Bifrost Documentation

Release 19.1.3.dev3

OpenStack Foundation

Feb 09, 2026

CONTENTS

CHAPTER

ONE

BIFROST

Bifrost (pronounced bye-frost) is a set of Ansible playbooks that automates the task of deploying a base image onto a set of known hardware using ironic. It provides modular utility for one-off operating system deployment with as few operational requirements as reasonably possible.

The mission of bifrost is to provide an easy path to deploy ironic in a stand-alone fashion, in order to help facilitate the deployment of infrastructure, while also being a configurable project that can consume other OpenStack components to allow users to easily customize the environment to fit their needs, and drive forward the stand-alone perspective.

Use cases include:

- Installation of ironic in standalone/noauth mode without other OpenStack components.
- Deployment of an operating system to a known pool of hardware as a batch operation.
- Testing and development of ironic in the standalone mode.

1.1 Useful Links

Bifrosts documentation can be found at:
https://docs.openstack.org/bifrost/latest

Release notes are at:
https://docs.openstack.org/releasenotes/bifrost/

The project source code repository is located at:
https://opendev.org/openstack/bifrost/

Bugs can be filed in launchpad:
https://launchpad.net/bifrost

CHAPTER

TWO

CONTENTS

2.1 Bifrost Installation

2.1.1 Introduction

This document will guide you through installing the Bare Metal Service (ironic) using Bifrost.

Supported operating systems

Full support (fully tested in the CI, no known or potential issues):

- CentOS Stream 9

> **Note**
>
> RHEL 9 and derivatives are assumed to work but not tested explicitly.

- Ubuntu 22.04 Jammy
- Debian 11 Bullseye and 12 Bookworm

> **Note**
>
> Operating systems evolve and so does the support for them, even on stable branches.

Bifrost structure

Installation and use of Bifrost is split into roughly three steps:

- **install**: prepare the local environment by downloading and/or building machine images, and installing and configuring the necessary services.
- **enroll-dynamic**: take as input a customizable hardware inventory file and enroll the listed hardware with ironic, configuring each appropriately for deployment with the previously-downloaded images.
- **deploy-dynamic**: instruct ironic to deploy the operating system onto each machine.

Installation of Bifrost can be done in three ways:

- Via the `bifrost-cli` command line tool.

  This is the path recommended for those who want something that just works. It provides minimum configuration and uses the recommended defaults.

- By directly invoking `ansible-playbook` on one of provided playbooks.
- By writing your own playbooks using Ansible roles provided with Bifrost.

If you want to understand what and how is installed by Bifrost, please see *Bifrost Architecture*.

2.1.2 Pre-install steps

Know your environment

Before you start, you need to gather certain facts about your bare metal environment (this step can be skipped if youre testing Bifrost on virtual machines).

For the machine that hosts Bifrost youll need to figure out:

- The network interface youre going to use for communication between the bare metal machines and the Bifrost services.

  On systems using firewalld (Fedora, CentOS and RHEL currently), a new zone `bifrost` will be created, and the network interface will be moved to it. DHCP, PXE and API services will only be added to this zone. If you need any of them available in other zones, you need to configure firewall yourself.

  Warning

  If you use the same NIC for bare metal nodes and external access, installing bifrost may lock you out of SSH to the node. You have two options:

  1. Pre-create the `bifrost` firewalld zone before installation and add the SSH service to it.
  2. Use the `public` zone by providing `firewalld_internal_zone=public` when installing.

- Whether to use the integrated DHCP server or an external DHCP service.
- Pool of IP addresses for DHCP (must be within the network configured on the chosen network interface).
- Whether you want the services to use authentication via Keystone.

For each machine that is going to be enrolled in the Bare Metal service youll need:

- The management technology you are going to use to control the machine (IPMI, Redfish, etc). See bare metal drivers for guidance.
- An IP address or a host name of its management controller (BMC).
- Credentials for the management controller.
- MAC address of the NIC the machine uses for PXE booting (optional for IPMI).
- Whether it boots in the UEFI or legacy (BIOS) mode.

> **Note**
>
> Some hardware types (like `redfish`) can enforce the desired boot mode, while the other (like `ipmi`) require the same boot mode to be set in ironic and on the machine.

Required packages

To start with Bifrost you will need Python 3.6 or newer and the `git` source code management tool.

On CentOS/RHEL/Fedora:

```
sudo dnf install -y git python3
```

On Ubuntu/Debian:

```
sudo apt-get update
sudo apt-get install -y python3 git
```

On openSUSE:

```
sudo zipper install -y python3 git
```

Enable additional repositories (RHEL only)

The `extras` and `optional` dnf repositories must be enabled to satisfy bifrosts dependencies. To check:

```
sudo dnf repolist | grep 'optional\|extras'
```

To view the status of repositories:

```
sudo dnf repolist all | grep 'optional\|extras'
```

The output will look like this:

```
!rhui-REGION-rhel-server-debug-extras/8Server/x86_64         Red H disabled
rhui-REGION-rhel-server-debug-optional/8Server/x86_64        Red H disabled
rhui-REGION-rhel-server-extras/8Server/x86_64                Red H disabled
rhui-REGION-rhel-server-optional/8Server/x86_64              Red H disabled
rhui-REGION-rhel-server-source-extras/8Server/x86_64         Red H disabled
rhui-REGION-rhel-server-source-optional/8Server/x86_64       Red H disabled
```

Use the names of the repositories (minus the version and architecture) to enable them:

```
sudo dnf config-manager --enable rhui-REGION-rhel-server-optional
sudo dnf config-manager --enable rhui-REGION-rhel-server-extras
```

Enable the EPEL repository (RHEL and CentOS)

Building Debian or Ubuntu based images on RHEL or CentOS requires a few extra pre-install steps, in order to have access to the additional packages contained in the EPEL repository.

Please refer to the official wiki page to install and configure them.

Note

Use of EPEL repositories may result in incompatible packages being installed by the package manager. Care should be taken when using a system with EPEL enabled.

Clone Bifrost

Bifrost is typically installed from git:

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

To install Bare Metal services from a specific release series (rather than the latest versions), check out the corresponding stable branch. For example, for Ussuri:

```
git checkout stable/ussuri
```

Testing on virtual machines

If you want to try Bifrost on virtual machines instead of real hardware, you need to prepare a testing environment. The easiest way is via `bifrost-cli`, available since the Victoria release series:

```
./bifrost-cli testenv
```

Then do not forget to pass `--testenv` flag to `bifrost-cli install`.

See *Testing Environment* for more details and for advanced ways of creating a virtual environment (also supported on Ussuri and older).

Warning

Testenv for bifrost will default to using the default libvirt network. If you are installing the testenv inside a VM using the default libvirt network, you will encounter errors.

2.1.3 Quick start with bifrost-cli

The `bifrost-cli` script, available since the Victoria release series, installs the Bare Metal service with the recommended defaults. Follow *Installation via playbooks* if using Ussuri or older or if you need a full control over your environment.

Using it is as simple as:

```
./bifrost-cli install \
    --network-interface <the network interface to use> \
    --dhcp-pool <DHCP start IP>-<DHCP end IP>
```

For example:

```
./bifrost-cli install --network-interface eno1 \
    --dhcp-pool 10.0.0.20-10.0.0.100
```

Note

See *Know your environment* for the guidance on the two required parameters.

If installing on a virtual environment, skip these two parameters:

```
./bifrost-cli install --testenv
```

Additionally, the following parameters can be useful:

`--hardware-types`
A comma separated list of hardware types to enable.

`--enable-keystone`
Whether to enable authentication with Keystone.

`--enable-tls`
Enable self-signed TLS on API endpoints.

Warning

If using Keystone, see *TLS notes* for important notes.

`--release`
If using a stable version of Bifrost, the corresponding version of Ironic is usually detected from the git checkout. If it is not possible (e.g. youre using Bifrost from a tarball), use this argument to provide the matching version.

Note

Using Bifrost to install older versions of Ironic may work, but is not guaranteed.

`--enable-prometheus-exporter`
Enable the Ironic Prometheus Exporter service.

`--uefi` / `--legacy-boot`
: Boot machines in the UEFI or BIOS mode by default (defaults to UEFI).

`--disable-dhcp`
: Disable the configuration of the integrated DHCP server, allowing to use an external DHCP service.

`--develop`
: Install services in develop mode, so that the changes to the repositories in `/opt` get immediately reflected in the environment.

See the built-in documentation for more details:

```
./bifrost-cli install --help
```

The Ansible variables generated for installation are stored in a JSON file (`baremetal-install-env.json` by default) that should be passed via the `-e` flag to subsequent playbook or command invocations.

Build Custom Ironic Python Agent (IPA) images

Bifrost supports the ability for a user to build a custom IPA ramdisk utilizing diskimage-builder and ironic-python-agent-builder. In order to utilize this feature, the `download_ipa` setting must be set to `false` and the create_ipa_image must be set to true. By default, the install playbook will build a Debian Bookworm based IPA image, if a pre-existing IPA image is not present on disk. If you wish to explicitly set a specific release to be passed to diskimage-create, then the setting `dib_os_release` can be set in addition to `dib_os_element`.

If you wish to include an extra element into the IPA disk image, such as a custom hardware manager, you can pass the variable `ipa_extra_dib_elements` as a space-separated list of elements. This defaults to an empty string.

Using Bifrost

After installation is done, export the following environment variable to configure the bare metal client to use the `bifrost` cloud configuration from the generated `clouds.yaml` (see *Use the baremetal CLI* for details):

```
export OS_CLOUD=bifrost
```

Now you can use Ironic directly, see the standalone guide for more details. Alternatively, you can use the provided playbooks to automate certain common operations - see *How-To*.

2.1.4 Advanced Topics

Installation via playbooks

Countrary to *Quick start with bifrost-cli*, this method of installation allows full control over all parameters, as well as injecting your own ansible playbooks.

Installation is split into four parts:

- Installation of Ansible
- Configuring settings for the installation

- Execution of the installation playbook

Installation of Ansible

Installation of Ansible can take place using the provided environment setup script located at `scripts/env-setup.sh` which is present in the bifrost repository. This may also be used if you already have ansible, as it will install ansible and various dependencies to a virtual environment in order to avoid overwriting or conflicting with a system-wide Ansible installation.

Alternatively, if you have a working Ansible installation, under normal circumstances the installation playbook can be executed, but you will need to configure the *Virtual environment*.

Note

All testing takes place utilizing the `scripts/env-setup.sh` script. Please feel free to submit bug reports or patches to OpenStack Gerrit for any issues encountered if you choose to directly invoke the playbooks without using `env-setup.sh`.

Virtual environment

To avoid conflicts between Python packages installed from source and system packages, Bifrost defaults to installing everything to a virtual environment. `scripts/env-setup.sh` will automatically create a virtual environment in `/opt/stack/bifrost` if it does not exist.

If you want to relocate the virtual environment, export the `VENV` variable before calling `env-setup.sh`:

```
export VENV=/path/to/my/venv
```

If youre using the ansible playbooks directly (without the helper scripts), set the `bifrost_venv_dir` variables accordingly.

Note

Because of Ansible dependencies Bifrost only supports virtual environments created with `--system-site-packages`.

Pre-installation settings

Before performing the installation, it is highly recommended that you edit `./playbooks/inventory/group_vars/*` to match your environment. Several files are located in this folder, and you may wish to review and edit the settings across multiple files:

- The `target` file is used by roles that execute against the target node upon which you are installing ironic and all required services.
- The `baremetal` file is geared for roles executed against baremetal nodes. This may be useful if you are automating multiple steps involving deployment and configuration of nodes beyond deployment via the same roles.

- The `localhost` file is similar to the `target` file, and likely contains identical settings. This file is referenced if no explicit target is defined, as it defaults to the localhost.

Duplication between variable names does occur within these files, as variables are unique to the group that the role is being executed upon.

- If MySQL is already installed, update `mysql_password` to match your local installation.
- Change `network_interface` to match the interface that will need to service DHCP requests.
- Set `service_password` which is used for communication between services. If unset, a random password is generated during the initial installation and stored on the controller in `~/.config/bifrost/service_password`.

The install process, when executed will either download, or build disk images for the deployment of nodes, and be deployed to the nodes.

If you wish to build an image, based upon the settings, you will need to set `create_image_via_dib` to `true`.

If you are running the installation behind a proxy, export the environment variables `http_proxy`, `https_proxy` and `no_proxy` so that ansible will use these proxy settings.

TLS support

Bifrost supports TLS for API services with two options:

- A self-signed certificate can be generated automatically. Set `enable_tls=true` and `generate_tls=true`.

  Note

  This is equivalent to the `--enable-tls` flag of `bifrost-cli`.

- Certificate paths can be provided via:

  `tls_certificate_path`
  : Path to the TLS certificate (must be world-readable).

  `tls_private_key_path`
  : Path to the private key (must not be password protected).

  `tls_csr_path`
  : Path to the certificate signing request file.

  Set `enable_tls=true` and do not set `generate_tls` to use this option.

Warning

If using Keystone, see *TLS notes* for important notes.

Dependencies

In order to really get started, you must install dependencies.

With the addition of ansible collections, the `env-setup.sh` will install the collections in the default ansible `collections_paths` (according to your ansible.cfg) or you can specify the location setting `ANSIBLE_COLLECTIONS_PATH`:

```
$ export ANSIBLE_COLLECTIONS_PATH=/mydir/collections
```

Note

If you are using a virtual environment ANSIBLE_COLLECTIONS_PATH is automatically set. After Ansible Collections are installed, a symbolic link to to the installation is created in the bifrost playbook directory.

The `env-setup.sh` script automatically invokes `install-deps.sh` and creates a virtual environment for you:

```
$ bash ./scripts/env-setup.sh
$ source /opt/stack/bifrost/bin/activate
$ cd playbooks
```

Once the dependencies are in-place, you can execute the ansible playbook to perform the actual installation. The playbook will install and configure ironic in a stand-alone fashion.

A few important notes:

- The OpenStack Identity service (keystone) is NOT installed by default, and ironics API is accessible without authentication. It is possible to put basic password authentication on ironics API by changing the nginx configuration accordingly.

Note

Bifrost playbooks can leverage and optionally install keystone. See *Keystone install details*.

- The OpenStack Networking service (neutron) is NOT installed. Ironic performs static IP injection via config-drive or DHCP reservation.
- Deployments are performed by the ironic python agent (IPA).
- dnsmasq is configured statically and responds to all PXE boot requests by chain-loading to iPXE, which then fetches the Ironic Python Agent ramdisk from nginx.
- By default, installation will build an Ubuntu-based image for deployment to nodes. This image can be easily customized if so desired.

The re-execution of the playbook will cause states to be re-asserted. If not already present, a number of software packages including MySQL will be installed on the host. Python code will be reinstalled regardless if it has changed.

Playbook Execution

Playbook based install provides a greater degree of visibility and control over the process and is suitable for advanced installation scenarios.

Examples:

First, make sure that the virtual environment is active (the example below assumes that bifrost venv is installed into the default path /opt/stack/bifrost).

$. /opt/stack/bifrost/bin/activate (bifrost) $

Verify if the ansible-playbook executable points to the one installed in the virtual environment:

(bifrost) $ which ansible-playbook /opt/stack/bifrost/bin/ansible-playbook (bifrost) $

change to the `playbooks` subdirectory of the cloned bifrost repository:

$ cd playbooks

If you have passwordless sudo enabled, run:

```
$ ansible-playbook -vvvv -i inventory/target install.yaml
```

Otherwise, add the `-K` to the ansible command line, to trigger ansible to prompt for the sudo password:

```
$ ansible-playbook -K -vvvv -i inventory/target install.yaml
```

With regard to testing, ironics node cleaning capability is enabled by default, but only metadata cleaning is turned on, as it can be an unexpected surprise for a new user that their test node is unusable for however long it takes for the disks to be wiped.

If you wish to enable full cleaning, you can achieve this by passing the option `-e cleaning_disk_erase=true` to the command line or executing the command below:

```
$ ansible-playbook -K -vvvv -i inventory/target install.yaml -e cleaning_disk_
↪erase=true
```

If installing a stable release, you need to set two more parameters, e.g.:

```
-e git_branch=stable/train -e ipa_upstream_release=stable-train
```

Note

Note the difference in format: git branch uses slashes, IPA release uses dashes.

After you have performed an installation, you can edit `/etc/ironic/ironic.conf` to enable or disable cleaning as desired. It is highly encouraged to utilize cleaning in any production environment.

Additional ironic drivers

An additional collection of drivers are maintained outside of the ironic source code repository, as they do not have Continuous Integration (CI) testing.

These drivers and information about them can be found in ironic-staging-drivers docs. If you would like to install the ironic staging drivers, simply pass `-e staging_drivers_include=true` when executing the install playbook:

```
$ ansible-playbook -K -vvvv -i inventory/target install.yaml -e staging_
↪drivers_include=true
```

Installation with Keystone

Bifrost can now install and make use of keystone. In order to enable this as part of the installation, the `enable_keystone` variable must be set to `true`, either in `playbooks/inventory/group_vars/target` or on the command line during installation. Note that enable_keystone and noauth_mode are mutually exclusive so they should have an opposite value of oneanother. Example:

```
ansible-playbook -vvvv -i inventory/target install.yaml -e enable_
↪keystone=true -e noauth_mode=false
```

However, prior to installation, overriding credentials should be set in order to customize the deployment to meet your needs. At the very least, the following parameters should be changed for a production environment:

admin_password
: Password for the bootstrap user (called `admin` by default).

default_password
: Password for the regular user (called `bifrost_user` by default).

service_password
: Password for communication between services (never exposed to end users).

If any of these values is not set, a random password is generated during the initial installation and stored on the controller in an accordingly named file in the `~/.config/bifrost` directory (override using `password_dir`).

See the following files for more settings that can be overridden:

- `playbooks/roles/bifrost-ironic-install/defaults/main.yml`
- `playbooks/roles/bifrost-keystone-install/defaults/main.yml`

TLS notes

There are two important limitations to keep in mind when using Keystone with TLS:

- Its not possible to enable TLS on upgrade from Bifrost < 9.0 (Ussuri and early Victoria). First do an upgrade to Bifrost >= 9.0, then enable TLS in a separate step.
- Automatically updating from a TLS environment to a non-TLS one may not be possible if using custom TLS certificates in a non-standard location (`/etc/bifrost/bifrost.crt`). You need to manually change identity endpoints in the catalog from `https` to `http` directly before an update. The `public` endpoint **must** be updated **last** or you may lock yourself out of keystone.

Using an existing Keystone

If you choose to install bifrost using an existing keystone, this should be possible, however it has not been tested. In this case you will need to set the appropriate defaults, via `playbooks/roles/bifrost-ironic-install/defaults/main.yml` which would be a good source for the role level defaults. Ideally, when setting new defaults, they should be set in the `playbooks/inventory/group_vars/target` file.

Creation of clouds.yaml

By default, during bifrost installation, a file will be written to the users home directory that is executing the installation. That file can be located at `~/.config/openstack/clouds.yaml`. The clouds that are written to that file are named `bifrost` (for regular users) and `bifrost-admin` (for administrators).

Creation of openrc

Also by default, after bifrost installation and again, when keystone is enabled, a file will be written to the users home directory that you can use to set the appropriate environment variables in your current shell to be able to use OpenStack utilities:

```
. ~/openrc bifrost
openstack baremetal driver list
```

Offline Installation

The ansible scripts that compose Bifrost download and install software via a number of means, which generally assumes connectivity to the internet. However, it is possible to use Bifrost without external connectivity.

If you want or need to install Bifrost without having a dependency on a connection to the internet, there are a number of steps that you will need to follow (many of which may have already been done in your environment anyway).

Those steps can be broken down into two general categories; the first being steps that need to be done in your inventory file, and the second being steps that need to be done on your target host outside of Ansible.

Ansible Specific Steps

The script `scripts/env-setup.sh` will do a `git clone` to create `/opt/stack/ansible`, if it doesnt already exist. You can use the environment variables `ANSIBLE_GIT_URL` and `ANSIBLE_GIT_BRANCH` to override the source URL and the branch name to pull from.

Ansible uses Git submodules, which means if you are cloning from anything other than the canonical location (GitHub), youll need to commit a patched `.gitmodules` to that repo so that submodules are also cloned from an alternate location - otherwise, the submodules will still try to clone from GitHub.

Bifrost Specific Steps

As a general rule, any URL referenced by Bifrost scripts is configured in a `playbook/roles/<role>/defaults/main.yml` file, which means that all of those can be redirected to point to a local copy by creating a file named `playbooks/host_vars/<hostname>.yml` and redirecting the appropriate variables.

As an example, the yaml files contents may look like something like this.

```
ipa_kernel_upstream_url: file:///vagrant/ipa-centos9-master.kernel
ipa_ramdisk_upstream_url: file:///vagrant/ipa-centos9-master.initramfs
custom_deploy_image_upstream_url: file:///vagrant/cirros-0.5.3-x86_64-disk.img
dib_git_url: file:///vagrant/git/diskimage-builder
ironicclient_git_url: file:///vagrant/git/python-ironicclient
ironic_git_url: file:///vagrant/git/ironic
```

If this list becomes out of date, its simple enough to find the things that need to be fixed by looking for any URLs in the `playbook/roles/<role>/defaults/main.yml` files, as noted above.

Currently you can grep the `defaults/main.yml` in Bifrost repo

For `kolla-ansible` you also need the `sha256sum` for the `ipa` images.

```
sha256sum /vagrant/ipa-centos8-master.kernel > /vagrant/ipa-centos8-master.
↪kernel.sha256
sha256sum /vagrant/ipa-centos8-master.initramfs > /vagrant/ipa-centos8-master.
↪initramfs.sha256
```

External Steps

Bifrost doesnt attempt to configure `apt`, `yum`, or `pip`, so if you are working in an offline mode, youll need to make sure those work independently.

`pip` in particular will be sensitive; Bifrost tends to use the most recent version of python modules, so youll want to make sure your cache isnt stale.

Virtualenv Installation Support

Virtual environments are now used by default, see *Bifrost Installation*.

2.2 Bifrost User Guide

As bifrost is primarilly intended to be a tool for use by administrators, this documentation serves as a blend of both Admin and User documentation.

2.2.1 Bifrost Architecture

This document describes what and how Bifrost installs in the default setup (e.g. one created by `./bifrost-cli install`).

Components

Bifrost installs the following main components:

ironic
: Ironic is the main service that provides bare metal capabilities. Its bare metal API is served on TCP port 6385.

ironic-inspector
: Inspector is an auxiliary service that provides in-band inspection. Its introspection API is served on TCP port 5050.

  Inspector is deprecated and can be enabled by setting `enable_inspector=true`. Otherwise, Ironics native in-band inspection is used.

mariadb
: MariaDB is used as a database to persistently store information.

nginx
: Nginx is used as a web server for three main purposes:

  - to serve iPXE configuration scripts when booting nodes over the network,
  - to serve virtual media ISO images to BMC when booting nodes over virtual CD,
  - to serve instance images to the ramdisk when deploying nodes.

  Uses HTTP port 8080 by default (can be changed via the `file_url_port` parameter).

  When TLS is enabled, Nginx serves as a TLS proxy for Ironic and Inspector. It listens on ports 6385 and 5050 and passes requests to the services via unix sockets.

dnsmasq
: Dnsmasq is used as a DHCP and TFTP server (but not for DNS by default) when booting nodes over the network. It can also be used to provide DHCP to deployed nodes.

  Dnsmasq can be disabled by setting `enable_dhcp=false` or passing `--disable-dhcp` to `bifrost-cli`.

The following components can be enabled if needed:

keystone
: Keystone is an OpenStack Identity service. It can be used to provide sophisticated authentication to Ironic and Inspector instead of HTTP basic authentication. Its identity API is served using uWSGI and Nginx on the port 5000, the systemd service is called `uwsgi@keystone-public`.

  See *Using Keystone* for details.

ironic-prometheus-exporter
: An exporter that exposes hardware metrics from the nodes BMC (using IPMI or Redfish) for consumption by Prometheus.

  The exporter can be enabled by setting `enable_prometheus_exporter=true` or passing `--enable-prometheus-exporter` to `bifrost-cli`.

Networking

Bifrost Installation requires to specify the network interface which will be used for communication with the nodes. On systems that use firewalld, a new zone `bifrost` is created with this interface, and firewall settings are adjusted to allow boot traffic (DHCP, TFTP, etc) through this interface.

Virtual testing environments use network interface `virbr0`, IP subnet `192.168.122.1/24` and the `libvirt` firewalld zone by default.

Parameters

network_interface
: the main network interface for provisioning nodes. It is recommended to use an isolated network for this purpose.

internal_ip
: IPv4 address of the Bifrost node on the provisioning interface. Normally derived automatically. You can use the following command to detect it:

```
$ sudo awk '/^tftp_server =/ { print $3 }' /etc/ironic/ironic.conf
192.168.122.1
```

  This IP address is used for all provisioning traffic: TFTP, iPXE, call-backs to Ironic and Inspector. It is also used for the traffic between the services.

public_ip
: IPv4 address of the Bifrost node that is accessible outside of the provisioning environment. Used e.g. to provide public URLs for the service catalog, when Keystone support is enabled.

> **Note**
>
> Bifrost does not currently support IPv6, contributions are welcome.

Locations

Note

Many locations are not writable or even readable by the regular user (even the one that started installation). You need to use `sudo` or add the user to `ironic` and `nginx` groups.

Installation locations

/opt/stack/bifrost
: is a Python virtual environment where all Python packages are installed.

/opt/stack/<reponame>
: is a source directory for project <reponame> (e.g. `ironic`) installed from source. Services are installed this way, while libraries are mostly installed as packages.

  Unlike most other locations, these paths will be owned by the user that installed Bifrost (i.e. your regular user).

Log locations

journald
: is used for logging from most services. For example, to get Inspector logs:

```
$ sudo journalctl -u ironic-inspector
```

/var/log/ironic/deploy
: contains tarballs with ramdisk logs from deployment or cleaning. The file name format is <node UUID>-<node name>-[cleaning-]<datatime>.tar.gz. You can access the main logs like this:

```
$ cd $(mktemp -d)
$ sudo tar -xzf \
    /var/log/ironic/deploy/493aacf2-90ec-5e3d-9ce5-ea496f12e2a5_testvm3_
↪2021-11-08-17-34-18.tar.gz
$ less journal  # for ramdisks that use systemd, e.g. DIB-built
$ less var/log/ironic-python-agent.log # for tinyIPA and similar
```

/var/log/ironic-inspector/ramdisk
: contains tarballs with ramdisk logs from inspection. They are very similar to ramdisk logs from deployment and cleaning.

/var/log/nginx/
: contains logs for serving files (iPXE scripts, images, virtual media ISOs).

/var/log/nginx/keystone
: contains HTTP logs for Keystone API, complementing the logs from the `uwsgi@keystone-public` systemd unit.

Runtime locations

/var/lib/ironic/httpboot
HTTP root directory. Contains iPXE scripts and images, including `deployment_image.qcow2` which is built or downloaded during Bifrost installation.

You can detect the root URL with the following command:

```
$ sudo awk '/^http_url =/ { print $3 }' /etc/ironic/ironic.conf
http://192.168.122.1:8080/
```

/var/lib/tftpboot
TFTP root directory. Normally contains only the binaries to run iPXE on systems that dont have an iPXE firmware built in. Can contain images when the `pxe` boot interface is used.

/var/lib/ironic/master_images
cache for instance images.

/var/lib/ironic/master_iso_images
cache for virtual media ISO images.

/var/lib/ironic/certificates
TLS certificates that are used to communicate to the ramdisk on the nodes when cleaning or deploying.

/run/ironic
When TLS is enabled, this directory contains unix sockets of Ironic and Inspector, which Nginx uses to pass requests.

2.2.2 How-To

Use the baremetal CLI

If you wish to utilize the baremetal CLI in no-auth mode, there are two options for configuring the authentication parameters.

clouds.yaml

During installation, Bifrost creates a `clouds.yaml` file with credentials necessary to access Ironic. A cloud called `bifrost` is always available. For example:

```
export OS_CLOUD=bifrost
baremetal node list
baremetal introspection list
```

Note

Previously, a separate cloud `bifrost-inspector` was provided for introspection commands. It is now deprecated, the main `bifrost` cloud should always be used.

Environment variables

For convenience, an environment file called `openrc` is created in the home directory of the current user that contains default values for these variables and can be sourced to allow the CLI to connect to a local Ironic installation. For example:

```
. ~/openrc bifrost
baremetal node list
```

This should display a table of nodes, or nothing if there are no nodes registered in Ironic.

Installing OpenStack CLI

Starting with the Victoria release, the `openstack` command is only installed when Keystone is enabled. Install the `python-openstackclient` Python package to get this command.

Inventory file format

In order to enroll hardware, you will naturally need an inventory of your hardware. When utilizing the dynamic inventory module and accompanying roles the inventory can be supplied in one of three ways, all of which ultimately translate to JSON data that Ansible parses.

The current method is to utilize a JSON or YAML document which the inventory parser will convert and provide to Ansible.

The JSON format resembles the data structure that ironic utilizes internally.

- The `name`, `uuid`, `driver`, and `properties` fields are directly mapped through to ironic. Only the `driver` is required.

  Note

  Most properties are automatically populated during inspection, if it is enabled. However, it is recommended to set the root device for nodes with multiple disks.

- The `driver_info` field format matches one of the OpenStack Ansible collection and for legacy reasons can use nested structures `power`, `deploy`, `management` and `console`.

  Note

  With newer versions of the collection you should just put all fields under `driver_info` directly).

- The `nics` field is a list of ports to create. The required field is `mac` - MAC address of the port.

Example:

```
{
    "testvm1": {
      "name": "testvm1",
```

(continues on next page)

(continued from previous page)

```
      "driver": "ipmi",
      "driver_info": {
        "ipmi_address": "192.168.122.1",
        "ipmi_username": "admin",
        "ipmi_password": "pa$$w0rd"
      },
      "nics": [
        {
          "mac": "52:54:00:f9:32:f6"
        }
      ],
      "properties": {
        "cpu_arch": "x86_64",
        "root_device": {"wwn": "0x4000cca77fc4dba1"}
      }
    }
}
```

Overriding instance information

The additional power of this format is easy configuration parameter injection, which could potentially allow a user to provision different operating system images onto different hardware chassis by defining the appropriate settings in an `instance_info` variable, for example:

```
{
    "testvm1": {
      "uuid": "00000000-0000-0000-0000-000000000001",
      "name": "testvm1",
      "driver": "redfish",
      "driver_info": {
        "redfish_address": "https://bmc.myhost.com",
        "redfish_system_id": "/redfish/v1/Systems/11",
        "redfish_username": "admin",
        "redfish_password": "pa$$w0rd",
      },
      "nics": [
        {
          "mac": "52:54:00:f9:32:f6"
        }
      ],
      "properties": {
        "cpu_arch": "x86_64",
        "root_device": {"wwn": "0x4000cca77fc4dba1"}
      },
      "instance_info": {
        "image_source": "http://image.server/image.qcow2",
        "image_checksum": "<md5/sha256/sha512 checksum>",
        "configdrive": {
```

(continues on next page)

(continued from previous page)

```
            "meta_data": {
              "public_keys": {"0": "ssh-rsa ..."},
              "hostname": "vm1.example.com"
            }
          }
        }
      }
}
```

The `instance_info` format is documented in the Ironic deploy guide. The ability to populate `configdrive` this way is a Bifrost-specific feature, but the `configdrive` itself follows the Ironic format.

Examples utilizing JSON and YAML formatting, along host specific variable injection can be found in the `playbooks/inventory/` folder.

Static network configuration

When building a configdrive, Bifrost can embed static networking configuration in it. This configuration will be applied by the first-boot service, such as cloud-init or glean. The following fields can be set:

ipv4_address
: The IPv4 address of the node. If missing, the configuration is not provided in the configdrive.

  When `ipv4_address` is set, its also used as the default for `ansible_ssh_host`. Because of this, you can run SSH commands against deployed hosts, as long as you use the Bifrosts inventory plugin.

  This parameter can also used for *DHCP configuration*.

ipv4_subnet_mask
: The subnet mask of the IP address. Defaults to *255.255.255.0*.

ipv4_interface_mac
: MAC address of the interface to configure. If missing, the MAC address of the first NIC defined in the inventory is used.

ipv4_gateway
: IPv4 address of the default router. A default value is only provided for testing case.

ipv4_nameserver
: The server to use for name resolution (a string or a list).

network_mtu
: MTU to use for the link.

For example:

```
{
    "testvm1": {
      "name": "testvm1",
      "driver": "redfish",
      "driver_info": {
        "redfish_address": "https://bmc.myhost.com",
        "redfish_system_id": "/redfish/v1/Systems/11",
```

(continues on next page)

(continued from previous page)

```
        "redfish_username": "admin",
        "redfish_password": "pa$$w0rd",
      },
      "ipv4_address": "192.168.122.42",
      "ipv4_subnet_mask": "255.255.255.0",
      "ipv4_gateway": "192.168.122.1",
      "ipv4_nameserver": "8.8.8.8",
      "nics": [
        {
          "mac": "52:54:00:f9:32:f6"
        }
      ],
      "properties": {
        "cpu_arch": "x86_64",
        "root_device": {"wwn": "0x4000cca77fc4dba1"}
      }
    }
}
```

Warning

Static network configuration only works this way if you let Bifrost generate the configdrive.

Enroll Hardware

Starting with the Wallaby cycle, you can use `bifrost-cli` for enrolling:

```
./bifrost-cli enroll /tmp/baremetal.json
```

Note that enrollment is a one-time operation. The Ansible module *does not* synchronize data for existing nodes. You should use the ironic CLI to do this manually at the moment.

Additionally, it is important to note that the playbooks for enrollment are split into three separate playbooks based on the `ipmi_bridging` setting.

Deploy Hardware

After the nodes are enrolled, they can be deployed upon. Bifrost is geared to utilize configuration drives to convey basic configuration information to the each host. This configuration information includes an SSH key to allow a user to login to the system.

Starting with the Yoga cycle, you can use `bifrost-cli` for deploying. If you used `bifrost-cli` for installation, you should pass its environment variables, as well as the inventory file (see *Inventory file format*):

```
./bifrost-cli deploy /tmp/baremetal.json \
    -e @baremetal-install-env.json
```

Note

By default, the playbook will return once the deploy has started. Pass the `--wait` flag to wait for completion.

The inventory file may override some deploy settings, such as images or even the complete `instance_info`, per node. If you omit it, all nodes from Ironic will be deployed using the Bifrost defaults:

```
./bifrost-cli deploy -e @baremetal-install-env.json
```

Command line parameters

By default the playbooks use the image, downloaded or built during installation. You can also use a custom image:

```
./bifrost-cli deploy -e @baremetal-install-env.json \
    --image http://example.com/images/my-image.qcow2 \
    --image-checksum 91ebfb80743bb98c59f787c9dc1f3cef \
```

Note

Please see the OpenStack Image Guide for options and locations for obtaining guest images.

You can also provide a custom configdrive URL (or its content) instead of the one Bifrost builds for you:

```
./bifrost-cli deploy -e @baremetal-install-env.json \
    --config-drive '{"meta_data": {"public_keys": {"0": "'"$(cat ~/.ssh/id_
↪rsa.pub)"'"}}}' \
```

File images do not require a checksum:

```
./bifrost-cli deploy -e @baremetal-install-env.json \
    --image file:///var/lib/ironic/custom-image.qcow2
```

Note

Files must be readable by Ironic. Your home directory is often not.

Partition images can de deployed by specifying an image type:

```
./bifrost-cli deploy -e @baremetal-install-env.json \
    --image http://example.com/images/my-image.qcow2 \
    --image-checksum 91ebfb80743bb98c59f787c9dc1f3cef \
    --partition
```

Note

The default root partition size is 10 GiB. Set the `deploy_root_gb` parameter to override or use a first-boot service such as cloud-init to grow the root partition automatically.

Redeploy Hardware

If the hosts need to be re-deployed, the dynamic redeploy playbook may be used:

```
export BIFROST_INVENTORY_SOURCE=/tmp/baremetal.json
cd playbooks
ansible-playbook -vvvv -i inventory/bifrost_inventory.py redeploy-dynamic.yaml
```

This playbook will undeploy the hosts, followed by a deployment, allowing a configurable timeout for the hosts to transition in each step.

Use playbooks instead of bifrost-cli

Using playbooks directly allows you full control over what is executed by Bifrost, with what variables and using what inventory.

Utilizing the dynamic inventory module, enrollment is as simple as setting the `BIFROST_INVENTORY_SOURCE` environment variable to your inventory data source, and then executing the enrollment playbook:

```
export BIFROST_INVENTORY_SOURCE=/tmp/baremetal.json
cd playbooks
ansible-playbook -vvvv -i inventory/bifrost_inventory.py enroll-dynamic.yaml
```

To utilize the dynamic inventory based deployment:

```
export BIFROST_INVENTORY_SOURCE=/tmp/baremetal.json
cd playbooks
ansible-playbook -vvvv -i inventory/bifrost_inventory.py deploy-dynamic.yaml
```

If you used `bifrost-cli` for installation, you should pass its environment variables:

```
export BIFROST_INVENTORY_SOURCE=/tmp/baremetal.json
cd playbooks
ansible-playbook -vvvv \
  -i inventory/bifrost_inventory.py \
  -e @../baremetal-install-env.json \
  deploy-dynamic.yaml
```

Deployment and configuration of operating systems

By default, Bifrost deploys a configuration drive which includes the user SSH public key, hostname, and the network configuration in the form of network_data.json that can be read/parsed by glean or cloud-init. This allows for the deployment of Ubuntu, CentOS, or Fedora tenants on baremetal.

By default, Bifrost utilizes a utility called *simple-init* which leverages the previously noted glean utility to apply network configuration. This means that by default, root file systems may not be automatically expanded to consume the entire disk, which may, or may not be desirable depending upon operational needs. This is dependent upon what base OS image you utilize, and if the support is included in that image or not. At present, the standard Ubuntu cloud image includes cloud-init which will grow the root partition, however the ubuntu-minimal image does not include cloud-init and thus will not automatically grow the root partition.

Due to the nature of the design, it would be relatively easy for a user to import automatic growth or reconfiguration steps either in the image to be deployed, or in post-deployment steps via custom Ansible playbooks.

To be able to access nodes via SSH, ensure that the value for *ssh_public_key_path* in `./playbooks/inventory/group_vars/baremetal` refers to a valid public key file, or set the `ssh_public_key_path` variable on the command line, e.g. `-e ssh_public_key_path=~/.ssh/id_rsa.pub`.

Advanced topics

Using a remote ironic

When ironic is installed on remote server, a regular ansible inventory with a target server should be added to ansible. This can be achieved by specifying a directory with files, each file in that directory will be part of the ansible inventory. Refer to ansible documentation http://docs.ansible.com/ansible/intro_dynamic_inventory.html#using-inventory-directories-and-multiple-inventory-sources. Example:

```
export BIFROST_INVENTORY_SOURCE=/tmp/baremetal.json
cd playbooks
rm inventory/*.example
ansible-playbook -vvvv -i inventory/ enroll-dynamic.yaml
```

Build Custom Ironic Python Agent (IPA) images

Content moved, see *Build Custom Ironic Python Agent (IPA) images*.

Configuring the integrated DHCP server

Content moved, see *Configuring the integrated DHCP server*.

Use Bifrost with Keystone

Content moved, see *Using Keystone*.

2.2.3 Troubleshooting

Firewalling

Due to the nature of firewall settings and customizations, bifrost does **not** change any local firewalling on the node. Users must ensure that their firewalling for the node running bifrost is such that the nodes that are being booted can connect to the following ports:

```
67/UDP for DHCP requests to be serviced
69/UDP for TFTP file transfers (Initial iPXE binary)
6385/TCP for the ironic API
8080/TCP for HTTP File Downloads (iPXE, Ironic-Python-Agent)
```

If you encounter any additional issues, use of `tcpdump` is highly recommended while attempting to deploy a single node in order to capture and review the traffic exchange between the two nodes.

NodeLocked Errors

This is due to node status checking thread in ironic, which is a locking action as it utilizes IPMI. The best course of action is to retry the operation. If this is occurring with a high frequency, tuning might be required.

Example error:

```
NodeLocked: Node 00000000-0000-0000-0000-046ebb96ec21 is locked by
host $HOSTNAME, please retry after the current operation is completed.
```

Building an IPA image

Troubleshooting issues involving IPA can be time consuming. The IPA developers **HIGHLY** recommend that users build their own custom IPA images in order to inject things such as SSH keys, and turn on agent debugging which must be done in a custom image as there is no mechanism to enable debugging via the kernel command line at present.

Custom IPA images can be built a number of ways, the most generally useful mechanism is with diskimage-builder as the distributions typically have better hardware support than Tiny Core Linux.

DIB images:
https://docs.openstack.org/ironic-python-agent-builder/latest/admin/dib.html

TinyIPA:
https://docs.openstack.org/ironic-python-agent-builder/latest/admin/tinyipa.html

For documentation on diskimage-builder, See::
https://docs.openstack.org/diskimage-builder/latest/.

It should be noted that the steps for diskimage-builder installation and use to create an IPA image for Bifrost are the same as for ironic. See: https://docs.openstack.org/ironic/latest/install/deploy-ramdisk.html

Once your build is completed, you will need to copy the images files into the `/var/lib/ironic/httpboot` folder.

Unexpected/Unknown failure with the IPA Agent

Many failures due to the IPA agent can be addressed by building a custom IPA Image. See *Building an IPA image* for information on building your own IPA image.

Obtaining IPA logs via the console

1) By default, bifrost sets the agent journal to be logged to the system console. Due to the variation in hardware, you may need to tune the parameters passed to the deployment ramdisk. This can be done, as shown below in ironic.conf:

```
kernel_append_params=nofb nomodeset vga=normal console=ttyS0 systemd.
↪journald.forward_to_console=yes
```

   Parameters will vary by your hardware type and configuration, however the `systemd.journald.forward_to_console=yes` setting is a default, and will only work for systemd based IPA images.

   The example above, effectively disables all attempts by the kernel to set the video mode, defines the console as ttyS0 or the first serial port, and instructs systemd to direct logs to the console.

2) Once set, restart the ironic service, e.g. `systemctl restart ironic` and attempt to redeploy the node. You will want to view the system console occurring. If possible, you may wish to use `ipmitool` and write the output to a log file.

Gaining access via SSH to the node running IPA for custom images

Custom built images will require a user to be burned into the image. Typically a user would use the diskimage-builder `devuser` element to achieve this. More detail on this can be located at https://docs.openstack.org/diskimage-builder/latest/elements/devuser/README.html.

Example:

```
export DIB_DEV_USER_USERNAME=customuser
export DIB_DEV_USER_PWDLESS_SUDO=yes
export DIB_DEV_USER_AUTHORIZED_KEYS=$HOME/.ssh/id_rsa.pub
ironic-python-agent-builder -o /path/to/custom-ipa -e devuser debian
```

ssh_public_key_path is not valid

Bifrost requires that the user who executes bifrost have an SSH key in their user home, or that the user defines a variable to tell bifrost where to identify this file. Once this variable is defined to a valid file, the deployment playbook can be re-run.

Generating a new ssh key

See the manual page for the `ssh-keygen` command.

Defining a specific public key file

A user can define a specific public key file by utilizing the `ssh_public_key_path` variable. This can be set in the `group_vars/inventory/all` file, or on the `ansible-playbook` command line utilizing the `-e` command line parameter.

Example:

```
ansible-playbook -i inventory/bifrost_inventory.py deploy-dynamic.yaml -e ssh_
↪public_key_path=~/path/to/public/key/id_rsa.pub
```

NOTE: The matching private key will need to be utilized to login to the machine deployed.

Changing from TinyIPA to another IPA Image

With-in the Newton cycle, the default IPA image for Bifrost was changed to TinyIPA, which is based on Tiny Core Linux. This has a greatly reduced boot time for testing, however should be expected to have less hardware support. In the Yoga cycle, the default image was changed to one based on CentOS.

If on a fresh install, or a re-install, you wish to change to DIB-based or any other IPA image, you will need to take the following steps:

1. Remove the existing IPA image ipa.kernel and ipa.initramfs.
2. Edit the `playbooks/roles/bifrost-ironic-install/defaults/main.yml` file and update the `ipa_kernel_upstream_url` and `ipa_kernel_upstream_url` settings to a

new URL. For DIB-based images, these urls would be, `https://tarballs.opendev.org/openstack/ironic-python-agent/dib/files/ipa-centos9-master.kernel` and `https://tarballs.opendev.org/openstack/ironic-python-agent/dib/files/ipa-centos9-master.initramfs` respectively.

3. Execute the installation playbook, and the set files will be automatically downloaded again. If the files are not removed prior to (re)installation, then they will not be replaced. Alternatively, the files can just be directly replaced on disk. The default where the kernel and ramdisk are located is in `/httboot/`.

2.2.4 Using Keystone

Ultimately, as bifrost was designed for relatively short-lived installations to facilitate rapid hardware deployment, the default operating mode is referred to as `noauth` mode. In order to leverage Keystone authentication for the roles, Bifrost reads configuration from `clouds.yaml`. If `clouds.yaml` has not been generated through the `bifrost-keystone-client-config` role, one of the following steps need to take place:

1. Update the role defaults for each role you plan to make use. This may not make much sense for most users, unless they are carrying such changes as downstream debt.
2. Invoke ansible-playbook with variables being set to override the default behavior. Example:

```
-e enable_keystone=true -e noauth_mode=false -e cloud_name=bifrost
```

3. Set the global defaults for target (`master/playbooks/inventory/group_vars/target`).

OpenStack Client usage

A user wishing to invoke OSC commands against the bifrost installation, should set the `OS_CLOUD` environment variable. An example of setting the environment variable and then executing the OSC command to list all baremetal nodes:

```
export OS_CLOUD=bifrost
openstack baremetal node list
```

For administration actions, use the `bifrost-admin` cloud:

```
export OS_CLOUD=bifrost-admin
openstack endpoint list
```

Keystone roles

Ironic, which is the underlying OpenStack component bifrost helps a user leverage, supports two different roles in keystone that helps govern the rights a user has in keystone.

These roles are `baremetal_admin` and `baremetal_observer` and a user can learn more about the roles from the ironic install guide.

Individual playbook use

The OpenStack Ansible modules utilize `clouds.yaml` file to obtain authentication details to connect to determine details. The bifrost roles that speak with Ironic for actions such as enrollment of nodes and deployment, automatically attempt to collect authentication data from `clouds.yaml`. A user can explicitly select the cloud they wish to deploy to via the `cloud_name` parameter.

2.2.5 Configuring the integrated DHCP server

Setting static DHCP assignments with the integrated DHCP server

You can set up a static DHCP reservation using the `ipv4_address` parameter and setting the `inventory_dhcp` setting to a value of `true`. This will result in the first MAC address defined in the list of hardware MAC addresses to receive a static address assignment in dnsmasq.

Forcing DNS to resolve to ipv4_address

dnsmasq will resolve all entries to the IP assigned to each server in the leases file. However, this IP will not always be the desired one, if you are working with multiple networks. To force DNS to always resolve to `ipv4_address` please set the `inventory_dns` setting to a value of `true`. This will result in each server to resolve to `ipv4_address` by explicitly using address capabilities of dnsmasq.

Extending dnsmasq configuration

Bifrost manages the dnsmasq configuration file in `/etc/dnsmasq.conf`. It is not recommended to make manual modifications to this file after it has been written. dnsmasq supports the use of additional configuration files in `/etc/dnsmasq.d`, allowing extension of the dnsmasq configuration provided by bifrost. It is possible to use this mechanism provide additional DHCP options to systems managed by ironic, or even to create a DHCP boot environment for systems not managed by ironic. For example, create a file `/etc/dnsmasq.d/example.conf` with the following contents:

```
dhcp-match=set:<tag>,<match criteria>
dhcp-boot=tag:<tag>,<boot options>
```

The tag, match criteria and boot options should be modified for your environment. Here we use dnsmasq tags to match against hosts that we want to manage. dnsmasq will use the last matching tagged `dhcp-boot` option for a host or an untagged default `dhcp-boot` option if there were no matches. These options will be inserted at the `conf-dir=/etc/dnsmasq.d` line of the dnsmasq configuration file. Once configured, send the `HUP` signal to dnsmasq, which will cause it to reread its configuration:

```
killall -HUP dnsmasq
```

Using Bifrost with your own DHCP server

The possibility exists that a user may already have a Dynamic Host Configuration Protocol (DHCP) server on their network.

Currently Ironic, when configured with Bifrost in standalone mode, does not utilize a DHCP provider. This would require a manual configuration of the DHCP server to deploy an image. Bifrost utilizes dnsmasq for this functionality; however, any DHCP server can be utilized. This is largely intended to function in the context of a single flat network although conceivably the nodes can be segregated.

What is required:

- DHCP server on the network segment
- Appropriate permissions to change DHCP settings
- Network access to the API and conductor. Keep in mind the iPXE image does not support ICMP redirects.

Example DHCP server configurations

In the examples below port 8080 is used. However, the port number may vary depending on the environment configuration.

dnsmasq:

```
dhcp-match=set:ipxe,175 # iPXE sends a 175 option.
dhcp-boot=tag:ipxe,http://<Bifrost Host IP Address>:8080/boot.ipxe
dhcp-boot=/undionly.kpxe,<TFTP Server Hostname>,<TFTP Server IP Address>
```

Internet Systems Consortium DHCPd:

```
if exists user-class and option user-class = "iPXE" {
      filename "http://<Bifrost Host IP Address>:8080/boot.ipxe";
} else {
      filename "/undionly.kpxe";
      next-server <TFTP Server IP Address>;
}
```

Architecture

It should be emphasized that Ironic in standalone mode is intended to be used only in a trusted environment.

```
         +-------------+
         | DHCP Server |
         +-------------+
                |
+--------Trusted-Network----------+
       |                        |
+-------------+       +-----------+
```

(continues on next page)

(continued from previous page)

```
|Ironic Server|       |   Server   |
+-------------+       +------------+
```

2.3 Contributor Guide

2.3.1 Contributing

Bifrost is a part of Ironic, which is an OpenStack project and thus follows OpenStack development procedures.

For a full (and official) description of the development workflow, see:

> https://docs.openstack.org/infra/manual/developers.html#development-workflow

For a highly abridged version, read on.

Communicating

Before you file a bug or new review set, its often helpful to chat with other developers. The #openstack-ironic channel on OFTC IRC network (irc://irc.oftc.net/#openstack-ironic) is a good place to start, and if you dont have IRC (or would prefer email), openstack-discuss@lists.openstack.org is the mailing list for all OpenStack projects. As the name implies, that mailing list is for all OpenStack development, so its often harder to get attention on your particular issue.

Filing Bugs

Bugs should be filed in launchpad, not GitHub:

> https://launchpad.net/bifrost

Contributing Code

Bifrost requires a valid OpenStack contributor agreement to be signed before code can be accepted. Details can be found in the development workflow link above.

Code isnt committed directly (so pull requests wont work); instead, the code is submitted for review through Gerrit via git review, and once its been sufficiently reviewed it will be merged from there.

Once thats done, the development workflow is, roughly:

```
$ git clone https://opendev.org/openstack/bifrost
$ cd bifrost
$ git checkout -b some-branch-name
... hack hack hack ...
$ git commit
$ git review
... The configuration details for this are in .gitreview.
... When the command runs, it will add a ChangeId to your commit
... message and print out a link for your reference
```

(continues on next page)

(continued from previous page)

```
...
... If you need to fix something in that commit, you can do:
$ git commit --amend
$ git review
```

From that point on, the link the git review command generated is the place to do final tweaks. When its approved, the code will be merged in automatically.

If you propose a new feature and are unable to complete it, please let the community know by commenting in the review set indicating that someone else is free to carry on your change. If the core reviewers observe reviews that are not being actively worked on, we are likely to inquire with you. If a review is untouched and the owner of the review is unreachable for a lengthy period of time, such as three to six months, the core reviewers may abandon the change as we do not utilize auto-abandon.

Code Style

Bifrost is a mix of Python, YaML, and bash thrown in for good measure.

The overall intent is to keep features, and changes simple to permit a user to easily understand and extend bifrost to meet their operational needs as we recognize needs may vary.

With this, we have a list of things that we would like people to keep in mind when contributing code.

1. Try to limit YaML to 79 characters per row, we understand this is not always possible, but please make an effort.
2. Try to keep change sets as short and to the point as possible.
3. Rather than pass key-value pair strings to Ansible modules, try to utilize key-value pair lists on a module command line. Example:

   ```
   - name: "Stat file for x reason"
     stat:
       file: '/path/to/file'
   ```

4. Playbook conditionals utilizing variables intended as booleans, should make use of the `| bool` casting feature. This is due to command line overrides are typically interpreted as strings instead of booleans. Example:

   ```
   - name: "Something something something"
     module:
       parameter: "value"
     when: boolean_value | bool == true
   ```

5. Be clear and explicit with actions in playbooks and comments.
6. Simplicity is favored over magic.
7. Documentation should generally be paired with code changes as we feel that it is important for us to be able to release the master branch at any time.
8. Documentation should always be limited to 79 characters per row.
9. If you have any questions, please ask in #openstack-ironic.

2.3.2 Bifrost via Vagrant

One of the main user audiences that weve found is for users to utilize vagrant in order to build quick development environments, or for their environments to facilitate deployments, as the intent is for relatively short lived installations.

As such, a virtual machine can be started with vagrant executing the following commands:

```
cd tools/vagrant_dev_env
vagrant up
```

This will bring up an Ubuntu based virtual machine, with bifrost installed.

Note

Virtual machine images, as well as all of the software used in bifrost can take some time to install. Typically expect `vagrant up` to take at least fifteen minutes if you do not already have the virtual machine image on your machine.

By default, the VM will have three interfaces:

- **eth0** - connected to a NAT network
- **eth1** - connected to Host-only network named: vboxnet1
- **eth2** - bridged - adapter must be set in Vagrantfile

Walkthrough done on OS X

Setup vagrant by:

- Installing git
- Installing virtualbox
- Installing vagrant
- Installing ansible

Configure Vagrant with the correct box:

```
vagrant box add ubuntu/bionic64
```

Clone bifrost repo:

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

Change into the bifrost directory:

```
cd bifrost/tools/vagrant_dev_env
```

Edit the Vagrantfile:

- Change the `bifrost.vm.network public_network` value to a valid network interface to allow Bare Metal connectivity

- Change `public_key` to correct key name
- Change `network_interface` to match your needs

Boot the VM with:

```
vagrant up
```

Installation Options

Ansible is installed within the VM directly from source or from the path set by `ANSIBLE_GIT_URL` into `/opt/stack`.

2.3.3 Testing Environment

Quick start with bifrost-cli

If you want to try Bifrost on virtual machines instead of real hardware, you need to prepare a testing environment. The easiest way is via `bifrost-cli`, available since the Victoria release series:

```
./bifrost-cli testenv
```

Additionally, the following parameters can be useful:

`--develop`
: Install services in develop mode, so that the changes to the repositories in `/opt` get immediately reflected in the virtual environment.

> **Note**
>
> You still need to restart services to apply any changes, e.g.:
>
> ```
> sudo systemctl restart ironic
> ```

`--driver=[ipmi|redfish]`
: Choose the default driver for the generated nodes inventory.

> **Note**
>
> Both IPMI and Redfish support is configured anyway, so you can switch the drivers on fly if needed.
>
> IPMI support uses VirtualBMC, Redfish - sushy-tools.

`--uefi`
: Makes the testing VMs boot with UEFI.

See the built-in documentation for more details:

```
./bifrost-cli testenv --help
```

The command generates two files with node inventory in the current directory:

- `baremetal-inventory.json` can be used with the provided playbooks, see *How-To* for details.
- `baremetal-nodes.json` can be used with the Ironic enrollment command:

```
export OS_CLOUD=bifrost
baremetal create baremetal-nodes.json
```

Reproduce CI testing locally

A simple `scripts/test-bifrost.sh` script can be utilized to install pre-requisite software packages, Ansible, and then execute the `test-bifrost-create-vm.yaml` and `test-bifrost.yaml` playbooks in order to provide a single step testing mechanism.

`playbooks/test-bifrost-create-vm.yaml` creates one or more VMs for testing and saves out a baremetal.json file which is used by `playbooks/test-bifrost.yaml` to execute the remaining roles. Two additional roles are invoked by this playbook which enables Ansible to connect to the new nodes by adding them to the inventory, and then logging into the remote machine via the users ssh host key. Once that has successfully occurred, additional roles will unprovision the host(s) and delete them from ironic.

Command:

```
scripts/test-bifrost.sh
```

Note:

- In order to cap requirements for installation, an `upper_constraints_file` setting is defined. This is consuming the `UPPER_CONSTRAINTS_FILE` or `TOX_CONSTRAINTS_FILE` env var by default, to properly integrate with CI systems, and will default to `/opt/stack/requirements/upper-constraints.txt` file if not present.

Manually test with Virtual Machines

Bifrost supports using virtual machines to emulate the hardware.

The VirtualBMC project is used as an IPMI proxy, so that the same `ipmi` hardware type can be used as for real hardware. Redfish emulator from sushy-tools is also installed.

1. Set `testing` to *true* in the `playbooks/inventory/group_vars/target` file.
2. You may need to adjust the value for `ssh_public_key_path`.
3. Execute the `ansible-playbook -vvvv -i inventory/target test-bifrost-create-vm.yaml` command to create a test virtual machine.
4. Run the install step, as documented in *Bifrost Installation*, however adding `-e testing=true` to the Ansible command line.
5. Set the environment variable of `BIFROST_INVENTORY_SOURCE` to the path to the JSON file, which by default has been written to `/tmp/baremetal.json`.
6. Run the *enrollment step*, using the JSON file you created in the previous step.
7. Run the deployment step, as documented in *Deploy Hardware*.