VMThunder 安装与使用指南

当前版本：VMThunder stable/0.3

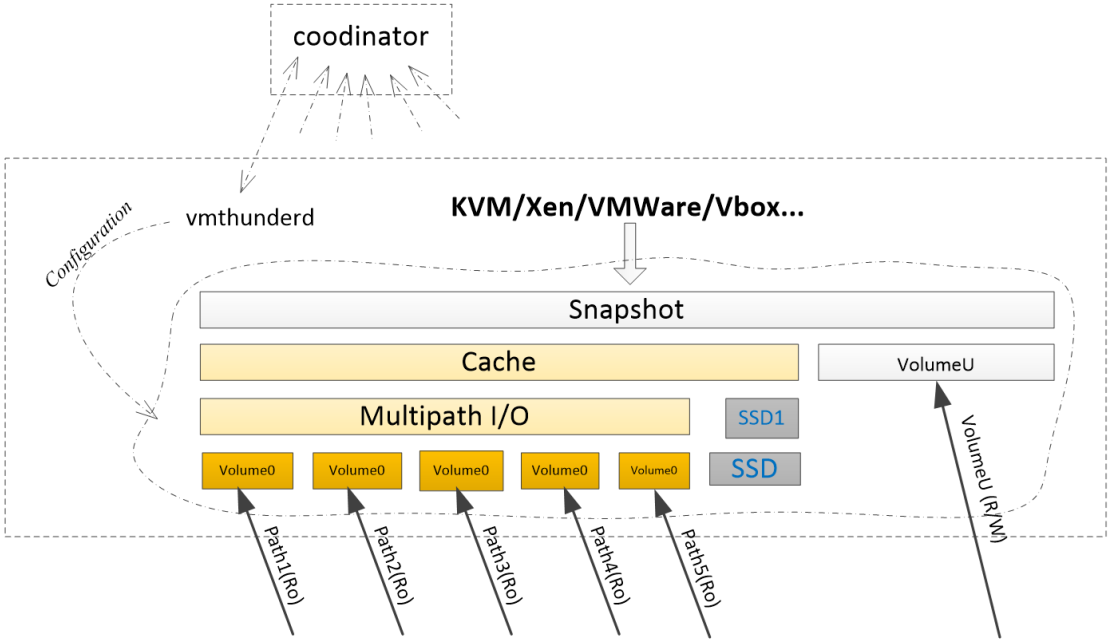
Jiaxin Li, Xiang Zhao, Zhihui Sun, Ziyang Li@PDL 10/26/2014

概述

本文档详细介绍了 VMThunder 系统的安装与部署过程，以及使用方法。同时，文档还对 VMThunder 系统的工作原理、源码结构、API 接口和存在的问题等进行了说明。

工作原理

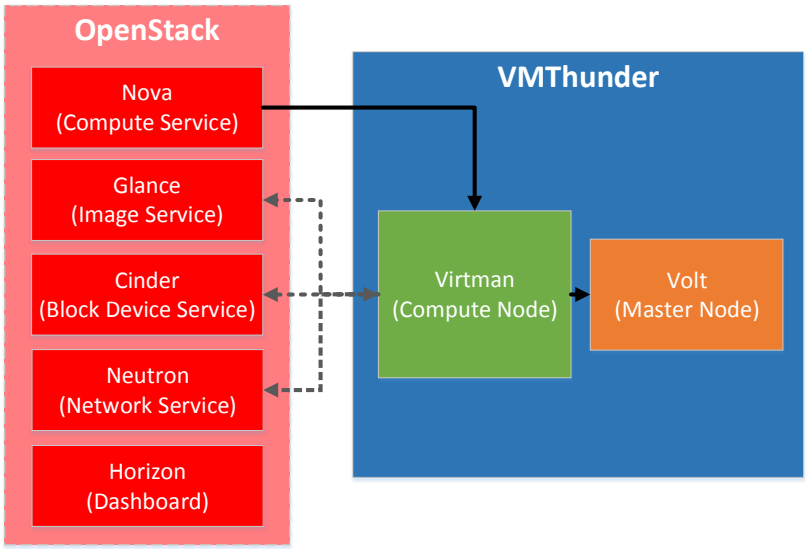
VMThunder 主要是通过在各个节点中缓存 image 数据，这些缓存的数据不仅可以在本机使用，也可以被其他节点读取。快照的存在保证了缓存里面的数据始终是干净的，而多路径设备则可以将 IO 请求分散到多个节点上，避免了单点的性能瓶颈和故障问题。



VMThunder 把 image 通过 iSCSI 协议挂载到本地，为该 image 创建多路径和缓存设备，然后创建快照，最后在快照上启动虚拟机。

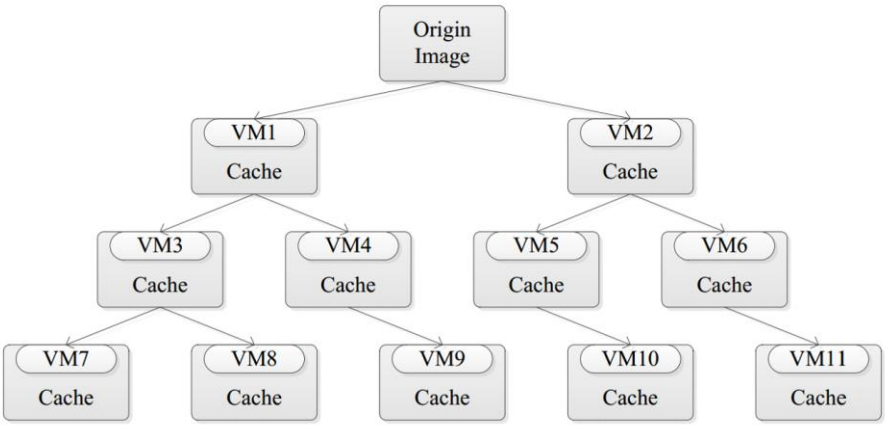
系统结构图

为了保证系统的独立性与模块的功能性，VMThunder 系统只实现 VMs 的快速部署功能，通过与开源云计算平台 Openstack 的整合，利用 Openstack 已有的镜像服务（Glance）、块存储服务（Cinder）和网络服务（Neutron）以及界面服务（Dashboard），以达到高功能性、易使用性。VMThunder 与 Openstack 整合的系统架构图如下：



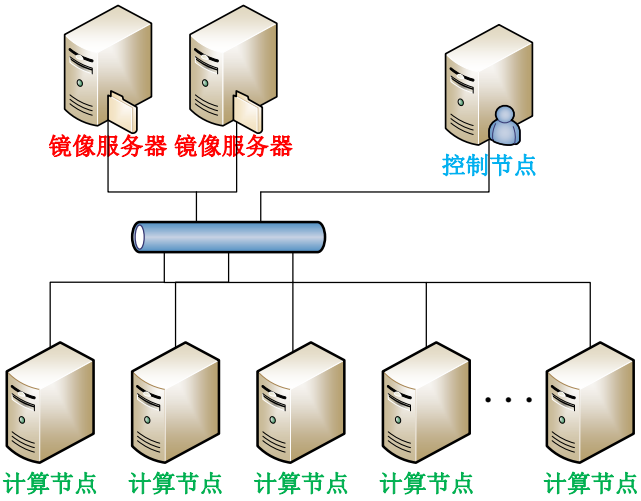
整个系统通过 Openstack 系统界面（Horizon）对 VMThunder 进行管理。Nova 通过调用 VMThunder API 负责初始化和管理 Virtman，Openstack 的 Glance、Cinder、Neutron 通过与 Virtman 交互为 VMThunder 提供镜像服务、块设备服务和网络服务。Virtman 通过与 Volt 进行交互，实时维护、动态调整 VMThunder 系统的网络结构。

VMThunder 系统采用简化的 P2P 网络结构，形成了以二叉树-树形网络结构，系统的组织结构如下。（简化版，实际结构支持多镜像节点，多父节点）

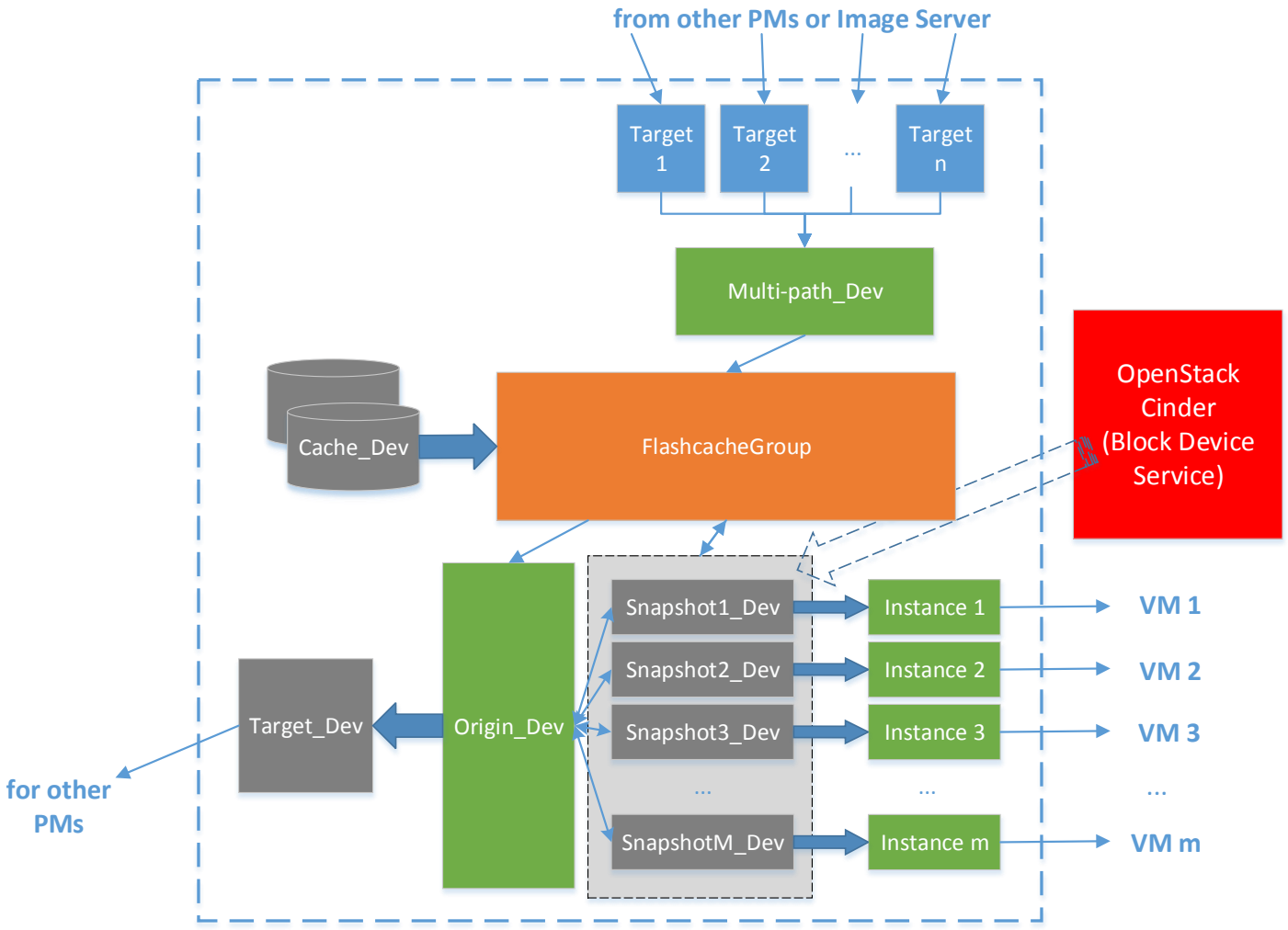


上图中的第一层节点为镜像服务器节点（Imager Server，可以多个），第二层至最底层节点为计算节点（Compute Node）。每个计算节点缓存一份 Cache 数据供下游节点使用。

VMThunder 系统涉及到的物理节点包括：控制节点（Master Node）、镜像服务器（Image Server）和计算节点（Compute Node）。系统的结构组织如下：



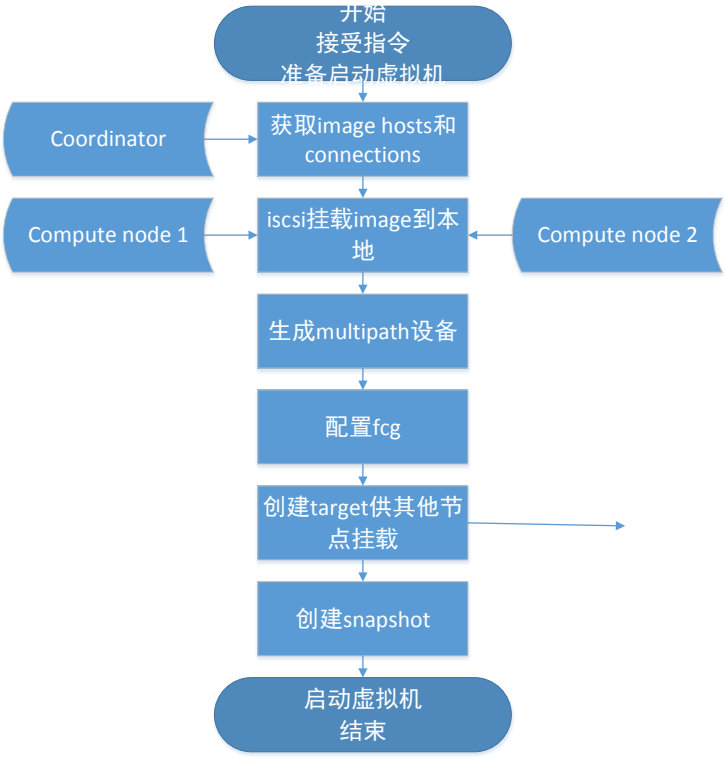
VMThunder 系统中，单个计算节点的内部体系结构图如下：



VMthunder 系统在部署、配置和启动虚拟机时的主要工作流程具体如下：

1. 从 master（即 volt）获取其他节点的缓存位置信息。通过 iSCSI 协议，将 Image 或其他节点的缓存挂载到本地。
2. 可能会挂载过来了多个设备，这些设备实际上是通向 image 的多条路径，为这些设备配置多路径。
3. 为多路径设备添加缓存。
4. 给新生成的缓存创建 Target，供其他设备挂载。向 master 注册该 target，这样其他节点就可以通过 volt 来获得该 target 的信息。
5. 创建快照。
6. 在快照上启动虚拟机。

其具体工作流程图如下：



系统的可靠性和稳定性

VMThunder 系统采用四大措施来保证系统的可靠性和稳定性。

1. 系统网络动态调整（Network Rebuild）

注：已实现

控制节点（Master Node）根据计算节点的变化（新增节点、关闭节点、节点出现故障等）自动调整 iSCSI 树形网络结构，以保证结构平衡、链路可靠。

2. 多路径支持（Multipath）

注：已实现

在 VMThunder 系统上进行数据传输的 iSCSI 树形结构中，每个计算节点（Compute Node）拥有多个上级父节点，可以从多条路径获取数据，以保证在部分节点或链路出现故障时，下游计算节点仍然可以正常获取数据，系统仍然可以正常运行。

3. 数据空闲时预取（Data Prefetching）

注：当前未实现

在虚拟机规模性快速启动完毕之后，系统能够根据网络状态情况，在网络较空闲时段，在后台自动下载镜像数据到本地计算节点，即镜像数据后台自动下载（可按照数据可能的先后读取顺序），直至完成整个镜像数据的传输。（完全可以同“数据实时预取”一起实现）

4. 系统网络动态重构（Network Reconstruction）

注：当前未实现

在虚拟机规模性快速启动完毕之后，为了提高 VMThunder 系统的稳定性，可以压缩树形网络的深度，形成具有多个超级计算节点的高度为 3 的多叉树结构。

源码结构

VMThunder

packages

VMThunder 工程项目的所有模块都放在一个 packages/目录下，该目录下主要包括如下几个部分。

（1） brick 模块：iscsi 服务模块（从 openstack/cinder 中独立出来）

（2） flashcache 模块：cache 服务模块（来源于 facebook/flashcache）

（3） volt 和 python-voltclient 是控制节点模块（MasterNode/Coordinator）

（4） virtman 和 python-virtman 是计算节点模块（ComputeNode）

a) 依赖包 pydm：用于设备映射的相关模块，已在 PyPi 中维护

b) 依赖包 flashcachegroup：用于创建缓存组的相关模块，已在 PyPi 中维护。

（5） VMT-demo/ ：VMThunder 独立版的运行、测试、使用脚本

在 packages 目录下有三个脚本：

（1）install.sh 安装 VMThunder 和所依赖的软件包

（2）remove.sh 卸载 VMThunder 和所依赖的软件包 （目前尚不完善）

（3）run-volt.sh 启动 volt 服务

VMT-demo

VMT-demo/	测试代码及 API 启动服务
./image.cfg	基础镜像配置项
./nodelist.cfg	虚拟机启动配置项
./vmt-initenv.sh	计算节点和镜像节点初始化脚本：Cache、snapshot 和 image 建立脚本
./vmt-reinstall.sh	VMThunder 计算节点卸载重装脚本
./vmtapi.py	VMThunder 的 API 服务接口，放置于客户端
./vmtimg.py	镜像 Target 管理服务：根据 image.cfg 创建、销毁 image target
./vmtrun.py	运行 VMThuner：根据 nodelist 创建 VMs

	的启动块设备

Virtman

源码结构：

文件路径	功能
bin/virtmanserver.py	Virtman 的 API 服务器，监听服务请求
etc/virtman/virtman.conf	配置文件
virtman/ ./baseimage.py ./compute.py ./image.py ./instance.py ./path.py ./snapshot.py ./blockservice.py ./imageservice.py	virtman 主目录 基础镜像管理模块 计算节点 镜像管理模块 虚拟机管理模块 iscsi 路径类 快照管理模块 VMThunder 块设备服务（for Standalone） VMThunder 镜像服务（for Standalone）
virtman/drivers/ ./connector.py ./dmsetup.py ./fcg.py ./iscsi.py ./volt.py	外部包接口 Brick 模块接口 Pydm 接口 Flashcachegroup 模块接口 Brick 模块接口 MasterNode 接口

依赖包：

详见 /packages/virtman/requirements.txt 和 /packages/install.sh

Virtman 副产品：

instance_path<snapshot_path> = /dev/mapper/snapshot_vm1 -> /dev/dm-8 (snapshot_name = "snapshot"+instance_name)

origin_path = /dev/mapper/origin_image-1 -> /dev/dm-7 (origin_name = "origin_" + image_name)

cached_path = /dev/mapper/cached_dm-5 -> /dev/dm-6 <5 放到 4 中得出 6?> (cached_name 应该就是 "cached_" + 映射名) #本机的 target_id = 0 是根据 cached_path 和 iqn 建立得来的

multipath_path = /dev/mapper/multipath_image-1 -> /dev/dm-5 (multipath_name = "multipath_" + image_name)

cache_fcg -> /dev/dm-4 (licyh:总的 fcg)

ssd_fcg -> /dev/dm-3 (licyh:每个 ssd 的本身映射) 即 ssds_path = /dev/loop1 -> /root/blocks/cache.blk

fcg -> /dev/dm-2 (licyh:fcg 初始)

snapshot_path<snapshot_dev> = /dev/loop2 -> /root/blocks/snap1.blk #或者放到 cached 里再出来，应该就变为/dev/mapper/cached_dm-*
image_path = /dev/loop7 -> /root/blocks/image.blk (image_name = "image-" + image-id)
device_path = /dev/disk/by-path/ip-192.168.137.101:3260-iscsi-iqn.2010-10.org.openstack:1-lun-1 -> ../../sdb (即/dev/sdb) 挂载过来到本地的：(注：变为本地的一个设备 /dev/sd*，可以有多个即多路径)

使用端口：
virtman 服务端口：7774 （VMThunder 独立版使用）
Path 连接时的端口：3260

Volt

（by Zhihui Sun）

使用端口：
volt 服务端口：7447

下载

VMThunder.org

VMThunder 官网地址：http://www.vmthunder.org

Github

VMThunder 的 Github 托管地址：https://github.com/vmthunder
Virtman 地址：https://github.com/vmthunder/virtman
Python-virtmanclient 地址：https://github.com/vmthunder/python-virtmanclient
Volt 地址：https://github.com/vmthunder/volt
Python-voltclient 地址：https://github.com/vmthunder/python-voltclient
Others：
Brick 地址：https://github.com/vmthunder/brick
Packages 地址：https://github.com/vmthunder/packages

安装与使用

本节详细讲述了 VMThunder 系统的多节点部署方法。该方法适用于所有

Linux 系统，强烈推荐使用 Ubuntu12.04 或 14.04 进行 VMThunder 部署。

VMThunder 整合版（Installing for Openstack）

（By Xiang Zhao）

VMThunder 独立版（Standalone）

说明：1）安装前请先 apt-get update，再进行 apt-get upgrade；2）安装与使用过程支持 root 用户和普通用户。

前期准备

安装源码包：（在所有节点上）

```
1. 拷贝 packages.tar 到/root 目录下并解压，得到的 packages/目录
      cp packages.tar /root/
      cd /root
      tar xvf packages.tar
2. 进入 packages/目录，运行 install.sh 安装脚本
      cd packages
      sh install.sh
3. 说明：必要时，可对 VMThunder 进行卸载重装
      cd /root/packages/VMT-demo
      sh vmt-reinstall.sh   # vmt-reinstall.sh 可以销毁 VMThunder 进行的配置，以便重新运行
```

配置

控制节点（Master Node）：
（假定为 192.168.137.101）
修改 volt 的配置文件（默认可不修改）

```
mkdir -p /etc/volt
cp /root/packages/volt/etc/* /etc/volt/
```

计算节点（Compute Node）/ 镜像节点（Image Server）：
（假定镜像节点为 192.168.37.102，可与控制节点或计算节点相同）

（假定计算节点为 192.168.137.103~192.168.137.109，可包括控制节点及镜像节点）

1. 修改配置文件

```
mkdir -p /etc/virtman
cp /root/packages/virtman/etc/virtman/* /etc/virtman/
vi /etc/virtman/virtman.conf
    修改 host_ip=192.168.137.102~109 和 master_ip=192.168.137.101
    # 还可设置用作缓存的设备，即 fcg_ssds，默认值为/dev/loop0
vi /etc/tgt/targets.conf
    加上一行 include /etc/tgt/stack.d/*
```

2. 创建用于启动 VM 的快照设备（可不做，提前创建可加快 VMs 启动速度）

```
cd /root/packages/VMT-demo
sh vmt-initenv.sh  #初始化生成创建 cache 和 VMs 所需的快照文件（注释部分可创建测试用 image），默认创建于/root/blocks/目录下
*注：由于目前镜像 Target 服务同计算节点服务绑定在一起，镜像节点也需要此步骤
```

启动服务

控制节点（Master Node）：
运行 volt

```
volt-api --config-file=/etc/volt/volt.conf
```

计算节点（Compute Node）/ 镜像节点（Image Server）：
启动 Virtman 的 RPC 服务器
（为方便调试，该服务器没有后台运行，Virtman 默认打开 debug，日志文件位于 /root/packages/virtman.log）
（若需要远程创建 Image Target，镜像节点也需启动 Virtman 服务）

```
cd /root/packages/virtman/bin
python virtmanserver.py
```

运行

镜像节点（Image Server）：
创建 Image Target，需要准备一个 RAW 格式的镜像（如/root/blocks/image1.img）

1. 修改 image.cfg 配置文件

```
cd /root/packages/VMT-demo
vi image.cfg #修改 image 源（RAW 格式）的创建位置信息
    # image_server_ip    image_name    file_path
    + 192.168.137.102 image1 /root/blocks/image1.blk
    + 192.168.137.102 image2 /root/blocks/image2.blk
    - 192.168.137.102 image1
    @ 192.168.137.102
*注：创建用于实验的虚拟镜像文件/root/blocks/image$i.blk 可借助 vmt-initenv 脚本
```

配置文件 image.cfg 参数说明：

“+” “-” “@”: 分别表示创建、销毁、列出 Image Target

:param image_server_ip:	string, the Image Sever node to create an image Target
:param image_name:	string, which image to use to create target in the Image Server
:param file_path:	string, the realpath of the image

2. 创建镜像 Target

```
python vmtimg.py   #创建镜像 Target（包括销毁、列出镜像 Target 功能）
*注：需要在镜像节点（Image Server）上也安装源码包（见上）
    并提前启动 virtmanserver.py
```

主控节点：(任意节点)
选定控制节点或镜像节点或任意一个计算节点作为“主控节点”，创建指定 VMs 的快照设备。其中 nodelist.cfg 文件定义了在哪个节点上启动哪台虚拟机，用于该虚拟机的快照设备。

```
cd /root/packages/VMT-demo
vi nodelist.cfg   #添加待启动 VMs 的位置信息
    # 格式为 computenode_ip    vm_name    image_server_ip    image_name
    + 192.168.137.103 vm1 192.168.137.102 image1
    + 192.168.137.103 vm2 192.168.137.102 image1
    + 192.168.137.104 vm1 192.168.137.102 image1
    + 192.168.137.104 vm2 192.168.137.102 image2
    - 192.168.137.103 vm1
```

```
    - 192.168.137.104 vm2
    @ 192.168.137.104
python vmtrun.py
    dmsetup table    #ok:有 snapshot_vm1 vm2 vm3
```

如果一切顺利的话，在每个计算节点上将生成可启动虚拟机的快照设备，命名样式为：snapshot_虚拟机名，设备路径为/dev/mapper/snapshot_虚拟机名。进一步可以用这些快照设备作为 image 来启动虚拟机。

启动 VMs

计算节点（Compute Node）：

1. 配置 VMs 启动文件

```
cd /root/packages/VMT-demo
vi windows-vm1.xml
    #修改 4 个地方，如下表红色显示
```

windows-vm1.xml 文件如下：

```
<domain type='kvm'>
  <name>win7_1</name>
  <memory>2048000</memory>
  <currentMemory>2048000</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source file='/dev/mapper/snapshot_vm1'/>
      <target dev='hda' bus='virtio'/>
    </disk>
    <interface type='bridge'>
```

```
        <source bridge='virbr0'/>
        <mac address='52:54:00:CF:6A:A1'/>
        <model type='virtio' />
      </interface>
      <graphics type='vnc' port='5901'>
        <listen type='address' address='0.0.0.0'/>
      </graphics>
  </devices>
</domain>
```

2. 采用 virsh 启动及管理 VMs：

```
virsh create windows-vm1.xml    #默认 vnc 地址  host_ip:5901
virsh create windows-vm2.xml    #默认 vnc 地址  host_ip:5902
      virsh destroy win7_1
virsh list
      virsh vncdisplay win7_1
```

调试相关

```
service open-iscsi restart/status
iscsiadm -m discovery -t sendtargets -p 192.168.2.40:3260    #发现(节点)设备
tgt-admin –s    #查看 target 设备  #lzy 意思好像和 tgtadm 一样  但是这个参数多一些
iscsiadm -m node --logout    #断掉所有 target
tgt-admin --force --delete iqn.2010-10.org.openstack:volume-image1
      同 tgtadm --lld iscsi --mode target --op delete --tid 1 #不行则  –force
losetup -d /dev/loop7
ls -l /dev/mapper    #列出/dev/mapper
ls -l /dev/disk/by-path/
附命令：
dmsetup table    #查看 Device Mapper 列表
dmsetup remove cache_fcg    #删除本机创建的 cache_fcg
dmsetup remove fcg
dmsetup remove ssd_fcg
losetup -a  查看所有循环设备
volt-api  --config-file=/etc/volt/volt.conf      #若 20 秒自动断  显示 0.0.0.0:7447 绑定不上
那应该是端口被占
用 netstat -nat 查看
scp -r 192.168.137.101:/root/packages/VMThunder ./
scp -r stack@10.107.19.13:/home/stack/packages ./
sudo scp -r stack@10.107.19.13:/opt/stack/nova ./
mkdir -p ~/blocks
```

VMThunder API

Vmthunder API for Openstack

主要模块

virtman.compute

主要类型

virtman.compute.Virtman

创建 Virtman 对象

示例：

```
from virtman.compute import Virtman
virtmanclient = Virtman(openstack_compatible=True)
```

参数包括：

- openstack_compatible=True: 表示是否启用 openstack 兼容模式

创建 VM 实例

示例：

```
virtmanclient.create(instance_name, image_name, image_connections, snapshot)
```

参数必须包括：

- instance_name: string 类型，待创建 VM 名称
- image_name: string 类型，待启动镜像名称
- image_connections: list/tuple/single dict 类型，形如：({},..) 或 [{},..] or {}
  每个{}形如{‘target_portal’:.., ‘target_iqn’:.., ‘target_lun’:.., ..}
  其中, target_portal （IP 及其可选端口），target_iqn（符合 iSCSI 规范的名字 ），target_lun （卷的的 lun）
- snapshot: list/string 类型，用来创建 snapshot 的块设备链接或地址，如 snapshot_connection or snapshot_dev

返回参数：(已变)

- instance_path: string 类型，创建成功的 VM 实例的路径地址

删除 VM 实例

示例：

```
virtmanclient.destroy(instance_name)
```

参数包括：

- instance_name：字符串类型，删除 VM 的名字

返回参数：（已变）

- rst: boolean 类型，是否成功删除 VM

列出已创建的 VM 实例

示例：

```
virtmanclient.list()
```

无参数要求

返回参数：

- instance_list: list 类型，每个元素为 string 类型，形如”instance_name:image_name”

Vmthunder API for Java (or Python or others)

主要模块

/packages/VMT-demo/vmtapi.py

主要函数

vmtapi.create
vmtapi.destroy
vmtapi.list
vmtapi.create_image_target
vmtapi.destroy_image_target
vmtapi.list_image_target

Java 调用 vmtapi.py 接口方法

示例：

```
//An Example for Calling Python Code in Java
import org.python.core.PyFunction;
import org.python.core.PyInteger;
import org.python.core.PyObject;
import org.python.util.PythonInterpreter;
public class Test
{
    public static void main(String args[])
    {
        PythonInterpreter interpreter = new PythonInterpreter();
        interpreter.execfile("E:\\vmtapi.py");
        PyFunction vmtcreate = (PyFunction)interpreter.get("create", PyFunction.class);
        PyFunction vmtdestroy = (PyFunction)interpreter.get("destroy", PyFunction.class);
        PyFunction vmtlist = (PyFunction)interpreter.get("list", PyFunction.class);
```

```
            //String node_ip = "192.168.63.101", instance_name = "vm1", image_name = "image1", image_server_ip = "192.168.63.16";
            PyObject pyobj = vmtcreate.__call__(new PyString(node_ip), new PyString(instance_name), new PyString(image_name), new PyString(image_server_ip));
            PyObject pyobj = vmtdestroy.__call__(new PyString(node_ip), new PyString(instance_name));
            PyObject pyobj = vmtlist.__call__(new PyString(node_ip));
            System.out.println("Anwser = " + pyobj.toString());
      }
}
```

参数包括：

- openstack_compatible=True: 表示是否启用 openstack 兼容模式

创建 VM 实例

示例：

```
import vmtapi
vmtapi.create(node_ip, instance_name, image_name, image_server_ip, image_server_port = 3260, iqn_prefix = 'iqn.2010-10.org.openstack:', target_lun = 1, snapshot_dev = None)
```

参数必须包括：

:param node_ip: string, the compute node to create a VM instance
:param instance_name: string, a name of the VM instance to create
:param image_name: string, which image to use in the Image Server
:param image_server_ip: string, the ip of the Image Server
:param image_server_port: (optional) int, the port of the image target in the Image Server
:param iqn_prefix: (optional) string, the prefix of the image target's name in the Image Server,
:param target_lun: (optional) int, target_lun of the image target in the Image Server
:param snapshot_dev: (optional) string, the block device in this compute node to create the VM
:returns : string
"0:info" specifies SUCCESS, info=instance_path, instance_path is " or like '/dev/mapper/snapshot_vm1' in local deployment
"1:info" specifies WARNING, info indicates instance_name exists

销毁 VM 实例

示例：

```
import vmtapi
vmtapi.destroy(node_ip, instance_name)
```

参数包括：

:param node_ip: the compute node to destroy a VM instance
:param instance_name: a name of the VM instance to destroy
:returns : string
"0:info" specifies SUCCESS, info=""
"1:info" specifies WARNING, info indicates instance_name not exists

列出已创建的 VM 实例

示例：

```
import vmtapi
vmtapi.list(node_ip)
```

参数包括：

:param node_ip: the compute node to list VM instances
:returns : a list, like ["instance_name+':'+image_id", …]

创建 Image Target 对象

示例：

```
import vmtapi
vmtapi.create_image_target(image_server_ip, image_name, file_path, loop_dev = '/dev/loop7', iqn_prefix = 'iqn.2010-10.org.openstack:')
```

参数包括：

:param image_server_ip: string, the Image Sever node to create an image Target
:param image_name: string, which image to use to create target in the Image Server
:param file_path: string, the realpath of the image
:param loop_dev: (optional) string, the loop device for the image file to bind
:param iqn_prefix: (optional) string, the prefix of the image target's name, like 'iqn.2010-10.org.openstack:'
:returns : string
"0:info" specifies SUCCESS, info="target_id:loop_dev", target_id (int) is the id of the target in the Image Server node
"1:info" specifies WARNING, info indicates image_name exists
"2:info" specifies WARNING, info indicates image file_path not exists

销毁 Image Target 对象

示例：

```
import vmtapi
vmtapi.destroy_image_target(image_server_ip, image_name)
```

参数包括：

:param image_server_ip: string, the Image Sever node to destroy an image Target

:param image_name: string, which image to use to destroy target in the Image Server
:returns : string
"0:info" specifies SUCCESS, info="nothing", nothing is None
"1:info" specifies WARNING, info indicates image_name not exists

列出已创建的 Image Target 对象

示例：

```
import vmtapi
vmtapi.list_image_target(image_server_ip)
```

参数包括：

:param image_server_ip: string, the Image Sever node to list an image Target
:returns : a list, like ["image_name+':'+target_id+':'+loop_dev",..]

Python-vmthunderclient API

（当前不能使用，待修改）

Main modules

vmthunderclient.client

Main types

vmthunderclient.client.Client

Creating a Client object

Sample code：

```
from vmthunderclient.client import Client
vmtclient = Client(endpoint)
```

Parameter must include:

- endpoint - a string, service endpoint, like "http://${host}:${port}"

Creating a VM instance

Sample code:

```
vmtclient.create(image_id, vm_name, connections, snapshot_dev)
```

Parameters for creating a VM must include:

- image_id – a string, id of the image
- vm_name – a string, the name of the VM to create
- connections – a list, connection properties, must include: [target_portal - ip and optional port, target_iqn - iSCSI Qualified Name, target_lun - LUN id of the volume]
- snapshot_dev – a string, the name of devices to use to create snapshot, like “/dev/loop1”

No return value.

Destroying a VM instance

Sample code:

```
vmtclient.destroy(vm_name)
```

Parameter for destroying a VM must include:

- vm_name – a string, the name of the VM to destroy

No return value.

Listing VMs

Sample code:

```
vmtclient.list()
```

None parameter required.

No return value.

存在的问题

当前版本 stable/0.3 较稳定