昵称：SammyLiu
园龄：2年6个月
荣誉：推荐博客
粉丝：470
关注：30
+加关注

< 2017年5月 >

日	一	二	三	四	五	六
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
GRE(1)
Neutron(1)
Open vSwitch(1)
OpenStack(1)

随笔分类(254)
Ceilometer(3)
Ceph(13)
Cinder(6)
Docker(8)
Glance
Heat(2)
K8S
Keystone(1)
KVM(10)
MessageQueue(4)
MySQL(1)
Neutron(17)
Nova(10)
OpenStack(33)
Sahara
Storage(1)
Swift(3)
Trove
Ubuntu(3)
VMware(3)
安装和配置(1)
版本(4)
备份(1)
大数据(5)
翻译(4)
高可用（HA）(6)
基础知识(19)
监控(1)
容器(4)
容器编排
使用案例(4)
网络(8)
问题定位(3)
行业(14)
性能(4)
虚拟化(7)
原理(22)
云Cloud(29)

随笔档案(121)
2017年5月 (1)

KVM 介绍（8）：使用 libvirt 迁移 QEMU/KVM 虚机和 Nova 虚机 [Nova Libvirt QEMU/KVM Live Migration]

学习 KVM 的系列文章：

- （1）介绍和安装
- （2）CPU 和 内存虚拟化
- （3）I/O QEMU 全虚拟化和准虚拟化（Para-virtulizaiton）
- （4）I/O PCI/PCIe设备直接分配和 SR-IOV
- （5）libvirt 介绍
- （6）Nova 通过 libvirt 管理 QEMU/KVM 虚机
- （7）快照（snapshot）
- （8）迁移（migration）

1. QEMU/KVM 迁移的概念

迁移（migration）包括系统整体的迁移和某个工作负载的迁移。系统整理迁移，是将系统上所有软件包括操作系统完全复制到另一个物理机硬件机器上。虚拟化环境中的迁移，可分为静态迁移（static migration，或者 冷迁移 cold migration，或者离线迁移 offline migration）和 动态迁移（live migration，或者 热迁移 hot migration 或者 在线迁移 online migration）。静态迁移和动态迁移的最大区别是，静态迁移有明显一段时间客户机中的服务不可用，而动态迁移则没有明显的服务暂停时间。

虚拟化环境中的静态迁移也可以分为两种，一种是关闭客户机后，将其硬盘镜像复制到另一台宿主机上然后恢复启动起来，这种迁移不能保留客户机中运行的工作负载；另一种是两台宿主机共享存储系统，这时候的迁移可以保持客户机迁移前的内存状态和系统运行的工作负载。

动态迁移，是指在保证客户机上应用服务正常运行的同时，让客户机在不同的宿主机之间进行迁移，其逻辑步骤和前面的静态迁移几乎一直，有硬盘存储和内存都复制的动态迁移，也有仅复制内存镜像的动态迁移。不同的是，为了保证迁移过程中客户机服务的可用性，迁移过程只能有非常短暂的停机时间。动态迁移允许系统管理员将客户机在不同物理机上迁移，同时不会断开访问客户机中服务的客户端或者应用程序的连接。一个成功的迁移，需要保证客户机的内存、硬盘存储和网络连接在迁移到目的主机后任然保持不变，而且迁移的过程的服务暂停时间较短。

1.1 迁移效率的衡量

（1）整体迁移时间

（2）服务器停机时间：这时间是指源主机上的客户机已经暂停服务，而目的主机上客户机尚未恢复服务的时间。

（3）对服务性能的影响：客户机迁移前后性能的影响，以及目的主机上其它服务的性能影响。

其中，整体迁移时间受很多因素的影响，比如 Hypervisor 和迁移工具的种类、磁盘存储的大小（是否需要复制磁盘镜像）、内存大小及使用率、CPU 的性能和利用率、网络带宽大小及是否拥塞等，整体迁移时间一般分为几秒钟到几十分钟不等。动态迁移的服务停机时间，也有这些因素的影响，往往在几毫秒到几百毫秒。而静态迁移，其暂停时间较长。因此，静态迁移一般适合于对服务可用性要求不高的场景，而动态迁移适合于对可用性要求高的场景。

动态迁移的应用场景包括：负载均衡、解除硬件依赖、节约能源 和异地迁移。

1.2 KVM 迁移的原理

1.2.1 静态迁移

对于静态迁移，你可以在宿主机上某客户机的 QEMU monitor 中，用 savevm my_tag 命令保存一个完整的客户机镜像快照，然后在宿主机中关闭或者暂停该客户机，然后将该客户机的镜像文件复制到另一台宿主机中，使用在源主机中启动该客户机时的命令来启动复制过来的镜像，在其 QEMU monitor 中 loadvm my_tag 命令恢复刚才保存的快照即可完全加载保存快照时的客户机状态。savevm 命令可以保证完整的客户机状态，包括 CPU 状态、内存、设备状态、科协磁盘中的内存等。注意，这种方式需要 qcow2、qed 等格式的磁盘镜像文件的支持。

1.2.2 动态迁移

如果源宿主机和目的宿主机共享存储系统，则只需要通过网络发送客户机的 vCPU 执行状态、内存中的内容、虚机设备的状态到目的主机上。否则，还需要将客户机的磁盘存储发到目的主机上。共享存储系统指的是源和目的虚机的镜像文件目录是在一个共享的存储上的。

在基于共享存储系统时，KVM 动态迁移的具体过程为：

1. 迁移开始时，客户机依然在宿主机上运行，与此同时，客户机的内存页被传输到目的主机上。
2. QEMU/KVM 会监控并记录下迁移过程中所有已被传输的内存页的任何修改，并在所有内存页都传输完成后即开始传输在前面过程中内存页的更改内容。
3. QEMU/KVM 会估计迁移过程中的传输速度，当剩余的内存数据量能够在一个可以设定的时间周期（默认 30 毫秒）内传输完成时，QEMU/KVM 会关闭源宿主机上的客户机，再将剩余的数据量传输到目的主机上，最后传输过来的内存内容在目的宿主机上恢复客户机的运行状态。
4. 至此，KVM 的动态迁移操作就完成了。迁移后的客户机尽可能与迁移前一直，除非目的主机上缺少一些配置，比如网桥等。

注意，当客户机中内存使用率非常大而且修改频繁时，内存中数据不断被修改的速度大于KVM能够传输的内存速度时，动态迁移的过程是完成不了的，这时候只能静态迁移。

关于实时迁移的效率，业界不少人提出了改进的建议，比如通过使用内存压缩技术，减少需要传输的内存的大小。这篇文章比较了各种方法，还是值得一读。

1.3 使用命令行的方式做动态迁移

2017年3月 (1)
2017年1月 (1)
2016年10月 (7)
2016年9月 (5)
2016年8月 (4)
2016年7月 (1)
2016年6月 (5)
2016年5月 (1)
2016年4月 (1)
2016年3月 (9)
2016年2月 (4)
2016年1月 (2)
2015年12月 (7)
2015年11月 (7)
2015年10月 (4)
2015年9月 (4)
2015年8月 (5)
2015年7月 (9)
2015年6月 (10)
2015年5月 (3)
2015年4月 (11)
2015年3月 (2)
2015年2月 (6)
2015年1月 (5)
2014年12月 (6)

文章分类(21)
Ceph(1)
GlusterFS
Web 服务器(2)
操作系统(1)
存储
大数据(2)
分布式系统
服务器(1)
网络(11)
虚拟化(3)
云

文章档案(42)
2016年10月 (2)
2016年9月 (1)
2016年6月 (1)
2016年5月 (3)
2015年12月 (4)
2015年10月 (5)
2015年9月 (2)
2015年6月 (1)
2015年4月 (23)

积分与排名
积分 - 286831
排名 - 535

最新评论
1. Re:Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network]
eth1 - 公共网络（untagged），管理网络（tag=102），存储网络（tag=103）不好意思，大家共用同一个eth1端口的时候，请问这里交换机端口是配置为tagged还是

untagged.....
--xianke9
2. Re:理解Docker（5）：Docker 网络
1.12版本上网络的表现如何？
--幽灵狼
3. Re:理解Docker（5）：Docker 网络
我想请问一下运行docker quickstart terminal时一直卡在"waiting for an IP"应该如何解决呢？希望楼主能解答一下。
--silentbell
4. Re:理解Docker（6）：若干企业生产环境中的容器网络方案
写得好！加油。
--itbj00

1.3.1 使用 NFS 共享存储

（1）在源宿主机上挂载 NFS 上的客户机镜像，并启动客户机

```
mount my-nfs:/raw-images/ /mnt/

kvm /mnt/rh1.img -smp 2 -m 2048 -net nic -net tap
```

（2）在目的宿主机上也挂载镜像目录，并启动一个客户机用于接收动态迁移过来的内存内容

```
mount my-nfs:/raw-images/ /mnt/

kvm /mnt/rh1.img -smp 2 -m 2048 -net nic -net tap -incoming tcp:0:6666
```

注意：（1）NFS 挂载目录必须一致 （2）“-incoming tcp:0:6666” 参数表示在 6666 端口建立一个 TCP socket 连接用于接收来自源主机的动态迁移的内容，其中 0 表示运行来自任何主机的连接。“-incoming“ 使 qemu-kvm 进程进入到监听模式，而不是真正以命令行中的文件运行客户机。

（3）在源宿主机的客户机的 QEMU monitor 中，使用命令 ” migrate tcp:host2:6666" 即可进入动态迁移的流程。

1.3.2 不使用共享存储的动态迁移

过程类似，包括使用相同backing file 的镜像的客户机迁移，以及完全不同镜像文件的客户机的迁移。唯一的区别是，migrate 命令中添加 “-b” 参数，它意味着传输块设备。

1.3.3 其它 QEMU monitor migrate 命令

- migrate_cancel：取消迁移
- migrate_set_speed：设置最大迁移速度，单位是 bytes
- migrate_set_downtime：设置最大允许的服务暂停时间，单位是 秒
- info migrate：显示迁移进度

2. OpenStack Nova QEMU/KVM 实例动态迁移的环境配置

除了直接拷贝磁盘镜像文件的冷迁移，OpenStack 还支持下面几种虚机热迁移模式：

- 不使用共享存储时的块实时迁移（Block live migration without shared storage）。这种模式不支持使用只读设备比如 CD-ROM 和 Config Drive。块实时迁移不需要 nova compute 节点都使用共享存储。它使用 TCP 来将虚机的镜像文件通过网络拷贝到目的主机上，因此和共享存储式的实时迁移相比，这种方式需要更长的时间。而且在迁移过程中，主机的性能包括网络和 CPU 会下降。
- 基于共享存储的实时迁移 （Shared storage based live migration）：两个主机可以访问共享的存储。
- 从卷启动的虚机的实时迁移（Volume backed VM live migration）。这种迁移也是一种块拷贝迁移。

实时迁移的过程并不复杂，复杂在于环境配置。

2.1 基础环境配置

2.1.1 SSH 权限配置

这种方式需要配置源(compute1)和目的主机(compute2)之间能够通过 SSH 相互访问，以确保能通过 TCP 拷贝文件，已经可以通过 SSH 在目的主机建立目录。

使用 nova 用户在compute1 上执行操作：

```
usermod -s /bin/bash nova
su nova
mkdir -p -m 700 .ssh

#创建 config 文件如下
nova@compute2:~/.ssh$ cat config
Host *
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null

#产生 key
ssh-keygen -f id_rsa -b 1024 -P ""
cat id_rsa.pub >> authorized_keys

#将 id_rsa id_rsa.pub 拷贝到 compute2 上面
cat id_rsa.pub >> authorized_keys
```

使用 root 用户在每个主机上进行操作：

```
root@compute1:/var/lib/nova/.ssh# chown -R nova:nova /var/lib/nova
root@compute1:/var/lib/nova/.ssh# chmod 700 /var/lib/nova/.ssh
root@compute1:/var/lib/nova/.ssh# chmod 600 /var/lib/nova/.ssh/authorized_keys
```

测试 SSH 无密码访问：

```
nova@compute1:~/.ssh$ ssh nova@compute2 ls
Warning: Permanently added 'compute2,192.168.1.29' (ECDSA) to the list of known hosts.
...

nova@compute2:~/.ssh$ ssh nova@compute1 ls
Warning: Permanently added 'compute1,192.168.1.15' (ECDSA) to the list of known hosts.
...
```

2.1.2 其它配置

每个node 上的 DNS 或者 /etc/hosts，确保互联互通。

2.2 Live migration 环境配置

2.2.1 libvirtd 配置

在 compute1 和 compute2 上做如下配置：

```
->Edit /etc/libvirt/libvirtd.conf
listen_tls = 0
listen_tcp = 1
auth_tcp = “none”

->Edit /etc/init/libvirt-bin.conf
env libvirtd_opts="-d -l"
->Edit /etc/default/libvirt-bin

  # options passed to libvirtd, add "-l" to listen on tcp
  libvirtd_opts="-d -l"

->Restart libvirtd
service libvirt-bin restart

root     12088     1  2 07:48 ?        00:00:00 /usr/sbin/libvirtd -d -l
```

做完上述操作后，可以使用如下命令来检查是否设置正确：

```
root@compute2:~# virsh -c qemu+tcp://compute1/system list  --all
 Id    Name                           State
----------------------------------------------------
 4     instance-0000000d              running
 5     instance-00000006              running
 -     instance-00000005              shut off

root@compute1:~# virsh -c qemu+tcp://compute2/system list  --all
 Id    Name                           State
----------------------------------------------------
```

Nova 设置：

```
->Edit /etc/nova/nova.conf, add following line:
  [libvirt]
  block_migration_flag = VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,
VIR_MIGRATE_LIVE,VIR_MIGRATE_TUNNELLED,VIR_MIGRATE_NON_SHARED_INC
  live_migration_flag = VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER,
VIR_MIGRATE_LIVE,VIR_MIGRATE_TUNNELLED
  live_migration_uri = qemu+tcp://%s/system
```

2.2.2 共享存储 Live migration 环境配置

其实共享存储的实时迁移配置的要求和块拷贝的实时迁移的配置差不多，除了下面几点：

1. Hypervisor 要求：目前只有部分 Hypervisor 支持 live migraiton，可查询该表。
2. 共享存储：存放虚机文件的文件夹 *NOVA-INST-DIR*/instances/ (比如 /var/lib/nova/instances，该路径可以由 state_path 配置变量来配置) 必须是挂载到共享存储上的。当Nova 使用 RBD 作为镜像的backend时，这个要求不是必须的，具体见下面的说明。
3. 必须在 nova.conf 中配置 vncserver_listen=0.0.0.0 (关于这个，社区认为这个配置具有安全风险，会通过这个 ticket 来解决)
4. 不使用默认配置的话，必须在每个 nova compute 上的 nova.conf 中配置相同的 instances_path 和 state_path。
5. 在 Kilo 版本之前，Nova 是默认不支持 live migriation 的。在做实时迁移之前，需要在 nova.conf 中做如下配置

```
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,VIR_MIGRATE_TUNNELLED
```

注意：对于上面第二点，在 Kilo 版本中（前面版本的情况未知），当 Nova 使用 RBD 作为 image backend 时，Nova 会认为是在共享存储上：

```
def check_instance_shared_storage_local(self, context, instance):
        """Check if instance files located on shared storage."""
        if self.image_backend.backend().is_shared_block_storage():
            return None
```

在 class Rbd(Image): 类中：

```
 @staticmethod
    def is_shared_block_storage():
        """True if the backend puts images on a shared block storage."""
        return True
```

目前，只有 RBD 作为 image backend 时，该函数才返回 true。对于其它类型的 backend，Nova 会在目的 host 上的 instance folder 创建一个临时文件，再在源 host 上查看该文件，通过判断是否该文件在共享存储上来判断是否在使用共享存储。

常见问题：

（1）在源 host 上，出现 "live Migration failure: operation failed: Failed to connect to remote libvirt URI qemu+tcp://compute2/system: unable to connect to server at 'compute2:16509': Connection refused"

其原因是 2.1.1 部分的 libvirt 设置不正确。

（2）在目的 host 上，出现 "libvirtError: internal error: process exited while connecting to monitor: 2015-09-21T14:17:31.840109Z qemu-system-x86_64: -drive file=rbd:vms/6bef8898-85f9-429d-9250-9291a2e4e5ac_disk:id=cinder:key=AQDaoPpVEDJZHhAAu8fuMR/OxHUV90Fm1MhONQ==:auth_supported=cephx\;none:mon_host=9.115.251.194\:6789\;9.115.251.195\:6789\;9.115.251.218\:6789,if=none,id=drive-virtio-disk0 format=raw cache=writeback discard=unmap: could not open disk image rbd:vms/6bef8898-85f9-429d-9250-

9291a2e4e5ac_disk:id=cinder:key=AQDaoPpVEDJZHhAAu8fuMR/OxHUV90Fm1MhONQ==:auth_supported=cephx\;none:mon_host=9.115.251.194\:6789\;9.115.251.195\:6789\;9.115.251.218\:6789: Could not open 'rbd:vms/6bef8898-85f9-429d-9250-9291a2e4e5ac_disk:id=cinder:key=AQDaoPpVEDJZHhAAu8fuMR/OxHUV90Fm1MhONQ==:auth_supported=cephx\;none:mon_host=9.115.251.194\:6789\;9.115.251.195\:6789\;9.115.251.218\:6789': Operation not permitted"

原因：目的 host 上的用户操作 RBD 的权限设置不正确，检查 secret 设置。

3. 迁移过程

3.0 Nova 有关迁移的命令

Nova 有三个与迁移有关的命令：migrate，live-migrate 和 resize。

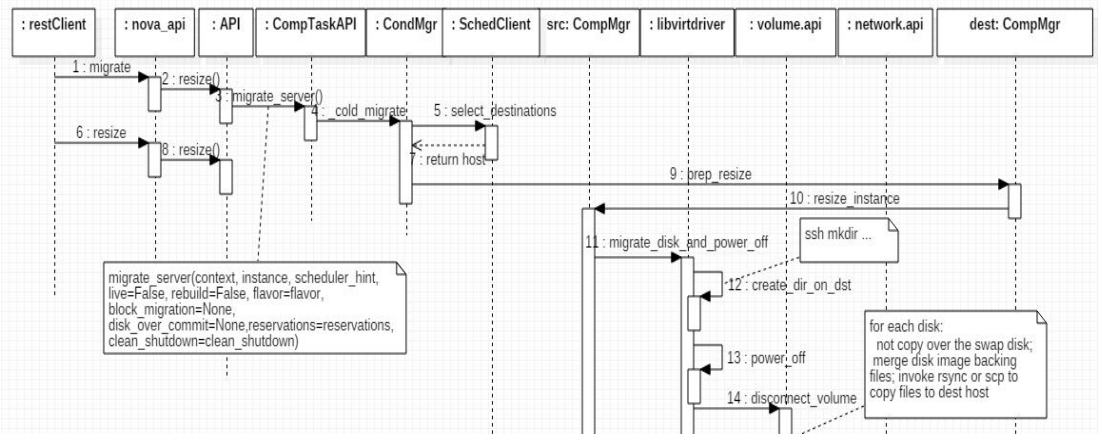
Nova CLI	REST API Action	行为
nova live-migration --block-migrate --disk_over_commit 8352e969-0a25-4abf-978f-d9d0ec4de0cd compute2	os-migrateLive	块拷贝动态迁移
nova live-migration 8352e969-0a25-4abf-978f-d9d0ec4de0cd compute2	os-migrateLive	共享存储动态迁移
nova migrate 8352e969-0a25-4abf-978f-d9d0ec4de0cd	migrate	静态迁移
nova resize --poll 8352e969-0a25-4abf-978f-d9d0ec4de0cd 1	resize	静态迁移并且改变 flavor
nova resize --poll 8352e969-0a25-4abf-978f-d9d0ec4de0cd	resize	静态迁移
nova resize-confirm 9eee079e-0353-44cb-b76c-ecf9be61890d	confirmResize	确认 resize 使得完整操作得以完成
nova resize-revert 9eee079e-0353-44cb-b76c-ecf9be61890d	revertResize	取消 resize 使得操作被取消虚机回到原始状态

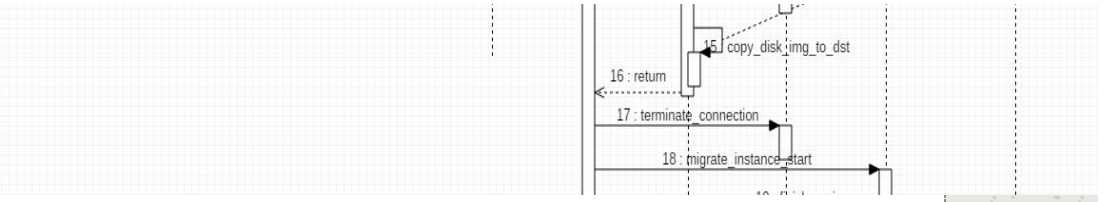
3.1 静态迁移（migrate 或者 resize 不使用新的 flavor）

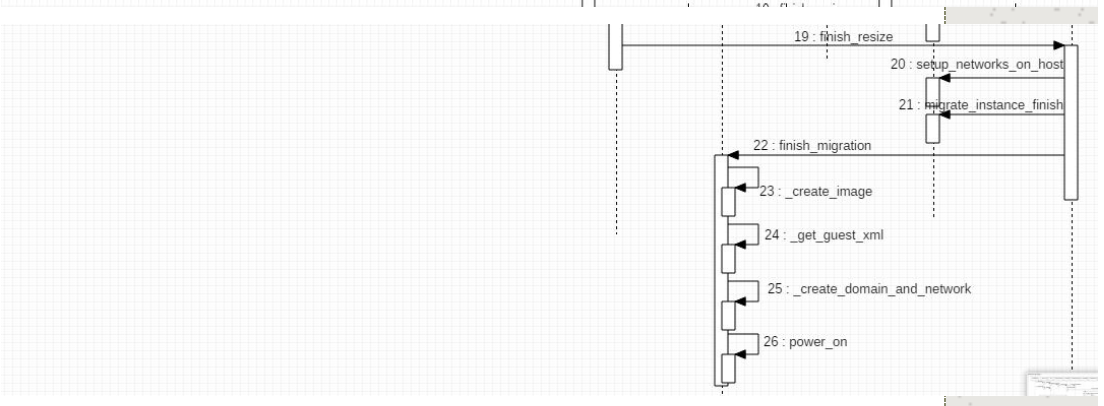
```
s1@controller:~$ nova migrate --poll 9eee079e-0353-44cb-b76c-ecf9be61890d

Server migrating... 100% complete
Finished
s1@controller:~$ nova list
+--------------------------------------+-------+---------------+------------+-------------+------------
-----------------------------+
| ID                                   | Name  | Status        | Task State | Power State | Networks
|
+--------------------------------------+-------+---------------+------------+-------------+------------
-----------------------------+
| 02699155-940f-4401-bc01-36220db80639 | vm10  | ACTIVE        | -          | Running     | demo-
net2=10.0.10.17; demo-net=10.0.0.39 |
| 9eee079e-0353-44cb-b76c-ecf9be61890d | vm100 | VERIFY_RESIZE | -          | Running     | demo-
net2=10.0.10.20                     |
+--------------------------------------+-------+---------------+------------+-------------+------------
-----------------------------+
s1@controller:~$ nova resize-confirm 9eee079e-0353-44cb-b76c-ecf9be61890d
s1@controller:~$ nova list
+--------------------------------------+-------+--------+------------+-------------+------------------
----------------------+
| ID                                   | Name  | Status | Task State | Power State | Networks
|
+--------------------------------------+-------+--------+------------+-------------+------------------
----------------------+

| 02699155-940f-4401-bc01-36220db80639 | vm10  | ACTIVE | -          | Running     | demo-
net2=10.0.10.17; demo-net=10.0.0.39 |
| 9eee079e-0353-44cb-b76c-ecf9be61890d | vm100 | ACTIVE | -          | Running     | demo-
net2=10.0.10.20                     |
+--------------------------------------+-------+--------+------------+-------------+------------------
----------------------+
```

3.1.1 迁移过程

直接使用流程图来说明：







1. migrate 和 resize 都是执行静态迁移。

2. 静态迁移分为三个阶段：

（1）调用 Scheduler 算法选择目的 node（步骤5），并通过 RPC 远程调用 prep_resize 做些迁移前的准备工作

（2）在源主机上，调用 libvirt driver 做一系列操作：

1. 使用 ssh 在目的 node 上建立虚机的镜像文件的目录
2. 将虚机关机
3. 断开所有 volume connections
4. 针对每一个非 swap 分区的磁盘，如果是 qcow2 格式，则执行 qemu-img merge 操作将稀疏文件和backing 文件合并成单个文件，并通过 “rysnc” 或者 “scp”命令将文件拷贝到目的 node 上
5. 开始迁移需要的网络工作

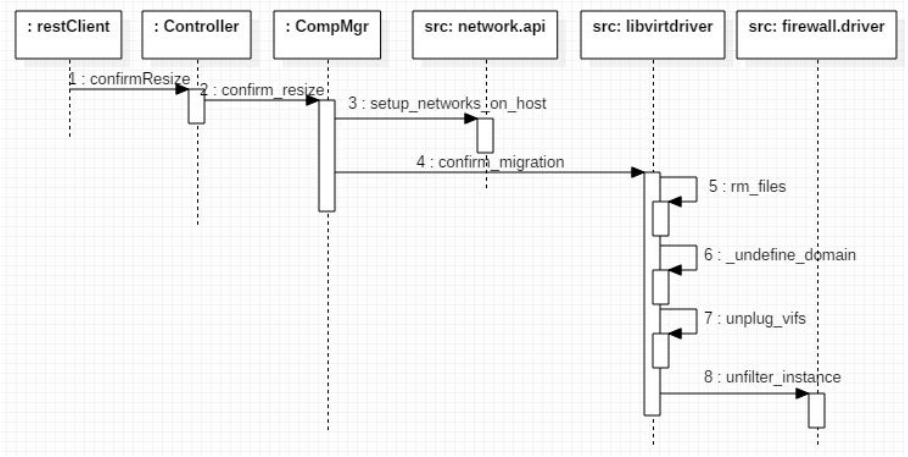
（3）通过 RPC，调用目的 node 上的 Nova 的 finish_resize 方法。该方法会在自己本机上设置网络、结束网络设置工作，并调用 libvirt driver 来：

1. 创建 image
2. 获取 guest xml
3. 创建 domain 和 network
4. 需要的话启动虚机

至此，虚机已经被拷贝到目的主机上了。接下来，用户有两个选择：resize_confirm 和 resize_revert。

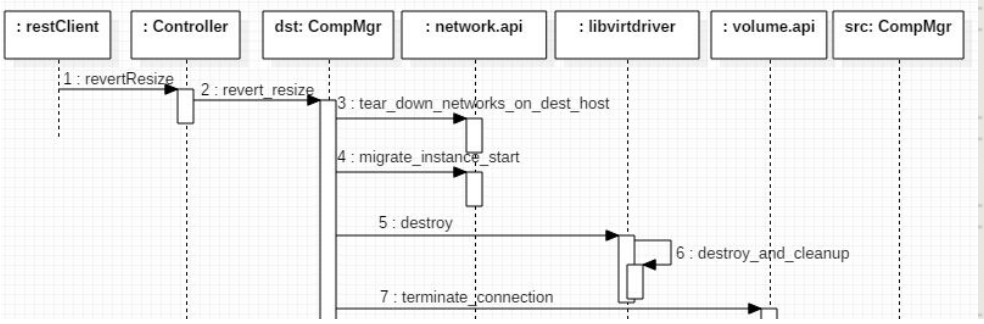
3.1.2 确认迁移（resize_confirm）

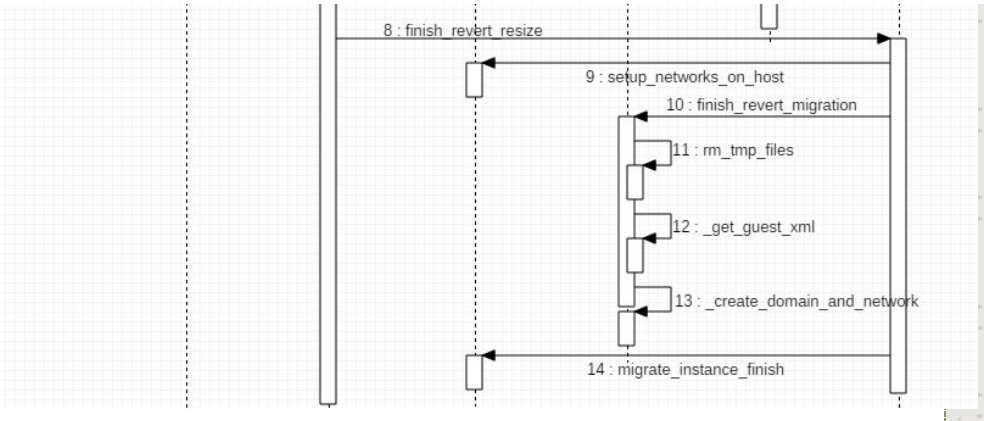
迁移确认后，在源主机上，虚机的文件会被删除，虚机被 undefine，虚机的 VIF 被从 OVS 上拔出，network filters 也会被删除。



3.1.3 取消迁移（resize_revert）

取消迁移的命令首先发到目的 node 上，依次 tear down network，删除 domain，断掉 volume connections，然后调用源主机上的方法来重建 network，删除临时文件，启动 domain。这样，虚机就会需要到 resize 之前的状态。





3.2 实时迁移（Live migration）

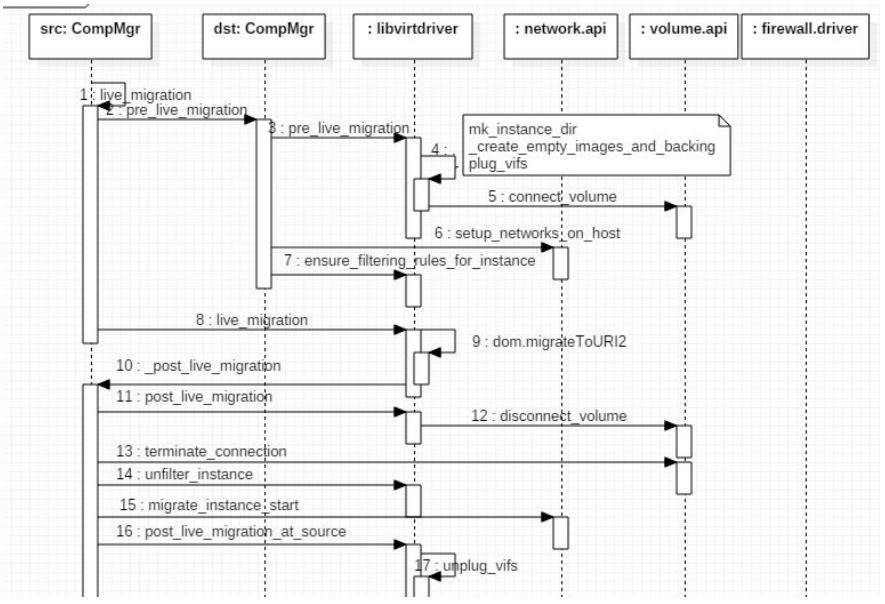
可以 Nova client 的 live-migration 命令来做实时迁移，除了要被迁移的 虚机 和 目的 node 外，它可以带两个额外的参数：

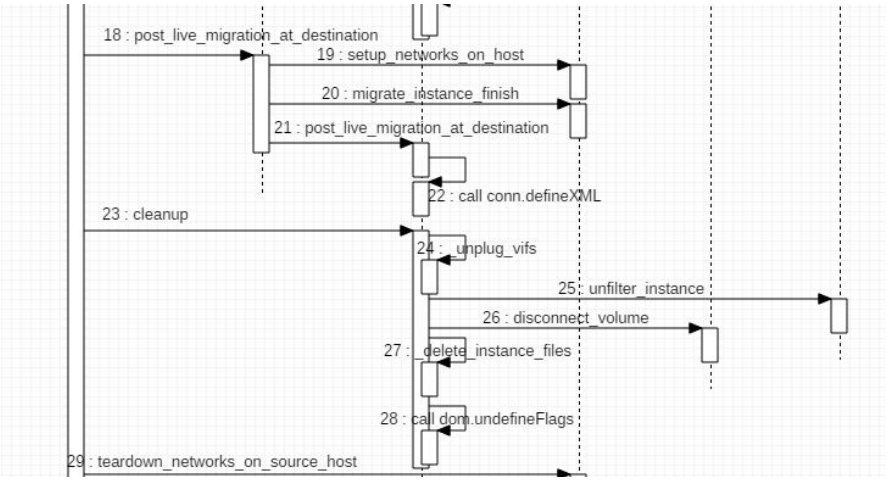
- “block-migrate“：使用的话，做 block copy live migration；不使用的话，做共享存储的 live migration；
- ”disk_over_commit“：使用的话，计算所有磁盘的 disk_size 来计算目的 node 上所需空间的大小；不使用的话，则计算磁盘的 virtual size。在下面的例子中，如果使用 disk_over_commit，那么计算在目的主机上需要的空间的时候，该 disk 的 size 为 324k，否则为 1G：

```
root@compute1:/home/s1# qemu-img info /var/lib/nova/instances/8352e969-0a25-4abf-978f-
d9d0ec4de0cd/disk.local
image: /var/lib/nova/instances/8352e969-0a25-4abf-978f-d9d0ec4de0cd/disk.local
file format: qcow2
virtual size: 1.0G (1073741824 bytes)
disk size: 324K
cluster_size: 65536
backing file: /var/lib/nova/instances/_base/ephemeral_1_default
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

REST API request body 实例： {"os-migrateLive": {"disk_over_commit": false, "block_migration": true, "host": "compute2"}}

实时迁移的主要步骤如下：





其过程也可以分为三个阶段：

3.2.1 实时迁移前的准备工作（步骤 2 - 7）

Nova 通过 RPC 调用目的主机上 nova comute manager 的 pre_live_migration 方法，它依次：

（1）准备 instance 目录：

（1）创建 instance dir

（2）如果源和目的虚机不共享 instance path：获取镜像类型，为每一个disk，如果不使用 backing file 的话则调用 “qemu-img create” 方法来创建空的磁盘镜像；否则，依次创建空的 Ephemeral disk 和 Swap disk，以及从 Glance 中获取 image 来创建 Root disk

（3）如果不是 block migration 而且 不 is_shared_instance_path，则 _fetch_instance_kernel_ramdisk

（2）调用 volumer driver api 为每一个volume 建立目的主机和 volume 的连接

（3）调用 plug_vifs(instance, network_info) 将每一个 vif plug 到 OVS 上

（4）调用 network_api.setup_networks_on_host 方法，该方法会为迁移过来的虚机准备 dhcp 和 gateway；

（5）调用 libvirt driver 的 ensure_filtering_rules_for_instance 方法去准备 network filters。

3.2.2 调用 libvirt API 开始迁移虚机（步骤 8 - 9）

这部分的实现在 libvirt driver 代码中。因为 libvirt 的一个 bug（说明在这里），当 libvirt 带有 VIR_DOMAIN_XML_MIGRATABLE flag 时，Nova 会调用 libvirt 的 virDomainMigrateToURI2 API，否则调用 virDomainMigrateToURI API。

首先比较一下 block live migration 和 live migration 的 flags 的区别：

```
#nova block live migration flags：VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER, VIR_MIGRATE_LIVE,
VIR_MIGRATE_TUNNELLED, VIR_MIGRATE_NON_SHARED_INC
#nova live migration flags：      VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER, VIR_MIGRATE_LIVE,
VIR_MIGRATE_TUNNELLED
```

各自的含义如下：

- VIR_MIGRATE_UNDEFINE_SOURCE： 迁移完成后，将源虚机删除掉。（If the migration is successful, undefine the domain on the source host.）
- VIR_MIGRATE_PEER2PEER 和 VIR_MIGRATE_TUNNELLED 一起使用：点对点迁移，必须指定目的 URI，QEMU在两者之间建立 TCP Tunnel 用于数据传输
- VIR_MIGRATE_LIVE： 执行 live migration，不要停机（Do not pause the VM during migration）
- VIR_MIGRATE_NON_SHARED_INC： 使用非共享存储式迁移即 block migration（Migration with non-shared storage with incremental disk copy）

再看看两个 API 的参数：

```
int    virDomainMigrateToURI2        (virDomainPtr domain,
                      const char * dconnuri, # 格式为 qemu+tcp://<desthost>/system
                      const char * miguri, #为 none
                      const char * dxml, #指定迁移后的虚机的 XML。Nova 对 “/devices/graphics” 部分做了一点点更
改。
                      unsigned long flags, # nova.conf 中的配置
                      const char * dname, #none
                      unsigned long bandwidth)  # 由 CONF.libvirt.live_migration_bandwidth 指定，默认为 0
表示由 libvirt 自己选择合适的值
```

如果 libvirt 不带 VIR_DOMAIN_XML_MIGRATABLE flag，则调用的 API 是：

```
int    virDomainMigrateToURI        (virDomainPtr domain,
                      const char * duri,
                      unsigned long flags,
                      const char * dname,
                      unsigned long bandwidth)
```

可见，两个 API 唯一的区别是不能指定新的虚机使用的 XML 配置。这时候你必须手动配置 VNC 或者 SPICE 地址为 0.0.0.0 or ::（接收全部）或者 127.0.0.1 or ::1（只限本机）。

调用 API 后，接下来就是等待其完成。这其中的过程应该主要包括：

（1）根据传入的 domain xml，启动一个虚机，它处于等待 TCP incoming 状态

（2）从源 node 上将 domain 的数据传过来

（3）快完成时，关闭源 node 上的虚机，传输最后一次数据，打开目的 node 上的虚机

（4）将源 node 上的虚机删除

Nova 每个0.5 秒检查源虚机的状态，直到它被删除。

迁移完成后，需要执行后续的操作（_post_live_migration）。

3.2.3 迁移完成后在源和目的主机上的后续操作（步骤 10 -29）

在源主机上，依次执行下面的操作：

1. 调用 volume driver 的 disconnect_volume 方法和 terminate_connection 方法，断开主机和所有 volume 的连接
2. 调用 firewall driver 的 unfilter_instance 方法，删除 domain 的 iptables 中的所有 security group ingress 规则（self.iptables.ipv4['filter'].remove_chain(chain_name)）
3. 调用 network api 的 migrate_instance_start 方法，开始将网络从源主机上迁移到目的主机上（实际上没做什么事情，只是 pass）
4. 调用 vif driver 的 unplug 方法，将每个 vif 从 OVS 上删除

```
brctl delif qbr59cfa0b8-2f qvb59cfa0b8-2f
```

```
ip link set qbr59cfa0b8-2f down
brctl delbr qbr59cfa0b8-2f
ovs-vsctl --timeout=120 -- --if-exists del-port br-int qvo59cfa0b8-2f
ip link delete qvo59cfa0b8-2f
```

5. 通过 RPC 调用目的主机上的 nova manager 的 post_live_migration_at_destination 方法，该方法会：
    1. 调用 network api 的 setup_networks_on_host 方法来设置网络（处理 vpn，dhcp，gateway）
    2. 调用 network api 的 migrate_instance_finish 方法
    3. 调用 libvirt driver 的 post_live_migration_at_destination方法，它会调用 libvirt _conn.listDefinedDomains 方法查看迁移过来的主机的 domain是否存在；不存在的话，生成其 xml，然后调用 libvirt API _conn.defineXML(xml) 去定义该 domain。
    4. 将新的 domain 数据更新到数据库（包括新的 host，power_state，vm_state，node）
    5. 调用 network api 的 setup_networks_on_host 方法（不理解为什么重复上面第1步）
6. 调用 libvirt driver 的 driver.cleanup 方法去 _unplug_vifs（如果上面第四步失败，则再次尝试删除所有 vif 相关的 bridge 和 OVS 连接），firewall_driver.unfilter_instance（和上面第2步重复了），_disconnect_volume（断开 domain 和 所有 volume 的连接），_delete_instance_files（删除 domain 相关的文件），_undefine_domain（删除 domain）
7. 调用 network_api.setup_networks_on_host 去 tear down networks on source host
8. 至此，live migration 完全结束。

3.2.4 迁移过程中失败时的回滚

迁移的三个步骤中，前面第一个和第二个步骤中出现失败的话，会调用 _rollback_live_migration 启动回滚操作。该方法

（1）将虚机的状态由 migrating 变为 running。

（2）调用 network_api.setup_networks_on_host 方法重做源主机上的网络设置

（3）通过 RPC 调用，去目的主机上将准备过程中建立的 volume 连接删除。

（4）通过 RPC 调用，去目的主机上调用 compute_rpcapi.rollback_live_migration_at_destination 函数，该方法会

（1）调用 network_api.setup_networks_on_host 方法去 tear down networks on destination host

（2）调用 libvirt driver 的 driver.rollback_live_migration_at_destination 方法，它会将 domain 删除，并且清理它所使用的资源，包括 unplug vif，firewall_driver.unfilter_instance，_disconnect_volume， _delete_instance_files， _undefine_domain。

3.2.5 测试

环境：准备两个虚机 vm1 和 vm2，操作系统为 cirros。打算将 vm1 迁移到另一个node 上。在 vm2 上查看 vm1 在迁移过程中的状态。

迁移前：在 vm1 中运行 “ping vm2”，并在 vm2 中 ssh 连接到 vm1。

结果：vm1 迁移过程中，vm2 上 ssh 的连接没有中断，vm1 中的 ping 命令继续执行。在另一次测试结果中，vm2 ping vm1 在整个迁移过程中 time 出现了一次 2ms 的时间增加。

3.3 遇到过的问题

3.3.1 apparmor

将虚机从 compute1 迁移到 compute2 成功，再从 compute2 迁移到 compute1 失败，报错如下：

```
An error occurred trying to live migrate. Falling back to legacy live migrate flow. Error: unsupported
configuration: Unable to find security driver for label apparmor
```

经比较迁移前后的虚机的 xml，发现 compute2 上的虚机的 xml 多了一项：<seclabel type='none' model='apparmor'/> 。

分别在 compute 1 和 2 上运行 “virsh capabilities”，发现 compute1 没有使用 apparmor，而 compute2 使用了 apparmor。

```
#compute 1 上
    <secmodel>
      <model>none</model>
      <doi>0</doi>
    </secmodel>

#compute2 上
    <secmodel>
      <model>apparmor</model>
      <doi>0</doi>
    </secmodel>
```

最简单的方法是在两个 node 上都 disable apparmor（在 /etc/libvirt/qemu.conf 中添加 ‘security_driver = “none” 然后重启 libvirtd），然后 destroy/start 虚机后，它的 xml 配置中的 apparmor 就没有了。这篇文章 详细介绍了 apparmor。

3.3.2 当虚机是 boot from volume 时，live migration 失败。

报错：

```
Command: iscsiadm -m node -T iqn.2010-10.org.openstack:volume-26446902-5a56-4c79-b839-a8e13a66dc7a -p
10.0.2.41:3260 --rescan
Exit code: 21
Stdout: u''
Stderr: u'iscsiadm: No session found.\n' to caller
```

原因是 cinder 代码中有 bug，导致目的主机无法建立和 volume 的连接。fix 在这里。

参考文档：

https://www.mirantis.com/blog/tutorial-openstack-live-migration-with-kvm-hypervisor-and-nfs-shared-storage/

http://www.sebastien-han.fr/blog/2015/01/06/openstack-configure-vm-migrate-nova-ssh/

KVM 原理技术 实战与原理解析 任永杰、单海涛著

OpenStack 官网

分类: KVM,Nova

SammyLiu
关注 - 30
粉丝 - 470
荣誉：推荐博客
+加关注

« 上一篇：KVM 介绍（7）：使用 libvirt 做 QEMU/KVM 快照和 Nova 实例的快照（Nova Instances Snapshot Libvirt）
» 下一篇：Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network]

posted on 2015-06-24 09:51 SammyLiu 阅读(5754) 评论(3) 编辑 收藏

评论:

#1楼 2016-05-23 14:38 | 泉水叮~咚
楼主请问以下需求：
将已有KVM hypervisor环境中的vm迁移到OpenStack环境中，这个可以做热迁移吗？网上搜了下，好像说只能冷迁移。
http://stackoverflow.com/questions/11383526/migrating-vms-from-kvm-to-openstack

支持(0) 反对(0)

#2楼[楼主] 2016-05-24 09:15 | SammyLiu
@ 泉水叮~咚
你好！我认为现在通过OpenStack是无法实现的，因为热迁移需要Nova通过libvirt去控制两边的KVM Hypervisor，而另外的KVM hypervisor环境是不受Nova控制的。即使你手工通过命令把虚机热迁移过去，但是怎么让OpenStack接受这个虚机，目前没有标准做法，但是如果一定要做，可能有hacking方法，比如先通过Nova 创建一个虚机，然后修改数据库指向你迁移过来的虚机这种。我觉得这个需求是有效的，将来社区应该在这种事情上做些事情，包括迁移、备份之类的功能。

支持(0) 反对(0)

#3楼 2016-05-24 09:37 | 泉水叮~咚
@ SammyLiu
好的，谢谢：）

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。