST_HOSTNAME | sudo tee -a /root/.ssh/known_hosts
```

3. Verify that login via ssh works without a password:

```
ssh -i /root/.ssh/id_rsa stack@DESTINATION
```

In essence, this means that every compute nodes root users public RSA key must exist in every other compute nodes stack users authorized_keys file and every compute nodes public ECDSA key needs to be in every other compute nodes root users known_hosts file. Please note that if the root or stack user does not have a SSH key, one can be generated using:

```
ssh-keygen -t rsa
```

The above steps are necessary because libvirtd runs as root when the live_migration_uri uses the qemu:///system family of URIs. For more information, see the libvirt documentation.

2.5.5 Using DevStack with neutron Networking

This guide will walk you through using OpenStack neutron with the ML2 plugin and the Open vSwitch mechanism driver.

Using Neutron with a Single Interface

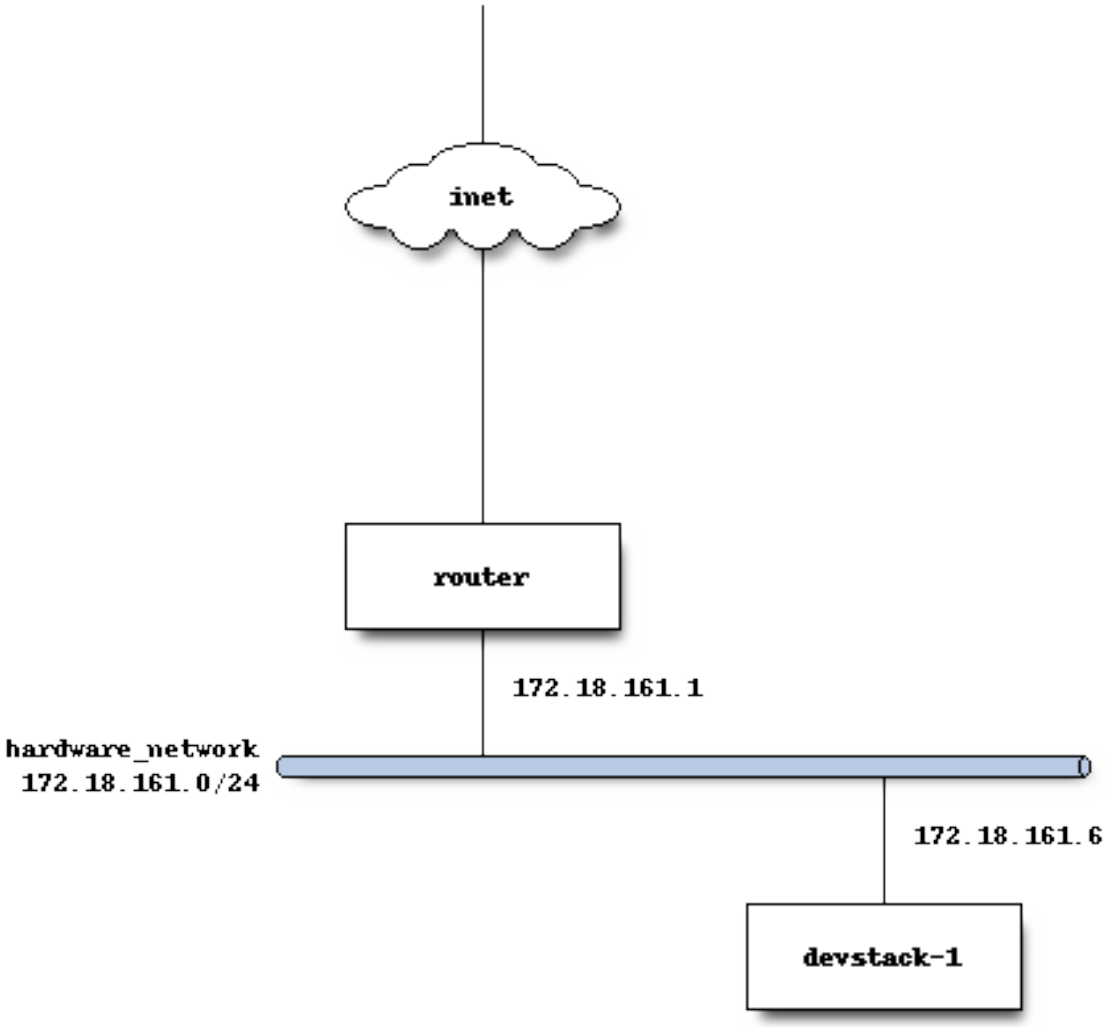
In some instances, like on a developer laptop, there is only one network interface that is available. In this scenario, the physical interface is added to the Open vSwitch bridge, and the IP address of the laptop is migrated onto the bridge interface. That way, the physical interface can be used to transmit self service project network traffic, the OpenStack API traffic, and management traffic.

Warning

When using a single interface networking setup, there will be a temporary network outage as your IP address is moved from the physical NIC of your machine, to the OVS bridge. If you are SSHd into the machine from another computer, there is a risk of being disconnected from your ssh session (due to arp cache invalidation), which would stop the stack.sh or leave it in an unfinished state. In these cases, start stack.sh inside its own screen session so it can continue to run.

Physical Network Setup

In most cases where DevStack is being deployed with a single interface, there is a hardware router that is being used for external connectivity and DHCP. The developer machine is connected to this network and is on a shared subnet with other machines. The *local.conf* exhibited here assumes that 1500 is a reasonable MTU to use on that network.



DevStack Configuration

The following is a complete *local.conf* for the host named *devstack-1*. It will run all the API and services, as well as serving as a hypervisor for guest instances.

```
[[local|localrc]]
HOST_IP=172.18.161.6
```

(continues on next page)

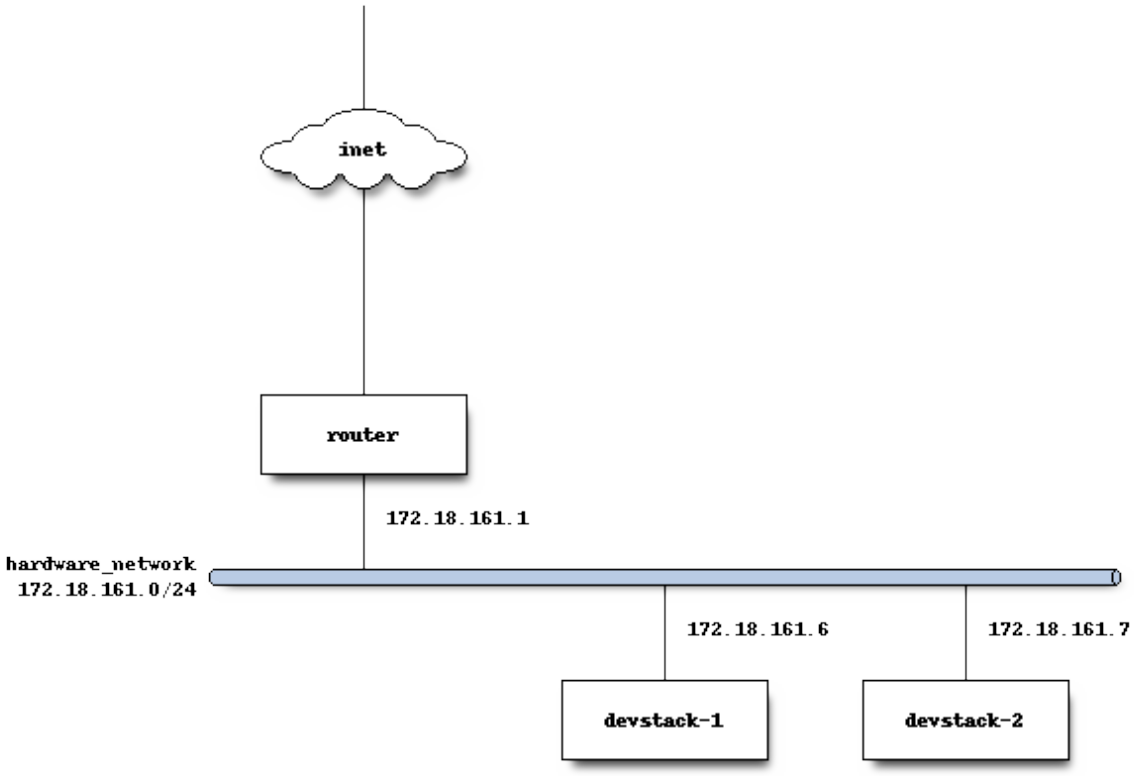
(continued from previous page)

```
SERVICE_HOST=172.18.161.6
MYSQL_HOST=172.18.161.6
RABBIT_HOST=172.18.161.6
GLANCE_HOSTPORT=172.18.161.6:9292
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_PASSWORD=secret

## Neutron options
Q_USE_SECGROUP=True
FLOATING_RANGE="172.18.161.0/24"
IPV4_ADDRS_SAFE_TO_USE="10.0.0.0/22"
Q_FLOATING_ALLOCATION_POOL=start=172.18.161.250,end=172.18.161.254
PUBLIC_NETWORK_GATEWAY="172.18.161.1"
PUBLIC_INTERFACE=eth0

# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex
```

Adding Additional Compute Nodes

Lets suppose that after installing DevStack on the first host, you also want to do multinode testing and networking.

Physical Network Setup



After DevStack installs and configures Neutron, traffic from guest VMs flows out of *devstack-2* (the compute node) and is encapsulated in a VXLAN tunnel back to *devstack-1* (the control node) where the L3 agent is running.

```
stack@devstack-2:~/devstack$ sudo ovs-vsctl show
8992d965-0ba0-42fd-90e9-20ecc528bc29
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
    Bridge br-tun
```

(continues on next page)

(continued from previous page)

```
        fail_mode: secure
        Port "vxlan-c0a801f6"
            Interface "vxlan-c0a801f6"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="172.18.
↪161.7", out_key=flow, remote_ip="172.18.161.6"}
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port br-tun
            Interface br-tun
                type: internal
    ovs_version: "2.0.2"
```

Open vSwitch on the control node, where the L3 agent runs, is configured to de-encapsulate traffic from compute nodes, then forward it over the *br-ex* bridge, where *eth0* is attached.

```
stack@devstack-1:~/devstack$ sudo ovs-vsctl show
422adeea-48d1-4a1f-98b1-8e7239077964
    Bridge br-tun
        fail_mode: secure
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "vxlan-c0a801d8"
            Interface "vxlan-c0a801d8"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="172.18.
↪161.6", out_key=flow, remote_ip="172.18.161.7"}
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port "eth0"
            Interface "eth0"
        Port br-ex
            Interface br-ex
                type: internal
    Bridge br-int
        fail_mode: secure
        Port "tapce66332d-ea"
            tag: 1
            Interface "tapce66332d-ea"
                type: internal
```

(continues on next page)

(continued from previous page)

```
        Port "qg-65e5a4b9-15"
            tag: 2
            Interface "qg-65e5a4b9-15"
                type: internal
        Port "qr-33e5e471-88"
            tag: 1
            Interface "qr-33e5e471-88"
                type: internal
        Port "qr-acbe9951-70"
            tag: 1
            Interface "qr-acbe9951-70"
                type: internal
        Port br-int
            Interface br-int
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
    ovs_version: "2.0.2"
```

br-int is a bridge that the Open vSwitch mechanism driver creates, which is used as the integration bridge where ports are created, and plugged into the virtual switching fabric. *br-ex* is an OVS bridge that is used to connect physical ports (like *eth0*), so that floating IP traffic for project networks can be received from the physical network infrastructure (and the internet), and routed to self service project network ports. *br-tun* is a tunnel bridge that is used to connect OpenStack nodes (like *devstack-2*) together. This bridge is used so that project network traffic, using the VXLAN tunneling protocol, flows between each compute node where project instances run.

DevStack Compute Configuration

The host *devstack-2* has a very minimal *local.conf*.

```
[[local|localrc]]
HOST_IP=172.18.161.7
SERVICE_HOST=172.18.161.6
MYSQL_HOST=172.18.161.6
RABBIT_HOST=172.18.161.6
GLANCE_HOSTPORT=172.18.161.6:9292
ADMIN_PASSWORD=secret
MYSQL_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_PASSWORD=secret

## Neutron options
PUBLIC_INTERFACE=eth0
```

(continues on next page)

(continued from previous page)

```
ENABLED_SERVICES=n-cpu,rabbit,q-agt,placement-client
```

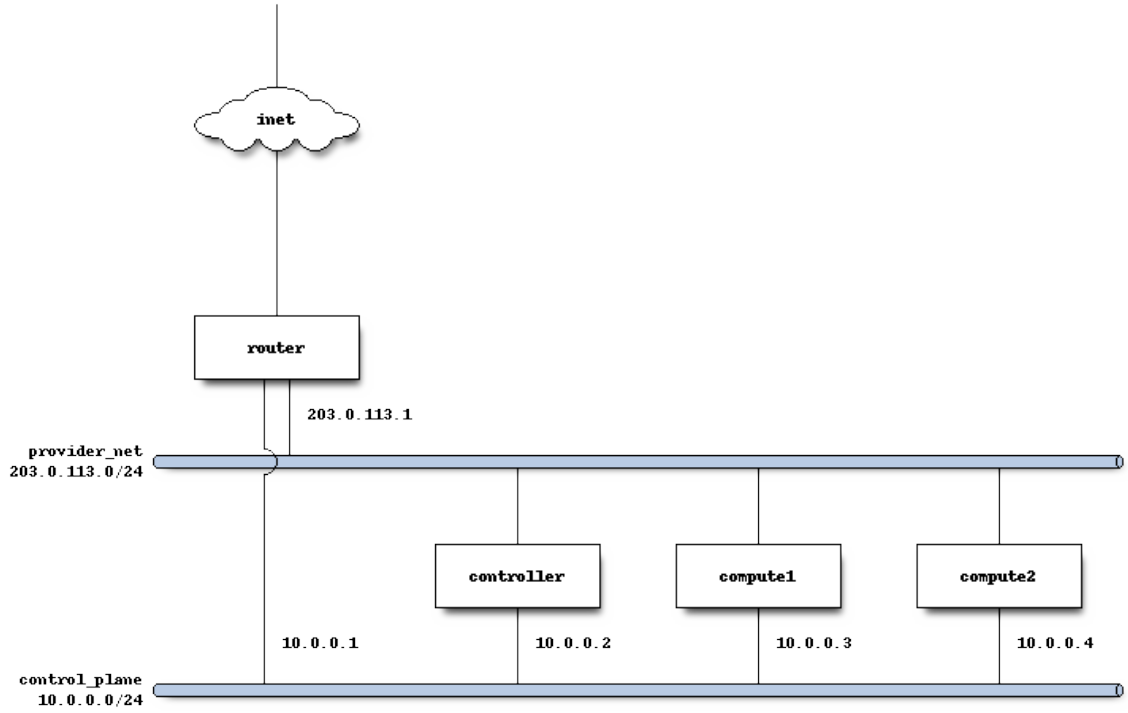
Network traffic from *eth0* on the compute nodes is then NATd by the controller node that runs Neutrons *neutron-l3-agent* and provides L3 connectivity.

Neutron Networking with Open vSwitch and Provider Networks

In some instances, it is desirable to use neutrons provider networking extension, so that networks that are configured on an external router can be utilized by neutron, and instances created via Nova can attach to the network managed by the external router.

For example, in some lab environments, a hardware router has been pre-configured by another party, and an OpenStack developer has been given a VLAN tag and IP address range, so that instances created via DevStack will use the external router for L3 connectivity, as opposed to the neutron L3 service.

Physical Network Setup



On a compute node, the first interface, eth0 is used for the OpenStack management (API, message bus, etc) as well as for ssh for an administrator to access the machine.

```
stack@compute:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr bc:16:65:20:af:fc
          inet addr:10.0.0.3
```

eth1 is manually configured at boot to not have an IP address. Consult your operating system documentation for the appropriate technique. For Ubuntu, the contents of */etc/network/interfaces* contains:

```
auto eth1
iface eth1 inet manual
        up ifconfig $IFACE 0.0.0.0 up
        down ifconfig $IFACE 0.0.0.0 down
```

The second physical interface, eth1 is added to a bridge (in this case named br-ex), which is used to forward network traffic from guest VMs.

```
stack@compute:~$ sudo ovs-vsctl add-br br-ex
stack@compute:~$ sudo ovs-vsctl add-port br-ex eth1
stack@compute:~$ sudo ovs-vsctl show
9a25c837-32ab-45f6-b9f2-1dd888abcf0f
    Bridge br-ex
        Port br-ex
            Interface br-ex
                type: internal
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port "eth1"
            Interface "eth1"
```

Service Configuration

Control Node

In this example, the control node will run the majority of the OpenStack API and management services (keystone, glance, nova, neutron)

Compute Nodes

In this example, the nodes that will host guest instances will run the `neutron-openvswitch-agent` for network connectivity, as well as the compute service `nova-compute`.

DevStack Configuration

The following is a snippet of the DevStack configuration on the controller node.

```
HOST_IP=10.0.0.2
SERVICE_HOST=10.0.0.2
MYSQL_HOST=10.0.0.2
RABBIT_HOST=10.0.0.2
GLANCE_HOSTPORT=10.0.0.2:9292
PUBLIC_INTERFACE=eth1

ADMIN_PASSWORD=secret
MYSQL_PASSWORD=secret
RABBIT_PASSWORD=secret
```

(continues on next page)

(continued from previous page)

```
SERVICE_PASSWORD=secret

## Neutron options
Q_USE_SECGROUP=True
ENABLE_TENANT_VLANS=True
TENANT_VLAN_RANGE=3001:4000
PHYSICAL_NETWORK=default
OVS_PHYSICAL_BRIDGE=br-ex

Q_USE_PROVIDER_NETWORKING=True

disable_service q-l3

## Neutron Networking options used to create Neutron Subnets

IPV4_ADDRS_SAFE_TO_USE="203.0.113.0/24"
NETWORK_GATEWAY=203.0.113.1
PROVIDER_SUBNET_NAME="provider_net"
PROVIDER_NETWORK_TYPE="vlan"
SEGMENTATION_ID=2010
USE_SUBNETPOOL=False
```

In this configuration we are defining IPV4_ADDRS_SAFE_TO_USE to be a publicly routed IPv4 subnet. In this specific instance we are using the special TEST-NET-3 subnet defined in RFC 5737, which is used for documentation. In your DevStack setup, IPV4_ADDRS_SAFE_TO_USE would be a public IP address range that you or your organization has allocated to you, so that you could access your instances from the public internet.

The following is the DevStack configuration on compute node 1.

```
HOST_IP=10.0.0.3
SERVICE_HOST=10.0.0.2
MYSQL_HOST=10.0.0.2
RABBIT_HOST=10.0.0.2
GLANCE_HOSTPORT=10.0.0.2:9292
ADMIN_PASSWORD=secret
MYSQL_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_PASSWORD=secret

# Services that a compute node runs
ENABLED_SERVICES=n-cpu,rabbit,q-agt

## Open vSwitch provider networking options
PHYSICAL_NETWORK=default
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_INTERFACE=eth1
Q_USE_PROVIDER_NETWORKING=True
```

Compute node 2s configuration will be exactly the same, except `HOST_IP` will be `10.0.0.4`

When DevStack is configured to use provider networking (via `Q_USE_PROVIDER_NETWORKING` is True) - DevStack will automatically add the network interface defined in `PUBLIC_INTERFACE` to the `OVS_PHYSICAL_BRIDGE`

For example, with the above configuration, a bridge is created, named `br-ex` which is managed by Open vSwitch, and the second interface on the compute node, `eth1` is attached to the bridge, to forward traffic sent by guest VMs.

Miscellaneous Tips

Non-Standard MTU on the Physical Network

Neutron by default uses a MTU of 1500 bytes, which is the standard MTU for Ethernet.

A different MTU can be specified by adding the following to the Neutron section of *local.conf*. For example, if you have network equipment that supports jumbo frames, you could set the MTU to 9000 bytes by adding the following

```
[[post-config|/$Q_PLUGIN_CONF_FILE]]
global_physnet_mtu = 9000
```

Disabling Next Generation Firewall Tools

DevStack does not properly operate with modern firewall tools. Specifically it will appear as if the guest VM can access the external network via ICMP, but UDP and TCP packets will not be delivered to the guest VM. The root cause of the issue is that both ufw (Uncomplicated Firewall) and firewalld (Fedoras firewall manager) apply firewall rules to all interfaces in the system, rather then per-device. One solution to this problem is to revert to iptables functionality.

To get a functional firewall configuration for Fedora do the following:

```
sudo service iptables save
sudo systemctl disable firewalld
sudo systemctl enable iptables
sudo systemctl stop firewalld
sudo systemctl start iptables
```

To get a functional firewall configuration for distributions containing ufw, disable ufw. Note ufw is generally not enabled by default in Ubuntu. To disable ufw if it was enabled, do the following:

```
sudo service iptables save
sudo ufw disable
```

Configuring Extension Drivers for the ML2 Plugin

Extension drivers for the ML2 plugin are set with the variable `Q_ML2_PLUGIN_EXT_DRIVERS`, and includes the port_security extension by default. If you want to remove all the extension drivers (even port_security), set `Q_ML2_PLUGIN_EXT_DRIVERS` to blank.

Using MacVTap instead of Open vSwitch

Security groups are not supported by the MacVTap agent. Due to that, devstack configures the Noop-Firewall driver on the compute node.

MacVTap agent does not support l3, dhcp and metadata agent. Due to that you can chose between the following deployment scenarios:

Single node with provider networks using config drive and external l3, dhcp

This scenario applies, if l3 and dhcp services are provided externally, or if you do not require them.

```
[[local|localrc]]
HOST_IP=10.0.0.2
SERVICE_HOST=10.0.0.2
MYSQL_HOST=10.0.0.2
RABBIT_HOST=10.0.0.2
ADMIN_PASSWORD=secret
MYSQL_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_PASSWORD=secret

Q_ML2_PLUGIN_MECHANISM_DRIVERS=macvtap
Q_USE_PROVIDER_NETWORKING=True

enable_plugin neutron https://opendev.org/openstack/neutron

## MacVTap agent options
Q_AGENT=macvtap
PHYSICAL_NETWORK=default

IPV4_ADDRS_SAFE_TO_USE="203.0.113.0/24"
NETWORK_GATEWAY=203.0.113.1
PROVIDER_SUBNET_NAME="provider_net"
PROVIDER_NETWORK_TYPE="vlan"
SEGMENTATION_ID=2010
USE_SUBNETPOOL=False

[[post-config|/$Q_PLUGIN_CONF_FILE]]
[macvtap]
physical_interface_mappings = $PHYSICAL_NETWORK:eth1

[[post-config|$NOVA_CONF]]
force_config_drive = True
```

Multi node with MacVTap compute node

This scenario applies, if you require OpenStack provided l3, dhcp or metadata services. Those are hosted on a separate controller and network node, running some other l2 agent technology (in this example Open vSwitch). This node needs to be configured for VLAN tenant networks.

For OVS, a similar configuration like described in the *OVS Provider Network* section can be used. Just

add the following line to this local.conf, which also loads the MacVTap mechanism driver:

```
[[local|localrc]]
...
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,macvtap
...
```

For the MacVTap compute node, use this local.conf:

```
HOST_IP=10.0.0.3
SERVICE_HOST=10.0.0.2
MYSQL_HOST=10.0.0.2
RABBIT_HOST=10.0.0.2
ADMIN_PASSWORD=secret
MYSQL_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_PASSWORD=secret

# Services that a compute node runs
disable_all_services
enable_plugin neutron https://opendev.org/openstack/neutron
ENABLED_SERVICES+=n-cpu,q-agt

## MacVTap agent options
Q_AGENT=macvtap
PHYSICAL_NETWORK=default

[[post-config|/$Q_PLUGIN_CONF_FILE]]
[macvtap]
physical_interface_mappings = $PHYSICAL_NETWORK:eth1
```

2.5.6 Configure DevStack with KVM-based Nested Virtualization

When using virtualization technologies like KVM, one can take advantage of Nested VMX (i.e. the ability to run KVM on KVM) so that the VMs in cloud (Nova guests) can run relatively faster than with plain QEMU emulation.

Kernels shipped with Linux distributions doesnt have this enabled by default. This guide outlines the configuration details to enable nested virtualization in KVM-based environments. And how to setup DevStack (thatll run in a VM) to take advantage of this.

Nested Virtualization Configuration

Configure Nested KVM for Intel-based Machines

Procedure to enable nested KVM virtualization on Intel-based machines.

Check if the nested KVM Kernel parameter is enabled:

```
cat /sys/module/kvm_intel/parameters/nested
N
```

Temporarily remove the KVM intel Kernel module, enable nested virtualization to be persistent across reboots and add the Kernel module back:

```
sudo rmmod kvm-intel
sudo sh -c "echo 'options kvm-intel nested=y' >> /etc/modprobe.d/dist.conf"
sudo modprobe kvm-intel
```

Ensure the Nested KVM Kernel module parameter for Intel is enabled on the host:

```
cat /sys/module/kvm_intel/parameters/nested
Y

modinfo kvm_intel | grep nested
parm:           nested:bool
```

Start your VM, now it should have KVM capabilities you can verify that by ensuring `/dev/kvm` character device is present.

Configure Nested KVM for AMD-based Machines

Procedure to enable nested KVM virtualization on AMD-based machines.

Check if the nested KVM Kernel parameter is enabled:

```
cat /sys/module/kvm_amd/parameters/nested
0
```

Temporarily remove the KVM AMD Kernel module, enable nested virtualization to be persistent across reboots and add the Kernel module back:

```
sudo rmmod kvm-amd
sudo sh -c "echo 'options kvm-amd nested=1' >> /etc/modprobe.d/dist.conf"
sudo modprobe kvm-amd
```

Ensure the Nested KVM Kernel module parameter for AMD is enabled on the host:

```
cat /sys/module/kvm_amd/parameters/nested
1

modinfo kvm_amd | grep -i nested
parm:           nested:int
```

To make the above value persistent across reboots, add an entry in /etc/modprobe.d/dist.conf so it looks as below:

```
cat /etc/modprobe.d/dist.conf
options kvm-amd nested=y
```

Expose Virtualization Extensions to DevStack VM

Edit the VMs libvirt XML configuration via `virsh` utility:

```
sudo virsh edit devstack-vm
```

Add the below snippet to expose the host CPU features to the VM:

```
<cpu mode='host-passthrough'>
</cpu>
```

Ensure DevStack VM is Using KVM

Before invoking `stack.sh` in the VM, ensure that KVM is enabled. This can be verified by checking for the presence of the file `/dev/kvm` in your VM. If it is present, DevStack will default to using the config attribute `virt_type = kvm` in `/etc/nova.conf`; otherwise, itll fall back to `virt_type=qemu`, i.e. plain QEMU emulation.

Optionally, to explicitly set the type of virtualization, to KVM, by the libvirt driver in nova, the below config attribute can be used in DevStacks `local.conf`:

```
LIBVIRT_TYPE=kvm
```

Once DevStack is configured successfully, verify if the Nova instances are using KVM by noticing the QEMU CLI invoked by Nova is using the parameter `accel=kvm`, e.g.:

```
ps -ef | grep -i qemu
root     29773     1  0 11:24 ?        00:00:00 /usr/bin/qemu-system-x86_64 -
↪machine accel=kvm [. . .]
```

2.5.7 Nova and DevStack

This is a rough guide to various configuration parameters for nova running with DevStack.

nova-serialproxy

In Juno, nova implemented a spec to allow read/write access to the serial console of an instance via nova-serialproxy.

The service can be enabled by adding `n-sproxy` to `ENABLED_SERVICES`. Further options can be enabled via `local.conf`, e.g.

```
[[post-config|$NOVA_CONF]]
[serial_console]
#
# Options defined in nova.cmd.serialproxy
#

# Host on which to listen for incoming requests (string value)
#serialproxy_host=0.0.0.0

# Port on which to listen for incoming requests (integer
# value)
#serialproxy_port=6083



#
# Options defined in nova.console.serial
#
```

(continues on next page)

(continued from previous page)

```
# Enable serial console related features (boolean value)
#enabled=false
# Do not set this manually.  Instead enable the service as
# outlined above.

# Range of TCP ports to use for serial ports on compute hosts
# (string value)
#port_range=10000:20000

# Location of serial console proxy. (string value)
#base_url=ws://127.0.0.1:6083/

# IP address on which instance serial console should listen
# (string value)
#listen=127.0.0.1

# The address to which proxy clients (like nova-serialproxy)
# should connect (string value)
#proxyclient_address=127.0.0.1
```

Enabling the service is enough to be functional for a single machine DevStack.

These config options are defined in nova.conf.serial_console.

For more information on OpenStack configuration see the OpenStack Compute Service Configuration Reference

Fake virt driver

Nova has a fake virt driver which can be used for scale testing the control plane services or testing move operations between fake compute nodes, for example cold/live migration, evacuate and unshelve.

The fake virt driver does not communicate with any hypervisor, it just reports some fake resource inventory values and keeps track of the state of the guests created, moved and deleted. It is not feature-complete with the compute API but is good enough for most API testing, and is also used within the nova functional tests themselves so is fairly robust.

Configuration

Set the following in your devstack `local.conf`:

```
[[local|localrc]]
VIRT_DRIVER=fake
NUMBER_FAKE_NOVA_COMPUTE=<number>
```

The `NUMBER_FAKE_NOVA_COMPUTE` variable controls the number of fake `nova-compute` services to run and defaults to 1.

When `VIRT_DRIVER=fake` is used, devstack will disable quota checking in nova and neutron automatically. However, other services, like cinder, will still enforce quota limits by default.

Scaling

The actual value to use for `NUMBER_FAKE_NOVA_COMPUTE` depends on factors such as:

- The size of the host (physical or virtualized) on which devstack is running.
- The number of API workers. By default, devstack will run `max($nproc/2, 2)` workers per API service. If you are running several fake compute services on a single host, then consider setting `API_WORKERS=1` in `local.conf`.

In addition, while quota will be disabled in neutron, there is no fake ML2 backend for neutron so creating fake VMs will still result in real ports being created. To create servers without networking, you can specify `--nic=none` when creating the server, for example:

```
$ openstack --os-compute-api-version 2.37 server create --flavor cirros256 \
    --image cirros-0.6.3-x86_64-disk --nic none --wait test-server
```

Note

`--os-compute-api-version` greater than or equal to 2.37 is required to use `--nic=none`.

To avoid overhead from other services which you may not need, disable them in your `local.conf`, for example:

```
disable_service horizon
disable_service tempest
```

2.5.8 Devstack with Octavia Load Balancing

Starting with the OpenStack Pike release, Octavia is now a standalone service providing load balancing services for OpenStack.

This guide will show you how to create a devstack with Octavia API enabled.

Phase 1: Create DevStack + 2 nova instances

First, set up a VM of your choice with at least 8 GB RAM and 16 GB disk space, make sure it is updated. Install git and any other developer tools you find useful.

Install devstack:

```
git clone https://opendev.org/openstack/devstack
cd devstack/tools
sudo ./create-stack-user.sh
cd ../..
sudo mv devstack /opt/stack
sudo chown -R stack.stack /opt/stack/devstack
```

This will clone the current devstack code locally, then setup the stack account that devstack services will run under. Finally, it will move devstack into its default location in /opt/stack/devstack.

Edit your `/opt/stack/devstack/local.conf` to look like:

```
[[local|localrc]]
# ===== BEGIN localrc =====
DATABASE_PASSWORD=password
ADMIN_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=password
RABBIT_PASSWORD=password
GIT_BASE=https://opendev.org
# Optional settings:
# OCTAVIA_AMP_BASE_OS=centos
# OCTAVIA_AMP_DISTRIBUTION_RELEASE_ID=9-stream
# OCTAVIA_AMP_IMAGE_SIZE=3
# OCTAVIA_LB_TOPOLOGY=ACTIVE_STANDBY
# OCTAVIA_ENABLE_AMPHORAV2_JOBBOARD=True
# LIBS_FROM_GIT+=octavia-lib,
# Enable Logging
LOGFILE=$DEST/logs/stack.sh.log
VERBOSE=True
LOG_COLOR=True
enable_service rabbit
enable_plugin neutron $GIT_BASE/openstack/neutron
# Octavia supports using QoS policies on the VIP port:
enable_service q-qos
enable_service placement-api placement-client
# Octavia services
enable_plugin octavia $GIT_BASE/openstack/octavia master
enable_plugin octavia-dashboard $GIT_BASE/openstack/octavia-dashboard
enable_plugin ovn-octavia-provider $GIT_BASE/openstack/ovn-octavia-provider
enable_plugin octavia-tempest-plugin $GIT_BASE/openstack/octavia-tempest-
↪plugin
enable_service octavia o-api o-cw o-hm o-hk o-da
# If you are enabling barbican for TLS offload in Octavia, include it here.
# enable_plugin barbican $GIT_BASE/openstack/barbican
# enable_service barbican
# Cinder (optional)
disable_service c-api c-vol c-sch
# Tempest
enable_service tempest
# ===== END localrc =====
```

Note

For best performance it is highly recommended to use KVM virtualization instead of QEMU. Also make sure nested virtualization is enabled as documented in *the respective guide*. By adding `LIBVIRT_CPU_MODE="host-passthrough"` to your `local.conf` you enable the guest VMs to make use of all features your hosts CPU provides.

Run stack.sh and do some sanity checks:

```
sudo su - stack
cd /opt/stack/devstack
./stack.sh
. ./openrc

openstack network list  # should show public and private networks
```

Create two nova instances that we can use as test http servers:

```
# create nova instances on private network
openstack server create --image $(openstack image list | awk '/ cirros-.*-x86_
↪64-.* / {print $2}') --flavor 1 --nic net-id=$(openstack network list | awk
↪'/ private / {print $2}') node1
openstack server create --image $(openstack image list | awk '/ cirros-.*-x86_
↪64-.* / {print $2}') --flavor 1 --nic net-id=$(openstack network list | awk
↪'/ private / {print $2}') node2
openstack server list # should show the nova instances just created

# add secgroup rules to allow ssh etc..
openstack security group rule create default --protocol icmp
openstack security group rule create default --protocol tcp --dst-port 22:22
openstack security group rule create default --protocol tcp --dst-port 80:80
```

Set up a simple web server on each of these instances. One possibility is to use the Golang test server that is used by the Octavia project for CI testing as well. Copy the binary to your instances and start it as shown below (username cirros, password gocubsgo):

```
INST_IP=<instance IP>
scp -O test_server.bin cirros@${INST_IP}:
ssh -f cirros@${INST_IP} ./test_server.bin -id ${INST_IP}
```

When started this way the test server will respond to HTTP requests with its own IP.

Phase 2: Create your load balancer

Create your load balancer:

```
openstack loadbalancer create --wait --name lb1 --vip-subnet-id private-subnet
openstack loadbalancer listener create --wait --protocol HTTP --protocol-port␣
↪80 --name listener1 lb1
openstack loadbalancer pool create --wait --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP --name pool1
openstack loadbalancer healthmonitor create --wait --delay 5 --timeout 2 --
↪max-retries 1 --type HTTP pool1
openstack loadbalancer member create --wait --subnet-id private-subnet --
↪address <web server 1 address> --protocol-port 80 pool1
openstack loadbalancer member create --wait --subnet-id private-subnet --
↪address <web server 2 address> --protocol-port 80 pool1
```

Please note: The <web server # address> fields are the IP addresses of the nova servers created in Phase 1. Also note, using the API directly you can do all of the above commands in one API call.

Phase 3: Test your load balancer

```
openstack loadbalancer show lb1 # Note the vip_address
curl http://<vip_address>
curl http://<vip_address>
```

This should show the Welcome to <IP> message from each member server.

2.5.9 Deploying DevStack with LDAP

The OpenStack Identity service has the ability to integrate with LDAP. The goal of this guide is to walk you through setting up an LDAP-backed OpenStack development environment.

Introduction

LDAP support in keystone is read-only. You can use it to back an entire OpenStack deployment to a single LDAP server, or you can use it to back separate LDAP servers to specific keystone domains. Users within those domains can authenticate against keystone, assume role assignments, and interact with other OpenStack services.

Configuration

To deploy an OpenLDAP server, make sure `ldap` is added to the list of `ENABLED_SERVICES` in the `local.conf` file:

```
enable_service ldap
```

Devstack will require a password to set up an LDAP administrator. This administrative user is also the bind user specified in keystones configuration files, similar to a `keystone` user for MySQL databases.

Devstack will prompt you for a password when running `stack.sh` if `LDAP_PASSWORD` is not set. You can add the following to your `local.conf`:

```
LDAP_PASSWORD=super_secret_password
```

At this point, devstack should have everything it needs to deploy OpenLDAP, bootstrap it with a minimal set of users, and configure it to back to a domain in keystone. You can do this by running the `stack.sh` script:

```
$ ./stack.sh
```

Once `stack.sh` completes, you should have a running keystone deployment with a basic set of users. It is important to note that not all users will live within LDAP. Instead, keystone will back different domains to different identity sources. For example, the `default` domain will be backed by MySQL. This is usually where youll find your administrative and services users. If you query keystone for a list of domains, you should see a domain called `Users`. This domain is set up by devstack and points to OpenLDAP.

User Management

Initially, there will only be two users in the LDAP server. The `Manager` user is used by keystone to talk to OpenLDAP. The `demo` user is a generic user that you should be able to see if you query keystone for users within the `Users` domain. Both of these users were added to LDAP using basic LDAP utilities installed by devstack (e.g. `ldap-utils`) and LDIFs. The LDIFs used to create these users can be found in `devstack/files/ldap/`.

Listing Users

To list all users in LDAP directly, you can use `ldapsearch` with the LDAP user bootstrapped by devstack:

```
$ ldapsearch -x -w LDAP_PASSWORD -D cn=Manager,dc=openstack,dc=org \
    -H ldap://localhost -b dc=openstack,dc=org
```

As you can see, devstack creates an OpenStack domain called `openstack.org` as a container for the `Manager` and `demo` users.

Creating Users

Since keystones LDAP integration is read-only, users must be added directly to LDAP. Users added directly to OpenLDAP will automatically be placed into the `Users` domain.

LDIFs can be used to add users via the command line. The following is an example LDIF that can be used to create a new LDAP user, lets call it `peter.ldif.in`:

```
dn: cn=peter,ou=Users,dc=openstack,dc=org
cn: peter
displayName: Peter Quill
givenName: Peter Quill
mail: starlord@openstack.org
objectClass: inetOrgPerson
objectClass: top
sn: peter
uid: peter
userPassword: im-a-better-pilot-than-rocket
```

Now, we use the `Manager` user to create a user for Peter in LDAP:

```
$ ldapadd -x -w LDAP_PASSWORD -D cn=Manager,dc=openstack,dc=org \
    -H ldap://localhost -c -f peter.ldif.in
```

We should be able to assign Peter roles on projects. After Peter has some level of authorization, he should be able to login to Horizon by specifying the `Users` domain and using his `peter` username and password. Authorization can be given to Peter by creating a project within the `Users` domain and giving him a role assignment on that project:

```
$ openstack project create --domain Users awesome-mix-vol-1
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 61a2de23107c46bea2d758167af707b9 |
| enabled     | True                             |
| id          | 7d422396d54945cdac8fe1e8e32baec4 |
| is_domain   | False                            |
| name        | awesome-mix-vol-1                |
| parent_id   | 61a2de23107c46bea2d758167af707b9 |
| tags        | []                               |
+-------------+----------------------------------+
```

(continues on next page)

(continued from previous page)

```
$ openstack role add --user peter --user-domain Users \
      --project awesome-mix-vol-1 --project-domain Users admin
```

Deleting Users

We can use the same basic steps to remove users from LDAP, but instead of using LDIFs, we can just pass the dn of the user we want to delete:

```
$ ldapdelete -x -w LDAP_PASSWORD -D cn=Manager,dc=openstack,dc=org \
    -H ldap://localhost cn=peter,ou=Users,dc=openstack,dc=org
```

Group Management

Like users, groups are considered specific identities. This means that groups also fall under the same read-only constraints as users and they can be managed directly with LDAP in the same way users are with LDIFs.

Adding Groups

Lets define a specific group with the following LDIF:

```
dn: cn=guardians,ou=UserGroups,dc=openstack,dc=org
objectClass: groupOfNames
cn: guardians
description: Guardians of the Galaxy
member: cn=peter,dc=openstack,dc=org
member: cn=gamora,dc=openstack,dc=org
member: cn=drax,dc=openstack,dc=org
member: cn=rocket,dc=openstack,dc=org
member: cn=groot,dc=openstack,dc=org
```

We can create the group using the same ldapadd command as we did with users:

```
$ ldapadd -x -w LDAP_PASSWORD -D cn=Manager,dc=openstack,dc=org \
    -H ldap://localhost -c -f guardian-group.ldif.in
```

If we check the group membership in Horizon, well see that only Peter is a member of the guardians group, despite the whole crew being specified in the LDIF. Once those accounts are created in LDAP, they will automatically be added to the guardians group. They will also assume any role assignments given to the guardians group.

Deleting Groups

Just like users, groups can be deleted using the dn:

```
$ ldapdelete -x -w LDAP_PASSWORD -D cn=Manager,dc=openstack,dc=org \
    -H ldap://localhost cn=guardians,ou=UserGroups,dc=openstack,dc=org
```

Note that this operation will not remove users within that group. It will only remove the group itself and the memberships any users had with that group.

2.5.10 All-In-One Single VM

Run *OpenStack in a VM*. The VMs launched in your cloud will be slow as they are running in QEMU (emulation), but it is useful if you dont have spare hardware laying around. *[Read]*

2.5.11 All-In-One Single Machine

Run *OpenStack on dedicated hardware* This can include a server-class machine or a laptop at home. *[Read]*

2.5.12 All-In-One LXC Container

Run *OpenStack in a LXC container*. Beneficial for intermediate and advanced users. The VMs launched in this cloud will be fully accelerated but not all OpenStack features are supported. *[Read]*

2.5.13 Multi-Node Lab

Setup a *multi-node cluster* with dedicated VLANs for VMs & Management. *[Read]*

2.5.14 DevStack with Neutron Networking

Building a DevStack cluster with *Neutron Networking*. This guide is meant for building lab environments with a dedicated control node and multiple compute nodes.

2.5.15 DevStack with KVM-based Nested Virtualization

Procedure to setup *DevStack with KVM-based Nested Virtualization*. With this setup, Nova instances will be more performant than with plain QEMU emulation.

2.5.16 Nova and devstack

Guide to working with nova features *Nova and devstack*.

2.5.17 Configure Octavia

Guide on *Configure Octavia*.

2.5.18 Deploying DevStack with LDAP

Guide to setting up *DevStack with LDAP*.

2.6 Contributing to DevStack

2.6.1 General

DevStack is written in UNIX shell script. It uses a number of bash-isms and so is limited to Bash (version 4 and up) and compatible shells. Shell script was chosen because it best illustrates the steps used to set up and interact with OpenStack components.

DevStacks official repository is located on opendev.org at https://opendev.org/openstack/devstack. Besides the master branch that tracks the OpenStack trunk branches a separate branch is maintained for all OpenStack releases starting with Diablo (stable/diablo).

Contributing code to DevStack follows the usual OpenStack process as described in How To Contribute in the OpenStack wiki. DevStacks LaunchPad project contains the usual links for blueprints, bugs, etc.

The Gerrit review queue is used for all commits.

The primary script in DevStack is `stack.sh`, which performs the bulk of the work for DevStacks use cases. There is a subscript `functions` that contains generally useful shell functions and is used by a number of the scripts in DevStack.

A number of additional scripts can be found in the `tools` directory that may be useful in supporting DevStack installations. Of particular note are `info.sh` to collect and report information about the installed system, and `install_prereqs.sh` that handles installation of the prerequisite packages for DevStack. It is suitable, for example, to pre-load a system for making a snapshot.

2.6.2 Repo Layout

The DevStack repo generally keeps all of the primary scripts at the root level.

`doc` - Contains the Sphinx source for the documentation. A complete doc build can be run with `tox -edocs`.

`extras.d` - Contains the dispatch scripts called by the hooks in `stack.sh`, `unstack.sh` and `clean.sh`. See *the plugins docs* for more information.

`files` - Contains a variety of otherwise lost files used in configuring and operating DevStack. This includes templates for configuration files and the system dependency information. This is also where image files are downloaded and expanded if necessary.

`lib` - Contains the sub-scripts specific to each project. This is where the work of managing a projects services is located. Each top-level project (Keystone, Nova, etc) has a file here. Additionally there are some for system services and project plugins. These variables and functions are also used by related projects, such as Grenade, to manage a DevStack installation.

`samples` - Contains a sample of the local files not included in the DevStack repo.

`tests` - the DevStack test suite is rather sparse, mostly consisting of test of specific fragile functions in the `functions` and `functions-common` files.

`tools` - Contains a collection of stand-alone scripts. While these may reference the top-level DevStack configuration they can generally be run alone.

2.6.3 Scripts

DevStack scripts should generally begin by calling `env(1)` in the shebang line:

```
#!/usr/bin/env bash
```

Sometimes the script needs to know the location of the DevStack install directory. `TOP_DIR` should always point there, even if the script itself is located in a subdirectory:

```
# Keep track of the current DevStack directory.
TOP_DIR=$(cd $(dirname "$0") && pwd)
```

Many scripts will utilize shared functions from the `functions` file. There are also rc files (`stackrc` and `openrc`) that are often included to set the primary configuration of the user environment:

```
# Keep track of the current DevStack directory.
TOP_DIR=$(cd $(dirname "$0") && pwd)

```

(continues on next page)

(continued from previous page)

```
# Import common functions
source $TOP_DIR/functions

# Import configuration
source $TOP_DIR/openrc
```

`stack.sh` is a rather large monolithic script that flows through from beginning to end. It has been broken down into project-specific subscripts (as noted above) located in `lib` to make `stack.sh` more manageable and to promote code reuse.

These library sub-scripts have a number of fixed entry points, some of which may just be stubs. These entry points will be called by `stack.sh` in the following order:

```
install_XXXX
configure_XXXX
init_XXXX
start_XXXX
stop_XXXX
cleanup_XXXX
```

There is a sub-script template in `lib/templates` to be used in creating new service sub-scripts. The comments in <> are meta comments describing how to use the template and should be removed.

In order to show the dependencies and conditions under which project functions are executed the top-level conditional testing for things like `is_service_enabled` should be done in `stack.sh`. There may be nested conditionals that need to be in the sub-script, such as testing for keystone being enabled in `configure_swift()`.

2.6.4 stackrc

`stackrc` is the global configuration file for DevStack. It is responsible for calling `local.conf` (or `localrc` if it exists) so local user configuration is recognized.

The criteria for what belongs in `stackrc` can be vaguely summarized as follows:

- All project repositories and branches handled directly in `stack.sh`
- Global configuration that may be referenced in `local.conf`, i.e. `DEST`, `DATA_DIR`
- Global service configuration like `ENABLED_SERVICES`
- Variables used by multiple services that do not have a clear owner, i.e. `VOLUME_BACKING_FILE_SIZE` (nova-compute and cinder) or `PUBLIC_NETWORK_NAME` (only neutron but formerly nova-network too)
- Variables that can not be cleanly declared in a project file due to dependency ordering, i.e. the order of sourcing the project files can not be changed for other reasons but the earlier file needs to dereference a variable set in the later file. This should be rare.

Also, variable declarations in `stackrc` before `local.conf` is sourced do NOT allow overriding (the form `FOO=${FOO:-baz}`); if they did then they can already be changed in `local.conf` and can stay in the project file.

2.6.5 Documentation

The DevStack repo now contains all of the static pages of devstack.org in the `doc/source` directory. The OpenStack CI system rebuilds the docs after every commit and updates devstack.org (now a redirect to https://docs.openstack.org/devstack/latest/).

All of the scripts are processed with shocco to render them with the comments as text describing the script below. For this reason we tend to be a little verbose in the comments _ABOVE_ the code they pertain to. Shocco also supports Markdown formatting in the comments; use it sparingly. Specifically, `stack.sh` uses Markdown headers to divide the script into logical sections.

The script used to drive <code>shocco</code> is <code>tools/build_docs.sh</code>. The complete docs build is also handled with <code>tox -edocs</code> per the OpenStack project standard.

Bash Style Guidelines

DevStack defines a bash set of best practices for maintaining large collections of bash scripts. These should be considered as part of the review process.

DevStack uses the bashate style checker to enforce basic guidelines, similar to pep8 and flake8 tools for Python. The list below is not complete for what bashate checks, nor is it all checked by bashate. So many lines of code, so little time.

2.6.6 Whitespace Rules

- lines should not include trailing whitespace
- there should be no hard tabs in the file
- indents are 4 spaces, and all indentation should be some multiple of them

2.6.7 Control Structure Rules

- then should be on the same line as the if
- do should be on the same line as the for

Example:

```
if [[ -r $TOP_DIR/local.conf ]]; then
    LRC=$(get_meta_section_files $TOP_DIR/local.conf local)
    for lfile in $LRC; do
        if [[ "$lfile" == "localrc" ]]; then
            if [[ -r $TOP_DIR/localrc ]]; then
                warn $LINENO "localrc and local.conf:[[local]] both exist,␣
↪using localrc"
            else
                echo "# Generated file, do not edit" >$TOP_DIR/.localrc.auto
                get_meta_section $TOP_DIR/local.conf local $lfile >>$TOP_DIR/.
↪localrc.auto
            fi
        fi
    done
fi
```

2.6.8 Variables and Functions

- functions should be used whenever possible for clarity
- functions should use `local` variables as much as possible to ensure they are isolated from the rest of the environment
- local variables should be lower case, global variables should be upper case
- function names should_have_underscores, NotCamelCase.
- functions should be declared as per the regex ^function foo {$ with code starting on the next line

2.6.9 Review Criteria

There are some broad criteria that will be followed when reviewing your change

- **Is it passing tests** your change will not be reviewed thoroughly unless the official CI has run successfully against it.
- **Does this belong in DevStack** DevStack reviewers have a default position of no but are ready to be convinced by your change.

  For very large changes, you should consider *the plugins system* to see if your code is better abstracted from the main repository.

  For smaller changes, you should always consider if the change can be encapsulated by per-user settings in `local.conf`. A common example is adding a simple config-option to an `ini` file. Specific flags are not usually required for this, although adding documentation about how to achieve a larger goal (which might include turning on various settings, etc) is always welcome.
- **Work-arounds** often things get broken and DevStack can be in a position to fix them. Work-arounds are fine, but should be presented in the context of fixing the root-cause of the problem. This means it is well-commented in the code and the change-log and mostly likely includes links to changes or bugs that fix the underlying problem.
- **Should this be upstream** DevStack generally does not override default choices provided by projects and attempts to not unexpectedly modify behavior.
- **Context in commit messages** DevStack touches many different areas and reviewers need context around changes to make good decisions. We also always want it to be clear to someone perhaps even years from now why we were motivated to make a change at the time.

2.6.10 Making Changes, Testing, and CI

Changes to Devstack are tested by automated continuous integration jobs that run on a variety of Linux Distros using a handful of common configurations. What this means is that every change to Devstack is self testing. One major benefit of this is that developers do not typically need to add new non voting test jobs to add features to Devstack. Instead the features can be added, then if testing passes with the feature enabled the change is ready to merge (pending code review).

A concrete example of this was the switch from screen based service management to systemd based service management. No new jobs were created for this. Instead the features were added to devstack, tested locally and in CI using a change that enabled the feature, then once the enabling change was passing and the new behavior communicated and documented it was merged.

Using this process has been proven to be effective and leads to quicker implementation of desired features.

2.7 DevStack Networking

An important part of the DevStack experience is networking that works by default for created guests. This might not be optimal for your particular testing environment, so this document tries its best to explain whats going on.

2.7.1 Defaults

If you dont specify any configuration you will get the following:

- neutron (including l3 with openvswitch)
- private project networks for each openstack project
- a floating ip range of 172.24.4.0/24 with the gateway of 172.24.4.1
- the demo project configured with fixed ips on a subnet allocated from the 10.0.0.0/22 range
- a `br-ex` interface controlled by neutron for all its networking (this is not connected to any physical interfaces).
- DNS resolution for guests based on the resolv.conf for your host
- an ip masq rule that allows created guests to route out

This creates an environment which is isolated to the single host. Guests can get to the external network for package updates. Tempest tests will work in this environment.

Note

By default all OpenStack environments have security group rules which block all inbound packets to guests. If you want to be able to ssh / ping your created guests you should run the following.

```
openstack security group rule create --proto icmp --dst-port 0 default
openstack security group rule create --proto tcp --dst-port 22 default
```

2.7.2 Locally Accessible Guests

If you want to make your guests accessible from other machines on your network, we have to connect `br-ex` to a physical interface.

Dedicated Guest Interface

If you have 2 or more interfaces on your devstack server, you can allocate an interface to neutron to fully manage. This **should not** be the same interface you use to ssh into the devstack server itself.

This is done by setting with the `PUBLIC_INTERFACE` attribute.

```
[[local|localrc]]
PUBLIC_INTERFACE=eth1
```

That will put all layer 2 traffic from your guests onto the main network. When running in this mode the ip masq rule is **not** added in your devstack, you are responsible for making routing work on your local network.

Shared Guest Interface

Warning

This is not a recommended configuration. Because of interactions between OVS and bridging, if you reboot your box with active networking you may lose network connectivity to your system.

If you need your guests accessible on the network, but only have 1 interface (using something like a NUC), you can share your one network. But in order for this to work you need to manually set a lot of addresses, and have them all exactly correct.

```
[[local|localrc]]
PUBLIC_INTERFACE=eth0
HOST_IP=10.42.0.52
FLOATING_RANGE=10.42.0.0/24
PUBLIC_NETWORK_GATEWAY=10.42.0.1
Q_FLOATING_ALLOCATION_POOL=start=10.42.0.250,end=10.42.0.254
```

In order for this scenario to work the floating ip network must match the default networking on your server. This breaks HOST_IP detection, as we exclude the floating range by default, so you have to specify that manually.

The `PUBLIC_NETWORK_GATEWAY` is the gateway that server would normally use to get off the network. `Q_FLOATING_ALLOCATION_POOL` controls the range of floating ips that will be handed out. As we are sharing your existing network, youll want to give it a slice that your local dhcp server is not allocating. Otherwise you could easily have conflicting ip addresses, and cause havoc with your local network.

2.7.3 Private Network Addressing

The private networks addresses are controlled by the `IPV4_ADDRS_SAFE_TO_USE` and the `IPV6_ADDRS_SAFE_TO_USE` variables. This allows users to specify one single variable of safe internal IPs to use that will be referenced whether or not subnetpools are in use.

For IPv4, `FIXED_RANGE` and `SUBNETPOOL_PREFIX_V4` will just default to the value of `IPV4_ADDRS_SAFE_TO_USE` directly.

For IPv6, `FIXED_RANGE_V6` will default to the first /64 of the value of `IPV6_ADDRS_SAFE_TO_USE`. If `IPV6_ADDRS_SAFE_TO_USE` is /64 or smaller, `FIXED_RANGE_V6` will just use the value of that directly. `SUBNETPOOL_PREFIX_V6` will just default to the value of `IPV6_ADDRS_SAFE_TO_USE` directly.

2.7.4 SSH access to instances

To validate connectivity, you can create an instance using the `$PRIVATE_NETWORK_NAME` network (default: `private`), create a floating IP using the `$PUBLIC_NETWORK_NAME` network (default: `public`), and attach this floating IP to the instance:

```
openstack keypair create --public-key ~/.ssh/id_rsa.pub test-keypair
openstack server create --network private --key-name test-keypair ... test-
↪server
fip_id=$(openstack floating ip create public -f value -c id)
openstack server add floating ip test-server ${fip_id}
```

Once done, ensure you have enabled SSH and ICMP (ping) access for the security group used for the instance. You can either create a custom security group and specify it when creating the instance or add it after creation, or you can modify the `default` security group created by default for each project. Lets do the latter:

```
openstack security group rule create --proto icmp --dst-port 0 default
openstack security group rule create --proto tcp --dst-port 22 default
```

Finally, SSH into the instance. If you used the Cirros instance uploaded by default, then you can run the following:

```
openstack server ssh test-server -- -l cirros
```

This will connect using the `cirros` user and the keypair you configured when creating the instance.

2.7.5 Remote SSH access to instances

You can also SSH to created instances on your DevStack host from other hosts. This can be helpful if you are e.g. deploying DevStack in a VM on an existing cloud and wish to do development on your local machine. There are a few ways to do this.

Configure instances to be locally accessible

The most obvious way is to configure guests to be locally accessible, as described above. This has the advantage of requiring no further effort on the client. However, it is more involved and requires either support from your cloud or some inadvisable workarounds.

Use your DevStack host as a jump host

You can choose to use your DevStack host as a jump host. To SSH to a instance this way, pass the standard `-J` option to the `openstack ssh` / `ssh` command. For example:

```
openstack server ssh test-server -- -l cirros -J username@devstack-host
```

(where `test-server` is name of an existing instance, as described *previously*, and `username` and `devstack-host` are the username and hostname of your DevStack host).

This can also be configured via your `~/.ssh/config` file, making it rather effortless. However, it only allows SSH access. If you want to access e.g. a web application on the instance, you will need to configure an SSH tunnel and forward select ports using the `-L` option. For example, to forward HTTP traffic:

```
openstack server ssh test-server -- -l cirros -L 8080:username@devstack-
↪host:80
```

(where `test-server` is name of an existing instance, as described *previously*, and `username` and `devstack-host` are the username and hostname of your DevStack host).

As you can imagine, this can quickly get out of hand, particularly for more complex guest applications with multiple ports.

Use a proxy or VPN tool

You can use a proxy or VPN tool to enable tunneling for the floating IP address range of the `$PUBLIC_NETWORK_NAME` network (default: `public`) defined by `$FLOATING_RANGE` (default: `172.24.4.0/24`). There are many such tools available to do this. For example, we could use a useful utility called shuttle. To enable tunneling using `shuttle`, first ensure you have allowed SSH and HTTP(S) traffic to your DevStack host. Allowing HTTP(S) traffic is necessary so you can use the OpenStack APIs remotely. How you do this will depend on where your DevStack host is running. Once this is done, install `sshuttle` on your localhost:

```
sudo apt-get install sshuttle || dnf install sshuttle
```

Finally, start `sshuttle` on your localhost using the floating IP address range. For example, assuming you are using the default value for `$FLOATING_RANGE`, you can do:

```
sshuttle -r username@devstack-host 172.24.4.0/24
```

(where `username` and `devstack-host` are the username and hostname of your DevStack host).

You should now be able to create an instance and SSH into it:

```
openstack server ssh test-server -- -l cirros
```

(where `test-server` is name of an existing instance, as described *previously*)

2.8 Overview

DevStack has evolved to support a large number of configuration options and alternative platforms and support services. That evolution has grown well beyond what was originally intended and the majority of configuration combinations are rarely, if ever, tested. DevStack is not a general OpenStack installer and was never meant to be everything to everyone.

Below is a list of what is specifically is supported (read that as tested) going forward.

2.8.1 Supported Components

Base OS

The OpenStack Technical Committee (TC) has defined the current CI strategy to include the latest Ubuntu release and the latest RHEL release.

- Ubuntu: current LTS release plus current development release
- RHEL/CentOS/RockyLinux: current major release
- Other OS platforms may continue to be included but the maintenance of those platforms shall not be assumed simply due to their presence. Having a listed point-of-contact for each additional OS will greatly increase its chance of being well-maintained.
- Patches for Ubuntu and/or RockyLinux will not be held up due to side-effects on other OS platforms.

Databases

As packaged by the host OS

- MySQL

Queues

As packaged by the host OS

- Rabbit

Web Server

As packaged by the host OS

- Apache

Services

The default services configured by DevStack are Identity (keystone), Object Storage (swift), Image Service (glance), Block Storage (cinder), Compute (nova), Placement (placement), Networking (neutron), Dashboard (horizon).

Additional services not included directly in DevStack can be tied in to `stack.sh` using the *plugin mechanism* to call scripts that perform the configuration and startup of the service.

Node Configurations

- single node
- multi-node configurations as are tested by the gate

2.9 DevStack Plugin Registry

The following list is an automatically-generated collection of available DevStack plugins. This includes, but is not limited to, official OpenStack projects.

Plugin Name	URL
openstack/aetos	https://opendev.org/openstack/aetos
openstack/aodh	https://opendev.org/openstack/aodh
openstack/barbican	https://opendev.org/openstack/barbican
openstack/blazar	https://opendev.org/openstack/blazar
openstack/ceilometer	https://opendev.org/openstack/ceilometer
openstack/cloudkitty	https://opendev.org/openstack/cloudkitty
openstack/cyborg	https://opendev.org/openstack/cyborg
openstack/designate	https://opendev.org/openstack/designate
openstack/designate-tempest-plugin	https://opendev.org/openstack/designate-tempest-plugin
openstack/devstack-plugin-amqp1	https://opendev.org/openstack/devstack-plugin-amqp1
openstack/devstack-plugin-ceph	https://opendev.org/openstack/devstack-plugin-ceph
openstack/devstack-plugin-container	https://opendev.org/openstack/devstack-plugin-container
openstack/devstack-plugin-kafka	https://opendev.org/openstack/devstack-plugin-kafka
openstack/devstack-plugin-nfs	https://opendev.org/openstack/devstack-plugin-nfs

continues on next page

Table 1 – continued from previous page

Plugin Name	URL
openstack/devstack-plugin-open-cas	https://opendev.org/openstack/devstack-plugin-open-cas
openstack/devstack-plugin-prometheus	https://opendev.org/openstack/devstack-plugin-prometheus
openstack/freezer	https://opendev.org/openstack/freezer
openstack/freezer-api	https://opendev.org/openstack/freezer-api
openstack/freezer-tempest-plugin	https://opendev.org/openstack/freezer-tempest-plugin
openstack/freezer-web-ui	https://opendev.org/openstack/freezer-web-ui
openstack/grian-ui	https://opendev.org/openstack/grian-ui
openstack/heat	https://opendev.org/openstack/heat
openstack/heat-dashboard	https://opendev.org/openstack/heat-dashboard
openstack/ironic	https://opendev.org/openstack/ironic
openstack/ironic-inspector	https://opendev.org/openstack/ironic-inspector
openstack/ironic-prometheus-exporter	https://opendev.org/openstack/ironic-prometheus-exporter
openstack/ironic-ui	https://opendev.org/openstack/ironic-ui
openstack/keystone	https://opendev.org/openstack/keystone
openstack/kuryr-libnetwork	https://opendev.org/openstack/kuryr-libnetwork
openstack/magnum	https://opendev.org/openstack/magnum
openstack/magnum-ui	https://opendev.org/openstack/magnum-ui
openstack/manila	https://opendev.org/openstack/manila
openstack/manila-tempest-plugin	https://opendev.org/openstack/manila-tempest-plugin
openstack/manila-ui	https://opendev.org/openstack/manila-ui
openstack/masakari	https://opendev.org/openstack/masakari
openstack/mistral	https://opendev.org/openstack/mistral
openstack/monasca-api	https://opendev.org/openstack/monasca-api
openstack/monasca-events-api	https://opendev.org/openstack/monasca-events-api
openstack/monasca-tempest-plugin	https://opendev.org/openstack/monasca-tempest-plugin
openstack/networking-bagpipe	https://opendev.org/openstack/networking-bagpipe
openstack/networking-baremetal	https://opendev.org/openstack/networking-baremetal
openstack/networking-bgpvpn	https://opendev.org/openstack/networking-bgpvpn
openstack/networking-generic-switch	https://opendev.org/openstack/networking-generic-switch
openstack/networking-sfc	https://opendev.org/openstack/networking-sfc
openstack/neutron	https://opendev.org/openstack/neutron
openstack/neutron-dynamic-routing	https://opendev.org/openstack/neutron-dynamic-routing
openstack/neutron-fwaas	https://opendev.org/openstack/neutron-fwaas
openstack/neutron-fwaas-dashboard	https://opendev.org/openstack/neutron-fwaas-dashboard
openstack/neutron-tempest-plugin	https://opendev.org/openstack/neutron-tempest-plugin
openstack/neutron-vpnaas	https://opendev.org/openstack/neutron-vpnaas
openstack/neutron-vpnaas-dashboard	https://opendev.org/openstack/neutron-vpnaas-dashboard
openstack/nova	https://opendev.org/openstack/nova
openstack/octavia	https://opendev.org/openstack/octavia
openstack/octavia-dashboard	https://opendev.org/openstack/octavia-dashboard
openstack/octavia-tempest-plugin	https://opendev.org/openstack/octavia-tempest-plugin
openstack/openstacksdk	https://opendev.org/openstack/openstacksdk
openstack/osprofiler	https://opendev.org/openstack/osprofiler
openstack/ovn-bgp-agent	https://opendev.org/openstack/ovn-bgp-agent
openstack/ovn-octavia-provider	https://opendev.org/openstack/ovn-octavia-provider
openstack/rally-openstack	https://opendev.org/openstack/rally-openstack
openstack/shade	https://opendev.org/openstack/shade
openstack/skyline-apiserver	https://opendev.org/openstack/skyline-apiserver

continues on next page

Table 1 – continued from previous page

Plugin Name	URL
openstack/storlets	https://opendev.org/openstack/storlets
openstack/tacker	https://opendev.org/openstack/tacker
openstack/tap-as-a-service	https://opendev.org/openstack/tap-as-a-service
openstack/telemetry-tempest-plugin	https://opendev.org/openstack/telemetry-tempest-plugin
openstack/trove	https://opendev.org/openstack/trove
openstack/trove-dashboard	https://opendev.org/openstack/trove-dashboard
openstack/venus	https://opendev.org/openstack/venus
openstack/venus-dashboard	https://opendev.org/openstack/venus-dashboard
openstack/vitrage	https://opendev.org/openstack/vitrage
openstack/vitrage-dashboard	https://opendev.org/openstack/vitrage-dashboard
openstack/vitrage-tempest-plugin	https://opendev.org/openstack/vitrage-tempest-plugin
openstack/watcher	https://opendev.org/openstack/watcher
openstack/watcher-dashboard	https://opendev.org/openstack/watcher-dashboard
openstack/whitebox-tempest-plugin	https://opendev.org/openstack/whitebox-tempest-plugin
openstack/zaqar	https://opendev.org/openstack/zaqar
openstack/zaqar-ui	https://opendev.org/openstack/zaqar-ui
openstack/zun	https://opendev.org/openstack/zun
openstack/zun-ui	https://opendev.org/openstack/zun-ui
performa/os-faults	https://opendev.org/performa/os-faults
starlingx/config	https://opendev.org/starlingx/config
starlingx/fault	https://opendev.org/starlingx/fault
starlingx/ha	https://opendev.org/starlingx/ha
starlingx/integ	https://opendev.org/starlingx/integ
starlingx/metal	https://opendev.org/starlingx/metal
starlingx/nfv	https://opendev.org/starlingx/nfv
starlingx/update	https://opendev.org/starlingx/update
vexxhost/openstack-operator	https://opendev.org/vexxhost/openstack-operator
x/almanach	https://opendev.org/x/almanach
x/bilean	https://opendev.org/x/bilean
x/broadview-collector	https://opendev.org/x/broadview-collector
x/collectd-openstack-plugins	https://opendev.org/x/collectd-openstack-plugins
x/devstack-plugin-additional-pkg-repos	https://opendev.org/x/devstack-plugin-additional-pkg-repos
x/devstack-plugin-glusterfs	https://opendev.org/x/devstack-plugin-glusterfs
x/devstack-plugin-hdfs	https://opendev.org/x/devstack-plugin-hdfs
x/devstack-plugin-libvirt-qemu	https://opendev.org/x/devstack-plugin-libvirt-qemu
x/devstack-plugin-mariadb	https://opendev.org/x/devstack-plugin-mariadb
x/devstack-plugin-tobiko	https://opendev.org/x/devstack-plugin-tobiko
x/devstack-plugin-vmax	https://opendev.org/x/devstack-plugin-vmax
x/drbd-devstack	https://opendev.org/x/drbd-devstack
x/fenix	https://opendev.org/x/fenix
x/gce-api	https://opendev.org/x/gce-api
x/glare	https://opendev.org/x/glare
x/group-based-policy	https://opendev.org/x/group-based-policy
x/gyan	https://opendev.org/x/gyan
x/horizon-mellanox	https://opendev.org/x/horizon-mellanox
x/ironic-staging-drivers	https://opendev.org/x/ironic-staging-drivers
x/kingbird	https://opendev.org/x/kingbird
x/meteos	https://opendev.org/x/meteos

continues on next page

Table 1 – continued from previous page

Plugin Name	URL
x/meteos-ui	https://opendev.org/x/meteos-ui
x/mixmatch	https://opendev.org/x/mixmatch
x/mogan	https://opendev.org/x/mogan
x/mogan-ui	https://opendev.org/x/mogan-ui
x/networking-6wind	https://opendev.org/x/networking-6wind
x/networking-ansible	https://opendev.org/x/networking-ansible
x/networking-arista	https://opendev.org/x/networking-arista
x/networking-brocade	https://opendev.org/x/networking-brocade
x/networking-cisco	https://opendev.org/x/networking-cisco
x/networking-cumulus	https://opendev.org/x/networking-cumulus
x/networking-dpm	https://opendev.org/x/networking-dpm
x/networking-fortinet	https://opendev.org/x/networking-fortinet
x/networking-hpe	https://opendev.org/x/networking-hpe
x/networking-huawei	https://opendev.org/x/networking-huawei
x/networking-infoblox	https://opendev.org/x/networking-infoblox
x/networking-l2gw	https://opendev.org/x/networking-l2gw
x/networking-lagopus	https://opendev.org/x/networking-lagopus
x/networking-mlnx	https://opendev.org/x/networking-mlnx
x/networking-nec	https://opendev.org/x/networking-nec
x/networking-omnipath	https://opendev.org/x/networking-omnipath
x/networking-opencontrail	https://opendev.org/x/networking-opencontrail
x/networking-ovs-dpdk	https://opendev.org/x/networking-ovs-dpdk
x/networking-plumgrid	https://opendev.org/x/networking-plumgrid
x/networking-spp	https://opendev.org/x/networking-spp
x/networking-vpp	https://opendev.org/x/networking-vpp
x/networking-vsphere	https://opendev.org/x/networking-vsphere
x/neutron-classifier	https://opendev.org/x/neutron-classifier
x/nova-dpm	https://opendev.org/x/nova-dpm
x/nova-mksproxy	https://opendev.org/x/nova-mksproxy
x/oaktree	https://opendev.org/x/oaktree
x/omni	https://opendev.org/x/omni
x/os-xenapi	https://opendev.org/x/os-xenapi
x/picasso	https://opendev.org/x/picasso
x/rsd-virt-for-nova	https://opendev.org/x/rsd-virt-for-nova
x/scalpels	https://opendev.org/x/scalpels
x/slogging	https://opendev.org/x/slogging
x/stackube	https://opendev.org/x/stackube
x/tatu	https://opendev.org/x/tatu
x/trio2o	https://opendev.org/x/trio2o
x/valet	https://opendev.org/x/valet
x/vmware-nsx	https://opendev.org/x/vmware-nsx
x/vmware-vspc	https://opendev.org/x/vmware-vspc
x/whitebox-neutron-tempest-plugin	https://opendev.org/x/whitebox-neutron-tempest-plugin

2.10 Plugins

The OpenStack ecosystem is wide and deep, and only growing more so every day. The value of DevStack is that its simple enough to understand what its doing clearly. And yet wed like to support as much of the OpenStack Ecosystem as possible. We do that with plugins.

DevStack plugins are bits of bash code that live outside the DevStack tree. They are called through a strong contract, so these plugins can be sure that they will continue to work in the future as DevStack evolves.

2.10.1 Prerequisites

If you are planning to create a plugin that is going to host a service in the service catalog (that is, your plugin will use the command `get_or_create_service`) please make sure that you apply to the service types authority to reserve a valid service-type. This will help to make sure that all deployments of your service use the same service-type.

2.10.2 Plugin Interface

DevStack supports a standard mechanism for including plugins from external repositories. The plugin interface assumes the following:

An external git repository that includes a `devstack/` top level directory. Inside this directory there can be 3 files.

- `override-defaults` - a file containing global variables that will be sourced before the lib/* files. This allows the plugin to override the defaults that are otherwise set in the lib/* files.

  For example, override-defaults may export CINDER_ENABLED_BACKENDS to include the plugin-specific storage backend and thus be able to override the default lvm only storage backend for Cinder.

- `settings` - a file containing global variables that will be sourced very early in the process. This is helpful if other plugins might depend on this one, and need access to global variables to do their work.

  Your settings should include any `enable_service` lines required by your plugin. This is especially important if you are kicking off services using `run_process` as it only works with enabled services.

  Be careful to allow users to override global-variables for customizing their environment. Usually it is best to provide a default value only if the variable is unset or empty; e.g. in bash syntax `FOO=${FOO:-default}`.

  The file should include a `define_plugin` line to indicate the plugins name, which is the name that should be used by users on enable_plugin lines. It should generally be the last component of the git repo path (e.g., if the plugins repo is openstack/foo, then the name here should be foo)

  ```
  define_plugin <YOUR PLUGIN>
  ```

  If your plugin depends on another plugin, indicate it in this file with one or more lines like the following:

  ```
  plugin_requires <YOUR PLUGIN> <OTHER PLUGIN>
  ```

  For a complete example, if the plugin foo depends on bar, the `settings` file should include:

```
define_plugin foo
plugin_requires foo bar
```

Devstack does not currently use this dependency information, so its important that users continue to add enable_plugin lines in the correct order in `local.conf`, however adding this information allows other tools to consider dependency information when automatically generating `local.conf` files.

- `plugin.sh` - the actual plugin. It is executed by devstack at well defined points during a `stack.sh` run. The plugin.sh internal structure is discussed below.

Plugins are registered by adding the following to the localrc section of `local.conf`.

They are added in the following format:

```
[[local|localrc]]
enable_plugin <NAME> <GITURL> [GITREF]
```

- `name` - an arbitrary name. (ex: glusterfs, docker, zaqar, congress)
- `giturl` - a valid git url that can be cloned
- `gitref` - an optional git ref (branch / ref / tag) that will be cloned. Defaults to master.

An example would be as follows:

```
enable_plugin ec2-api https://opendev.org/openstack/ec2-api
```

2.10.3 plugin.sh contract

`plugin.sh` is a bash script that will be called at specific points during `stack.sh`, `unstack.sh`, and `clean.sh`. It will be called in the following way:

```
source $PATH/TO/plugin.sh <mode> [phase]
```

`mode` can be thought of as the major mode being called, currently one of: `stack`, `unstack`, `clean`. `phase` is used by modes which have multiple points during their run where its necessary to be able to execute code. All existing `mode` and `phase` points are considered **strong contracts** and wont be removed without a reasonable deprecation period. Additional new `mode` or `phase` points may be added at any time if we discover we need them to support additional kinds of plugins in devstack.

The current full list of `mode` and `phase` are:

- **stack** - Called by `stack.sh` four times for different phases of its run:
  - **pre-install** - Called after system (OS) setup is complete and before project source is installed.
  - **install** - Called after the layer 1 and 2 projects source and their dependencies have been installed.
  - **post-config** - Called after the layer 1 and 2 services have been configured. All configuration files for enabled services should exist at this point.
  - **extra** - Called near the end after layer 1 and 2 services have been started.
  - **test-config** - Called at the end of devstack used to configure tempest or any other test environments

- **unstack** - Called by `unstack.sh` before other services are shut down.
- **clean** - Called by `clean.sh` before other services are cleaned, but after `unstack.sh` has been called.

2.10.4 Example plugin

An example plugin would look something as follows.

`devstack/settings`:

```
# settings file for template
enable_service template
```

`devstack/plugin.sh`:

```
# plugin.sh - DevStack plugin.sh dispatch script template

function install_template {
    ...
}

function init_template {
    ...
}

function configure_template {
    ...
}

# check for service enabled
if is_service_enabled template; then

    if [[ "$1" == "stack" && "$2" == "pre-install" ]]; then
        # Set up system services
        echo_summary "Configuring system services Template"
        install_package cowsay

    elif [[ "$1" == "stack" && "$2" == "install" ]]; then
        # Perform installation of service source
        echo_summary "Installing Template"
        install_template

    elif [[ "$1" == "stack" && "$2" == "post-config" ]]; then
        # Configure after the other layer 1 and 2 services have been␣
↪configured
        echo_summary "Configuring Template"
        configure_template

    elif [[ "$1" == "stack" && "$2" == "extra" ]]; then
        # Initialize and start the template service
        echo_summary "Initializing Template"
```

(continues on next page)

(continued from previous page)

```
        init_template
    fi

    if [[ "$1" == "unstack" ]]; then
        # Shut down template services
        # no-op
        :
    fi

    if [[ "$1" == "clean" ]]; then
        # Remove state and transient data
        # Remember clean.sh first calls unstack.sh
        # no-op
        :
    fi
fi
```

2.10.5 Plugin Execution Order

Plugins are run after in tree services at each of the stages above. For example, if you need something to happen before Keystone starts, you should do that at the `post-config` phase.

Multiple plugins can be specified in your `local.conf`. When that happens the plugins will be executed **in order** at each phase. This allows plugins to conceptually depend on each other through documenting to the user the order they must be declared. A formal dependency mechanism is beyond the scope of the current work.

2.10.6 System Packages

Devstack based

Devstack provides a custom framework for getting packages installed at an early phase of its execution. These packages may be defined in a plugin as files that contain new-line separated lists of packages required by the plugin

Supported packaging systems include apt and dnf across multiple distributions. To enable a plugin to hook into this and install package dependencies, packages may be listed at the following locations in the top-level of the plugin repository:

- `./devstack/files/debs/$plugin_name` - Packages to install when running on Ubuntu or Debian.
- `./devstack/files/rpms/$plugin_name` - Packages to install when running on Red Hat, Fedora, or CentOS.

Although there a no plans to remove this method of installing packages, plugins should consider it deprecated for `bindep` support described below.

bindep

The bindep project has become the defacto standard for OpenStack projects to specify binary dependencies.

A plugin may provide a `./devstack/files/bindep.txt` file, which will be called with the *default* profile to install packages. For details on the syntax, etc. see the bindep documentation.

It is also possible to use the `bindep.txt` of projects that are being installed from source with the `-bindep` flag available in install functions. For example

```
if use_library_from_git "diskimage-builder"; then
   GITREPO["diskimage-builder"]=$DISKIMAGE_BUILDER_REPO_URL
   GITDIR["diskimage-builder"]=$DEST/diskimage-builder
   GITBRANCH["diskimage-builder"]=$DISKIMAGE_BUILDER_REPO_REF
   git_clone_by_name "diskimage-builder"
   setup_dev_lib -bindep "diskimage-builder"
fi
```

will result in any packages required by the `bindep.txt` of the `diskimage-builder` project being installed. Note however that jobs that switch projects between source and released/pypi installs (e.g. with a `foo-dsvm` and a `foo-dsvm-src` test to cover both released dependencies and master versions) will have to deal with `bindep.txt` being unavailable without the source directory.

2.10.7 Using Plugins in the OpenStack Gate

For everyday use, DevStack plugins can exist in any git tree thats accessible on the internet. However, when using DevStack plugins in the OpenStack gate, they must live in projects in OpenStacks gerrit. This allows testing of the plugin as well as provides network isolation against upstream git repository failures (which we see often enough to be an issue).

Ideally a plugin will be included within the `devstack` directory of the project they are being tested. For example, the openstack/ec2-api project has its plugin support in its own tree.

However, some times a DevStack plugin might be used solely to configure a backend service that will be used by the rest of OpenStack, so there is no project tree per say. Good examples include: integration of back end storage (e.g. ceph or glusterfs), integration of SDN controllers (e.g. ovn, OpenDayLight), or integration of alternate RPC systems (e.g. zmq, qpid). In these cases the best practice is to build a dedicated `openstack/devstack-plugin-F00` project.

Legacy project-config jobs

To enable a plugin to be used in a gate job, the following lines will be needed in your `jenkins/jobs/<project>.yaml` definition in project-config:

```
# Because we are testing a non standard project, add the
# our project repository. This makes zuul do the right
# reference magic for testing changes.
export PROJECTS="openstack/ec2-api $PROJECTS"

# note the actual url here is somewhat irrelevant because it
# caches in nodepool, however make it a valid url for
# documentation purposes.
export DEVSTACK_LOCAL_CONFIG="enable_plugin ec2-api https://opendev.org/
↪openstack/ec2-api"
```

Zuul v3 jobs

See the `devstack_plugins` example in *Migrating Zuul V2 CI jobs to V3*.

2.10.8 See Also

For additional inspiration on devstack plugins you can check out the *Plugin Registry*.

2.11 Using Systemd in DevStack

By default DevStack is run with all the services as systemd unit files. Systemd is now the default init system for nearly every Linux distro, and systemd encodes and solves many of the problems related to poorly running processes.

2.11.1 Why this instead of screen?

The screen model for DevStack was invented when the number of services that a DevStack user was going to run was typically < 10. This made screen hot keys to jump around very easy. However, the landscape has changed (not all services are stoppable in screen as some are under Apache, there are typically at least 20 items)

There is also a common developer workflow of changing code in more than one service, and needing to restart a bunch of services for that to take effect.

2.11.2 Unit Structure

Note

Originally we actually wanted to do this as user units, however there are issues with running this under non interactive shells. For now, well be running as system units. Some user unit code is left in place in case we can switch back later.

All DevStack user units are created as a part of the DevStack slice given the name `devstack@$servicename.service`. This makes it easy to understand which services are part of the devstack run, and lets us disable / stop them in a single command.

2.11.3 Manipulating Units

Assuming the unit `n-cpu` to make the examples more clear.

Enable a unit (allows it to be started):

```
sudo systemctl enable devstack@n-cpu.service
```

Disable a unit:

```
sudo systemctl disable devstack@n-cpu.service
```

Start a unit:

```
sudo systemctl start devstack@n-cpu.service
```

Stop a unit:

```
sudo systemctl stop devstack@n-cpu.service
```

Restart a unit:

```
sudo systemctl restart devstack@n-cpu.service
```

See status of a unit:

```
sudo systemctl status devstack@n-cpu.service
```

Operating on more than one unit at a time

Systemd supports wildcarding for unit operations. To restart every service in devstack you can do that following:

```
sudo systemctl restart devstack@*
```

Or to see the status of all Nova processes you can do:

```
sudo systemctl status devstack@n-*
```

Well eventually make the unit names a bit more meaningful so that its easier to understand what you are restarting.

2.11.4 Querying Logs

One of the other major things that comes with systemd is journald, a consolidated way to access logs (including querying through structured metadata). This is accessed by the user via `journalctl` command.

Logs can be accessed through `journalctl`. journalctl has powerful query facilities. Well start with some common options.

Follow logs for a specific service:

```
sudo journalctl -f --unit devstack@n-cpu.service
```

Following logs for multiple services simultaneously:

```
sudo journalctl -f --unit devstack@n-cpu.service --unit devstack@n-cond.
↪service
```

or you can even do wild cards to follow all the nova services:

```
sudo journalctl -f --unit devstack@n-*
```

Use higher precision time stamps:

```
sudo journalctl -f -o short-precise --unit devstack@n-cpu.service
```

By default, journalctl strips out unprintable characters, including ASCII color codes. To keep the color codes (which can be interpreted by an appropriate terminal/pager - e.g. `less`, the default):

```
sudo journalctl -a --unit devstack@n-cpu.service
```

When outputting to the terminal using the default pager, long lines will be truncated, but horizontal scrolling is supported via the left/right arrow keys. You can override this by setting the `SYSTEMD_LESS` environment variable to e.g. `FRXM`.

You can pipe the output to another tool, such as `grep`. For example, to find a server instance UUID in the nova logs:

```
sudo journalctl -a --unit devstack@n-* | grep 58391b5c-036f-44d5-bd68-
↪21d3c26349e6
```

See `man 1 journalctl` for more.

2.11.5 Debugging

Using pdb

In order to break into a regular pdb session on a systemd-controlled service, you need to invoke the process manually - that is, take it out of systemds control.

Discover the command systemd is using to run the service:

```
systemctl show devstack@n-sch.service -p ExecStart --no-pager
```

Stop the systemd service:

```
sudo systemctl stop devstack@n-sch.service
```

Inject your breakpoint in the source, e.g.:

```
import pdb; pdb.set_trace()
```

Invoke the command manually:

```
/usr/local/bin/nova-scheduler --config-file /etc/nova/nova.conf
```

Some executables, such as **`nova-compute`**, will need to be executed with a particular group. This will be shown in the systemd unit file:

```
sudo systemctl cat devstack@n-cpu.service | grep Group
```

```
Group = libvirt
```

Use the **`sg`** tool to execute the command as this group:

```
sg libvirt -c '/usr/local/bin/nova-compute --config-file /etc/nova/nova-cpu.
↪conf'
```

Using remote-pdb

remote-pdb works while the process is under systemd control.

Make sure you have remote-pdb installed:

```
sudo pip install remote-pdb
```

Inject your breakpoint in the source, e.g.:

```
import remote_pdb; remote_pdb.set_trace()
```

Restart the relevant service:

```
sudo systemctl restart devstack@n-api.service
```

The remote-pdb code configures the telnet port when `set_trace()` is invoked. Do whatever it takes to hit the instrumented code path, and inspect the logs for a message displaying the listening port:

```
Sep 07 16:36:12 p8-100-neo devstack@n-api.service[772]: RemotePdb session␣
↪open at 127.0.0.1:46771, waiting for connection ...
```

Telnet to that port to enter the pdb session:

```
telnet 127.0.0.1 46771
```

See the remote-pdb home page for more options.

2.11.6 Future Work

user units

It would be great if we could do services as user units, so that there is a clear separation of code being run as not root, to ensure running as root never accidentally gets baked in as an assumption to services. However, user units interact poorly with devstack-gate and the way that commands are run as users with ansible and su.

Maybe someday we can figure that out.

2.11.7 References

- Arch Linux Wiki - https://wiki.archlinux.org/index.php/Systemd/User
- Python interface to journald - https://www.freedesktop.org/software/systemd/python-systemd/journal.html
- Systemd documentation on service files - https://www.freedesktop.org/software/systemd/man/systemd.service.html
- Systemd documentation on exec (can be used to impact service runs) - https://www.freedesktop.org/software/systemd/man/systemd.exec.html

2.12 Tempest

Tempest is the OpenStack Integration test suite. It is installed by default and is used to provide integration testing for many of the OpenStack services. Just like DevStack itself, it is possible to extend Tempest with plugins. In fact, many Tempest plugin packages also include DevStack plugin to do things like pre-create required static resources.

The Tempest documentation provides a thorough guide to using Tempest. However, if you simply wish to run the standard set of Tempest tests against an existing deployment, you can do the following:

```
cd /opt/stack/tempest
/opt/stack/data/venv/bin/tempest run ...
```

The above assumes you have installed DevStack in the default location (configured via the `DEST` configuration variable) and have enabled virtualenv-based installation in the standard location (configured via the `USE_VENV` and `VENV_DEST` configuration variables, respectively).

2.13 Migrating Zuul V2 CI jobs to V3

The OpenStack CI system moved from Zuul v2 to Zuul v3, and all CI jobs moved to the new CI system. All jobs have been migrated automatically to a format compatible with Zuul v3; the jobs produced in this way however are suboptimal and do not use the capabilities introduced by Zuul v3, which allow for re-use of job parts, in the form of Ansible roles, as well as inheritance between jobs.

DevStack hosts a set of roles, plays and jobs that can be used by other repositories to define their DevStack based jobs. To benefit from them, jobs must be migrated from the legacy v2 ones into v3 native format.

This document provides guidance and examples to make the migration process as painless and smooth as possible.

2.13.1 Where to host the job definitions.

In Zuul V3 jobs can be defined in the repository that contains the code they excercise. If you are writing CI jobs for an OpenStack service you can define your DevStack based CI jobs in one of the repositories that host the code for your service. If you have a branchless repo, like a Tempest plugin, that is a convenient choice to host the job definitions since job changes do not have to be backported. For example, see the beginning of the `.zuul.yaml` from the sahara Tempest plugin repo:

```
# In https://opendev.org/openstack/sahara-tests/src/branch/master/.zuul.yaml:
- job:
    name: sahara-tests-tempest
    description: |
      Run Tempest tests from the Sahara plugin.
    parent: devstack-tempest
```

2.13.2 Which base job to start from

If your job needs an OpenStack cloud deployed via DevStack, but you dont plan on running Tempest tests, you can start from one of the base *jobs* defined in the DevStack repo.

The `devstack` job can be used for both single-node jobs and multi-node jobs, and it includes the list of services used in the integrated gate (keystone, glance, nova, cinder, neutron and swift). Different topologies can be achieved by switching the nodeset used in the child job.

The `devstack-base` job is similar to `devstack` but it does not specify any required repo or service to be run in DevStack. It can be useful to setup children jobs that use a very narrow DevStack setup.

If your job needs an OpenStack cloud deployed via DevStack, and you do plan on running Tempest tests, you can start from one of the base jobs defined in the Tempest repo.

The `devstack-tempest` job can be used for both single-node jobs and multi-node jobs. Different topologies can be achieved by switching the nodeset used in the child job.

Jobs can be customized as follows without writing any Ansible code:

- add and/or remove DevStack services
- add or modify DevStack and services configuration
- install DevStack plugins
- extend the number of sub-nodes (multinode only)
- define extra log files and/or directories to be uploaded on logs.o.o
- define extra log file extensions to be rewritten to .txt for ease of access

Tempest jobs can be further customized as follows:

- define the Tempest tox environment to be used
- define the test concurrency
- define the test regular expression

Writing Ansible code, or importing existing custom roles, jobs can be further extended by:

- adding pre and/or post playbooks
- overriding the run playbook, add custom roles

The (partial) example below extends a Tempest single node base job devstack-tempest in the Kuryr repository. The parent job name is defined in job.parent.

```
# https://opendev.org/openstack/kuryr-kubernetes/src/branch/master/.zuul.d/
↪base.yaml:
- job:
    name: kuryr-kubernetes-tempest-base
    parent: devstack-tempest
    description: Base kuryr-kubernetes-job
    required-projects:
      - openstack/devstack-plugin-container
      - openstack/kuryr
      - openstack/kuryr-kubernetes
      - openstack/kuryr-tempest-plugin
      - openstack/neutron-lbaas
    vars:
      tempest_test_regex: '^(kuryr_tempest_plugin.tests.)'
      tox_envlist: 'all'
      devstack_localrc:
        KURYR_K8S_API_PORT: 8080
      devstack_services:
        kubernetes-api: true
        kubernetes-controller-manager: true
        kubernetes-scheduler: true
        kubelet: true
        kuryr-kubernetes: true
        (...)
      devstack_plugins:
        kuryr-kubernetes: https://opendev.org/openstack/kuryr
        devstack-plugin-container: https://opendev.org/openstack/devstack-
↪plugin-container
```

(continues on next page)

(continued from previous page)

```
      neutron-lbaas: https://opendev.org/openstack/neutron-lbaas
    tempest_plugins:
      - kuryr-tempest-plugin
    (...)
```

2.13.3 Job variables

Variables can be added to the job in three different places:

- job.vars: these are global variables available to all node in the nodeset
- job.host-vars.[HOST]: these are variables available only to the specified HOST
- job.group-vars.[GROUP]: these are variables available only to the specified GROUP

Zuul merges dict variables through job inheritance. Host and group variables override variables with the same name defined as global variables.

In the example below, for the sundaes job, hosts that are not part of the subnode group will run vanilla and chocolate. Hosts in the subnode group will run stracciatella and strawberry.

```
- job:
    name: ice-creams
    vars:
      devstack_service:
        vanilla: true
        chocolate: false
    group-vars:
      subnode:
        devstack_service:
          pistacchio: true
          stracciatella: true

- job:
    name: sundaes
    parent: ice-creams
    vars:
      devstack_service:
        chocolate: true
    group-vars:
      subnode:
        devstack_service:
          strawberry: true
          pistacchio: false
```

2.13.4 DevStack Gate Flags

The old CI system worked using a combination of DevStack, Tempest and devstack-gate to setup a test environment and run tests against it. With Zuul V3, the logic that used to live in devstack-gate is moved into different repos, including DevStack, Tempest and grenade.

DevStack-gate exposes an interface for job definition based on a number of DEVSTACK_GATE_* environment variables, or flags. This guide shows how to map DEVSTACK_GATE flags into the new

system.

The repo column indicates in which repository is hosted the code that replaces the devstack-gate flag. The new implementation column explains how to reproduce the same or a similar behaviour in Zuul v3 jobs. For localrc settings, devstack-gate defined a default value. In ansible jobs the default is either the value defined in the parent job, or the default from DevStack, if any.

Table 2: **DevStack Gate Flags**

DevStack gate flag	Repo	New implementation
OVERRIDE_ZUUL_BRAN	zuul	override-checkout: [branch] in the job definition.
DEVSTACK_GATE_NET_	zuul-jobs	A bridge called br-infra is set up for all jobs that inherit from multinode with a dedicated bridge role.
DEVSTACK_CINDER_VO	devstack	*CINDER_VOLUME_CLEAR: true/false* in devstack_localrc in the job vars.
DEVSTACK_GATE_NEUT	devstack	True by default. To disable, disable all neutron services in devstack_services in the job definition.
DEVSTACK_GATE_CONF	devstack	*FORCE_CONFIG_DRIVE: true/false* in devstack_localrc in the job vars.
DEVSTACK_GATE_INSTA	devstack	*INSTALL_TESTONLY_PACKAGES: true/false* in devstack_localrc in the job vars.
DEVSTACK_GATE_VIRT_	devstack	*VIRT_DRIVER: [virt driver]* in devstack_localrc in the job vars.
DEVSTACK_GATE_LIBV	devstack	*LIBVIRT_TYPE: [libvirt type]* in devstack_localrc in the job vars.
DEVSTACK_GATE_TEMP	devstack and tempest	Defined by the job that is used. The `devstack` job only runs devstack. The `devstack-tempest` one triggers a Tempest run as well.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: full* in the job vars.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: all* in the job vars.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: all-plugin* in the job vars.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: scenario* in the job vars.
TEMPEST_CONCURREN	tempest	*tempest_concurrency: [value]* in the job vars. This is available only on jobs that inherit from `devstack-tempest` down.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: venv-tempest* in the job vars. This will create Tempest virtual environment but run no tests.
DEVSTACK_GATE_SMOK	tempest	*tox_envlist: smoke-serial* in the job vars.
DEVSTACK_GATE_TEMP	tempest	*tox_envlist: full-serial* in the job vars. *TEMPEST_ALLOW_TENANT_ISOLATION: false* in devstack_localrc in the job vars.

The following flags have not been migrated yet or are legacy and wont be migrated at all.

Table 3: **Not Migrated DevStack Gate Flags**

DevStack gate flag	Status	Details
DEVS-TACK_GATE_TOPC	WIP	The topology depends on the base job that is used and more specifically on the nodeset attached to it. The new job format allows project to define the variables to be passed to every node/nodegroup that exists in the topology. Named topologies that include the nodeset and the matching variables can be defined in the form of base jobs.
DEVS-TACK_GATE_GREN	TBD	Grenade Zuul V3 jobs will be hosted in the grenade repo.
GRENADE_BASE_I	TBD	Grenade Zuul V3 jobs will be hosted in the grenade repo.
DEVS-TACK_GATE_NEUT	TBD	Depends on multinode support.
DEVS-TACK_GATE_EXER	TBD	Can be done on request.
DEVS-TACK_GATE_IRON	TBD	This will probably be implemented on ironic side.
DEVS-TACK_GATE_IRON	TBD	This will probably be implemented on ironic side.
DEVS-TACK_GATE_IRON	TBD	This will probably be implemented on ironic side.
DEVS-TACK_GATE_POST	Legacy	This flag exists in d-g but the only thing that it does is capture postgres logs. This is already supported by the roles in post, so the flag is useless in the new jobs. postgres itself can be enabled via the devstack_service job variable.
DEVS-TACK_GATE_ZERO	Legacy	This has no effect in d-g.
DEVS-TACK_GATE_MQ_I	Legacy	This has no effect in d-g.
DEVS-TACK_GATE_TEMI	Legacy	Stress is not in Tempest anymore.
DEVS-TACK_GATE_TEMI	Legacy	This is not used anywhere.
DEVS-TACK_GATE_CELL	Legacy	This has no effect in d-g.
DEVS-TACK_GATE_NOVA	Legacy	This has no effect in d-g.

2.14 Zuul CI Jobs

devstack-unit-tests

Runs unit tests on devstack project.

It runs `run_tests.sh`.

devstack-tox-functional-consumer

Base job for devstack-based functional tests for projects that consume the devstack cloud.

This base job should only be used by projects that are not involved in the devstack deployment step,

but are instead projects that are using devstack to get a cloud against which they can test things.

Runs devstack in pre-run, then runs the tox `functional` environment, then collects tox/testr build output like normal tox jobs.

Turns off tox sibling installation. Projects may be involved in the devstack deployment and so may be in the required-projects list, but may not want to test against master of the other projects in their tox env. Child jobs can set tox_install_siblings to True to re-enable sibling processing.

devstack-tox-functional

Base job for devstack-based functional tests that use tox.

Runs devstack, then runs the tox `functional` environment, then collects tox/testr build output like normal tox jobs.

Turns off tox sibling installation. Projects may be involved in the devstack deployment and so may be in the required-projects list, but may not want to test against master of the other projects in their tox env. Child jobs can set tox_install_siblings to True to re-enable sibling processing.

devstack-tox-base

Base job for devstack-based functional tests that use tox.

This job is not intended to be run directly. Its just here for organizational purposes for devstack-tox-functional and devstack-tox-functional-consumer.

devstack-no-tls-proxy

Tempest job with tls-proxy off.

Some gates run devstack like this and it follows different code paths.

devstack-platform-ubuntu-noble-ovs

Ubuntu 24.04 LTS (noble) platform test (OVS)

devstack-platform-ubuntu-noble-ovn-source

Ubuntu 24.04 LTS (noble) platform test (OVN from source)

devstack-platform-ubuntu-jammy

Ubuntu 22.04 LTS (Jammy) platform test

devstack-platform-rocky-blue-onyx

Rocky Linux 9 Blue Onyx platform test

devstack-platform-debian-bookworm

Debian Bookworm platform test

devstack-platform-centos-9-stream

CentOS 9 Stream platform test

devstack-multinode

Simple multinode test to verify multinode functionality on devstack side. This is not meant to be used as a parent job.

devstack-enforce-scope

This job runs the devstack with scope checks enabled.

devstack-ipv6

Devstack single node job for integration gate with IPv6, all services and tunnels using IPv6 addresses.

devstack

Base devstack job for integration gate.

This base job can be used for single node and multinode devstack jobs.

With a single node nodeset, this job sets up an all-in-one (aio) devstack with the seven OpenStack services included in the devstack tree: keystone, glance, cinder, neutron, nova, placement, and swift.

With a two node nodeset, this job sets up an aio + compute node. The controller can be customised using host-vars.controller, the sub-nodes can be customised using group-vars.subnode.

Descendent jobs can enable / disable services, add devstack configuration options, enable devstack plugins, configure log files or directories to be transferred to the log server.

The job assumes that there is only one controller node. The number of subnodes can be scaled up seamlessly by setting a custom nodeset in job.nodeset.

The run playbook consists of a single role, so it can be easily rewritten and extended.

devstack-minimal

Minimal devstack base job, intended for use by jobs that need less than the normal minimum set of required-projects.

devstack-base

Base abstract Devstack job.

Defines plays and base variables, but it does not include any project and it does not run any service by default. This is a common base for all single Devstack jobs, single or multinode. Variables are defined in job.vars, which is what is then used by single node jobs and by multi node jobs for the controller, as well as in job.group-vars.peers, which is what is used by multi node jobs for subnode nodes (everything but the controller).

2.15 Zuul CI Roles

apache-logs-conf

Prepare apache configs and logs for staging

Make sure apache config files and log files are available in a linux flavor independent location. Note that this relies on hard links, to the staging directory must be in the same partition where the logs and configs are.

Role Variables

stage_dir
Default: `{{ ansible_user_dir }}`

The base stage directory.

capture-performance-data

Generate performance logs for staging

Captures usage information from mysql, systemd, apache logs, and other parts of the system and generates a performance.json file in the staging directory.

Role Variables

stage_dir

Default: `{{ ansible_user_dir }}`

The base stage directory

devstack_conf_dir
Default: `/opt/stack`

The base devstack destination directory

debian_suse_apache_deref_logs

The apache logs found in the debian/suse locations

redhat_apache_deref_logs

The apache logs found in the redhat locations

capture-system-logs

Stage a number of system type logs

Stage a number of different logs / reports: - snapshot of iptables - disk space available - pip[2|3] freeze - installed packages (dpkg/rpm) - ceph, openswitch, gluster - coredumps - dns resolver - listen53 - services - unbound.log - deprecation messages

Role Variables

stage_dir
Default: `{{ ansible_user_dir }}`

The base stage directory.

devstack-ipv6-only-deployments-verification

Verify all addresses in IPv6-only deployments

This role needs to be invoked from a playbook that runs tests. This role verifies the IPv6 settings on the devstack side and that devstack deploys with all addresses being IPv6. This role is invoked before tests are run so that if there is any missing IPv6 setting, deployments can fail the job early.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack-project-conf

Prepare OpenStack project configurations for staging

Prepare all relevant config files for staging. This is helpful to avoid staging the entire /etc.

Role Variables

stage_dir
Default: `{{ ansible_user_dir }}`

The base stage directory.

export-devstack-journal

Export journal files from devstack services

This performs a number of logging collection services

- Export the systemd journal in native format

- For every devstack service, export logs to text in a file named `screen-*` to maintain legacy compatability when devstack services used to run in a screen session and were logged separately.
- Export a syslog-style file with kernel and sudo messages for legacy compatability.

Writes the output to the `logs/` subdirectory of `stage_dir`.

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory. This is used to obtain the `log-start-timestamp.txt`, used to filter the systemd journal.

`stage_dir`
Default: `{{ ansible_user_dir }}`

The base stage directory.

`fetch-devstack-log-dir`

Fetch content from the devstack log directory

Copy logs from every host back to the zuul executor.

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory.

`get-devstack-os-environment`

Reads the OS_* variables set by devstack through openrc for the specified user and project and exports them as the os_env_vars fact.

WARNING: this role is meant to be used as porting aid for the non-unified python-<service>client jobs which are already around, as those clients do not use clouds.yaml as openstackclient does. When those clients and their jobs are deprecated and removed, or anyway when the new code is able to read from clouds.yaml directly, this role should be removed as well.

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory.

`openrc_file`
Default: `{{ devstack_base_dir }}/devstack/openrc`

The location of the generated openrc file.

`openrc_user`
Default: `admin`

The user whose credentials should be retrieved.

`openrc_project`
Default: `admin`

The project (which openrc_user is part of) whose access data should be retrieved.

`openrc_enable_export`

Default: `false`

Set it to true to export os_env_vars.

`orchestrate-devstack`

Orchestrate a devstack

Runs devstack in a multinode scenario, with one controller node and a group of subnodes.

The reason for this role is so that jobs in other repository may run devstack in their plays with no need for re-implementing the orchestration logic.

The run-devstack role is available to run devstack with no orchestration.

This role sets up the controller and CA first, it then pushes CA data to sub-nodes and run devstack there. The only requirement for this role is for the controller inventory_hostname to be controller and for all sub-nodes to be defined in a group called subnode.

This role needs to be invoked from a playbook that uses a linear strategy.

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory.

`process-stackviz`

Generate stackviz report.

Generate stackviz report using subunit and dstat data, using the stackviz archive embedded in test images.

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory.

`stage_dir`
Default: `"{{ ansible_user_dir }}"`

The stage directory where the input data can be found and the output will be produced.

`zuul_work_dir`
Default: `{{ devstack_base_dir }}/tempest`

Directory to work in. It has to be a fully qualified path.

`run-devstack`

Run devstack

Role Variables

`devstack_base_dir`
Default: `/opt/stack`

The devstack base directory.

`setup-devstack-cache`

Set up the devstack cache directory

If the node has a cache of devstack image files, copy it into place.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack_cache_dir
Default: `/opt/cache`

The directory with the cached files.

setup-devstack-log-dir

Set up the devstack log directory

Create a log directory on the ephemeral disk partition to save space on the root device.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

setup-devstack-source-dirs

Set up the devstack source directories

Ensure that the base directory exists, and then move the source repos into it.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack_sources_branch
Default: `None`

The target branch to be setup (where available).

setup-stack-user

Set up the *stack* user

Create the stack user, set up its home directory, and allow it to sudo.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack_stack_home_dir
Default: `{{ devstack_base_dir }}`

The home directory for the stack user.

setup-tempest-user

Set up the *tempest* user

Create the tempest user and allow it to sudo.

Role Variables

devstack_base_dir

Default: `/opt/stack`

The devstack base directory.

start-fresh-logging

Restart logging on all hosts

Restart syslog so that the system logs only include output from the job.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

sync-controller-ceph-conf-and-keys

Sync ceph config and keys between controller and subnodes

Simply copy the contents of /etc/ceph on the controller to subnodes.

sync-devstack-data

Sync devstack data for multinode configurations

Sync any data files which include certificates to be used if TLS is enabled. This role must be executed on the controller and it pushes data to all subnodes.

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack_data_base_dir
Default: `{{ devstack_base_dir }}`

The devstack base directory for data/. Useful for example when multiple executions of devstack (i.e. grenade) share the same data directory.

write-devstack-local-conf

Write the local.conf file for use by devstack

Role Variables

devstack_base_dir
Default: `/opt/stack`

The devstack base directory.

devstack_local_conf_path
Default: `{{ devstack_base_dir }}/devstack/local.conf`

The path of the local.conf file.

devstack_localrc
Type: *dict*

A dictionary of variables that should be written to the localrc section of local.conf. The values (which are strings) may contain bash shell variables, and will be ordered so that variables used by later entries appear first.

As a special case, the variable `LIBS_FROM_GIT` will be constructed automatically from the projects which appear in the `required-projects` list defined by the job plus the project of

the change under test. To instruct devstack to install a library from source rather than pypi, simply add that library to the jobs `required-projects` list. To override the automatically-generated value, set `LIBS_FROM_GIT` in `devstack_localrc` to the desired value.

devstack_local_conf
Type: *dict*

A complex argument consisting of nested dictionaries which combine to form the meta-sections of the local_conf file. The top level is a dictionary of phases, followed by dictionaries of filenames, then sections, which finally contain key-value pairs for the INI file entries in those sections.

The keys in this dictionary are the devstack phases.

devstack_local_conf{}.**[phase]**
Type: *dict*

The keys in this dictionary are the filenames for this phase.

devstack_local_conf{}.[phase]{}.**[filename]**
Type: *dict*

The keys in this dictionary are the INI sections in this file.

devstack_local_conf{}.[phase]{}.[filename]{}.**[section]**
Type: *dict*

This is a dictionary of key-value pairs which comprise this section of the INI file.

devstack_base_services
Default: `{{ base_services | default(omit) }}`
Type: *list*

A list of base services which are enabled. Services can be added or removed from this list via the `devstack_services` variable. This is ignored if `base` is set to `False` in `devstack_services`.

devstack_services
Type: *dict*

A dictionary mapping service names to boolean values. If the boolean value is `false`, a `disable_service` line will be emitted for the service name. If it is `true`, then `enable_service` will be emitted. All other values are ignored.

The special key `base` can be used to enable or disable the base set of services enabled by default. If `base` is found, it will processed before all other keys. If its value is `False` a `disable_all_services` will be emitted; if its value is `True` services from `devstack_base_services` will be emitted via `ENABLED_SERVICES`.

devstack_plugins
Type: *dict*

A dictionary mapping a plugin name to a git repo location. If the location is a non-empty string, then an `enable_plugin` line will be emmitted for the plugin name.

If a plugin declares a dependency on another plugin (via `plugin_requires` in the plugins settings file), this role will automatically emit `enable_plugin` lines in the correct order.

tempest_plugins
Type: *list*

A list of tempest plugins which are installed alongside tempest.

The list of values will be combined with the base devstack directory and used to populate the `TEMPEST_PLUGINS` variable. If the variable already exists, its value is *not* changed.

INDEX