

目 录

1	nova 调用 libvirt	2
1.1	nova 创建虚拟机时对 libvirt 的调用	2
1.1.1	创建虚拟机的函数	3
1.2	nova 动态迁移时对 libvirt 的调用	6
1.3	nova 创建快照时对 libvirt 的调用	7
1.3.1	获得虚拟机抽象对象	8
1.3.2	获得虚拟机磁盘路径	8
1.3.3	获得虚拟机磁盘类型	8
1.3.4	静态创建快照	9
1.3.5	动态创建快照	11
2	查看 libvirt 的 python API	13

1 nova 调用 libvirt

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

1.1 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)
```

1.1.1 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：

```

1  def __create_domain_and_network(self, context, xml, instance, network_info,
2                                disk_info, block_device_info=None,
3                                power_on=True, reboot=False,
4                                vifs_already_plugged=False,
5                                post_xml_callback=None,
6                                destroy_disks_on_failure=False):
7
8      ...
9
10     # 创建虚拟机
11     guest = None
12     try:
13         with self.virtapi.wait_for_instance_event(
14             instance, events, deadline=timeout,
15             error_callback=self.__neutron_failed_callback):
16             ...
17             with self._lxc_disk_handler(instance, instance.image_meta,
18                                         block_device_info, disk_info):
19                 guest = self._create_domain(
20                     xml, pause=pause, power_on=power_on,
21                     post_xml_callback=post_xml_callback)
22             ...
23     ...
24
25     return guest

```

创建虚拟机的函数代码如下：

```

1  def __create_domain(self, xml=None, domain=None,
2                    power_on=True, pause=False, post_xml_callback=None):
3
4      # 创建虚拟机
5      # 这里的 libvirt_guest 在 driver.py 中有引用: from nova.virt.libvirt import
6      #      guest as libvirt_guest
7      if xml:
8          guest = libvirt_guest.Guest.create(xml, self._host)
9      else:
10         guest = libvirt_guest.Guest(domain)
11
12     ...
13
14     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：

```

1 class Guest(object):
2     @classmethod
3     def create(cls, xml, host):
4
5         try:
6             ...
7             # 这里使用host创建虚拟机, host是nova/virt/libvirt/host.py中的Host类
8             guest = host.write_instance_config(xml)
9
10            ...
11
12            return guest

```

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

```

1 def write_instance_config(self, xml):
2     # 直接告诉我们, get_connection()函数返回的是与libvirt相关的对象
3     # 下面这行代码可以让我们想起: virsh define demo.xml这个命令
4     # 实际上self.get_connection()返回的是一个virConnect对象
5     domain = self.get_connection().defineXML(xml)
6
7     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:

```

1 def get_connection(self):
2
3     try:
4         # 很清楚地看到conn对象是由__get_connection()创建的
5         conn = self.__get_connection()
6
7         ...
8
9     return conn

```

进一步看 __get_connection() 函数:

```

1 def __get_connection(self):
2     # __wrapped_conn_lock是一个互斥锁
3     with self.__wrapped_conn_lock:
4         ...
5         # __wrapped_conn是一个连接到libvirt的对象
6         # 当前服务没有调用libvirt是, __wrapped_conn是None
7         if self.__wrapped_conn is None:
8             try:
9                 # 连接到libvirt并返回一个对象
10                self.__wrapped_conn = self.__get_new_connection()
11
12                ...
13
14            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。

1.2 nova 动态迁移时对 libvirt 的调用

LibvirtDriver 类中动态迁移的函数代码如下:

```
1  def live_migration(self, context, instance, dest,
2                      post_method, recover_method, block_migration=False,
3                      migrate_data=None):
4      """Spawning live_migration operation for distributing high-load.
5
6      :param context: security context
7      :param instance:
8          nova.db.sqlalchemy.models.Instance object
9          instance object that is migrated.
10     :param dest: destination host
11     :param post_method:
12         post operation method.
13         expected nova.compute.manager._post_live_migration.
14     :param recover_method:
15         recovery method when any exception occurs.
16         expected nova.compute.manager._rollback_live_migration.
17     :param block_migration: if true, do block migration.
18     :param migrate_data: a LibvirtLiveMigrateData object
19
20     """
21
22     # 'dest' will be substituted into 'migration_uri' so ensure
23     # it doesn't contain any characters that could be used to
24     # exploit the URI accepted by libvirt
25     if not libvirt_utils.is_valid_hostname(dest):
26         raise exception.InvalidHostname(hostname=dest)
27
28     self._live_migration(context, instance, dest,
29                          post_method, recover_method, block_migration,
30                          migrate_data)
```

1.3 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：

```
1  def snapshot(self, context, instance, image_id, update_task_state):
2
3      try:
4          # 得到虚拟机的抽象对象 virt_dom
5          virt_dom = self._lookup_by_name(instance['name'])
6          ...
7
8          # 得到虚拟机的磁盘路径
9          disk_path = libvirt_utils.find_disk(virt_dom)
10
11         # 得到虚拟机的磁盘类型
12         source_format = libvirt_utils.get_disk_type(disk_path)
13
14         # 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
15         if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
16             if state == power_state.RUNNING or state == power_state.PAUSED:
17                 ...
18                 virt_dom.managedSave(0)
19
20         # snapshot_backend 也将用于静态创建快照，稍后分析
21         snapshot_backend = self.image_backend.snapshot(instance,
22                                                         disk_path,
23                                                         image_type=source_format)
24
25         with utils.tmpdir(dir=snapshot_directory) as tmpdir:
26             try:
27                 ...
28                 if live_snapshot:
29                     # 动态创建快照
30                     self._live_snapshot(virt_dom, disk_path, out_path,
31                                         image_format)
32             else:
33                 # 静态创建快照
34                 snapshot_backend.snapshot_extract(out_path, image_format)
35             ...
```

1.3.1 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。

1.3.2 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。

1.3.3 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。

1.3.4 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.tmpdir(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：

```
1 # virt_dom是libvirt的virDomain对象
2 # 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
3 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：

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

进一步看 snapshot() 函数：

```
1 def snapshot(self, instance, disk_path, image_type=None):
2     # self.backend()根据image_type返回相应的类
3     # 具体细节在此不详述，有兴趣的可以看Backend类
4     backend = self.backend(image_type)
5     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：

```
1 snapshot_backend.snapshot_extract(out_path, image_format)
2
3 # snapshot_extract()函数如下
4 class Qcow2(Image):
5     ...
6     def snapshot_extract(self, target, out_format):
7         libvirt_utils.extract_snapshot(self.path, 'qcow2',
8             target,
9             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：

```

1      def extract_snapshot(disk_path, source_fmt, out_path, dest_fmt):
2          # 函数相当于执行了 “qemu-img convert -f source_fmt -O dest_fmt
            disk_path out_path”
3          # qemu-img convert 就是用来提取快照的命令
4          qemu_img_cmd = ('qemu-img', 'convert', '-f', source_fmt, '-O',
            dest_fmt)
5          ...
6          qemu_img_cmd += (disk_path, out_path)
7          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。

1.3.5 动态创建快照

`snapshot()` 函数中动态创建快照的相关语句如下：

```

1      ...
2      with utils.tmpdir(dir=snapshot_directory) as tmpdir:
3          try:
4              ...
5              if live_snapshot:
6                  # 动态创建快照
7                  self._live_snapshot(virt_dom, disk_path, out_path,
8                                      image_format)
9              ...

```

进一步看 `_live_snapshot()` 函数：

```

1      def _live_snapshot(self, domain, disk_path, out_path, image_format):
2          # domain是虚拟机的virDomain对象
3
4          # 传入VIR_DOMAIN_XML_INACTIVE和VIR_DOMAIN_XML_SECURE参数
5          # 获得domain的xml配置文件
6          # 这个xml用于之后的undefine()和define()函数
7          xml = domain.XMLDesc(
8              libvirt.VIR_DOMAIN_XML_INACTIVE |
9              libvirt.VIR_DOMAIN_XML_SECURE)
10
11         try:
12             # 调用virDomain.blockJobAbort()停止活动块操作
13             domain.blockJobAbort(disk_path, 0)
14             ...
15             # 获得虚拟机backing_file的路径
16             src_back_path = libvirt_utils.get_disk_backing_file(disk_path,

```

```

17                                                                 basenane=False)
18     disk_delta = out_path + '.delta'
19     # 这个函数位于nova/virt/libvirt/utils.py中
20     # 函数中调用 “qemu-img create -f qcow2 -o src_disk_size,src_back_path
21     #     src_back_path” 创建镜像文件
22     libvirt_utils.create_cow_image(src_back_path, disk_delta,
23                                   src_disk_size)
24
25     try:
26         # 因为virDomain.blockRebase()不能对persistent的domain操作
27         # 所以调用virDomain.undefine()将domain变为transient
28         if domain.isPersistent():
29             domain.undefine()
30
31         # 将虚拟机镜像文件的内容拷贝到disk_delta中的镜像文件中
32         domain.blockRebase(disk_path, disk_delta, 0,
33                           libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
34                           libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
35                           libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)
36
37         ...
38
39         # 调用virDomain.blockJobAbort()终止数据拷贝
40         domain.blockJobAbort(disk_path, 0)
41         ...
42     finally:
43         # 将domain从transient变为persistent
44         self._conn.defineXML(xml)
45
46     # 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
47     libvirt_utils.extract_snapshot(disk_delta, 'qcow2',
48                                   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。

2 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:

```
pengsida@scholes:~  
pengsida@scholes:~$ python  
Python 2.7.12 (default, Nov 19 2016, 06:48:10)  
[GCC 5.4.0 20160609] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> help()  
  
Welcome to Python 2.7! This is the online help utility.  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, or topics, type "modules",  
"keywords", or "topics". Each module also comes with a one-line summary  
of what it does; to list the modules whose summaries contain a given word  
such as "spam", type "modules spam".  
  
help> |
```

2. 查看 libvirt 的帮助文档:

```
pengsida@scholes:~$ python  
Python 2.7.12 (default, Nov 19 2016, 06:48:10)  
[GCC 5.4.0 20160609] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> help()  
  
Welcome to Python 2.7! This is the online help utility.  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, or topics, type "modules",  
"keywords", or "topics". Each module also comes with a one-line summary  
of what it does; to list the modules whose summaries contain a given word  
such as "spam", type "modules spam".  
  
help> libvirt
```

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

```
pengsida@scholes:~  
class virConnect(_builtin_.object)  
    Methods defined here:  
  
    __del__(self)  
        # virConnect methods from virConnect.py (hand coded)  
  
    __init__(self, _obj=None)  
  
    allocPages(self, pages, startCell=0, cellCount=0, flags=0)  
        Allocate or free some pages in the huge pages pool  
  
    baselineCPU(self, xmlCPUs, flags=0)  
        Computes the most feature-rich CPU which is compatible with all give  
n host CPUs.  
  
    c_pointer(self)  
        Get C pointer to underlying object  
  
    changeBegin(self, flags=0)  
        This function creates a restore point to which one can return  
later by calling virInterfaceChangeRollback(). This function should  
be called before any transaction with interface configuration.  
Once it is known that a new configuration works, it can be committed  
/virConnect
```