ceph-ansible

Ansible playbooks for Ceph, the distributed filesystem.

Installation

github

You can install directly from the source on github by following these steps:

- Clone the repository:

```
git clone https://github.com/ceph/ceph-ansible.git
```

- Next, you must decide which branch of `ceph-ansible` you wish to use. There are stable branches to choose from or you could use the master branch:

```
git checkout $branch
```

Ansible on RHEL and CentOS

You can acquire Ansible on RHEL and CentOS by installing from Extras.

On RHEL:

```
subscription-manager repos --enable=rhel-7-server-extras-rpms
```

(CentOS does not use subscription-manager and already has “Extras” enabled by default.)

```
sudo yum install ansible
```

Ansible on Ubuntu

You can acquire Ansible on Ubuntu by using the Ansible PPA.

```
sudo add-apt-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

Releases

The following branches should be used depending on your requirements. The `stable-*` branches have been QE tested and sometimes recieve backport fixes throughout their lifecycle. The `master` branch should be considered experimental and used with caution.

- `stable-2.1` Support for ceph version `jewel`. This branch supports ansible versions `2.1` and `2.2.1`.
- `stable-2.2` Support for ceph versions `jewel` and `luminous`. This branch supports ansible versions `2.1` and `2.2.2`.
- `stable-3.0` Support for ceph versions `jewel` and `luminous`. This branch supports ansible versions `2.3.1`, `2.3.2` and `2.4.2`.
- `master` Support for ceph versions `jewel`, and `luminous`. This branch supports ansible version `2.4.2`.

Configuration and Usage

This project assumes you have a basic knowledge of how ansible works and have already prepared your hosts for configuration by ansible.

After you've cloned the `ceph-ansible` repository, selected your branch and installed ansible then you'll need to create your inventory file, playbook and configuration for your ceph cluster.

Inventory

The ansible inventory file defines the hosts in your cluster and what roles each host plays in your ceph cluster. The default location for an inventory file is `/etc/ansible/hosts` but this file can be placed anywhere and used with the `-i` flag of ansible-playbook. An example inventory file would look like:

```
[mons]
mon1
mon2
mon3

[osds]
osd1
osd2
osd3
```

> **Note:**
>
> For more information on ansible inventories please refer to the ansible documentation: http://docs.ansible.com/ansible/latest/intro_inventory.html

Playbook

You must have a playbook to pass to the `ansible-playbook` command when deploying your cluster. There is a sample playbook at the root of the `ceph-ansible` project called `site.yml.sample`. This playbook should work fine for most usages, but it does include by default every daemon group which might not be appropriate for your cluster setup. Perform the following steps to prepare your playbook:

- Rename the sample playbook: `mv site.yml.sample site.yml`
- Modify the playbook as necessary for the requirements of your cluster

> **Note:**
>
> It's important the playbook you use is placed at the root of the `ceph-ansible` project. This is how ansible will be able to find the roles that `ceph-ansible` provides.

ceph-ansible - choose installation method

Ceph can be installed through several methods.

- Installation methods

ceph-ansible Configuration

The configuration for your ceph cluster will be set by the use of ansible variables that `ceph-ansible` provides. All of these options and their default values are defined in the `group_vars/` directory at the root of the `ceph-ansible` project. Ansible will use configuration in a `group_vars/` directory that is relative to your inventory file or your playbook. Inside of the

`group_vars/` directory there are many sample ansible configuration files that relate to each of the ceph daemon groups by their filename. For example, the `osds.yml.sample` contains all the default configuation for the OSD daemons. The `all.yml.sample` file is a special `group_vars` file that applies to all hosts in your cluster.

> **Note:**
>
> For more information on setting group or host specific configuration refer to the ansible documentation: http://docs.ansible.com/ansible/latest/intro_inventory.html#splitting-out-host-and-group-specific-data

At the most basic level you must tell `ceph-ansible` what version of ceph you wish to install, the method of installation, your clusters network settings and how you want your OSDs configured. To begin your configuration rename each file in `group_vars/` you wish to use so that it does not include the `.sample` at the end of the filename, uncomment the options you wish to change and provide your own value.

An example configuration that deploys the upstream `jewel` version of ceph with OSDs that have collocated journals would look like this in `group_vars/all.yml`:

```
ceph_origin: repository
ceph_repository: community
ceph_stable_release: jewel
public_network: "192.168.3.0/24"
cluster_network: "192.168.4.0/24"
monitor_interface: eth1
devices:
  - '/dev/sda'
  - '/dev/sdb'
osd_scenario: collocated
```

The following config options are required to be changed on all installations but there could be other required options depending on your OSD scenario selection or other aspects of your cluster.

- `ceph_origin`
- `ceph_stable_release`
- `public_network`
- `osd_scenario`
- `monitor_interface` or `monitor_address`
- `radosgw_interface` or `radosgw_address`

ceph.conf Configuration

The supported method for defining your ceph.conf is to use the `ceph_conf_overrides` variable. This allows you to specify configuration options using an INI format. This variable can be used to override sections already defined in ceph.conf (see: `roles/ceph-common/templates/ceph.conf.j2`) or to provide new configuration options. The following sections in ceph.conf are supported: [global], [mon], [osd], [mds] and [rgw].

An example:

```
ceph_conf_overrides:
   global:
     foo: 1234
     bar: 5678
   osd:
     osd_mkfs_type: ext4
```

> **Note:**
>
> We will no longer accept pull requests that modify the ceph.conf template unless it helps the deployment. For simple configuration tweaks please use the `ceph_conf_overrides` variable.

Full documentation for configuring each of the ceph daemon types are in the following sections.

OSD Configuration

OSD configuration is set by selecting an osd scenario and providing the configuration needed for that scenario. Each scenario is different in it's requirements. Selecting your OSD scenario is done by setting the `osd_scenario` configuration option.

- OSD Scenarios

Contribution

See the following section for guidelines on how to contribute to `ceph-ansible`.

- Contribution Guidelines

Testing

Documentation for writing functional testing scenarios for ceph-ansible.

- Testing with ceph-ansible
- Glossary

Demos

Vagrant Demo

Deployment from scratch on bare metal machines: https://youtu.be/E8-96NamLDo

Bare metal demo

Deployment from scratch on bare metal machines: https://youtu.be/dv_PEp9qAqg

Index

Search

From here you can search these documents. Enter your search words into the box below and click "search". Note that the search function will automatically search for all of the words. Pages containing fewer words won't appear in the result list.

search

Installation methods

The following are all of the available options for the installing Ceph through different channels. We support 3 main installation methods, all managed by the `ceph_origin` variable:

- `repository`: means that you will get ceph installed through a new repository. Later below choose between `community`, `rhcs` or `dev`. These options will be exposed through the `ceph_repository` variable.
- `distro`: means that no separate repo file will be added and you will get whatever version of Ceph is included in your Linux distro.
- `local`: means that the ceph binaries will be copied over from the local machine (not well tested, use at your own risk)

Origin: Repository

If `ceph_origin` is set to `repository`, you now have the choice between a couple of repositories controlled by the `ceph_repository` option:

- `community`: fetches packages from http://download.ceph.com, the official community Ceph repositories
- `rhcs`: means you are a Red Hat customer, additionally you will have to select a repository type through `ceph_repository_type` (`cdn` or `iso`)
- `dev`: fetches packages from shaman, a gitbuilder based package system
- `uca`: fetches packages from Ubuntu Cloud Archive
- `custom`: fetches packages from a specific repository

Community repository

If `ceph_repository` is set to `community`, packages you will be by default installed from http://download.ceph.com, this can be changed by tweaking `ceph_mirror`. Final step is to select which Ceph release you want to install, for this you have to set `ceph_stable_release` accordingly. For example, `ceph_stable_release: luminous`.

RHCS repository

RHCS is the Red Hat Ceph Storage product from Red Hat, the enterprise version of Ceph. If `ceph_repository` is set to `rhcs`, packages you will be installed from Red Hat sources.

Additionally you will have to select a repository type through `ceph_repository_type`, it can be `cdn` or `iso`. To choose a specific version of RHCS you can set the `ceph_rhcs_version` variable accordingly, e.g: `ceph_rhcs_version: 2`.

UCA repository

If `ceph_repository` is set to `uca`, packages you will be by default installed from http://ubuntu-cloud.archive.canonical.com/ubuntu, this can be changed by tweaking `ceph_stable_repo_uca`. You can also decide which OpenStack version the Ceph packages should come from by tweaking `ceph_stable_openstack_release_uca`. For example, `ceph_stable_openstack_release_uca: liberty`.

Dev repository

If `ceph_repository` is set to `dev`, packages you will be by default installed from https://shaman.ceph.com/, this can not be tweaked. You can obviously decide which branch to install with the help of `ceph_dev_branch` (defaults to ‘master’). Additionally, you can

specify a SHA1 with `ceph_dev_sha1`, defaults to ‘latest’ (as in latest built).

Custom repository

If `ceph_repository` is set to `custom`, packages you will be by default installed from a desired repository. This repository is specifie with `ceph_custom_repo`, e.g: `ceph_custom_repo: https://server.domain.com/ceph-custom-repo`.

Origin: Distro

If `ceph_origin` is set to `distro`, no separate repo file will be added and you will get whatever version of Ceph is included in your Linux distro.

Origin: Local

If `ceph_origin` is set to `local`, the ceph binaries will be copied over from the local machine (not well tested, use at your own risk)

Glossary

Contents:

OSD Scenarios

The following are all of the available options for the `osd_scenario` config setting. Defining an `osd_scenario` is mandatory for using `ceph-ansible`.

collocated

This OSD scenario uses `ceph-disk` to create OSDs with collocated journals from raw devices.

Use `osd_scenario: collocated` to enable this scenario. This scenario also has the following required configuration options:

- `devices`

This scenario has the following optional configuration options:

- `osd_objectstore`: defaults to `filestore` if not set. Available options are `filestore` or `bluestore`. You can only select `bluestore` if the ceph release is Luminous or greater.
- `dmcrypt`: defaults to `false` if not set.

This scenario supports encrypting your OSDs by setting `dmcrypt: True`.

If `osd_objectstore: filestore` is enabled both 'ceph data' and 'ceph journal' partitions will be stored on the same device.

If `osd_objectstore: bluestore` is enabled 'ceph data', 'ceph block', 'ceph block.db', 'ceph block.wal' will be stored on the same device. The device will get 2 partitions:

- One for 'data', called 'ceph data'
- One for 'ceph block', 'ceph block.db', 'ceph block.wal' called 'ceph block'

Example of what you will get:

```
[root@ceph-osd0 ~]# blkid /dev/sda*
/dev/sda: PTTYPE="gpt"
/dev/sda1: UUID="9c43e346-dd6e-431f-92d8-cbed4ccb25f6" TYPE="xfs" PARTLABEL="ceph data
/dev/sda2: PARTLABEL="ceph block" PARTUUID="e6ca3e1d-4702-4569-abfa-e285de328e9d"
```

An example of using the `collocated` OSD scenario with encryption would look like:

```
osd_scenario: collocated
dmcrypt: true
devices:
  - /dev/sda
  - /dev/sdb
```

non-collocated

This OSD scenario uses `ceph-disk` to create OSDs from raw devices with journals that exist on a dedicated device.

Use `osd_scenario: non-collocated` to enable this scenario. This scenario also has the following required configuration options:

- `devices`

This scenario has the following optional configuration options:

- `dedicated_devices`: defaults to `devices` if not set
- `osd_objectstore`: defaults to `filestore` if not set. Available options are `filestore` or

`bluestore`. You can only select `bluestore` with the ceph release is Luminous or greater.
- `dmcrypt`: defaults to `false` if not set.

This scenario supports encrypting your OSDs by setting `dmcrypt: True`.

If `osd_objectstore: filestore` is enabled 'ceph data' and 'ceph journal' partitions will be stored on different devices: - 'ceph data' will be stored on the device listed in `devices` - 'ceph journal' will be stored on the device listed in `dedicated_devices`

Let's take an example, imagine `devices` was declared like this:

```
devices:
  - /dev/sda
  - /dev/sdb
  - /dev/sdc
  - /dev/sdd
```

And `dedicated_devices` was declared like this:

```
dedicated_devices:
  - /dev/sdf
  - /dev/sdf
  - /dev/sdg
  - /dev/sdg
```

This will result in the following mapping:

- /dev/sda will have /dev/sdf1 as journal
- /dev/sdb will have /dev/sdf2 as a journal
- /dev/sdc will have /dev/sdg1 as a journal
- /dev/sdd will have /dev/sdg2 as a journal

> ## Note:
>
> On a containerized scenario we only support A SINGLE journal for all the OSDs on a given machine. If you don't, bad things will happen This is a limitation we plan to fix at some point.

If `osd_objectstore: bluestore` is enabled, both 'ceph block.db' and 'ceph block.wal' partitions will be stored on a dedicated device.

So the following will happen:

- The devices listed in `devices` will get 2 partitions, one for 'block' and one for 'data'. 'data' is only 100MB big and do not store any of your data, it's just a bunch of Ceph metadata. 'block' will store all your actual data.
- The devices in `dedicated_devices` will get 1 partition for RocksDB DB, called 'block.db' and one for RocksDB WAL, called 'block.wal'

By default `dedicated_devices` will represent block.db

Example of what you will get:

```
[root@ceph-osd0 ~]# blkid /dev/sd*
/dev/sda: PTTYPE="gpt"
/dev/sda1: UUID="c6821801-2f21-4980-add0-b7fc8bd424d5" TYPE="xfs" PARTLABEL="ceph data
/dev/sda2: PARTLABEL="ceph block" PARTUUID="ea454807-983a-4cf2-899e-b2680643bc1c"
/dev/sdb: PTTYPE="gpt"
/dev/sdb1: PARTLABEL="ceph block.db" PARTUUID="af5b2d74-4c08-42cf-be57-7248c739e217"
/dev/sdb2: PARTLABEL="ceph block.wal" PARTUUID="af3f8327-9aa9-4c2b-a497-cf0fe96d126a"
```

There is more device granularity for Bluestore ONLY if `osd_objectstore: bluestore` is

enabled by setting the `bluestore_wal_devices` config option.

By default, if `bluestore_wal_devices` is empty, it will get the content of `dedicated_devices`. If set, then you will have a dedicated partition on a specific device for block.wal.

Example of what you will get:

```
[root@ceph-osd0 ~]# blkid /dev/sd*
/dev/sda: PTTYPE="gpt"
/dev/sda1: UUID="39241ae9-d119-4335-96b3-0898da8f45ce" TYPE="xfs" PARTLABEL="ceph data
/dev/sda2: PARTLABEL="ceph block" PARTUUID="bff8e54e-b780-4ece-aa16-3b2f2b8eb699"
/dev/sdb: PTTYPE="gpt"
/dev/sdb1: PARTLABEL="ceph block.db" PARTUUID="0734f6b6-cc94-49e9-93de-ba7e1d5b79e3"
/dev/sdc: PTTYPE="gpt"
/dev/sdc1: PARTLABEL="ceph block.wal" PARTUUID="824b84ba-6777-4272-bbbd-bfe2a25cecf3"
```

An example of using the `non-collocated` OSD scenario with encryption, bluestore and dedicated wal devices would look like:

```
osd_scenario: non-collocated
osd_objectstore: bluestore
dmcrypt: true
devices:
  - /dev/sda
  - /dev/sdb
dedicated_devices:
  - /dev/sdc
  - /dev/sdc
bluestore_wal_devices:
  - /dev/sdd
  - /dev/sdd
```

lvm

This OSD scenario uses `ceph-volume` to create OSDs from logical volumes and is only available when the ceph release is Luminous or newer.

> ## Note:
>
> The creation of the logical volumes is not supported by `ceph-ansible`, `ceph-volume` only creates OSDs from existing logical volumes.

`lvm_volumes` is the config option that needs to be defined to configure the mappings for devices to be deployed. It is a list of dictionaries which expects a volume name and a volume group for logical volumes, but can also accept a partition in the case of `filestore` for the `journal`.

This scenario supports encrypting your OSDs by setting `dmcrypt: True`. If set, all OSDs defined in `lvm_volumes` will be encrypted.

The `data` key represents the logical volume name, raw device or partition that is to be used for your OSD data. The `data_vg` key represents the volume group name that your `data` logical volume resides on. This key is required for purging of OSDs created by this scenario.

> ## Note:
>
> Any logical volume or logical group used in `lvm_volumes` must be a name and not a path.

> ## Note:

You can not use the same journal for many OSDs.

filestore

There is filestore support which can be enabled with:

```
osd_objectstore: filestore
```

To configure this scenario use the `lvm_volumes` config option. `lvm_volumes` is a list of dictionaries which expects a volume name and a volume group for logical volumes, but can also accept a parition in the case of `filestore` for the `journal`.

The following keys are accepted for a `filestore` deployment:

- `data`
- `data_vg` (not required if `data` is a raw device or partition)
- `journal`
- `journal_vg` (not required if `journal` is a partition and not a logical volume)
- `crush_device_class` (optional, sets the crush device class for the OSD)

The `journal` key represents the logical volume name or partition that will be used for your OSD journal.

For example, a configuration to use the `lvm` osd scenario would look like:

```
osd_objectstore: filestore
osd_scenario: lvm
lvm_volumes:
  - data: data-lv1
    data_vg: vg1
    journal: journal-lv1
    journal_vg: vg2
    crush_device_class: foo
  - data: data-lv2
    journal: /dev/sda
    data_vg: vg1
  - data: data-lv3
    journal: /dev/sdb1
    data_vg: vg2
  - data: /dev/sda
    journal: /dev/sdb1
  - data: /dev/sda1
    journal: journal-lv1
    journal_vg: vg2
```

For example, a configuration to use the `lvm` osd scenario with encryption would look like:

```
osd_objectstore: filestore
osd_scenario: lvm
dmcrypt: True
lvm_volumes:
  - data: data-lv1
    data_vg: vg1
    journal: journal-lv1
    journal_vg: vg2
    crush_device_class: foo
```

bluestore

This scenario allows a combination of devices to be used in an OSD. `bluestore` can work just with a single “block” device (specified by the `data` and optionally `data_vg`) or additionally with a `block.wal` and `block.db` (interchangeably)

The following keys are accepted for a `bluestore` deployment:

- `data` (required)
- `data_vg` (not required if `data` is a raw device or partition)
- `db` (optional for `block.db`)
- `db_vg` (optional for `block.db`)
- `wal` (optional for `block.wal`)
- `wal_vg` (optional for `block.wal`)
- `crush_device_class` (optional, sets the crush device class for the OSD)

A `bluestore` lvm deployment, for all four different combinations supported could look like:

```
  osd_objectstore: bluestore
  osd_scenario: lvm
  lvm_volumes:
    - data: data-lv1
      data_vg: vg1
      crush_device_class: foo
    - data: data-lv2
      data_vg: vg1
      wal: wal-lv1
      wal_vg: vg2
    - data: data-lv3
      data_vg: vg2
      db: db-lv1
      db_vg: vg2
    - data: data-lv4
      data_vg: vg4
      db: db-lv4
      db_vg: vg4
      wal: wal-lv4
      wal_vg: vg4
    - data: /dev/sda
```

Contribution Guidelines

The repository centralises all the Ansible roles. The roles are all part of the Galaxy. We love contribution and we love giving visibility to our contributors, this is why all the **commits must be signed-off**.

Mailing list

Please register the mailing list at http://lists.ceph.com/listinfo.cgi/ceph-ansible-ceph.com

IRC

Feel free to join us in the channel #ceph-ansible of the OFTC servers

Github

The main github account for the project is at https://github.com/ceph/ceph-ansible/

Submit a patch

To start contributing just do:

```
$ git checkout -b my-working-branch
$ # do your changes #
$ git add -p
```

If your change impacts a variable file in a role such as `roles/ceph-common/defaults/main.yml`, you need to generate a `group_vars` file:

```
$ ./generate_group_vars_sample.sh
```

You are finally ready to push your changes on Github:

```
$ git commit -s
$ git push origin my-working-branch
```

Worked on a change and you don't want to resend a commit for a syntax fix?

```
$ # do your syntax change #
$ git commit --amend
$ git push -f origin my-working-branch
```

PR Testing

Pull Request testing is handled by jenkins. All test must pass before your PR will be merged.

All of tests that are running are listed in the github UI and will list their current status.

If a test fails and you'd like to rerun it, comment on your PR in the following format:

```
jenkins test $scenario_name
```

For example:

```
jenkins test luminous-ansible2.3-journal_collocation
```

Backporting changes

If a change should be backported to a `stable-*` Git branch:

- Mark your PR with the GitHub label “Backport” so we don’t lose track of it.
- Fetch the latest updates into your clone: `git fetch`
- Determine the latest available stable branch: `git branch -r --list "origin/stable-[0-9].[0-9]" | sort -r | sed 1q`
- Create a new local branch for your PR, based on the stable branch: `git checkout --no-track -b my-backported-change origin/stable-3.0`
- Cherry-pick your change: `git cherry-pick -x (your-sha1)`
- Create a new pull request against the `stable-3.0` branch.
- Ensure that your PR’s title has the prefix “backport:”, so it’s clear to reviewers what this is about.
- Add a comment in your backport PR linking to the original (master) PR.

All changes to the stable branches should land in master first, so we avoid regressions.

Once this is done, one of the project maintainers will tag the tip of the stable branch with your change. For example:

```
git checkout stable-3.0
git pull --ff-only
git tag v3.0.12
git push origin v3.0.12
```

Testing

ceph-ansible has the ability to test different scenarios (collocated journals or dmcrypt OSDs for example) in an isolated, repeatable, and easy way.

These tests can run locally with VirtualBox or via libvirt if available, which removes the need to solely rely on a CI system like Jenkins to verify a behavior.

- **Getting started:** Running a Test Scenario | Dependencies
- **Configuration and structure:** Layout and conventions | Test Files | Scenario Files | Scenario Wiring
- **Adding or modifying tests:** Conventions | testinfra |
- **Adding or modifying a scenario:** Conventions | Environment configuration | Ansible configuration |
- **Custom/development repositories and packages:** Environment variables |

Glossary

- Testing
- Running Tests
- ceph-ansible testing for development
- Test Scenarios
- Scenario Files
- Modifying (or adding) tests
- Layout and conventions
- Conventions
- **`testinfra`**
- Tests
- Test Fixtures
- **`tox`**

ceph-ansible

Ansible playbooks for Ceph, the distributed filesystem.

Installation

github

You can install directly from the source on github by following these steps:

- Clone the repository:

```
git clone https://github.com/ceph/ceph-ansible.git
```

- Next, you must decide which branch of `ceph-ansible` you wish to use. There are stable branches to choose from or you could use the master branch:

```
git checkout $branch
```

Ansible on RHEL and CentOS

You can acquire Ansible on RHEL and CentOS by installing from Extras.

On RHEL:

```
subscription-manager repos --enable=rhel-7-server-extras-rpms
```

(CentOS does not use subscription-manager and already has “Extras” enabled by default.)

```
sudo yum install ansible
```

Ansible on Ubuntu

You can acquire Ansible on Ubuntu by using the Ansible PPA.

```
sudo add-apt-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

Releases

The following branches should be used depending on your requirements. The `stable-*` branches have been QE tested and sometimes recieve backport fixes throughout their lifecycle. The `master` branch should be considered experimental and used with caution.

- `stable-2.1` Support for ceph version `jewel`. This branch supports ansible versions `2.1` and `2.2.1`.
- `stable-2.2` Support for ceph versions `jewel` and `luminous`. This branch supports ansible versions `2.1` and `2.2.2`.
- `stable-3.0` Support for ceph versions `jewel` and `luminous`. This branch supports ansible versions `2.3.1`, `2.3.2` and `2.4.2`.
- `master` Support for ceph versions `jewel`, and `luminous`. This branch supports ansible version `2.4.2`.

Configuration and Usage

This project assumes you have a basic knowledge of how ansible works and have already prepared your hosts for configuration by ansible.

After you've cloned the `ceph-ansible` repository, selected your branch and installed ansible then you'll need to create your inventory file, playbook and configuration for your ceph cluster.

Inventory

The ansible inventory file defines the hosts in your cluster and what roles each host plays in your ceph cluster. The default location for an inventory file is `/etc/ansible/hosts` but this file can be placed anywhere and used with the `-i` flag of ansible-playbook. An example inventory file would look like:

```
[mons]
mon1
mon2
mon3

[osds]
osd1
osd2
osd3
```

> **Note:**
>
> For more information on ansible inventories please refer to the ansible documentation: http://docs.ansible.com/ansible/latest/intro_inventory.html

Playbook

You must have a playbook to pass to the `ansible-playbook` command when deploying your cluster. There is a sample playbook at the root of the `ceph-ansible` project called `site.yml.sample`. This playbook should work fine for most usages, but it does include by default every daemon group which might not be appropriate for your cluster setup. Perform the following steps to prepare your playbook:

- Rename the sample playbook: `mv site.yml.sample site.yml`
- Modify the playbook as necessary for the requirements of your cluster

> **Note:**
>
> It's important the playbook you use is placed at the root of the `ceph-ansible` project. This is how ansible will be able to find the roles that `ceph-ansible` provides.

ceph-ansible - choose installation method

Ceph can be installed through several methods.

- Installation methods

ceph-ansible Configuration

The configuration for your ceph cluster will be set by the use of ansible variables that `ceph-ansible` provides. All of these options and their default values are defined in the `group_vars/` directory at the root of the `ceph-ansible` project. Ansible will use configuration in a `group_vars/` directory that is relative to your inventory file or your playbook. Inside of the

`group_vars/` directory there are many sample ansible configuration files that relate to each of the ceph daemon groups by their filename. For example, the `osds.yml.sample` contains all the default configuation for the OSD daemons. The `all.yml.sample` file is a special `group_vars` file that applies to all hosts in your cluster.

> **Note:**
>
> For more information on setting group or host specific configuration refer to the ansible documentation: http://docs.ansible.com/ansible/latest/intro_inventory.html#splitting-out-host-and-group-specific-data

At the most basic level you must tell `ceph-ansible` what version of ceph you wish to install, the method of installation, your clusters network settings and how you want your OSDs configured. To begin your configuration rename each file in `group_vars/` you wish to use so that it does not include the `.sample` at the end of the filename, uncomment the options you wish to change and provide your own value.

An example configuration that deploys the upstream `jewel` version of ceph with OSDs that have collocated journals would look like this in `group_vars/all.yml`:

```
ceph_origin: repository
ceph_repository: community
ceph_stable_release: jewel
public_network: "192.168.3.0/24"
cluster_network: "192.168.4.0/24"
monitor_interface: eth1
devices:
  - '/dev/sda'
  - '/dev/sdb'
osd_scenario: collocated
```

The following config options are required to be changed on all installations but there could be other required options depending on your OSD scenario selection or other aspects of your cluster.

- `ceph_origin`
- `ceph_stable_release`
- `public_network`
- `osd_scenario`
- `monitor_interface` or `monitor_address`
- `radosgw_interface` or `radosgw_address`

ceph.conf Configuration

The supported method for defining your ceph.conf is to use the `ceph_conf_overrides` variable. This allows you to specify configuration options using an INI format. This variable can be used to override sections already defined in ceph.conf (see: `roles/ceph-common/templates/ceph.conf.j2`) or to provide new configuration options. The following sections in ceph.conf are supported: [global], [mon], [osd], [mds] and [rgw].

An example:

```
ceph_conf_overrides:
   global:
     foo: 1234
     bar: 5678
   osd:
     osd_mkfs_type: ext4
```

> **Note:**
>
> We will no longer accept pull requests that modify the ceph.conf template unless it helps the deployment. For simple configuration tweaks please use the `ceph_conf_overrides` variable.

Full documentation for configuring each of the ceph daemon types are in the following sections.

OSD Configuration

OSD configuration is set by selecting an osd scenario and providing the configuration needed for that scenario. Each scenario is different in it's requirements. Selecting your OSD scenario is done by setting the `osd_scenario` configuration option.

- OSD Scenarios

Contribution

See the following section for guidelines on how to contribute to `ceph-ansible`.

- Contribution Guidelines

Testing

Documentation for writing functional testing scenarios for ceph-ansible.

- Testing with ceph-ansible
- Glossary

Demos

Vagrant Demo

Deployment from scratch on bare metal machines: https://youtu.be/E8-96NamLDo

Bare metal demo

Deployment from scratch on bare metal machines: https://youtu.be/dv_PEp9qAqg

Running Tests

Although tests run continuously in CI, a lot of effort was put into making it easy to run in any environment, as long as a couple of requirements are met.

Dependencies

There are some Python dependencies, which are listed in a `requirements.txt` file within the `tests/` directory. These are meant to be installed using Python install tools (pip in this case):

```
pip install -r tests/requirements.txt
```

For virtualization, either libvirt or VirtualBox is needed (there is native support from the harness for both). This makes the test harness even more flexible as most platforms will be covered by either VirtualBox or libvirt.

Running a scenario

Tests are driven by `tox`, a command line tool to run a matrix of tests defined in a configuration file (`tox.ini` in this case at the root of the project).

For a thorough description of a scenario see Test Scenarios.

To run a single scenario, make sure it is available (should be defined from `tox.ini`) by listing them:

```
tox -l
```

In this example, we will use the `luminous-ansible2.4-xenial_cluster` one. The harness defaults to `VirtualBox` as the backend, so if you have that installed in your system then this command should just work:

```
tox -e luminous-ansible2.4-xenial_cluster
```

And for libvirt it would be:

```
tox -e luminous-ansible2.4-xenial_cluster -- --provider=libvirt
```

Warning:

Depending on the type of scenario and resources available, running these tests locally in a personal computer can be very resource intensive.

Note:

Most test runs take between 20 and 40 minutes depending on system resources

The command should bring up the machines needed for the test, provision them with ceph-ansible, run the tests, and tear the whole environment down at the end.

The output would look something similar to this trimmed version:

```
luminous-ansible2.4-xenial_cluster create: /Users/alfredo/python/upstream/ceph-ansible
luminous-ansible2.4-xenial_cluster installdeps: ansible==2.4.2, -r/Users/alfredo/pytho
luminous-ansible2.4-xenial_cluster runtests: commands[0] | vagrant up --no-provision -
Bringing machine 'client0' up with 'virtualbox' provider...
Bringing machine 'rgw0' up with 'virtualbox' provider...
```

```
Bringing machine 'mds0' up with 'virtualbox' provider...
Bringing machine 'mon0' up with 'virtualbox' provider...
Bringing machine 'mon1' up with 'virtualbox' provider...
Bringing machine 'mon2' up with 'virtualbox' provider...
Bringing machine 'osd0' up with 'virtualbox' provider...
...
```

After all the nodes are up, ceph-ansible will provision them, and run the playbook(s):

```
...
PLAY RECAP *********************************************************************
client0                    : ok=4    changed=0    unreachable=0    failed=0
mds0                       : ok=4    changed=0    unreachable=0    failed=0
mon0                       : ok=4    changed=0    unreachable=0    failed=0
mon1                       : ok=4    changed=0    unreachable=0    failed=0
mon2                       : ok=4    changed=0    unreachable=0    failed=0
osd0                       : ok=4    changed=0    unreachable=0    failed=0
rgw0                       : ok=4    changed=0    unreachable=0    failed=0
...
```

Once the whole environment is all running the tests will be sent out to the hosts, with output similar to this:

```
luminous-ansible2.4-xenial_cluster runtests: commands[4] | testinfra -n 4 --sudo -v --
============================= test session starts ==============================
platform darwin -- Python 2.7.8, pytest-3.0.7, py-1.4.33, pluggy-0.4.0 -- /Users/alfre
cachedir: ../../../../.cache
rootdir: /Users/alfredo/python/upstream/ceph-ansible/tests, inifile: pytest.ini
plugins: testinfra-1.5.4, xdist-1.15.0
[gw0] darwin Python 2.7.8 cwd: /Users/alfredo/python/upstream/ceph-ansible/tests/funct
[gw1] darwin Python 2.7.8 cwd: /Users/alfredo/python/upstream/ceph-ansible/tests/funct
[gw2] darwin Python 2.7.8 cwd: /Users/alfredo/python/upstream/ceph-ansible/tests/funct
[gw3] darwin Python 2.7.8 cwd: /Users/alfredo/python/upstream/ceph-ansible/tests/funct
[gw0] Python 2.7.8 (v2.7.8:ee879c0ffa11, Jun 29 2014, 21:07:35)  -- [GCC 4.2.1 (Apple
[gw1] Python 2.7.8 (v2.7.8:ee879c0ffa11, Jun 29 2014, 21:07:35)  -- [GCC 4.2.1 (Apple
[gw2] Python 2.7.8 (v2.7.8:ee879c0ffa11, Jun 29 2014, 21:07:35)  -- [GCC 4.2.1 (Apple
[gw3] Python 2.7.8 (v2.7.8:ee879c0ffa11, Jun 29 2014, 21:07:35)  -- [GCC 4.2.1 (Apple
gw0 [154] / gw1 [154] / gw2 [154] / gw3 [154]
scheduling tests via LoadScheduling

../../../tests/test_install.py::TestInstall::test_ceph_dir_exists[ansible:/mon0]
../../../tests/test_install.py::TestInstall::test_ceph_dir_is_a_directory[ansible:/mon
../../../tests/test_install.py::TestInstall::test_ceph_conf_is_a_file[ansible:/mon0]
../../../tests/test_install.py::TestInstall::test_ceph_dir_is_a_directory[ansible:/mon
[gw2] PASSED ../../../tests/test_install.py::TestCephConf::test_ceph_config_has_mon_ho
../../../tests/test_install.py::TestInstall::test_ceph_conf_exists[ansible:/mon1]
[gw3] PASSED ../../../tests/test_install.py::TestCephConf::test_mon_host_line_has_corr
../../../tests/test_install.py::TestInstall::test_ceph_conf_is_a_file[ansible:/mon1]
[gw1] PASSED ../../../tests/test_install.py::TestInstall::test_ceph_command_exists[ans
../../../tests/test_install.py::TestCephConf::test_mon_host_line_has_correct_value[ans
...
```

Finally the whole environment gets torn down:

```
luminous-ansible2.4-xenial_cluster runtests: commands[5] | vagrant destroy --force
==> osd0: Forcing shutdown of VM...
==> osd0: Destroying VM and associated drives...
==> mon2: Forcing shutdown of VM...
==> mon2: Destroying VM and associated drives...
==> mon1: Forcing shutdown of VM...
==> mon1: Destroying VM and associated drives...
==> mon0: Forcing shutdown of VM...
==> mon0: Destroying VM and associated drives...
==> mds0: Forcing shutdown of VM...
==> mds0: Destroying VM and associated drives...
==> rgw0: Forcing shutdown of VM...
==> rgw0: Destroying VM and associated drives...
==> client0: Forcing shutdown of VM...
```

```
    ==> client0: Destroying VM and associated drives...
```

And a brief summary of the scenario(s) that ran is displayed:

```
    _____________________________________________ summary ______________________________
      luminous-ansible2.4-xenial_cluster: commands succeeded
      congratulations :)
```

ceph-ansible testing for development

Test Scenarios

Scenarios are distinct environments that describe a Ceph deployment and configuration. Scenarios are isolated as well, and define what machines are needed aside from any ceph-ansible configuration.

Scenario Files

The scenario is described in a `vagrant_variables.yml` file, which is consumed by `Vagrant` when bringing up an environment.

This yaml file is loaded in the `Vagrantfile` so that the settings can be used to bring up the boxes and pass some configuration to ansible when running.

> **Note:**
>
> The basic layout of a scenario is covered in Layout and conventions There are just a handful of required files, this is the most basic layout.

There are just a handful of required files, these sections will cover the required (most basic) ones. Alternatively, other ceph-ansible files can be added to customize the behavior of a scenario deployment.

`vagrant_variables.yml`

There are a few sections in the `vagrant_variables.yml` file which are easy to follow (most of them are 1 line settings).

- **docker**: (bool) Indicates if the scenario will deploy docker daemons
- **VMS**: (int) These integer values are just a count of how many machines will be needed. Each supported type is listed, defaulting to 0:

  ```
  mon_vms: 0
  osd_vms: 0
  mds_vms: 0
  rgw_vms: 0
  nfs_vms: 0
  rbd_mirror_vms: 0
  client_vms: 0
  iscsi_gw_vms: 0
  mgr_vms: 0
  ```

  For a deployment that needs 1 MON and 1 OSD, the list would look like:

  ```
  mon_vms: 1
  osd_vms: 1
  ```

- **RESTAPI**: (bool) Deploy RESTAPI on each of the monitor(s) restapi: true
- **CEPH SOURCE**: (string) indicate whether a `dev` or `stable` release is needed. A `stable` release will use the latest stable release of Ceph, a `dev` will use `shaman` (http://shaman.ceph.com)
- **SUBNETS**: These are used for configuring the network availability of each server that will be booted as well as being used as configuration for ceph-ansible (and eventually ceph). The two values that are **required**:

  ```
  public_subnet: 192.168.13
  cluster_subnet: 192.168.14
  ```

- **MEMORY**: Memory requirements (in megabytes) for each server, e.g. `memory: 512`
- **interfaces**: some vagrant boxes (and linux distros) set specific interfaces. For Ubuntu releases older than Xenial it was common to have `eth1`, for CentOS and some Xenial boxes `enp0s8` is used. **However** the public Vagrant boxes normalize the interface to `eth1` for all boxes, making it easier to configure them with Ansible later.

> **Warning:**
>
> Do *not* change the interface from `eth1` unless absolutely certain that is needed for a box. Some tests that depend on that naming will fail.

- **disks**: The disks that will be created for each machine, for most environments `/dev/sd*` style of disks will work, like: `[ '/dev/sda', '/dev/sdb' ]`
- **vagrant_box**: We have published our own boxes to normalize what we test against. These boxes are published in Atlas (https://atlas.hashicorp.com/ceph/). Currently valid values are: `ceph/ubuntu-xenial`, and `ceph/centos7`

The following aren't usually changed/enabled for tests, since they don't have an impact, however they are documented here for general knowledge in case they are needed:

- **ssh_private_key_path** : The path to the `id_rsa` (or other private SSH key) that should be used to connect to these boxes.
- **vagrant_sync_dir** : what should be "synced" (made available on the new servers) from the host.
- **vagrant_disable_synced_folder** : (bool) when disabled, it will make booting machines faster because no files need to be synced over.
- **os_tuning_params** : These are passed onto ceph-ansible as part of the variables for "system tunning". These shouldn't be changed.

Vagrantfile

The `Vagrantfile` should not need to change, and it is symlinked back to the `Vagrantfile` that exists in the root of the project. It is linked in this way so that a vagrant environment can be isolated to the given scenario.

hosts

The `hosts` file should contain the hosts needed for the scenario. This might seem a bit repetitive since machines are already defined in vagrant_variables.yml but it allows granular changes to hosts (for example defining an interface vs. an IP on a monitor) which can help catch issues in ceph-ansible configuration. For example:

```
[mons]
mon0 monitor_address=192.168.5.10
mon1 monitor_address=192.168.5.11
mon2 monitor_interface=eth1
```

group_vars

This directory holds any configuration change that will affect ceph-ansible deployments in the same way as if ansible was executed from the root of the project.

The file that will need to be defined always is `all` where (again) certain values like `public_network` and `cluster_network` will need to be defined along with any customizations that ceph-ansible supports.

Scenario Wiring

Scenarios are just meant to provide the Ceph environment for testing, but they do need to be defined in the `tox.ini` so that they are available to the test framework. To see a list of available scenarios, the following command (ran from the root of the project) will list them, shortened for brevity:

```
    $ tox -l
    ...
    luminous-ansible2.4-centos7_cluster
    ...

These scenarios are made from different variables, in the above command there
are 3:
```

- jewel: the Ceph version to test
- ansible2.4: the Ansible version to install
- `centos7_cluster`: the name of the scenario

The last one is important in the *wiring up* of the scenario. It is a variable that will define in what path the scenario lives. For example, the `changedir` section for `centos7_cluster` that looks like:

```
centos7_cluster: {toxinidir}/tests/functional/centos/7/cluster
```

The actual tests are written for specific daemon types, for all daemon types, and for specific use cases (e.g. journal collocation), those have their own conventions as well which are explained in detail in Conventions and Test Files.

As long as a test scenario defines OSDs and MONs, the OSD tests and MON tests will run.

Conventions

Environment configuration

Ansible configuration

Modifying (or adding) tests

Layout and conventions

Test files and directories follow a few conventions, which makes it easy to create (or expect) certain interactions between tests and scenarios.

All tests are in the `tests` directory. Scenarios are defined in `tests/functional/` and use the following convention for directory structure:

```
tests/functional/<distro>/<distro version>/<scenario name>/
```

For example: `tests/functional/centos/7/journal-collocation`

Within a test scenario there are a few files that define what that specific scenario needs for the tests, like how many OSD nodes or MON nodes. Tls

At the very least, a scenario will need these files:

- `Vagrantfile`: must be symlinked from the root directory of the project
- `hosts`: An Ansible hosts file that defines the machines part of the cluster
- `group_vars/all`: if any modifications are needed for deployment, this would override them. Additionally, further customizations can be done. For example, for OSDs that would mean adding `group_vars/osds`
- `vagrant_variables.yml`: Defines the actual environment for the test, where machines, networks, disks, linux distro/version, can be defined.

Conventions

Python test files (unlike scenarios) rely on paths to *map* where they belong. For example, a file that should only test monitor nodes would live in `ceph-ansible/tests/functional/tests/mon/`. Internally, the test runner (`py.test`) will *mark* these as tests that should run on a monitor only. Since the configuration of a scenario already defines what node has a given role, then it is easier for the system to only run tests that belong to a particular node type.

The current convention is a bit manual, with initial path support for:

- mon
- osd
- mds
- rgw
- journal_collocation
- all/any (if none of the above are matched, then these are run on any host)

testinfra

Tests

Actual tests are written in Python methods that accept optional fixtures. These fixtures come with interesting attributes to help with remote assertions.

As described in Conventions, tests need to go into `tests/functional/tests/`. These are collected and *mapped* to a distinct node type, or *mapped* to run on all nodes.

Simple Python asserts are used (these tests do not need to follow the Python `unittest.TestCase` base class) that make it easier to reason about failures and errors.

The test run is handled by `py.test` along with testinfra for handling remote execution.

Test Files

Test Fixtures

Test fixtures are a powerful feature of `py.test` and most tests depend on this for making assertions about remote nodes. To request them in a test method, all that is needed is to require it as an argument.

Fixtures are detected by name, so as long as the argument being used has the same name, the fixture will be passed in (see pytest fixtures for more in-depth examples). The code that follows shows a test method that will use the `node` fixture that contains useful information about a node in a ceph cluster:

```
    def test_ceph_conf(self, node):
        assert node['conf_path'] == "/etc/ceph/ceph.conf"
```

The test is naive (the configuration path might not exist remotely) but explains how simple it is to “request” a fixture.

For remote execution, we can rely further on other fixtures (tests can have as many fixtures as needed) like `File`:

```
    def test_ceph_config_has_inital_members_line(self, node, File):
        assert File(node["conf_path"]).contains("^mon initial members = .*$")
```

node fixture

The `node` fixture contains a few useful pieces of information about the node where the test is being executed, this is captured once, before tests run:

- `address`: The IP for the `eth1` interface
- `subnet`: The subnet that `address` belongs to
- `vars`: all the ansible vars set for the current run
- `osd_ids`: a list of all the OSD IDs
- `num_mons`: the total number of monitors for the current environment
- `num_devices`: the number of devices for the current node
- `num_osd_hosts`: the total number of OSD hosts
- `total_osds`: total number of OSDs on the current node
- `cluster_name`: the name of the Ceph cluster (which defaults to ‘ceph’)
- `conf_path`: since the cluster name can change the file path for the Ceph configuration, this gets sets according to the cluster name.
- `cluster_address`: the address used for cluster communication. All environments are set up with 2 interfaces, 1 being used exclusively for the cluster
- `docker`: A boolean that identifies a Ceph docker cluster

- `osds`: A list of OSD IDs, unless it is a docker cluster, where it gets the name of the devices (e.g. `sda1`)

Other Fixtures

There are a lot of other fixtures provided by testinfra as well as `py.test`. The full list of `testinfra` fixtures are available in testinfra_fixtures

`py.test` builtin fixtures can be listed with `pytest -q --fixtures` and they are described in pytest builtin fixtures

tox

`tox` is an automation project we use to run our testing scenarios. It gives us the ability to create a dynamic matrix of many testing scenarios, isolated testing environments and a provides a single entry point to run all tests in an automated and repeatable fashion.

Documentation for tox can be found here.

Environment variables

When running `tox` we've allowed for the usage of environment variables to tweak certain settings of the playbook run using Ansible's `--extra-vars`. It's helpful in Jenkins jobs or for manual test runs of ceph-ansible.

The following environent variables are available for use:

- `FETCH_DIRECTORY` : (default: `changedir`) This would configure the ceph-ansible variable `fetch_directory`. This defaults to the `changedir` of the given scenario and should not need to be changed.
- `CEPH_STABLE_RELEASE`: (default: `jewel`) This would configure the ceph-ansible variable `ceph_stable_relese`. This is set automatically when using the `jewel-*` or `kraken-*` testing scenarios.
- `UPDATE_CEPH_STABLE_RELEASE`: (default: `kraken`) This would configure the ceph-ansible variable `ceph_stable_relese` during an `update` scenario. This is set automatically when using the `jewel-*` or `kraken-*` testing scenarios.
- `CEPH_DOCKER_REGISTRY`: (default: `docker.io`) This would configure the ceph-ansible variable `ceph_docker_registry`.
- `CEPH_DOCKER_IMAGE`: (default: `ceph/daemon`) This would configure the ceph-ansible variable `ceph_docker_image`.
- `CEPH_DOCKER_IMAGE_TAG`: (default: `latest`) This would configure the ceph-ansible variable `ceph_docker_image_name`.
- `CEPH_DEV_BRANCH`: (default: `master`) This would configure the ceph-ansible variable `ceph_dev_branch` which defines which branch we'd like to install from shaman.ceph.com.
- `CEPH_DEV_SHA1`: (default: `latest`) This would configure the ceph-ansible variable `ceph_dev_sha1` which defines which sha1 we'd like to install from shaman.ceph.com.
- `UPDATE_CEPH_DEV_BRANCH`: (default: `master`) This would configure the ceph-ansible variable `ceph_dev_branch` which defines which branch we'd like to update to from shaman.ceph.com.
- `UPDATE_CEPH_DEV_SHA1`: (default: `latest`) This would configure the ceph-ansible variable `ceph_dev_sha1` which defines which sha1 we'd like to update to from shaman.ceph.com.

Sections

The `tox.ini` file has a number of top level sections defined by `[ ]` and subsections within those. For complete documentation on all subsections inside of a tox section please refer to the tox documentation.

- `tox` : This section contains the `envlist` which is used to create our dynamic matrix. Refer to the section here for more information on how the `envlist` works.
- `purge` : This section contains commands that only run for scenarios that purge the cluster and redeploy. You'll see this section being reused in `testenv` with the following syntax: `{[purge]commands}`
- `update` : This section contains commands taht only run for scenarios that deploy a cluster and then upgrade it to another ceph version.
- `testenv` : This is the main section of the `tox.ini` file and is run on every scenario. This section contains many *factors* that define conditional settings depending on the scenarios defined in the `envlist`. For example, the factor `centos7_cluster` in the

`changedir` subsection of `testenv` sets the directory that tox will change do when that factor is selected. This is an important behavior that allows us to use the same `tox.ini` and reuse commands while tweaking certain sections per testing scenario.

Modifying or Adding environments

The tox environments are controlled by the `envlist` subsection of the `[tox]` section. Anything inside of `{}` is considered a *factor* and will be included in the dynamic matrix that tox creates. Inside of `{}` you can include a comma separated list of the *factors*. Do not use a hyphen (`-`) as part of the *factor* name as those are used by tox as the separator between different factor sets.

For example, if wanted to add a new test *factor* for the next ceph release of luminious this is how you'd accomplish that. Currently, the first factor set in our `envlist` is used to define the ceph release (`{jewel,kraken,rhcs}-...`). To add luminous you'd change that to look like `{luminous,kraken,rhcs}-...`. In the `testenv` section this is a subsection called `setenv` which allows you to provide environment variables to the tox environment and we support an environment variable called `CEPH_STABLE_RELEASE`. To ensure that all the new tests that are created by adding the luminous *factor* you'd do this in that section: `luminous: CEPH_STABLE_RELEASE=luminous`.