bareon Documentation

Release

OpenStack Foundation

February 24, 2016

Contents

CHAPTER 1

Bareon functional testing

https://blueprints.launchpad.net/bareon/+spec/bareon-functional-testing

1.1 Problem description

Currently there are no functional tests in bareon. Tests that would cover full on-metal provisioning and a result - actual partition scheme / image applied to the node

1.2 Proposed change

NOTE: This is a contribution of the feature developed within Cray OpenStack project. We will try to make a minimum changes to existing code.

We are adding the new framework "bareon-func-test" to allow writing such kind of tests. This is a virsh-based tool that allows to walk through the full provisioning cycle and read the results from the node at each step. An overview of the framework can be found at [1]

The framework will reside in a separate repo called "bareon-func-test".

Functional tests, as well as node templates, provision.json templates (if any) are stored in the bareon tree (bareon/tests_functional/). The main idea behind this is that you can commit the new piece of functionality together with funcitonal tests covering it, and it's easier to track in CI.

We are also adding a set of tests covering:

- partitioning operations on Ironic data driver
- lvm operations on Ironic data driver
- full provisioning on both swift/rsync deploy drivers and Ironic data driver.

Since the framework requires a ramdisk to run, test integration also includes a job to build a ramdisk from the current repo, and a tox env that manages build/test-run.

1.2.1 Alternatives

None.

1.3 Implementation

1.3.1 Assignee(s)

- max_lobur

1.3.2 Milestones

See blueprint ref above.

1.3.3 Work Items

- rebase onto Bareon master.

1.4 Dependencies

- rsync image deployment

1.5 Links

[1] http://www.slideshare.net/MaxLobur/bareon-functional-testing-ci-58066411

CHAPTER 2

Multi image deployment

https://blueprints.launchpad.net/bareon/+spec/multi-image-deployment

2.1 Problem description

Currently Bareon allows to deploy only one bootable image, other are 'utils' images. It's impossible to deploy two images and switch between them. This blocks a very useful use-case, where you can do a live update/modification of the system with a downtime to one reboot only.

2.2 Proposed change

NOTE: This is a contribution of the feature developed within Cray OpenStack project. We will try to make a minimum changes to existing code.

Requirements: - It should be possible to deploy multiple, co-resident images to a node. - It should be possible to set the default boot partition of a node/instance. - It should be possible to list valid bootable partitions. - It should be possible to switch between valid bootable partitions on a node.

The 'images' attribute of the provision.json schema in Ironic driver is extended to the following:

```
{
    "images": [
        {
            "name": "centos",
            "boot": true,
            "image_name": "centos-7.1.1503",
            "image_uuid": "2a86b00d-cfa4-49d9-a008-13c7940ed02d",
            "image_pull_url": "http://10.211.55.8:8080/v1/AUTH_319...",
            "target": "/"
        },
        {
            "name": "ubuntu",
            "boot": false,
            "image_name": "ubuntu",
            "image_uuid": "157636d8-62ad-499d-aecc-2ea4917ee396",
            "image_pull_url": "http://10.211.55.8:8080/v1/AUTH_319...",
            "target": "/"
        }
    ],
}
```

All the elements that have mount point in provision.json schema partitions (e.g. *partition* and *lv*) are extended to include the new attribute **'images'**. It will determine the set of images this partition belongs to. It is a list value like ["os 1", "os 2"] that holds image names. In different OSs, mount points may overlap. For example it will allow to define "/" for ["os 1"] and "/" for ["os 2"], while allow them have the same "/usr/share/utils" (e.g. ["os 1", "os 2"]). The attribute is optional, and by default the partition belongs to the first image in deploy_config. Fstab is created basing on this mapping as well. At the end, we will do a single grub install so it takes all the available OSs and allow you to choose which one to boot.

For example:

```
"volumes": [
    {
       "mount": "/",
       "images": [
           "ubuntu 14.04"
       ],
       "type": "partition",
       "file_system": "ext4",
       "size": "4000"
    },
    {
       "mount": "/",
       "images": [
           "centos 7.1"
       ],
       "type": "partition",
       "file_system": "ext4",
       "size": "5000"
    },
    {
       "mount": "/usr/share/common",
       "images": [
           "centos 7.1",
           "ubuntu 14.04"
       ],
       "type": "partition",
       "file_system": "ext4",
       "size": "10000"
    }
```

2.2.1 Deploy flow:

Schema passed to bareon will have one more implicit partition: a partition with "mount": "multiboot" is a 100 Mb partition used for grub installation. It is added to the first disk referenced in schema. It is not mounted into the images. Instead grub.cfg there refers kernels/ramdisks which reside at each image's boot dir. They are detected by os-proper. The flow is the following:

- Mount all partitions/lvs linked with "ubuntu 14.04" (/boot can't be separate partition, otherwise will be skipped by os-prober).
- Deploy "ubuntu 14.04" (rsync or swift)
- Create fstab for "ubuntu 14.04" basing on partition<->image mapping
- Unmount all
- Mount all partitions/lvs linked with "centos 7.1" (/boot can't be separate partition, otherwise will be skipped by os-prober).

- Deploy “centos 7.1” (rsync or swift)
- Create fstab for “centos 7.1” basing on partition<->image mapping.
- Unmount all
- Mount multiboot partition.
- Run grub install with os-prober
- Run grub mkconfig
- Unmount all
- Shut down the node.
- The disk where ‘multiboot’ partition resides is marked as bootable device in BIOS.
- Turn on the node.

After the deployment, bareon will write found images to a separate file, /tmp/boot-info.json. Example below:

```
{
    u'elements': [
        {
            u'grub_id': 0,
            u'image_name': u'centos-7.1.1503',
            u'os_id': u'centos',
            u'image_uuid': u'2a86b00d-cfa4-49d9-a008-13c7940ed02d',
            u'boot_name': u'CentOSLinuxrelease7.1.1503(Core)(on/dev/vda3)',
            u'root_uuid': u'2abf123d-d52f-4f62-a351-7358221bc51f'
        },
        {
            u'grub_id': 2,
            u'image_name': u'ubuntu',
            u'os_id': u'ubuntu',
            u'image_uuid': u'157636d8-62ad-499d-aecc-2ea4917ee396',
            u'boot_name': u'Ubuntu14.04.3LTS(14.04)(on/dev/vda4)',
            u'root_uuid': u'688c5f1e-dc46-4aca-a90e-be21ba8aa3e2'
        }
    ],
    u'current_element': 0,
    u'multiboot_partition': u'3b360901-7896-48d4-a14d-fc35e1582c74'
}
```

This json can be pulled out of the ramdisk and used for further management. The multiboot_partition attribute holds a UUID of the implicit partition, where grub.cfg is written. Any image can mount this partition and switch grub default index, which will lead node to boot another image after the next power cycle.

2.2.2 Alternatives

None.

2.3 Implementation

2.3.1 Assignee(s)

- max_lobur

2.3.2 Milestones

See blueprint ref above.

2.3.3 Work Items

- rebase onto Bareon master.

2.4 Dependencies

Code is rebased on the following patches:

- Rsync image deployment
- Functional tests
- Split deploy driver
- Policy-based partitioning

thus needs to be proposed after these.

CHAPTER 3

Pluggable architecture for bareon

https://blueprints.launchpad.net/bareon/+spec/pluggable-do-actions

At the current state bareon is monolitic tool. If one wants to add new drivers or do actions, then those changes should be landed into bareon's repo first. There's no convenient way to develop something as a 3rd party component or re-use actual codebase as a framework. That's why we need to introduce pluggable architecture.

3.1 Problem description

Currently, there're few flaws in current bareon architecture:

- It's purely monolitic tool. It can't be split to few tools like bareon-base, bareon-provisioning, bareon-image-building, bareon-partitioning. The idea is to provide the way to separate these few stages like building OS root file system and putting this root file system on a node. It is easy to imagine when a user wants to install OS building root file system directly on a disk.
- Like it was said above, there's no convenient way to develop out of core. This is the main stopper for external contributors.
- It's hard to introduce new functionality or re-use existing one.
- It glues few do_actions into single combo and prevents from running actions separately. For example, run partitioning without provisioning.
- Current architecture is not expandable (maybe even maintainable). Pluggable extensions is one of the ways to resolve the problem.

Therefore, pluggable architecture is really necessary.

3.2 Proposed change

Manager's do actions should be re-introduced as pluggable extensions. Therefore new directory for actions will be created:

    bareon/actions/

Manager will be improved in order to work with these pluggable do actions. At least, manager should know how to check the presence of pluggable extensions and validate that input data already contains all necessary information for being processed later.

stevedore will be used as a convenient extension manager.

Moving towards with data-driven approach, the base class for all drivers will have only one method used only for initializing the data. Drivers still be free to implement any types of collections (schemes) of objects they needed.

Regarding data intersections. Sometimes objects are the result of an action. For example when we want to use cloud-init with configdrive for provisioning, we should somehow tell the driver responsible for partitioning to reserve the space for config drive. In order to resolve that:

- All objects are shared among multiple do actions
- These objects could be modified in run-time inside any of do action
- Every do action should have a validate method for these objects, which will asure us that provided data already satisfied all necessary requirements. For example, configdrive action should expect special type of partition to be created by partitioning action. Manager that performs partitioning action, in turn, will check that configdrive action queued and create the partition and reflect that in the objects.

3.2.1 Alternatives

The other ways are:

- Just make every do action independent from others, but then the node we want to only provision will have to have the dependencies for e.g. partitioning also. And it won't be expendable for Users.
- Create separate project for every usecase, but there will be lots of code duplications and hard to maintain.

We could use other plugin system like:

- PluginBase
- Yapsy

But they're not part of openstack' ecosystem. The idea is that having all projects use the same approach is more important than the objections to the approach. Sharing code between projects is great, but by also having projects use the same idioms for stuff like this it makes it much easier for people to work on multiple projects. Therefore, stevedore is the essential choice.

3.3 Implementation

3.3.1 Assignee(s)

Primary assignee: Alexander Gordeev

Mandatory Design Reviewers: Evgeny Li Vladimir Kozhukalov

3.3.2 Milestones

Target Milestone for completion: 1.0.0

3.3.3 Work Items

- Introduce pluggable extension.
- Rework existing do actions according to the new architecture.

3.4 Dependencies

Doesn't require new dependencies as stevedore has been already included into dependencies.

CHAPTER 4

Policy-based partitioning

https://blueprints.launchpad.net/bareon/+spec/policy-based-partitioning

4.1 Problem description

We need the ability to verify the disposition of existing partitions during deployment. Bareon agent should support the prevention of modification of existing partitions during deployment when requested by the user or administrator.

4.2 Proposed change

NOTE: This change is added with a standalone "Ironic" data driver.

NOTE: This is a contribution of the feature developed within Cray OpenStack project. We will try to make a minimum changes to existing code.

We introduce the new attribute in provision.json, called partitions_policy. This policy will control the way of how partitions are applied. The partitions_policy, working together with "keep_data" attribute of partition/lv will control the data retention. The policy can be either:

- **verify**
  - Do verification: Compare partitions schema with existing partitions on the disk(disks). If there is an additional disk, not mentioned in schema (unknown) - it is ignored.
  - Do partitioning: partitions which do not have "keep_data" attribute are left as is (keep_data attribute is true by default); partitions which have "keep_data":false attribute in schema are wiped;
- **clean**

  Ignore existing partitions on the disk(disks). Clean the disk and create partitions according to the schema. If there is an additional disk, not mentioned in schema (unknown) - it is ignored.

For example:

```
{
  "partitions_policy": "<verify|clean>",
  "partitions": [{
         "type":"disk",
         "id": ...,
         "volumes":[
            {
               "type":"partition",
```

```
                "id": ...,
                 # "keep_data": true - by default
            },
            {
                "id": ...,
                "keep_data": false, # this partition will be wiped
                # (explicitly specified keep_data false here)
            },
        ],
}
```

4.2.1 Alternatives

There's already a keep_data flag in Nailgun driver, however it is impossible to verify HW partitions. So no alternatives to the current proposal.

4.3 Implementation

4.3.1 Assignee(s)

- max_lobur

4.3.2 Milestones

See blueprint ref above.

4.3.3 Work Items

- Rebase onto Bareon master

4.4 Dependencies

The code of this feature is on top of the following patches:

- rsync image deployment
- functional tests

thus needs to be prosed after these.

CHAPTER 5

Rsync image deployment

https://blueprints.launchpad.net/bareon/+spec/rsync-image-deployment

5.1 Problem description

Current version of bareon agent deploys an image on block-device level. Thus an image can be deployed only to a single partition. Usually the provided image is a single partition, however it has dirs like /usr or /var inside, that could map to a provided partition schema. To achieve this on the current agent we would need to upload multiple images - one per partition.

5.2 Proposed change

NOTE: This is a contribution of the feature developed within Cray OpenStack project. We are trying to make as minimum changes to existing code as possible.

We propose to use rsync to transfer the image to a baremetal node. Rsync server might be an Ironic Conductor (If Ironic is used), or any other calling server accessible from the baremetal node. Rsync does file-level copying, thus allows to deploy an image across partitions. This would also allow incremental image-updates.

To achieve this we are splitting the new kind of driver - deploy driver: provision –data-driver <ironic|nailgun> –deploy_driver <swift|rsync>

Deploy driver is basically a manager (on the current code base) converted to a driver. Abstract driver would include:

```
@abc.abstractmethod
def do_partitioning(self):

@abc.abstractmethod
def do_configdrive(self):

@abc.abstractmethod
def do_copyimage(self):

@abc.abstractmethod
def do_reboot(self):

@abc.abstractmethod
def do_provisioning(self):
```

And every deploy driver will add their own implementation of these. Currently we are moving most of the code to base driver, differentiating only do_copyimage method (rsync VS swift).

5.2.1 Alternatives

None.

5.3 Implementation

5.3.1 Assignee(s)

- max_lobur

5.3.2 Milestones

See blueprint ref above.

5.3.3 Work Items

- rebase onto Bareon master.

5.4 Dependencies

None.

CHAPTER 6

Size unit conversion and relative sizing

https://blueprints.launchpad.net/bareon/+spec/size-unit-conversion-and-relative-sizing

6.1 Problem description

The only size unit Bareon currently supports is MiB. Users often want to use both GiB and MiB, as well as relative sizes like 50%, which is impossible.

6.2 Proposed change

NOTE: This is a contribution of the feature developed within Cray OpenStack project. We will try to make a minimum changes to existing code.

All "size" values are strings containing either an integer number and size unit (e.g., "100 MiB" or 100MiB").

Available measurement units are:

- 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB',
- 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB'

Also relative values are supported for partition, pv and lv. Relative values use the size of the containing device or volume group as a base. For example, specifying "40%" for a 100MiB disk would result in a 40MiB partition. Obviously, relative sizes cannot be used for disks.

The user can also specify "remaining" as a size value for a volume in a disk or in a volume group. When "remaining" is specified, all remaining free space on the drive after allocations are made for all other volumes will be used for this volume.

All the conversion is done in Ironic data driver, before the schema is mapped to object model. Internally we continue to use MiB everywhere.

6.2.1 Alternatives

None.

6.3 Implementation

6.3.1 Assignee(s)

- max_lobur

6.3.2 Milestones

See blueprint ref above.

6.3.3 Work Items

- rebase on bareon master.

6.4 Dependencies

None.

CHAPTER 7

bareon Repository Information

7.1 bareon

Bareon provides flexible and data driven interface to perform actions which are related to operating system installation

- Free software: Apache license
- Documentation: http://docs.openstack.org/developer/bareon

7.1.1 Features

- TODO

7.2 Contributing to: bareon

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

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

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

> https://bugs.launchpad.net/bareon

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*