

## 目 录

<b>1</b>	<b>前言</b>	<b>2</b>
<b>2</b>	<b>snapshot 的流程</b>	<b>2</b>
2.1	快照入口函数 . . . . .	2
2.2	使用 RPC 与 nova-compute 服务通信 . . . . .	3
2.3	nova-compute 中的快照动作 . . . . .	4
2.4	实现快照功能的核心函数 . . . . .	5
<b>3</b>	<b>snapshot 优化方案</b>	<b>8</b>
3.1	最基本的情况 . . . . .	8
3.1.1	最基本的情况的描述 . . . . .	8
3.1.2	入口函数的改动 . . . . .	8
3.1.3	修改 RPC 与 nova-compute 服务通信部分的代码 . . . . .	10
3.1.4	在 nova-compute 中修改相应的快照动作 . . . . .	11
3.1.5	新增加一个核心函数 snapshot_overlay . . . . .	12
3.1.6	在 virt/libvirt/utils.py 创建新函数 . . . . .	15
3.2	镜像链为 3 的情况 . . . . .	16
3.2.1	镜像链为 3 的情况的描述 . . . . .	16
3.2.2	核心函数的修改 . . . . .	16

# 1 前言

在提出 snapshot 优化方案之前，我先分析一下 snapshot 的流程，以便之后优化快照时思路更加清晰。

## 2 snapshot 的流程

### 2.1 快照入口函数

```
1 # nova/api/openstack/compute/servers.py Controller._action_create_image()
2 def _action_create_image(self, req, id, body):
3     # id是instance id
4     context = req.environ['nova.context']
5     # body == {u'createImage': {u'name': u'snap1', u'metadata': {}}}
6     entity = body.get("createImage", {})
7
8     image_name = entity.get("name")
9
10    ...
11
12    props = {}
13    metadata = entity.get('metadata', {})
14
15    try:
16        props.update(metadata)
17        ...
18        # 根据context和req得到Instance类
19        instance = self._get_server(context, req, id)
20
21        bdms = objects.BlockDeviceMappingList.get_by_instance_uuid(
22            context, instance.uuid)
23
24        try:
25            # 判断root分区是否是volume
26            if self.compute_api.is_volume_backed_instance(context, instance,
27                                                         bdms):
28                img = instance['image_ref']
29                if not img:
30                    properties = bdms.root_metadata(
31                        context, self.compute_api.image_api,
32                        self.compute_api.volume_api)
33                    image_meta = {'properties': properties}
34                else:
35                    image_meta = self.compute_api.image_api.get(context, img)
36
37                # Snapshot the given volume-backed instance
38                image = self.compute_api.snapshot_volume_backed(
39                    context,
40                    instance,
41                    image_meta,
```

```

42                                     image_name,
43                                     extra_properties=props)
44     else:
45         # 做快照的正常流程
46         image = self.compute_api.snapshot(context,
47                                           instance,
48                                           image_name,
49                                           extra_properties=props)
50     ...
51
52     # 以下代码的功能: build location of newly-created image entity
53     image_id = str(image['id'])
54     url_prefix = self._view_builder._update_glance_link_prefix(
55         req.application_url)
56     image_ref = os.path.join(url_prefix,
57                             context.project_id,
58                             'images',
59                             image_id)
60
61     resp = webob.Response(status_int=202)
62     resp.headers['Location'] = image_ref
63     return resp

```

## 2.2 使用 RPC 与 nova-compute 服务通信

```

1     # nova/compute/api.py API.snapshot()
2     def snapshot(self, context, instance, name, extra_properties=None):
3         # instance: nova.db.sqlalchemy.models.Instance
4         # name: name of the snapshot
5         # extra_properties: dict of extra image properties to include
6         #                     when creating the image.
7         # returns: A dict containing image metadata
8
9         # 函数功能: Create new image entry in the image service.
10        #           This new image will be reserved for the compute
11        #           manager to upload a snapshot or backup.
12        image_meta = self._create_image(context, instance, name,
13                                       'snapshot',
14                                       extra_properties=extra_properties)
15
16        # 更改instance的task_state
17        instance.task_state = task_states.IMAGE_SNAPSHOT_PENDING
18        # 调用Instance.save()方法更改数据库
19        instance.save(expected_task_state=[None])
20
21        self.compute_rpcapi.snapshot_instance(context, instance,
22                                              image_meta['id'])
23
24        return image_meta

```

```

1     # nova/compute/rpcapi.py ComputeAPI.snapshot_instance()
2     def snapshot_instance(self, ctxt, instance, image_id):

```

```

3      # server: the destination host for a message.
4      # server == instance['host']
5      cctx = self.client.prepare(server=_compute_host(None, instance),
6                                version=version)
7      cctx.cast(ctxt, 'snapshot_instance',
8                 instance=instance,
9                 image_id=image_id)

```

## 2.3 nova-compute 中的快照动作

```

1      # nova/compute/manager.py ComputeManager.snapshot_instance()
2      def snapshot_instance(self, context, image_id, instance):
3          # context: security context
4          # instance: a nova.objects.instance.Instance object
5          # image_id: glance.db.sqlalchemy.models.Image.Id
6          try:
7              # 修改instance的task_state
8              instance.task_state = task_states.IMAGE_SNAPSHOT
9              # 调用Instance.save()方法更改数据库
10             instance.save(
11                 expected_task_state=task_states.IMAGE_SNAPSHOT_PENDING)
12         ...
13
14         self._snapshot_instance(context, image_id, instance,
15                                task_states.IMAGE_SNAPSHOT)

```

\_snapshot\_instance() 函数如下:

```

1      # nova/compute/manager.py ComputeManager._snapshot_instance()
2      def _snapshot_instance(self, context, image_id, instance,
3                             expected_task_state):
4          # self.driver.get_info(instance)["state"]
5          current_power_state = self._get_power_state(context, instance)
6          try:
7              # 修改instance的power_state
8              instance.power_state = current_power_state
9              instance.save()
10             if instance.power_state != power_state.RUNNING:
11                 state = instance.power_state
12                 running = power_state.RUNNING
13             def update_task_state(task_state,
14                                   expected_state=expected_task_state):
15                 instance.task_state = task_state
16                 instance.save(expected_task_state=expected_state)
17             # 调用LibvirtDriver.snapshot()函数实现快照功能
18             self.driver.snapshot(context, instance, image_id,
19                                  update_task_state)
20             # 记录instance状态
21             instance.task_state = None
22             instance.save(expected_task_state=task_states.IMAGE_UPLOADING)
23         ...

```

## 2.4 实现快照功能的核心函数

```

1  def snapshot(self, context, instance, image_id, update_task_state):
2      try:
3          # 调用virConnect.lookupByName() 返回virDomain对象
4          virt_dom = self._lookup_by_name(instance['name'])
5          ...
6
7          base_image_ref = instance['image_ref']
8
9          # 得到instance的image的相关数据
10         base = compute_utils.get_image_metadata(
11             context, self._image_api, base_image_ref, instance)
12
13         # Retrieves the information record for a single disk image by image_id
14         snapshot = self._image_api.get(context, image_id)
15
16         # 通过virtDomain.XMLDesc(0)得到xml配置文件，从而得到instance的磁盘路径
17         disk_path = libvirt_utils.find_disk(virt_dom)
18         # 使用“qemu-img info disk_path”获得磁盘信息
19         source_format = libvirt_utils.get_disk_type(disk_path)
20
21         image_format = CONF.libvirt.snapshot_image_format or source_format
22
23         # NOIE(bfilippov): save lvm and rbd as raw
24         if image_format == 'lvm' or image_format == 'rbd':
25             image_format = 'raw'
26
27         # metadata = {'is_public': False,
28         #             'status': 'active',
29         #             'name': snp_name,
30         #             'properties': {
31         #                 'kernel_id': instance['kernel_id'],
32         #                 'image_location': 'snapshot',
33         #                 'image_state': 'available',
34         #                 'owner_id': instance['project_id'],
35         #                 'ramdisk_id': instance['ramdisk_id'],
36         #             }
37         #
38         metadata = self._create_snapshot_metadata(base,
39                                                     instance,
40                                                     image_format,
41                                                     snapshot['name'])
42
43         # 获得快照的名称
44         snapshot_name = uuid.uuid4().hex
45
46         # LIBVIRT_POWER_STATE = {
47         #     VIR_DOMAIN_NOSTATE: power_state.NOSTATE,
48         #     VIR_DOMAIN_RUNNING: power_state.RUNNING,
49         #     VIR_DOMAIN_BLOCKED: power_state.RUNNING,
50         #     VIR_DOMAIN_PAUSED: power_state.PAUSED,
51         #     VIR_DOMAIN_SHUTDOWN: power_state.SHUTDOWN,
52         #     VIR_DOMAIN_SHUTOFF: power_state.SHUTDOWN,
53         #     VIR_DOMAIN_CRASHED: power_state.CRASHED,

```

```

54     # VIR_DOMAIN_PMSUSPENDED: power_state.SUSPENDED,
55     # }
56     # 调用virDomain.info()返回[state, maxMemory, memory, nbVirtCPU, cpuTime]
57     state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]
58
59     # 动态快照要求QEMU 1.3 and Libvirt 1.0.0
60     # Instances with LVM encrypted ephemeral storage只支持静态快照
61     if (self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
62                               MIN_QEMU_LIVESNAPSHOT_VERSION,
63                               REQ_HYPERVISOR_LIVESNAPSHOT)
64         and source_format not in ('lvm', 'rbd')
65         and not CONF.ephemeral_storage_encryption.enabled):
66         live_snapshot = True
67     try:
68         # 终止虚拟机磁盘上的active block job
69         virt_dom.blockJobAbort(disk_path, 0)
70         ...
71     else:
72         live_snapshot = False
73
74     if state == power_state.SHUTDOWN:
75         live_snapshot = False
76
77     # virDomain.managedSave() does not work for LXC
78     # 如果是静态快照，需要执行virDomain.managedSave()函数
79     if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
80         if state == power_state.RUNNING or state == power_state.PAUSED:
81             # 卸载虚拟机的pci设备
82             self._detach_pci_devices(virt_dom,
83                                     pci_manager.get_instance_pci_devs(instance))
84             # 关闭SR-IOV端口
85             self._detach_sriov_ports(context, instance, virt_dom)
86             # This method will suspend a domain and save its memory contents to
87             # a file on disk.
88             virt_dom.managedSave(0)
89
90     # 返回Qcow2类
91     # snapshot_backend是nova.virt.libvirt.imagebackend.Qcow2 object
92     snapshot_backend = self.image_backend.snapshot(instance,
93                                                    disk_path,
94                                                    image_type=source_format)
95     ...
96
97     update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
98     # snapshot_directory == /var/lib/nova/instances/snapshots
99     snapshot_directory = CONF.libvirt.snapshots_directory
100    # 确保有这个目录存在
101    fileutils.ensure_tree(snapshot_directory)
102    with utils.tmpdir(dir=snapshot_directory) as tmpdir:
103        try:
104            # 得到快照路径
105            out_path = os.path.join(tmpdir, snapshot_name)
106            if live_snapshot:
107                os.chmod(tmpdir, 0o701)
108            # 动态快照的步骤如下:

```

```

109         # disk_path为/var/lib/nova/instances/vm-uuid/disk
110         # out_path为/var/lib/nova/instances/snapshots/snapshot_name
111         # 首先调用“qemu-img create -f qcow2 -o backing_file=disk_path
            的backing_file,size=disk_path的virtual_size out_path.delta
            ”创建镜像
112         # 然后调用domain.blockRebase(disk_path, disk_delta, 0,
113         #         libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
114         #         libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
115         #         libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)函数将
            disk_path的内容拷贝给out_path.delta
116         # 然后使用“qemu-img convert”命令将out_path.delta拷贝到
            out_path
117         # 这条命令在拷贝过程中,会先把后端镜像和增量镜像合并,然后在拷
            贝到另外一个镜像文件中,所以会比较久
118         self._live_snapshot(virt_dom, disk_path, out_path,
119                             image_format)
120     else:
121         # 使用“qemu-img convert”命令将虚拟机磁盘拷贝到out_path
122         # 这条命令在拷贝过程中,会先把后端镜像和增量镜像合并,然后在拷
            贝到另外一个镜像文件中,所以会比较久
123         # 如果不想新建一个镜像,只是想做个快照,那么这是可以改进的一
            点
124         snapshot_backend.snapshot_extract(out_path, image_format)
125     finally:
126         new_dom = None
127         # NOTE(dkang): because previous managedSave is not called
128         #         for LXC, _create_domain must not be called.
129         if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
130             if state == power_state.RUNNING:
131                 # 这种情况new_dom == virt_dom
132                 new_dom = self._create_domain(domain=virt_dom)
133             elif state == power_state.PAUSED:
134                 # 这种情况new_dom == virt_dom.createWithFlags(libvirt.
                    VIR_DOMAIN_START_PAUSED)
135                 new_dom = self._create_domain(domain=virt_dom,
136                                                 launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
137             if new_dom is not None:
138                 # 安装原有的pci设备
139                 self._attach_pci_devices(new_dom,
140                                           pci_manager.get_instance_pci_devs(instance))
141                 # 开启SR-IOV端口
142                 self._attach_sriov_ports(context, instance, new_dom)
143
144         # Upload that image to the image service
145         with libvirt_utils.file_open(out_path) as image_file:
146             self._image_api.update(context,
147                                     image_id,
148                                     metadata,
149                                     image_file)

```

## 3 snapshot 优化方案

### 3.1 最基本的情况

#### 3.1.1 最基本的情况的描述

假设现在我们想迅速地备份一个虚拟机，而且现在这个虚拟机的镜像链只有两个，也就是只有一个当初创建虚拟机用的后端镜像和一个增量文件。

在这里我说一下我们想实现哪些功能：

1. 虚拟机原先有一个磁盘镜像，它的路径是 `disk_path`，现在我们要把它作为后端镜像，并且在它的基础上创建一块增量镜像。
2. 将后端镜像上传到镜像服务器上，以备之后根据这块镜像创建虚拟机。

原先我们 `horizon` 上已经有一个创建快照的按键，现在我希望把这个按键名称改为“创建完整的虚拟机镜像”。然后在添加一个按键，名称为“迅速备份虚拟机”。

因为从网页命令到底层函数的消息的传递我不了解，这里就不提，而是把自己在底层函数的开发方案的实现方法提出来。这里的修改方法是根据原先的 `snapshot` 流程更改的，如果不熟悉 `snapshot` 流程，可以参看上一节。

当然不局限于我这种实现方法，如果有更好的实现方法，当然可以把我这个方案完全推翻。

#### 3.1.2 入口函数的改动

我的想法是新的快照动作与原先的快照动作共用一个入口函数，只是传入的 `body` 参数改一下。

本来传入的 `body` 参数是：

```
1 body == {'createImage': {'u'name': 'u'snap1', 'u'metadata': {}}}
```

现在新的快照动作对应的 `body` 参数可以是：

```
1 body == {'u'snapshot': {'u'name': 'u'snap1', 'u'metadata': {}}}
```

```
1 # nova/api/openstack/compute/servers.py Controller._action_create_image()
2 def _action_create_image(self, req, id, body):
3     # id是instance id
4     context = req.environ['nova.context']
5     # body == {'createImage': {'u'name': 'u'snap1', 'u'metadata': {}}}
6     entity = body.get("createImage", {})
7
8     # 新增加的代码
```



```

9      action = "createImage"
10     if not entity:
11         entity = body.get("snapshot", {})
12         action = "snapshot"
13
14     image_name = entity.get("name")
15
16     # 更改的代码
17     if not image_name:
18         msg = _("%(action) entity requires name attribute") % {"action": action}
19         raise exc.HTTPBadRequest(explanation=msg)
20
21     props = {}
22     metadata = entity.get('metadata', {})
23     common.check_img_metadata_properties_quota(context, metadata)
24     try:
25         props.update(metadata)
26     except ValueError:
27         msg = _("Invalid metadata")
28         raise exc.HTTPBadRequest(explanation=msg)
29
30     instance = self._get_server(context, req, id)
31
32     bdms = objects.BlockDeviceMappingList.get_by_instance_uuid(
33         context, instance.uuid)
34
35     try:
36         # 因为对linux的device mapping机制不了解，这里先不动
37         if self.compute_api.is_volume_backed_instance(context, instance,
38                                                         bdms):
39             img = instance['image_ref']
40             if not img:
41                 properties = bdms.root_metadata(
42                     context, self.compute_api.image_api,
43                     self.compute_api.volume_api)
44                 image_meta = {'properties': properties}
45             else:
46                 image_meta = self.compute_api.image_api.get(context, img)
47
48             # Snapshot the given volume-backed instance
49             image = self.compute_api.snapshot_volume_backed(
50                                                         context,
51                                                         instance,
52                                                         image_meta,
53                                                         image_name,
54                                                         extra_properties=props)
55         else:
56             # 做快照的正常流程
57             # 为这个函数添加一个action的参数
58             image = self.compute_api.snapshot(context,
59                                                         instance,
60                                                         image_name,
61                                                         extra_properties=props,
62                                                         action=action)
63     except exception.InstanceInvalidState as state_error:
64         # 修改了这里的代码

```

```

65         common.raise_http_conflict_for_instance_invalid_state(state_error,
66             action)
67     except exception.Invalid as err:
68         raise exc.HTTPBadRequest(explanation=err.format_message())
69
70     image_id = str(image['id'])
71     url_prefix = self._view_builder._update_glance_link_prefix(
72         req.application_url)
73     image_ref = os.path.join(url_prefix,
74                             context.project_id,
75                             'images',
76                             image_id)
77
78     resp = webob.Response(status_int=202)
79     resp.headers['Location'] = image_ref
80     return resp

```

### 3.1.3 修改 RPC 与 nova-compute 服务通信部分的代码

在上面的 `_action_create_image()` 函数中调用 `compute_api.snapshot()` 函数时，我们增加了一个 `action` 参数，对它的修改如下：

```

1  # 之所以将action参数设置为关键字参数，是怕有其他地方调用snapshot，关键字参数能
2  # 把影响降到最小
3  def snapshot(self, context, instance, name, extra_properties=None, action="
4  # createImage"):
5  image_meta = self._create_image(context, instance, name,
6  # 'snapshot',
7  # extra_properties=extra_properties)
8  instance.task_state = task_states.IMAGE_SNAPSHOT_PENDING
9  instance.sace(excepted_task_state=[None])
10
11 # 仍然是多传递一个参数action
12 self.compute_rpcapi.snapshot_instance(context, instance, image_meta['id'],
13 # action=action)
14
15 return image_meta

```

相应地修改 `compute.rpcapi.py` 文件中的 `snapshot_instance()` 函数：

```

1  # 增加一个关键字参数action
2  def snapshot_instance(self, ctxt, instance, image_id, action="createImage"):
3  version = '3.0'
4  cctxt = self.client.prepare(server=_compute_host(None, instance),
5  # version=version)
6  cctxt.cast(ctxt, 'snapshot_instance', instance=instance, image_id=image_id,
7  # action=action)

```

随后在 `nova-compute` 服务中修改相应的函数。

### 3.1.4 在 nova-compute 中修改相应的快照动作

在 compute.manager.py 文件中添加 snapshot\_instance 函数，只是增加一个关键字参数：

```

1  def snapshot_instance(self, context, image_id, instance, action="createImage"):
2      try:
3          instance.task_state = task_states.IMAGE_SNAPSHOT
4          instance.save(exceptd_task_state=task_states.IMAGE_SNAPSHOT_PENDING)
5      except exception.InstanceNotFound:
6          LOG.debug("Instance not found, could not set state %s for instance",
7                    task_states.IMAGE_SNAPSHOT, instance=instance)
8          return
9      except exception.UnexpectedDeletingTaskStateError:
10         LOG.debug("Instance being deleted, snapshot cannot continue", instance=
11                   instance)
12         return
13
14     # 仍然是多传递一个参数 action
15     self._snapshot_instance(context, image_id, instance, task_states.
16                             IMAGE_SNAPSHOT, action=action)

```

仍然是将 \_snapshot\_instance() 函数多增加一个关键字参数 action，同时因为这个函数已经开始调用 LibvirtDriver 类中的函数，在这里开始我觉得就可以和原先的快照流程分开了。

也就是在 LibvirtDriver 类中新增加一个 snapshot\_overlay 函数，然后在 \_snapshot\_instance 函数根据 action 参数选择跳转到 snapshot() 函数还是跳转到 snapshot\_overlay 函数。

```

1  def _snapshot_instance(self, context, image_id, instance, exceptd_task_state,
2                        action="createImage"):
3      context = context.elevated()
4
5      current_power_state = self._get_power_state(context, instance)
6      try:
7          instance.power_state = current_power_state
8          instance.save()
9
10         LOG.audit(_('instance snapshotting'), context=context, instance=
11                   instance)
12
13         if instance.power_state != power_state.RUNNING:
14             state = instance.power_state
15             running = power_state.RUNNING
16             LOG.warn(_('trying to snapshot a non-running instance:: '
17                       '(state: %(state)s expected: %(running)s)'),
18                      {'state': state, 'running': running},
19                      instance=instance)
20
21         self._notify_about_instance_usage(context, instance, "snapshot.start")
22
23     def update_task_state(task_state, exceptd_state=expected_task_state):

```

```

22         instance.task_state = task_state
23         instance.save(expected_task_state=expected_state)
24
25     # 修改的代码：根据action选择LibvirtDriver中的参数
26     if action == "createImage":
27         self.driver.snapshot(context, instance, image_id, update_task_state)
28     else:
29         self.driver.snapshot_overlay(context, instance, image_id,
30                                     update_task_state)
31
32     instance.task_state = None
33     instance.save(expected_task_state=task_states.IMAGE_UPLOADING)
34
35     self._notify_about_instance_usage(context, instance, "snapshot.end")
36 except (exception.InstanceNotFound,
37        exception.UnexpectedDeletingTaskStateError):
38     msg = 'Instance disappeared during snapshot'
39     LOG.debug(msg, instance=instance)
40     try:
41         iamge_service = glance.get_default_image_service()
42         image = image_service.show(context, image_id)
43         if image['status'] != 'active':
44             image_service.delete(context, image_id)
45     except Exception:
46         LOG.warning(_("Error while trying to clean up image %s"),
47                     image_id, instance=instance)
48 except exception.ImageNotFound:
49     instance.task_state = None
50     instance.save()
51     msg = _("Image not found during snapshot")
52     LOG.warn(msg, instance=instance)

```

### 3.1.5 新增加一个核心函数 snapshot\_overlay

根据上一节的描述我们知道，我们需要在 LibvirtDriver 类中增加一个 snapshot\_overlay 函数，用于实现我们想实现的功能。

在这里我再说一下我们想实现哪些功能：

1. 虚拟机原先有一个磁盘镜像，它的路径是 disk\_path，现在我们要把它作为后端镜像，并且在它的基础上创建一块增量镜像。
2. 将后端镜像上传到镜像服务器上，以备之后根据这块镜像创建虚拟机。

针对功能一的实现细节，我有以下的想法：

1. 首先将虚拟机磁盘镜像改名，加个后缀“.base”，它的路径也就变成了 disk\_path.base。
2. 然后根据这个镜像文件创建增量文件，这个增量文件的名字和原先磁盘镜像的名字相同，所以它的路径也就是 disk\_path。

### 3. 重启虚拟机，虚拟机就会开始使用这个增量文件。

针对可能的疑问，我这里作出回答：

```

1  问：虚拟机运行时，将磁盘镜像更名是否会影响它的运行？
2  答：我在kvm中进行过操作，发现是不会影响虚拟机的运行的。
3
4  问：虚拟机磁盘镜像更名后，虚拟机的数据改动是否会保存？
5  答：会保存，而且是保存到原先的镜像文件中。
6
7  问：根据原先的虚拟机镜像创建增量镜像后，虚拟机的数据改动是否会保存到增量镜像
8  中？
9  答：数据的改动会保存到当前的后端镜像中，也保存到增量镜像中。
    不过需要知道的是，当虚拟机重启后，虚拟机开始使用这个增量镜像，所以虚拟机的
    数据改动会保存到增量镜像中，不会再保存到原先的镜像文件中。

```

针对功能二我有想补充的话：

```

1  当后端镜像传到镜像服务器以后，我们应该知道一个事实，这个后端镜像也是有自己的
    backing file 的。如果想要根据这个后端镜像创建原先的虚拟机，必须保证它的
    backing file 是存在的。

```

新增加 snapshot\_overlay 函数如下：

```

1  def snapshot_overlay(self, context, instance, image_id, update_task_state):
2      try:
3          virt_dom = self._lookup_by_name(instance['name'])
4      except exception.InstanceNotFound:
5          raise exception.InstanceNotRunning(instance_id=instance['uuid'])
6
7      base_image_ref = instance['image_ref']
8
9      base = compute_utils.get_image_metadata(context, self._image_api,
10                                              base_image_ref, instance)
11
12      snapshot = self._image_api.get(context, image_id)
13
14      disk_path = libvirt_utils.find_disk(virt_dom)
15      source_format = libvirt_utils.get_disk_type(disk_path)
16
17      image_format = CONF.libvirt.snapshot_image_format or source_format
18
19      if image_format == 'lvm' or image_format == 'rbd':
20          image_format = 'raw'
21
22      metadata = self._create_snapshot_metadata(base,
23                                              instance,
24                                              image_format,
25                                              snapshot['name'])
26
27      snapshot_name = uuid.uuid4().hex
28
29      state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]
30
31      if (self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,

```

```

31             MIN_QEMU_LIVESNAPSHOT_VERSION,
32             REQ_HYPERVISOR_LIVESNAPSHOT)
33         and source_format not in ('lvm', 'rbd')
34         and not CONF.ephemeral_storage_encryption.enabled):
35             live_snapshot = True
36
37         try:
38             virt_dom.blockJobAbort(disk_path, 0)
39         except libvirt.libvirtError as ex:
40             error_code = ex.get_error_code()
41             if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
42                 live_snapshot = False
43             else:
44                 pass
45         else:
46             live_snapshot = False
47
48         if state == power_state.SHUTDOWN:
49             live_snapshot = False
50
51         if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
52             if state == power_state.RUNNING or state == power_state.PAUSED:
53                 self._detach_pci_devices(virt_dom, pci_manager.
54                                         get_instance_pci_devs(instance))
55                 self._detach_sriov_ports(context, instance, virt_dom)
56                 virt_dom.managedSave(0)
57
58             snapshot_backend = self.image_backend.snapshot(instance,
59                                                            disk_path,
60                                                            image_type=source_format)
61
62             if live_snapshot:
63                 LOG.info(_LI("Beginning live snapshot process"),
64                          instance=instance)
65             else:
66                 LOG.info(_LI("Beginning cold snapshot process"),
67                          instance=instance)
68
69             update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
70             snapshot_directory = CONF.libvirt.snapshots_directory
71             fileutils.ensure_tree(snapshot_directory)
72             with utils.tmpdir(dir=snapshot_directory) as tmpdir:
73                 try:
74                     out_path = os.path.join(tmpdir, snapshot_name)
75                     if live_snapshot:
76                         # openstack貌似不支持动态创建快照，所以不修改这里的代码
77                         os.chmod(tmpdir, 0o701)
78                         self._live_snapshot(virt_dom, disk_path, out_path, image_format)
79                     else:
80                         # 修改的代码
81                         # 首先将虚拟机磁盘镜像改名，加个后缀“.base”
82                         disk_path_base = disk_path + '.base'
83                         out_path = disk_path
84                         utils.execute('mv', disk_path, disk_path_base)
85                         disk_path = disk_path_base

```

```

85         # 然后在原磁盘文件上创建增量文件
86         libvirt_utils.create_overlay(disk_path, out_path, image_format)
87         # 强制重启虚拟机
88         virt_dom.reset()
89     finally:
90         new_dom = None
91         if CONF.libvirt.type != 'lxc' and not live_snapshot:
92             if state == power_state.RUNNING:
93                 new_dom = self._create_domain(domain=virt_dom)
94             elif state == power_state.PAUSED:
95                 new_dom = self._create_domain(domain=virt_dom,
96                                                 launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
97             if new_dom is not None:
98                 self._attach_pci_devices(new_dom, pci_manager.
99                                         get_instance_pci_devs(instance))
100                 self._attach_sriov_ports(context, instance, new_dom)
101             LOG.info(_LI("Snapshot extracted, beginning image upload"),
102                     instance=instance)
103
104         update_task_state(task_state=task_states.IMAGE_UPLOADING,
105                          expected_state=task_states.IMAGE_PENDING_UPLOAD)
106
107         # 将原先的磁盘文件上传到镜像服务器
108         with libvirt_utils.file_open(disk_path) as image_file:
109             self._image_api.update(context,
110                                   image_id,
111                                   metadata,
112                                   image_file)
113             LOG.info(_LI("Snapshot image upload complete"),
114                     instance=instance)

```

### 3.1.6 在 virt/libvirt/utils.py 创建新函数

```

1     def create_overlay(disk_path, out_path, dest_format):
2         if dest_fmt != 'qcow2':
3             raise
4
5         qemu_img_cmd = ('qemu-img', 'create', '-b', disk_path, '-f', dest_fmt,
6                         out_path)
7         execute(*qemu_img_cmd)

```

## 3.2 镜像链为 3 的情况

### 3.2.1 镜像链为 3 的情况的描述

原先最基本的情况是：虚拟机的镜像链只有两个，也就是只有一个当初创建虚拟机用的后端镜像和一个增量文件。

现在的情况是：我们已经为备份过一次虚拟机，也就是现在虚拟机的镜像链为 3，有最初创建虚拟机用的后端镜像，一个备份用的镜像和一个最新的增量文件，也就是 `base <- overlay1 <- overlay2`。

如果再备份一次虚拟机的话，虚拟机的镜像链就会变成 4，为了保持虚拟机的性能，我们决定在备份虚拟机时，能够将 `overlay1` 和 `overlay2` 合并，保持镜像链为 3。

### 3.2.2 核心函数的修改

这里的修改方法参考了《快照相关原理》的第六节“删除快照”。

当镜像链为 4 时，镜像链为：`base <- overlay1 <- overlay2 <- overlay3`。为了保持镜像链为 3，我们现在需要先把 `overlay2` 的内容融入 `overlay1` 中，然后把 `overlay3` 的后端镜像变为 `overlay1`。

修改后的核心函数如下：

```

1  def snapshot_overlay(self, context, instance, image_id, update_task_state):
2      try:
3          virt_dom = self._lookup_by_name(instance['name'])
4      except exception.InstanceNotFound:
5          raise exception.InstanceNotRunning(instance_id=instance['uuid'])
6
7      base_image_ref = instance['image_ref']
8      base = compute_utils.get_image_metadata(context, self._image_api,
9          base_image_ref, instance)
9      snapshot = self._image_api.get(context, image_id)
10     disk_path = libvirt_utils.find_disk(virt_dom)
11     source_format = libvirt_utils.get_disk_type(disk_path)
12
13     image_format = CONF.libvirt.snapshot_image_format or source_format
14
15     if image_format == 'lvm' or image_format == 'rbd':
16         image_format = 'raw'
17
18     metadata = self._create_snapshot_metadata(base,
19         instance,
20         image_format,
21         snapshot['name'])
22
23     snapshot_name = uuid.uuid4().hex
24
25     state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]
26
27     if (self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
```



```

28             MIN_QEMU_LIVESNAPSHOT_VERSION,
29             REQ_HYPERVISOR_LIVESNAPSHOT)
30         and source_format not in ('lvm', 'rbd')
31         and not CONF.ephemeral_storage_encryption.enabled):
32             live_snapshot = True
33
34         try:
35             virt_dom.blockJobAbort(disk_path, 0)
36         except libvirt.libvirtError as ex:
37             error_code = ex.get_error_code()
38             if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
39                 live_snapshot = False
40             else:
41                 pass
42         else:
43             live_snapshot = False
44
45         if state == power_state.SHUTDOWN:
46             live_snapshot = False
47
48         if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
49             if state == power_state.RUNNING or state == power_state.PAUSED:
50                 self._detach_pci_devices(virt_dom, pci_manager.
51                     get_instance_pci_devs(instance))
52                 self._detach_sriov_ports(context, instance, virt_dom)
53                 virt_dom.managedSave(0)
54
55             snapshot_backend = self.image_backend.snapshot(instance,
56                 disk_path,
57                 image_type=source_format)
58
59             if live_snapshot:
60                 LOG.info(_LI("Beginning live snapshot process"),
61                     instance=instance)
62             else:
63                 LOG.info(_LI("Beginning cold snapshot process"),
64                     instance=instance)
65
66             update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
67             snapshot_directory = CONF.libvirt.snapshots_directory
68             fileutils.ensure_tree(snapshot_directory)
69             with utils.tmpdir(dir=snapshot_directory) as tmpdir:
70                 try:
71                     out_path = os.path.join(tmpdir, snapshot_name)
72                     if live_snapshot:
73                         # openstack貌似不支持动态创建快照，所以不修改这里的代码
74                         os.chmod(tmpdir, 0o701)
75                         self._live_snapshot(virt_dom, disk_path, out_path, image_format)
76                     else:
77                         # 修改的代码
78                         # 获得虚拟机镜像的后端镜像的文件名
79                         backing_file = libvirt_utils.get_disk_backing_file(disk_path,
80                             basename=False)
81                         # 获得后端镜像的后缀
82                         suffix = backing_file.split(".")[1]

```

```

81         # 我们本身备份虚拟机时，会把原先虚拟机磁盘加一个“.base”后缀
82         # 现在如果suffix == “.base”，那么说明这个虚拟机已经备份过一次，
            # 那么之后我们需要合并overlay1和overlay2
83         # 如果虚拟机备份过一次，我们还需要将overlay1再加个后缀“.base”
84         if suffix == “.base”:
85             utils.execute('mv', backing_file, backing_file + '.base')
86             backing_file = backing_file + '.base'
87         out_path = disk_path
88         utils.execute('mv', disk_path, disk_path + '.base')
89         disk_path = disk_path + '.base'
90         # 然后在原磁盘文件上创建增量文件
91         libvirt_utils.create_overlay(disk_path, out_path, image_format)
92         # 强制重启虚拟机
93         virt_dom.reset()
94         # 判断虚拟机是否备份过一次
95         # 如果备份过一次，现在我们的快照链如下：
96         # base <- overlay1 <- overlay2 <- overlay3
97         if suffix == “.base”:
98             # 因为overlay1改名了，所以先要更改overlay2的后端镜像名
99             utils.execute('qemu-img', 'rebase', '-b', backing_file,
                disk_path)
100             # 然后将overlay2融入overlay1中
101             utils.execute('qemu-img', 'commit', disk_path)
102             # 然后将overlay1作为overlay3的后端镜像
103             utils.execute('qemu-img', 'rebase', '-u', '-b',
                backing_file, out_path)
104             # 最后删除overlay2这个镜像
105             utils.execute('rm', disk_path)
106             # 这是为了之后上传后端镜像时，上传的是backing_file
107             disk_path = backing_file
108         finally:
109             new_dom = None
110             if CONF.libvirt.type != 'lxc' and not live_snapshot:
111                 if state == power_state.RUNNING:
112                     new_dom = self._create_domain(domain=virt_dom)
113                 elif state == power_state.PAUSED:
114                     new_dom = self._create_domain(domain=virt_dom,
115                         launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
116                 if new_dom is not None:
117                     self._attach_pci_devices(new_dom, pci_manager.
118                         get_instance_pci_devs(instance))
119                     self._attach_sriov_ports(context, instance, new_dom)
120             LOG.info(_LI("Snapshot extracted, beginning image upload"),
121                 instance=instance)
122             update_task_state(task_state=task_states.IMAGE_UPLOADING,
123                 expected_state=task_states.IMAGE_PENDING_UPLOAD)
124             # 将原先的磁盘文件上传到镜像服务器
125             with libvirt_utils.file_open(disk_path) as image_file:
126                 self._image_api.update(context,
127                     image_id,
128                     metadata,
129                     image_file)
130             LOG.info(_LI("Snapshot image upload complete"),
131                 instance=instance)

```