目 录

1 前言

在 nova/virt/libvirt/driver.py 中有 LibvirtDriver 类，用于调用 libvirt。

2 nova 创建虚拟机时对 libvirt 的调用

LibvirtDriver 类中创建虚拟机的函数代码如下：

```
1   def spawn(self, context, instance, image_meta, injected_files,
2             admin_password, network_info=None, block_device_info=None):
3
4       # 获取disk配置信息
5       disk_info = blockinfo.get_disk_info(CONF.libvirt.virt_type,
6                                           instance,
7                                           image_meta,
8                                           block_device_info)
9
10      ...
11
12      # 创建镜像
13      self._create_image(context, instance, disk_info['mapping'],
14                         injection_info=injection_info,
15                         block_device_info=block_device_info)
16
17      ...
18
19      # 创建xml配置文件
20      xml = self._get_guest_xml(context, instance, network_info,
21                                disk_info, image_meta,
22                                block_device_info=block_device_info)
23
24      ...
25
26      # 创建虚拟机和网络
27      self._create_domain_and_network(
28          context, xml, instance, network_info, disk_info,
29          block_device_info=block_device_info,
30          post_xml_callback=gen_confdrive,
31          destroy_disks_on_failure=True)
```

2.1 创建虚拟机的函数

LibvirtDriver 类中调用 libvirt 创建虚拟机和网络的函数代码如下，因为我更关注创建虚拟机的流程，所以把与创建网络有关的代码略去：

```
    def _create_domain_and_network(self, context, xml, instance, network_info,
                                   disk_info, block_device_info=None,
                                   power_on=True, reboot=False,
                                   vifs_already_plugged=False,
                                   post_xml_callback=None,
                                   destroy_disks_on_failure=False):

        ...

        # 创建虚拟机
        guest = None
        try:
            with self.virtapi.wait_for_instance_event(
                    instance, events, deadline=timeout,
                    error_callback=self._neutron_failed_callback):
                ...
                with self._lxc_disk_handler(instance, instance.image_meta,
                                            block_device_info, disk_info):
                    guest = self._create_domain(
                        xml, pause=pause, power_on=power_on,
                        post_xml_callback=post_xml_callback)
                    ...
        ...

        return guest
```

创建虚拟机的函数代码如下：

```
    def _create_domain(self, xml=None, domain=None,
                       power_on=True, pause=False, post_xml_callback=None):

        # 创建虚拟机
        # 这里的libvirt_guest在driver.py中有引用: from nova.virt.libvirt import
            guest as libvirt_guest
        if xml:
            guest = libvirt_guest.Guest.create(xml, self._host)
        else:
            guest = libvirt_guest.Guest(domain)

        ...

        return guest
```

从上面的代码可以看出，nova/virt/libvirt/guest.py 中的 Guest 类就是对虚拟机 instance 的抽象。

因为我们这里是新建虚拟机，所以 domain 应该是 None，是调用 Guest.create() 函数来创建虚拟机。下面来看 Guest 类的 create() 函数：

```
class Guest(object):
    @classmethod
    def create(cls, xml, host):

        try:
            ...
            # 这里使用host创建虚拟机，host是nova/virt/libvirt/host.py中的Host类
            guest = host.write_instance_config(xml)

        ...

        return guest
```

这里的 Host 类的 write_instance_config() 的代码如下所示：

```
    def write_instance_config(self, xml):
        # 直接告诉我们，get_connection()函数返回的是与libvirt相关的对象
        # 下面这行代码可以让我们想起：virsh define demo.xml这个命令
        # 实际上self.get_connection()返回的是一个virConnect对象
        domain = self.get_connection().defineXML(xml)

        return libvirt_guest.Guest(domain)
```

如果想知道 openstack 怎么创建一个 virConnect 对象，就继续看 get_connection() 函数：

```
    def get_connection(self):

        try:
            # 很清楚地看到conn对象是由_get_connection()创建的
            conn = self._get_connection()

        ...

        return conn
```

进一步看 _get_connection() 函数：

```
    def _get_connection(self):
        # _wrapped_conn_lock是一个互斥锁
        with self._wrapped_conn_lock:
            ...
            # _wrapped_conn是一个连接到libvirt的对象
            # 当前服务没有调用libvirt是，_wrapped_conn是None
            if self._wrapped_conn is None:
                try:
                    # 连接到libvirt并返回一个对象
                    self._wrapped_conn = self._get_new_connection()

                ...

        return self._wrapped_conn
```

再来看 _get_new_connection() 函数：

```
1     def _get_new_connection(self):
2         ...
3         # 这里的uri是'qemu:///system'，根据之前的代码追踪过程可以轻易得知，在此不再
              详述如何追踪到它的值
4         # _read_only为False
5         # 使用_connect()函数创建wrapped_conn对象
6         # 如果熟悉libvirt API，那么看到这条语句会想到下面C语言中的这条语句：
7         #   conn = virConnectPtr("qemu:///system")
8         #   conn是virConnectPtr对象，python中是virConnect类
9         wrapped_conn = self._connect(self._uri, self._read_only)
10
11        ...
12
13        return wrapped_conn
```

分析到这，应该算是比较清楚了。我们如果是熟悉 virConnect 对象，那么就能使用 libvirt 的 API 在 openstack 中呼风唤雨了。

3 nova 扩容时对 libvirt 的调用

LibvirtDriver 类中创建虚拟机快照的函数是 migrate_disk_and_power_off()。

```
# nova/virt/libvirt/driver.py LibvirtDriver.migrate_disk_and_power_off()
def migrate_disk_and_power_off(self, context, instance, dest,
                               flavor, network_info,
                               block_device_info=None,
                               timeout=0, retry_interval=0):

    # 获取disk配置信息
    # block_device_info是一个字典，如下所示:
    # {'swap': swap,
    # 'root_device_name': root_device_name,
    # 'ephemerals': ephemerals,
    # 'block_device_mapping': block_device_mapping}
    #
    # disk_info_text数据类型是一个list，里面有多个字典，字典格式如下
    # {'type': disk_type,
    # 'path': path,
    # 'virt_disk_size': virt_size,
    # 'backing_file': backing_file,
    # 'disk_size': dk_size,
    # 'over_committed_disk_size': over_commit_size}
    disk_info_text = self.get_instance_disk_info(instance['name'],
            block_device_info=block_device_info)

    # 调用json.loads()
    disk_info = jsonutils.loads(disk_info_text)

    # 调用os.path.join(CONF.instances_path, instance['name'])获得inst_base
    inst_base = libvirt_utils.get_instance_path(instance)
    # 创建resize以后instance后备文件的存放路径
    inst_base_resize = inst_base + "_resize"
    # 判断是否共用storage
    shared_storage = self._is_storage_shared_with(dest, inst_base)

    # 在目的主机下根据inst_base创建增量文件的文件夹
    if not shared_storage:
        utils.execute('ssh', dest, 'mkdir', '-p', inst_base)

    # 关闭虚拟机
    self.power_off(instance, timeout, retry_interval)

    # block_device_info是一个字典
    # 相当于调用block_device_info.get('block_device_mapping')
    block_device_mapping = driver.block_device_info_get_mapping(
        block_device_info)

    # 迁移前，通过特定类型的卷驱动卸载卷设备。对于
    # rbd (LibvirtNetVolumeDriver)，什么都没有做;
    # 对于iscsi (LibvirtISCSIVolumeDriver)，做了两个工作:
    # 1. echo '1' > /sys/block/{dev_name}/device/delete
    # 2. 通过iscsiadm工具删除相关的端点信息
```

```
51          for vol in block_device_mapping:
52              connection_info = vol['connection_info']
53              disk_dev = vol['mount_device'].rpartition("/")[2]
54              self._disconnect_volume(connection_info, disk_dev)
55
56          try:
57              # 重命名inst_base为inst_base_resize
58              utils.execute('mv', inst_base, inst_base_resize)
59              if shared_storage:
60                  dest = None
61                  utils.execute('mkdir', '-p', inst_base)
62
63              active_flavor = flavors.extract_flavor(instance)
64
65              # 迁移虚拟机磁盘内容
66              for info in disk_info:
67                  # assume inst_base == dirname(info['path'])
68                  img_path = info['path']
69                  fname = os.path.basename(img_path)
70                  from_path = os.path.join(inst_base_resize, fname)
71
72                  if (fname == 'disk.swap' and
73                      active_flavor.get('swap', 0) != flavor.get('swap', 0)):
74                      # To properly resize the swap partition, it must be
75                      # re-created with the proper size.  This is acceptable
76                      # because when an OS is shut down, the contents of the
77                      # swap space are just garbage, the OS doesn't bother about
78                      # what is in it.
79
80                      # We will not copy over the swap disk here, and rely on
81                      # finish_migration/_create_image to re-create it for us.
82                      continue
83
84                  on_execute = lambda process: self.job_tracker.add_job(
85                      instance, process.pid)
86                  on_completion = lambda process: self.job_tracker.remove_job(
87                      instance, process.pid)
88
89                  # 如果类型是qcow2并且有后备文件
90                  if info['type'] == 'qcow2' and info['backing_file']:
91                      tmp_path = from_path + "_rbase"
92                      # merge backing file
93                      utils.execute('qemu-img', 'convert', '-f', 'qcow2',
94                                    '-O', 'qcow2', from_path, tmp_path)
95
96                      if shared_storage:
97                          utils.execute('mv', tmp_path, img_path)
98                      else:
99                          libvirt_utils.copy_image(tmp_path, img_path, host=dest,
100                                                   on_execute=on_execute,
101                                                   on_completion=on_completion)
102                         utils.execute('rm', '-f', tmp_path)
103                 # 如果类型是raw或者是没有后备文件的qcow2
104                 else:
105                     libvirt_utils.copy_image(from_path, img_path, host=dest,
106                                              on_execute=on_execute,
```

```
107                                                 on_completion=on_completion)
108         ...
```

3.1 关闭虚拟机

相关语句如下:

```
1     self.power_off(instance, timeout, retry_interval)
```

4 nova 创建快照时对 libvirt 的调用

LibvirtDriver 类中创建虚拟机快照的函数是 snapshot()。

这个函数主要功能的实现都与 libvirt 有关，我留取了一些主要的代码，并在之后逐个分析：

```
    def snapshot(self, context, instance, image_id, update_task_state):

        try:
            # 得到虚拟机的抽象对象virt_dom
            virt_dom = self._lookup_by_name(instance['name'])
        ...

        # 得到虚拟机的磁盘路径
        disk_path = libvirt_utils.find_disk(virt_dom)

        # 得到虚拟机的磁盘类型
        source_format = libvirt_utils.get_disk_type(disk_path)

        # 如果是静态创建快照，将会执行下面这段代码，稍后会在“静态创建快照”一节分析
        if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
            if state == power_state.RUNNING or state == power_state.PAUSED:
                ...
                virt_dom.managedSave(0)

        # snapshot_backend也将用于静态创建快照，稍后分析
        snapshot_backend = self.image_backend.snapshot(instance,
                disk_path,
                image_type=source_format)

        with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                ...
                if live_snapshot:
                    # 动态创建快照
                    self._live_snapshot(virt_dom, disk_path, out_path,
                                        image_format)
                else:
                    # 静态创建快照
                    snapshot_backend.snapshot_extract(out_path, image_format)
            ...
```

4.1 获得虚拟机抽象对象

函数中的语句如下：

```
1    # instance是Intance类，Instance类定义在nova/objects/instance.py中
2    # 返回了virDomain对象
3    virt_dom = self._lookup_by_name(instance['name'])
```

_lookup_by_name() 函数如下：

```
1    def _lookup_by_name(self, instance_name):
2        try:
3            # 这里的_conn就是libvirt API中的virConnect对象
4            return self._conn.lookupByName(instance_name)
5        ...
```

lookupByName 是 libvirt 的一个 API 函数，可以根据域的名字返回 virDomain 对象。

4.2 获得虚拟机磁盘路径

函数中的语句如下：

```
1    # libvirt_utils在driver.py中有引用: from nova.virt.libvirt import utils as
         libvirt_utils
2    disk_path = libvirt_utils.find_disk(virt_dom)
```

find_disk 函数如下：

```
1    def find_disk(virt_dom):
2        # 在这里，我们只关心下面这条语句
3        # virt_dom是libvirt的virDomain对象
4        xml_desc = virt_dom.XMLDesc(0)
5        ...
```

这里的 XMLDesc() 是 virDomain 类的一个成员函数，用于提供域的 XML 配置文件。也就是说，获得虚拟机磁盘路径的核心函数是 virDomain.XMLDesc()。

虽然我这里省略了 find_disk() 函数接下来的代码，但是大家应该也能知道，接下来就是解析这个 xml 文件，得到域的磁盘路径。

4.3 获得虚拟机磁盘类型

函数中的语句如下：

```
1    # libvirt_utils在driver.py中有引用: from nova.virt.libvirt import utils as
         libvirt_utils
2    # 返回了虚拟机磁盘的类型
```

```
3     source_format = libvirt_utils.get_disk_type(disk_path)
```

get_disk_type() 函数如下：

```
1     def get_disk_type(path):
2     ...
3     # 这个函数用于得到磁盘的类型
4     # qemu_img_info()返回了是QemuImgInfo类
5     # QemuImgInfo类定义于nova/openstack/common/imageutils.py中
6     # QemuImgInfo.file_format存放着磁盘类型的信息
7     return images.qemu_img_info(path).file_format
```

继续追踪 qemu_img_info() 函数：

```
1     def qemu_img_info(path):
2     ...
3     # 这条语句相当于执行命令行：qemu-img info disk_path，返回镜像文件的信息
4     # utils是nova/utils.py文件
5     out, err = utils.execute('env', 'LC_ALL=C', 'LANG=C',
6                              'qemu-img', 'info', path)
7     ...
8     # 返回QemuImgInfo类，这个类定义于nova/openstack/common/imageutils.py中
9     return imageutils.QemuImgInfo(out)
```

所以说，openstack 获得虚拟机磁盘信息，最终是调用了“qemu-img info <filename>”这条命令。

4.4 静态创建快照

可以从 snapshot() 函数知道静态创建快照的相关语句如下：

```
1     ...
2     if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
3             if state == power_state.RUNNING or state == power_state.PAUSED:
4                 ...
5                 #
6                 virt_dom.managedSave(0)
7     # 获得与静态创建快照有关的数据结构
8     snapshot_backend = self.image_backend.snapshot(instance,
9                 disk_path,
10                image_type=source_format)
11    ...
12    with utils.tempdir(dir=snapshot_directory) as tmpdir:
13            try:
14                ...
15                else:
16                    # 静态创建快照
17                    snapshot_backend.snapshot_extract(out_path, image_format)
```

我们逐句分析这三句注释了的代码：

1. 第一句注释代码：

```
    # virt_dom是libvirt的virDomain对象
    # virDomain.managedSave()函数用于将虚拟机的域挂起，然后将它的内存信息保存
        到虚拟机镜像的一个文件中
    virt_dom.managedSave(0)
```

其实第一步已经完成了对虚拟机创建快照，只是现在这个快照保存在镜像文件中，还需要将它提取出来。

2. 第二句注释代码：

```
    # image_backend是nova/libvirt/imagebackend.py文件中定义的Backend类
    # snapshot()函数根据image_type返回相应的类
    snapshot_backend = self.image_backend.snapshot(instance,
                disk_path,
                image_type=source_format)
```

进一步看 snapshot() 函数：

```
    def snapshot(self, instance, disk_path, image_type=None):
        # self.backend()根据image_type返回相应的类
        # 具体细节在此不详述，有兴趣的可以看Backend类
        backend = self.backend(image_type)
        return backend(instance=instance, path=disk_path)
```

3. 在分析第三句注释代码前，我先说一些准备信息。

我分析代码的时候，使用的磁盘类型是 qcow2 类型，所以现在 snapshot_backend 是 Qcow2 类。而且现在我们已经通过 managedSave() 函数将它的快照保存到虚拟机镜像的一个文件中。所以使用 snapshot_extract() 函数就很好理解了，就是提取快照文件：

```
    snapshot_backend.snapshot_extract(out_path, image_format)

    # snapshot_extract()函数如下
class Qcow2(Image):
    ...
    def snapshot_extract(self, target, out_format):
        libvirt_utils.extract_snapshot(self.path, 'qcow2',
                                       target,
                                       out_format)
```

代码到了这一步，已经是即将完成快照的提取。进一步看 extract_snapshot() 函数：

```
    def extract_snapshot(disk_path, source_fmt, out_path, dest_fmt):
        # 函数相当于执行了“qemu-img convert -f source_fmt -O dest_fmt
            disk_path out_path”
        # qemu-img convert就是用来提取快照的命令
        qemu_img_cmd = ('qemu-img', 'convert', '-f', source_fmt, '-O',
            dest_fmt)
        ...
        qemu_img_cmd += (disk_path, out_path)
        execute(*qemu_img_cmd)
```

总结一下，openstack 静态创建快照步骤如下：

1. 调用 virDomain.managedSave() 创建快照。
2. 用“qemu-img convert”命令提取快照。

4.5 动态创建快照

snapshot() 函数中动态创建快照的相关语句如下：

```
    ...
    with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                ...
                if live_snapshot:
                    # 动态创建快照
                    self._live_snapshot(virt_dom, disk_path, out_path,
                                        image_format)
                ...
```

进一步看 _live_snapshot() 函数：

```
    def _live_snapshot(self, domain, disk_path, out_path, image_format):
        # domain是虚拟机的virDomain对象

        # 传入VIR_DOMAIN_XML_INACTIVE和VIR_DOMAIN_XML_SECURE参数
        # 获得domain的xml配置文件
        # 这个xml用于之后的undefine()和define()函数
        xml = domain.XMLDesc(
            libvirt.VIR_DOMAIN_XML_INACTIVE |
            libvirt.VIR_DOMAIN_XML_SECURE)

        try:
            # 调用virDomain.blockJobAbort()停止活动块操作
            domain.blockJobAbort(disk_path, 0)
        ...
        # 获得虚拟机backing_file的路径
        src_back_path = libvirt_utils.get_disk_backing_file(disk_path,
```

```
                                                        basename=False)
        disk_delta = out_path + '.delta'
        # 这个函数位于nova/virt/libvirt/utils.py中
        # 函数中调用“qemu-img create -f qcow2 -o src_disk_size,src_back_path
            src_back_path”创建镜像文件
        libvirt_utils.create_cow_image(src_back_path, disk_delta,
                                       src_disk_size)

        try:
            # 因为virDomain.blockRebase()不能对persistent的domain操作
            # 所以调用virDomain.undefine()将domain变为transient
            if domain.isPersistent():
                domain.undefine()

            # 将虚拟机镜像文件的内容拷贝到disk_delta中的镜像文件中
            domain.blockRebase(disk_path, disk_delta, 0,
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)

            ...

            # 调用virtDomain.blockJobAbort()终止数据拷贝
            domain.blockJobAbort(disk_path, 0)
            ...
        finally:
            # 将domain从transient变为persistent
            self._conn.defineXML(xml)

        # 这个函数调用“qemu-img convert”命令将disk_delta中的文件变为一个qcow2文件
        libvirt_utils.extract_snapshot(disk_delta, 'qcow2',
                                       out_path, image_format)
```

总结一下，openstack **动态创建快照步骤如下：**

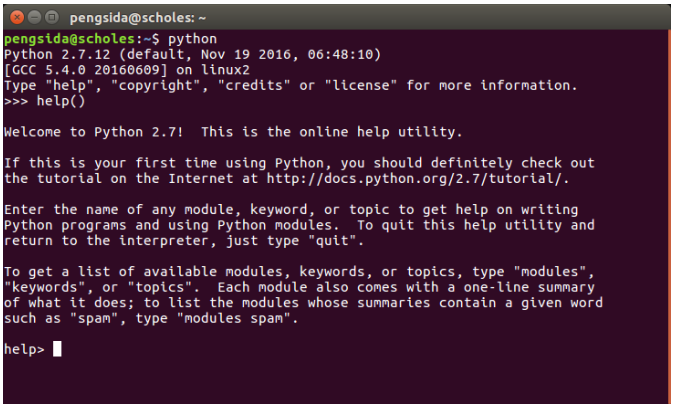
1. **使用“**qemu-img create**”命令创建用于备份的镜像。**
2. **调用** virtDomain.blockRebase() **函数将虚拟机的磁盘内容拷贝到备份镜像中。**
3. **使用**”qemu-img convert” **命令将备份镜像转换为一个** qcow2 **文件。**

5 openstack 中对 libvirt 调用的框架

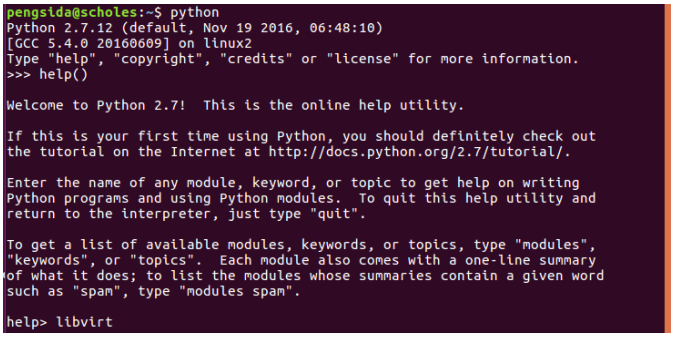
6 查看 libvirt 的 python API

可以通过 python 的 help() 函数来查看 virConnect 类：

1. 首先进入 python 的 help 界面：



2. 查看 libvirt 的帮助文档：



3. 通过“virConnect”找到对 virConnect 各个函数的介绍：