liekkas notes Documentation

发布 *1*

liekkas

2018 年 **05** 月 **17** 日

Contents

Contents:

CHAPTER 1

前言

本书主要介绍了 ironic 的基本原理，以及 ironic 在实际环境中的使用。本书采用的环境如下：

- Openstack: Ocata 版本
- OS: CentOS7.3

1.1 内容

本书主要从以下几个方面介绍 ironic

- Ironic 环境搭建；
- Ironic-api 原理介绍；
- Ironic-conductor 原理介绍；
- Ironic-inspector 原理介绍；
- Ironic-python-agent 原理介绍;
- Ironic 镜像制作；
- 测试环境搭建；
- Cloud-init 使用；
- 常见问题；

1.2 术语

名称	含义
ironic	openstack 组件，主要用来管理裸机
baremetal	裸金属，裸机，一般只物理服务器
provision	部署
inspector	主机发现

1.3 openstack 学习

关于 openstack 的学习，笔者主要从以下几个方面学习：

- 文档: https://docs.openstack.org/developer/ironic/
- IRC 记录: http://eavesdrop.openstack.org/meetings/ironic
- code review: https://review.openstack.org/#/q/status:open
- 查看 bp: https://launchpad.net/
- 阅读源码

1.4 Ironic 常用链接

- IRC LOG: http://eavesdrop.openstack.org/meetings/ironic/2017/
- WIKI: https://wiki.openstack.org/wiki/Ironic
- API: https://developer.openstack.org/api-ref/baremetal/index.html
- 状态机: https://docs.openstack.org/ironic/latest/_images/states.svg
- 近期Bug: http://ironic-divius.rhcloud.com/
- 相关项目: https://governance.openstack.org/tc/reference/projects/ironic.html#mission
- White board: https://etherpad.openstack.org/p/IronicWhiteBoard

1.5 About

- Author: liekkas
- Email: leetpy2@gmail.com

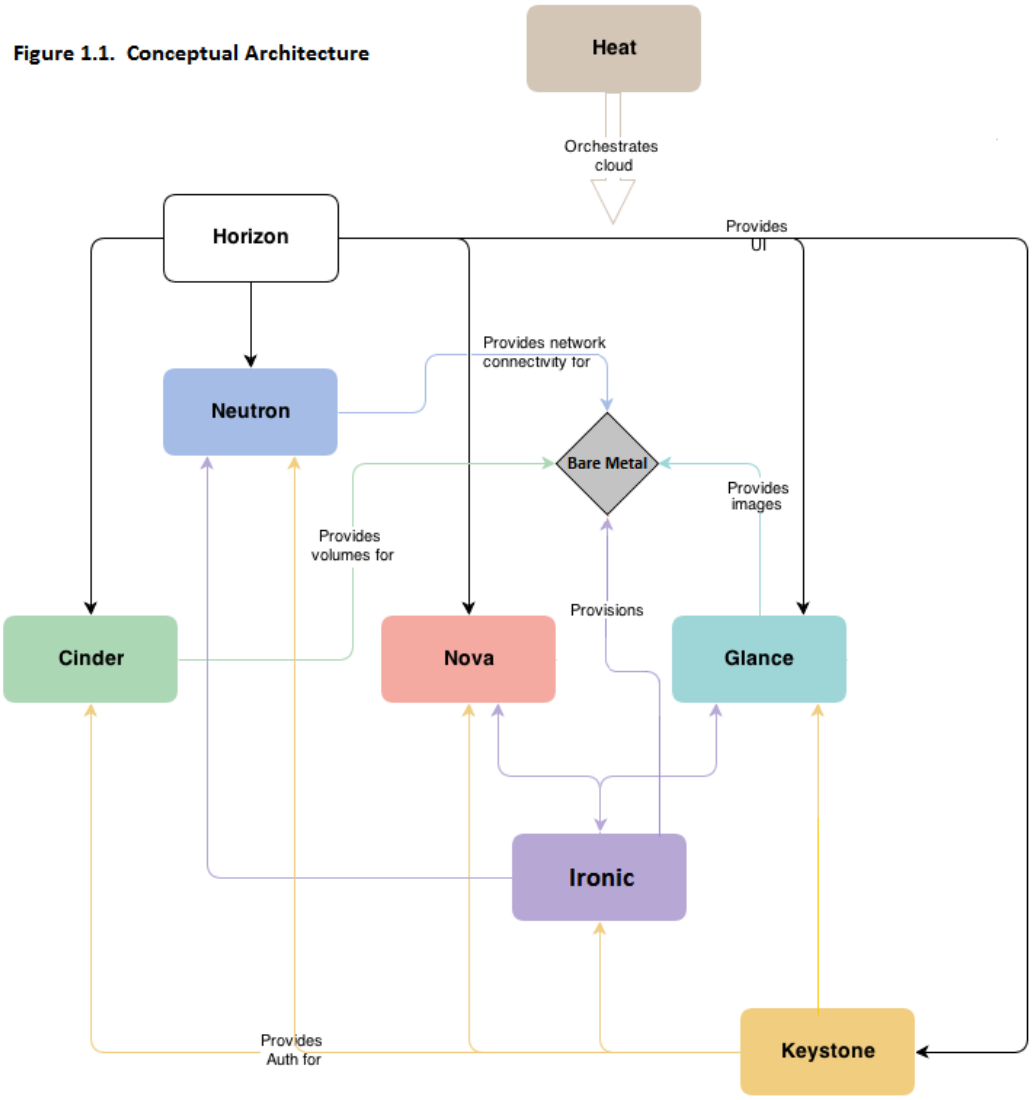
CHAPTER 2

1.0 Ironic 简介

Ironic 是一个 openstack 项目，主要用来管理裸机。包括裸机的电源控制，系统部署，网络配置等。

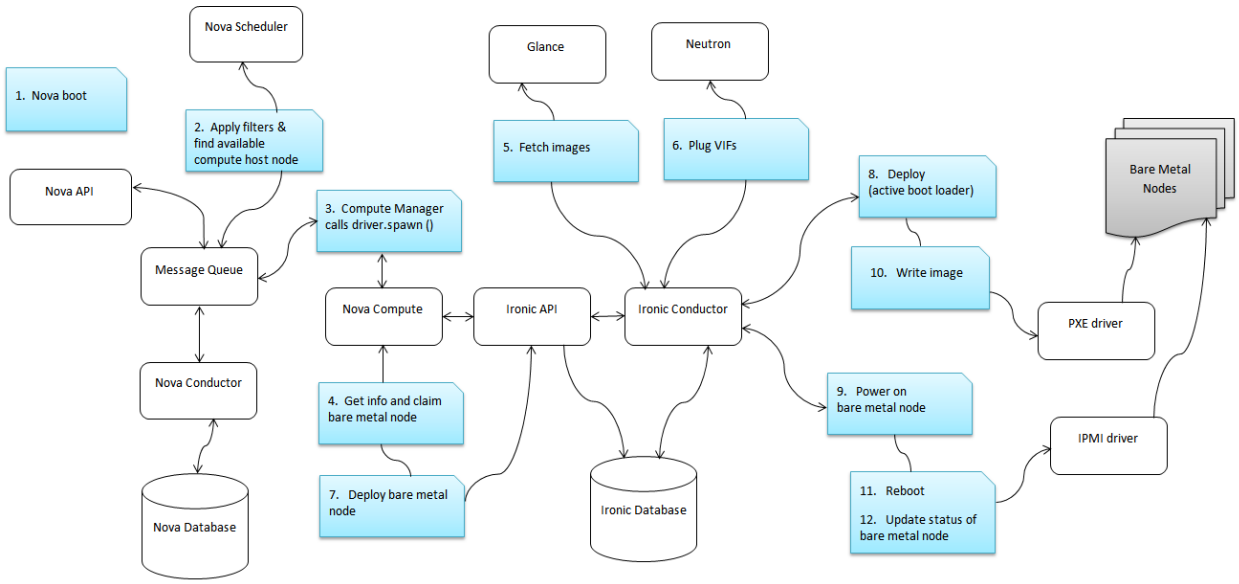
2.1 ironic 作用

- 裸机部署: 有时候虚机的性能不能满足我们的要求，这时候可以使用裸机代替;
- 异地重生: 当 nova computer 节点挂了，能及时检查，并迁移虚机;



Ironic 部署架构图如图所示：

Figure 1.3.3. Bare Metal Deployment Steps



CHAPTER 3

1.1 安装 ironic 环境

本文以 packstack 来安装 openstack 测试环境。

3.1 环境准备

这里我们以CentOS7为例，安装openstack ocata版本，其它版本安装方法类似。packstack目前对NetworkManager 还不支持，我们修改下配置:

```
systemctl disable firewalld
systemctl stop firewalld
systemctl disable NetworkManager
systemctl stop NetworkManager
systemctl enable network
systemctl start network
```

3.2 安装 packstack

```
# 添加packstack yum源
yum -y install centos-release-openstack-ocata

# 安装openstack-packstack
yum -y install openstack-packstack

# 生成answer-file
packstack --gen-answer-file=filename
```

3.3 安装 openstack

如果你安装的是ocata版本，这里packstack有点小bug， 有几个文件需要修改一下， 参考： https://www.redhat.com/archives/rdo-list/2017-March/msg00011.html

两个bug的review链接：

- https://review.openstack.org/#/c/440258/
- https://review.openstack.org/442551

照着review提交的内容修改一下就可以了。接着使用下面的命令安装openstack。

3.4 修改 answer-file

packstack 默认是不安装 ironic 的，需要做如下修改：

```
CONFIG_IRONIC_INSTALL=y
```

3.5 安装 openstack

```
packstack --answer-file=filename
```

CHAPTER 4

1.2 Ironic 配置

使用 packstack 安装完 Ironic 默认使用 Flat 网络，我们这里主要以 Flat 网络为例来介绍，关于 VLAN/VXLAN 将在后面章节介绍。

Ironic 有两种使用方式，一种是 standalone 模式, 另一种是结合 openstack. 如果和 openstack 其它组建集成，ironic 需要做一些配置。

默认情况下这些配置 packstack 都已经配置好了，我们这里还是介绍一下， 一是可以理解 Ironic 和其它组建怎么结合的，另一个是方便自己根据实际环境进行修改。

4.1 Keystone 配置

1. 注册 Bare Metal 服务用户:

```
$ openstack user create --password IRONIC_PASSWORD \
    --email ironic@example.com ironic
$ openstack role add --project service --user ironic admin
```

2. 注册服务:

```
$ openstack service create --name ironic --description \
    "Ironic baremetal provisioning service" baremetal
```

3. 创建 endpoint:

```
$ openstack endpoint create --region RegionOne \
    baremetal admin http://$IRONIC_NODE:6385

$ openstack endpoint create --region RegionOne \
    baremetal public http://$IRONIC_NODE:6385

$ openstack endpoint create --region RegionOne \
    baremetal internal http://$IRONIC_NODE:6385
```

如果使用 keystone v2 API, 使用如下命令:

```
$ openstack endpoint create --region RegionOne \
    --publicurl http://$IRONIC_NODE:6385 \
    --internalurl http://$IRONIC_NODE:6385 \
    --adminurl http://$IRONIC_NODE:6385 \
    baremetal
```

4. 创建角色:

```
$ openstack role create baremetal_admin
$ openstack role create baremetal_observer
```

5. 如果你想限制访问，可以创建一个 “baremetal” 的 Project. 只有这个 project 下的成员才能访问 Ironic 的资源（Nodes, ports 等）:

```
$ openstack project create baremetal
```

给特定的用户授权:

```
$ openstack user create \
    --domain default --project-domain default --project baremetal \
    --password PASSWORD USERNAME
$ openstack role add \
    --user-domain default --project-domain default --project baremetal \
    --user USERNAME baremetal_observer
```

4.2 Compute 配置

社区的 openstack 默认只能管理裸机或者虚机的一种，不能同时管理。 这时由于 nova 没法区分要部署的是裸机还是虚机， 当然修改代码可以达到同时管理裸机和虚机，这超出了本书范围，就不多介绍了。

Nova 的控制节点和计算节点需要做如下配置:

1. `default` 组配置:

```
[default]

# Driver to use for controlling virtualization. Options
# include: libvirt.LibvirtDriver, xenapi.XenAPIDriver,
# fake.FakeDriver, baremetal.BareMetalDriver,
# vmwareapi.VMwareESXDriver, vmwareapi.VMwareVCDriver (string
# value)
#compute_driver=<None>
compute_driver=ironic.IronicDriver

# Firewall driver (defaults to hypervisor specific iptables
# driver) (string value)
#firewall_driver=<None>
firewall_driver=nova.virt.firewall.NoopFirewallDriver

# The scheduler host manager class to use (string value)
#scheduler_host_manager=host_manager
scheduler_host_manager=ironic_host_manager

# Virtual ram to physical ram allocation ratio which affects
```

(continues on next page)

(续上页)

```
# all ram filters. This configuration specifies a global ratio
# for RamFilter. For AggregateRamFilter, it will fall back to
# this configuration value if no per-aggregate setting found.
# (floating point value)
#ram_allocation_ratio=1.5
ram_allocation_ratio=1.0

# Amount of disk in MB to reserve for the host (integer value)
#reserved_host_disk_mb=0
reserved_host_memory_mb=0

# Flag to decide whether to use baremetal_scheduler_default_filters or not.
# (boolean value)
#scheduler_use_baremetal_filters=False
scheduler_use_baremetal_filters=True

# Determines if the Scheduler tracks changes to instances to help with
# its filtering decisions (boolean value)
#scheduler_tracks_instance_changes=True
scheduler_tracks_instance_changes=False

# New instances will be scheduled on a host chosen randomly from a subset
# of the N best hosts, where N is the value set by this option.  Valid
# values are 1 or greater. Any value less than one will be treated as 1.
# For ironic, this should be set to a number >= the number of ironic nodes
# to more evenly distribute instances across the nodes.
#scheduler_host_subset_size=1
scheduler_host_subset_size=9999999
```

2. `ironic` 组配置:

   - 把 `IRONIC_PASSWORD` 换成前面注册的密码;
   - 把 `IRONIC_NODE` 换成 ironic-api 所在节点的 IP 地址;
   - 把 `IDENTITY_IP` 换成 keystone 所在节点的 IP 地址;

```
[ironic]

# Ironic authentication type
auth_type=password

# Keystone API endpoint
auth_url=http://IDENTITY_IP:35357/v3

# Ironic keystone project name
project_name=service

# Ironic keystone admin name
username=ironic

# Ironic keystone admin password
password=IRONIC_PASSWORD

# Ironic keystone project domain
# or set project_domain_id
project_domain_name=Default

```

(continues on next page)

(续上页)

```
# Ironic keystone user domain
# or set user_domain_id
user_domain_name=Default
```

3. 重启 nova 相关服务:

```
sudo systemctl restart openstack-nova-scheduler
sudo systemctl restart openstack-nova-compute
```

4.3 Networking 配置

Ironic 在部署的时候需要使用 Neutron 的 DHCP 服务。

1. 编辑并配置 `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ml2]
type_drivers = flat
tenant_network_types = flat
mechanism_drivers = openvswitch

[ml2_type_flat]
flat_networks = physnet1

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
↪OVSHybridIptablesFirewallDriver
enable_security_group = True

[ovs]
bridge_mappings = physnet1:br-eth2
# Replace eth2 with the interface on the neutron node which you
# are using to connect to the bare metal server
```

2. 如果 neutron-openstack-agent 服务使用 `ovs_neutron_plugin.in` 文件， 则编辑该文件的 [ovs] 组。
3. 添加 ovs 网桥:

```
$ ovs-vsctl add-br br-int
```

4. 处理裸机和 openstack 之间的通信:

```
$ ovs-vsctl add-br br-eth2
$ ovs-vsctl add-port br-eth2 eth2
```

这里的 br-eth2 要和前面的配置文件里的 `bridge_mappings` 对应， eth2 环境实际的物理网卡名。

5. 重启 Open vSwitch agent:

```
# service neutron-plugin-openvswitch-agent restart
```

6. 重启 Open vSwitch agent 服务之后，应该能看到 br-int 和 br-eth2.

```
$ ovs-vsctl show

Bridge br-int
```

(continues on next page)

(续上页)

```
    fail_mode: secure
    Port "int-br-eth2"
        Interface "int-br-eth2"
            type: patch
            options: {peer="phy-br-eth2"}
    Port br-int
        Interface br-int
            type: internal
Bridge "br-eth2"
    Port "phy-br-eth2"
        Interface "phy-br-eth2"
            type: patch
            options: {peer="int-br-eth2"}
    Port "eth2"
        Interface "eth2"
    Port "br-eth2"
        Interface "br-eth2"
            type: internal
ovs_version: "2.3.0"
```

7. 创建租户网络:

```
$ neutron net-create --tenant-id $TENANT_ID sharednet1 --shared \
      --provider:network_type flat --provider:physical_network physnet1

$ neutron subnet-create sharednet1 $NETWORK_CIDR --name $SUBNET_NAME \
      --ip-version=4 --gateway=$GATEWAY_IP --allocation-pool \
      start=$START_IP,end=$END_IP --enable-dhcp
```

4.4 Image 配置

如果使用 agent 驱动，Ironic 要使用 swift 的 temporary URLS, 因此必须要用 swift 做 glance 后端，关于Ironic驱动，后面章节会介绍。

CHAPTER 5

1.3 TFTP 配置

ironic 使用 pxe 流程进行部署。pxe 主要由 DHCP 和 TFTP 两个服务来 完成。ironic 自己并未提供这两个服务，其中 DHCP 由 neutorn 来提供。 TFTP 则是用户自己配置 xinet 和 tftp-server 来完成。

5.1 安装

1. 安装 rpm 包

```
# 创建目录并修改权限
$ sudo mkdir -p /tftpboot/pxelinux.cfg
$ sudo chown -R ironic /tftpboot

# 安装相关 rpm 包
$ sudo yum install tftp-server xinetd syslinux-tftpboot

# 复制 pxe 引导文件到 tftp 目录
$ sudo cp /usr/lib/PXELINUX/pxelinux.0 /tftpboot
$ sudo cp /boot/extlinux/chain.c32 /tftpboot
```

2. 创建 map-file 文件，内容如下：

```
$ cat /tftpboot/map-file
re ^(/tftpboot/) /tftpboot/\2
re ^/tftpboot/ /tftpboot/
re ^(^/) /tftpboot/\1
re ^([^/]) /tftpboot/\1
```

3. 修改 tftp 配置文件

```
$ cat /etc/xinetd.d/tftp
{
    socket_type     = dgram
    protocol        = udp
```

(continues on next page)

(续上页)

```
    wait            = yes
    user            = root
    server          = /usr/sbin/in.tftpd
    server_args = -v -v -v -v -v --map-file /tftpboot/map-file /tftpboot
    disable = no
    per_source      = 11
    cps         = 100 2
    flags           = IPv4
}
```

5.2 防火墙

centos 默认开启了 selinux， selinux 会拦截 tftp 报文， 我们需要做如下配置，让 tftp 报文通过。

```
setsebool -P tftp_home_dir 1
```

CHAPTER 6

2.0 Ironic 部署架构

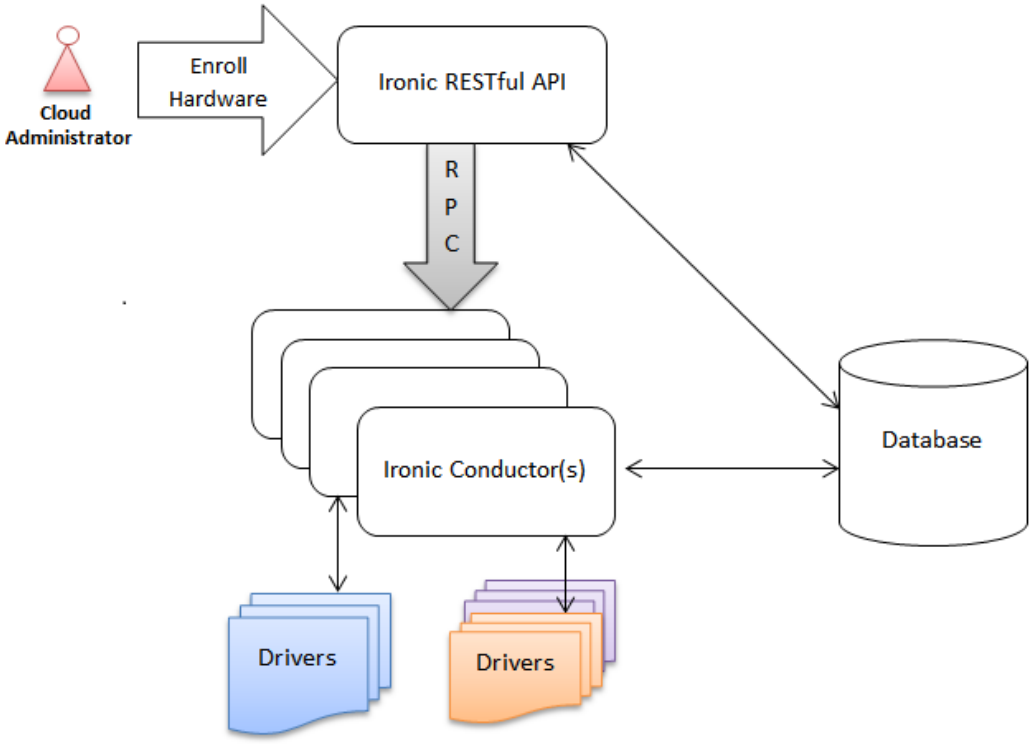
Openstack 早期的项目采用 `Paste + PasteDeploy + Routes + WebOb` 架构。 后来，OpenStack社区的人受不了这么嗦的代码了，决定换一个框架，他们最终选中了Pecan。 Pecan 框架有如下好处：[1]

- 不用自己写WSGI application了
- 请求路由很容易就可以实现了

总的来说，用上Pecan框架以后，很多重复的代码不用写了，开发人员可以专注于业务，也就是实现每个API的功能。

Ironic 对外提供 restful api 接口，来响应外部请求， ironic-api 和 ironic-conductor 之间通信则采用 RPC。 我们先看一下 ironic 的部署架构图:

[1] http://www.infoq.com/cn/articles/OpenStack-UnitedStack-API2


Cloud
Administrator
Enroll
Hardware
Ironic RESTful API
R
P
C
Ironic Conductor(s)
Database
Drivers
Drivers

6.1 参考文献

CHAPTER 7

2.1 Ironic-api

目前社区新项目都使用 pecan 框架，通常 pecan 的目录结构如下:

```
ironic/api/
├── app.py
├── app.wsgi
├── config.py
├── controllers
│   ├── base.py
│   ├── __init__.py
│   ├── link.py
│   ├── root.py
│   └── v1
│       ├── driver.py
│       ├── __init__.py
│       ├── node.py
│       ├── port.py
│       ├── state.py
│       ├── types.py
│       ├── versions.py
│       ├── volume.py
│       └── volume_target.py
├── expose.py
├── hooks.py
├── __init__.py
└── middleware
    ├── auth_token.py
    ├── __init__.py
    └── parsable_error.py
```

- app.py 一般包含了Pecan应用的入口，包含应用初始化代码;
- config.py 包含Pecan的应用配置，会被app.py使用;
- controllers/ 这个目录会包含所有的控制器，也就是API具体逻辑的地方;
- controllers/root.py 这个包含根路径对应的控制器;

- controllers/v1/ 这个目录对应v1版本的API的控制器。如果有多个版本的API，你一般能看到v2等目录;[1]

7.1 application 配置

Pecan的配置很容易，通过一个Python源码式的配置文件就可以完成基本的配置。 这个配置的主要目的是指定应用程序的root，然后用于生成WSGI application。

```
# file: ironic/api/config.py
server = {
    'port': '6385',
    'host': '0.0.0.0'
}

app = {
    'root': 'ironic.api.controllers.root.RootController',
    'modules': ['ironic.api'],
    'static_root': '%(confdir)s/public',
    'debug': False,
    'acl_public_routes': [
        '/',
        '/v1',
        # IPA ramdisk methods
        '/v1/lookup',
        '/v1/heartbeat/[a-z0-9\-]+',
    ],
}
```

上面这个app对象就是Pecan的配置，每个Pecan应用都需要有这么一个名为app的配置。 app配置中最主要的就是root的值，这个值表示了应用程序的入口，也就是从哪个地方开始解析HTTP的根path：/。 hooks对应的配置是一些Pecan的hook，作用类似于WSGI Middleware。

有了app配置后，就可以让Pecan生成一个WSGI application。在 Ironci 项目中， ironic/api/app.py文件就是生成WSGI application的地方，我们来看一下这个的主要内容:

7.1.1 参考文献

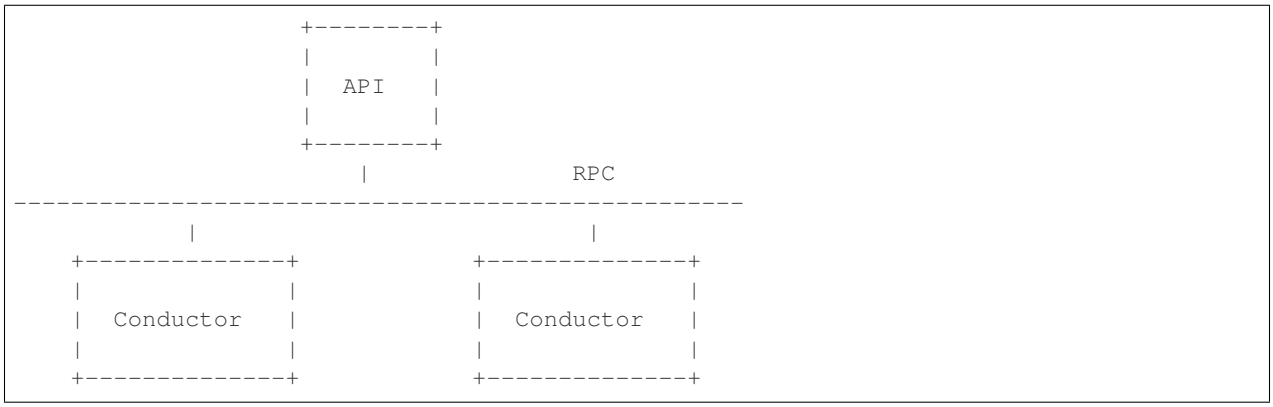
[1] http://www.infoq.com/cn/articles/OpenStack-demo-API3

CHAPTER 8

3.1 Conductor 初始化

8.1 Ironic 架构

Ironic 由两部分组成: API 和 Conductor, 其中 API 采用 pecan 框架， 使用 restful 方式访问，而 API 和 Conductor 之间采用 RPC 通信， 通常是 rabbitmq.



8.2 RPC 初始化

这里 Conductor 是 RPC 的 server 端(消费者)，API 是 RPC 的 client 端(生产者)。 我们先看看 ironic-conductor 的入口函数:

```
def main():
    assert 'ironic.conductor.manager' not in sys.modules

    # Parse config file and command line options, then start logging
    # 配置文件的初始化
    ironic_service.prepare_service(sys.argv)
```

(continues on next page)

(续上页)

```
    # 告警初始化
    gmr.TextGuruMeditation.setup_autorun(version)

    # 创建 RPC Server
    mgr = rpc_service.RPCService(CONF.host,
                                 'ironic.conductor.manager',
                                 'ConductorManager')

    issue_startup_warnings(CONF)

    profiler.setup('ironic_conductor', CONF.host)

    launcher = service.launch(CONF, mgr)
    launcher.wait()
```

8.3 RPCService

上面的代码中创建了一个 RPCService 对象。然后设置了 RPC 的 topic, endpoints, serializer。 这里通过反射的方式设置了 manager 属性。

即: `ironic.conductor.manager.ConductorManager(host, 'ironic.conductor_manager')`

Ironic conductor 通过 oslo.messaging 来创建了 RPC Server， 并把 ConductorManager 注册为 RPC 的 endpoint. Ironic api 通过 RPC 调用的时候，就调用到了 ConductorManager 里对应的方法。

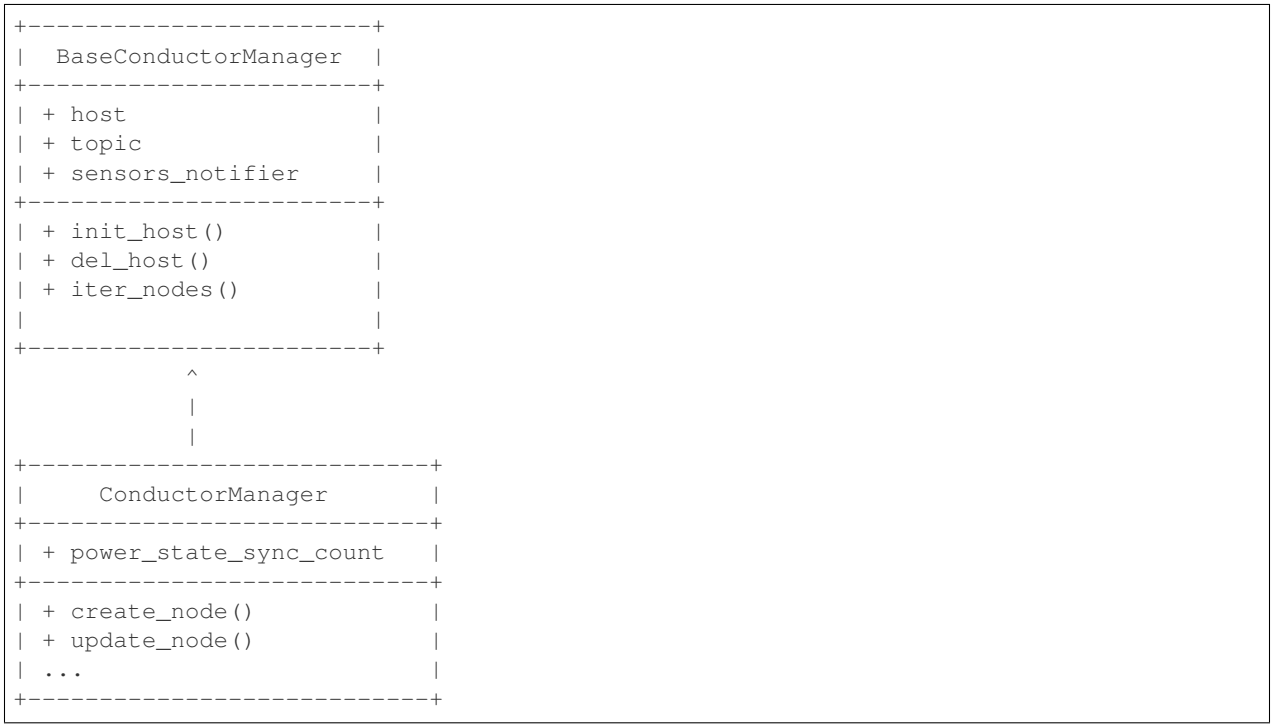
```
class RPCService(service.Service):

    def __init__(self, host, manager_module, manager_class):
        super(RPCService, self).__init__()
        self.host = host
        manager_module = importutils.try_import(manager_module)
        manager_class = getattr(manager_module, manager_class)
        self.manager = manager_class(host, manager_module.MANAGER_TOPIC)
        self.topic = self.manager.topic
        self.rpcserver = None
        self.deregister = True

    def start(self):
        super(RPCService, self).start()
        admin_context = context.get_admin_context()

        target = messaging.Target(topic=self.topic, server=self.host)
        endpoints = [self.manager]
        serializer = objects_base.IronicObjectSerializer(is_server=True)
        self.rpcserver = rpc.get_server(target, endpoints, serializer)
        self.rpcserver.start()

        self.handle_signal()
        self.manager.init_host(admin_context)

        LOG.info('Created RPC server for service %(service)s on host '
                 '%(host)s.',
                 {'service': self.topic, 'host': self.host})
```

8.4 Manager 初始化

在 start RPCServer 的时候会调用 manager 的 init_host 函数。 init_host 中主要完成如下操作:

- dbapi 初始化;
- Conductor 保活线程;
- 协程池初始化;
- 哈希环初始化;
- drivers;
- hardware_types;
- NetworkInterfaceFactory;
- StorageInterfaceFactory;

ConductorManager 类图如下图所示:



CHAPTER 9

3.2 Ironic driver

9.1 驱动类型

从 Ocata 开始 ironic 支持两种类型的驱动: *classic drivers* 和 *hardware types*, 以后 ironic 会停止 *classic drivers* 的支持，并只支持 *hardware types*, 详情参考 /install/enabling-drivers

9.2 驱动初始化

openstack 的请求流程是 `restful-api -> rpc` ,在 ironic 每个 rpc 请求处理中，一般会创建一个 *TaskManager* 的对象，后续的大部分 操作都是通过这个对象来完成的。其中最主要的是 *ironic drive* 的使用。 先看看初始化代码:

```
# file: task_manager.py
class TaskManager(object):

    def __init__(self, context, node_id, shared=False, driver_name=None,
                 purpose='unspecified action'):

        self.driver = driver_factory.build_driver_for_task(
            self, driver_name=driver_name)
```

可以看到这里的 `task.driver` 是采用工厂模式初始化的。看看工厂的具体实现:

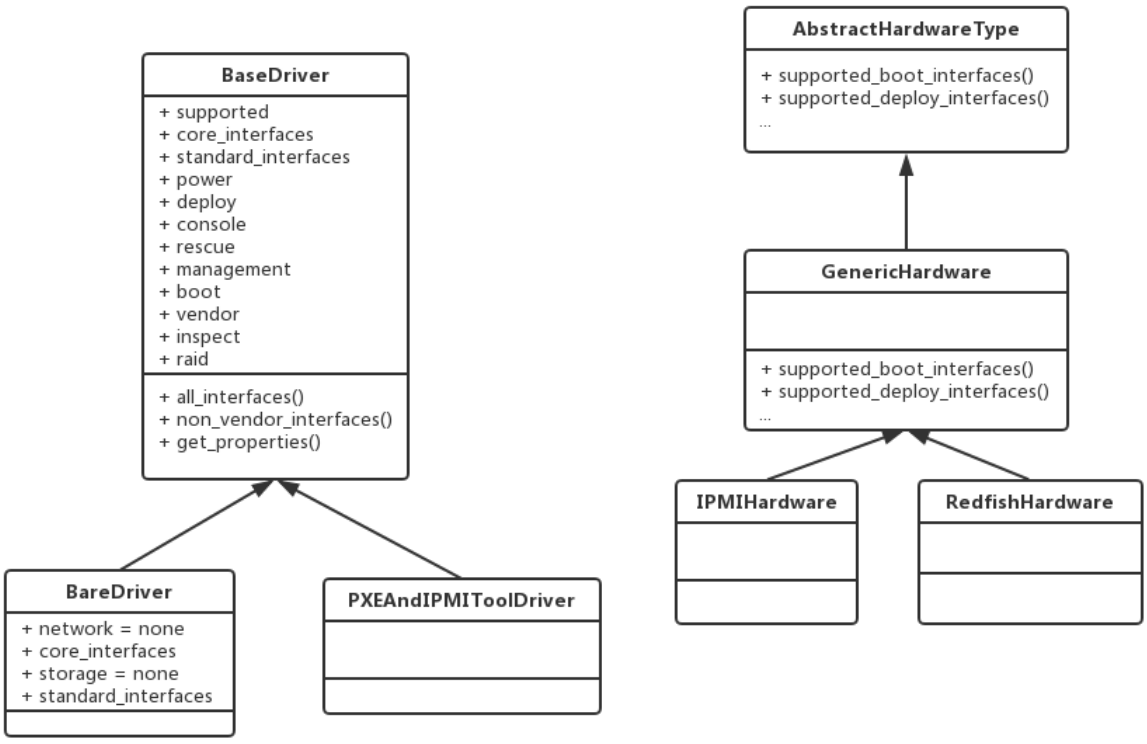
```
# file: driver_factory.py
def build_driver_for_task(task, driver_name=None):
    node = task.node
    driver_name = driver_name or node.driver

    driver_or_hw_type = get_driver_or_hardware_type(driver_name)
    try:
        check_and_update_node_interfaces(
```

(continues on next page)

(续上页)

```
            node, driver_or_hw_type=driver_or_hw_type)
    except exception.MustBeNone as e:
        LOG.warning('%s They will be ignored. To avoid this warning, '
                    'please set them to None.', e)

    bare_driver = driver_base.BareDriver()
    _attach_interfaces_to_driver(bare_driver, node, driver_or_hw_type)

    return bare_driver
```

Ironic driver 类图如下图所示:



假设我们 node 里使用的是 `pxe_ipmitool` 驱动，ironic 会根据这个名字到 setup.cfg 中找到对应的类名，这里是: `ironic.drivers.ipmi:PXEAndIPMIToolDriver`.

初始化 task 的驱动步骤如下:

1. 首先根据 node 里的 driver 名称加载对应的 driver;
2. 检查并更新 node interfaces，这里主要是检查驱动的 hardware interface 有没有设置;
3. 创建 BareDriver 对象 bare_driver;
4. 把 driver 里所有的 interface 复制到 bare_driver 中;

   这里的 interface由如下三部分构成:

   - core_interfaces
   - standard_interfaces

- vendor

5. 获取 node 里的 dynamic_interfaces 并复制到 bare_drivre 中;

> **注解：** 默认 dynamic_interfaces 是 ['network', 'storage']

这里的 `driver_or_hw_type` 就是我们注册 node 使用的 driver。 *classic drivers* 的 driver 都是直接继承 `BaseDriver` 的， 而 *hardware types* 的 driver 继承的 `GenericHardware`. 最终我们在 task 使用的 driver 是 `BareDriver` 类型。

CHAPTER 10

3.3 Ironic 一致性 Hash 算法

本文并不详细介绍一致性 Hash 算法，这里只介绍Ironic 如何使用一致性 Hash 算法。 有兴趣的朋友可以自行查阅资料 。

10.1 Hash 环构造过程

1. 首先 Ironic 获取所有 online 的 conductor 节点；
2. 根据 conductor 支持的驱动，把 conducor 划到不同的组里；

   存为字典结构： {driver_name1: [host1, host2], driver_name2: [host2, host3]}

   注解： 当一个 conductor 配置支持多个驱动时，这个 conductor 会出现在不同组里。
3. 每个驱动创建一个 Hash 环，环的默认大小是 2^5。

   Hash 环的大小可以通过配置项 `hash_partition_exponent` 修改。
4. 将某个驱动所有的 conductor 节点加入到 Hash 环上；
   (a) 将 conductor 的 hostname 进行 utf-8 编码，得到一个 key;
   (b) 将 key 做 md5 运算，得到一个 key_hash;
   (c) key_hash.update(key);
   (d) 取 key_hask 的十六进制字符串并转换成整数 Ci;
   (e) 把所有得到的整数 Ci 按大小排序，并报数到 partitions 数组中;

10.2 Node 映射到 Conductor

1. 根据 node 驱动找到对应的 Hash 环;

注解： 每次操作时，Hash 都会重新生成，防止某个 conductor 离线而导致操作失败。

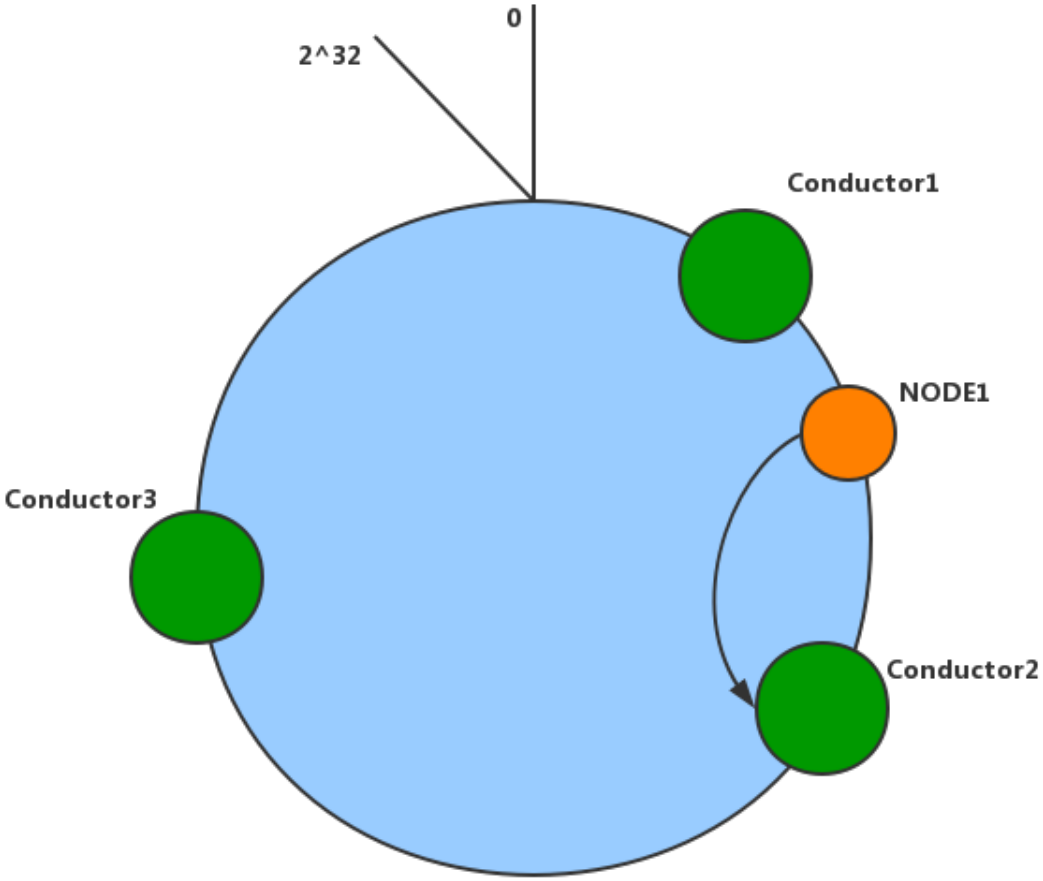
2. 根据 node_uuid hash 算出 hashed_key, 算法同上面;
3. 根据 hashed_key 在上面构造的 partitions 数组中找到合适的位置 p;
4. 根据 p 获取 Conductor 节点;

10.3 Example

这里我们看一个例子，假设有三个 conductor 节点， 节点的 hostname 分别是: conductor1, conductor2, conductor3. 这三个 conductor 都支持 `pxe_ipmitool` 驱动。

首先根据 hostname 构造一个 Hash 环，如下图所示。 然后根据 Ironic node 的 uuid hash 出一个 position, 离这个 position 最近的 conductor 就是要找的节点。

注解： 如果希望异常是快速恢复，可以通过配置 `hash_distribution_replicas` 调度多个 conductor。



CHAPTER 11

4.0 Inspector 介绍

在我们注册完 ironic-node 之后，我们需要把裸机的硬件信息更新到 node 表里。如果数量比较少，我们可以手动添加，但是如果是大规模部署，显然不适合手动添加。另外还有 port 的创建，裸机网口和交换机的链接关系等，都不适合自动添加。

ironic inspector 主要是帮助我们收集裸机信息的。

11.1 Inspector 安装

选择一个计算节点，安装 ironic-inspector，inspector 可以和 ironic-conducor 在同一个节点，也可以在不同节点。

```
sudo yum install openstack-ironic-inspector
```

11.1.1 创建数据库

```
CREATE DATABASE ironic_inspector CHARACTER SET utf8;

GRANT ALL PRIVILEGES ON ironic_inspector.* TO 'inspector'@'localhost' \
IDENTIFIED BY 'inspector';

GRANT ALL PRIVILEGES ON ironic_inspector.* TO 'inspector'@'%' \
IDENTIFIED BY 'inspector';
```

创建完数据库之后，配置数据库连接，并生成数据库表。

```
$ cat /etc/ironic-inspector/inspector.conf

[database]
connection =  mysql+pymysql://inspector:inspector@10.43.210.22/ironic_inspector?
↪charset=utf8
```

调用 ironic-inspector-dbsync 生成表

```
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf upgrade
```

11.1.2 keystone 注册

```
openstack user create --password ironic_inspector \
--email ironic_inspector@example.com ironic_inspector

openstack role add --project service --user ironic_inspector admin

openstack service create --name ironic-inspector --description \
"Ironic inspector baremetal provisioning service" baremetal-introspection
```

11.2 Inspector 配置

inspector 提供了两个服务 openstack-ironic-inspector 和 openstack-ironic-inspector-dnsmasq

11.2.1 Dnsmasq 配置

```
$ cat /etc/ironic-inspector/dnsmasq.conf

port=0
interface=eth0
bind-interfaces
dhcp-range=192.168.2.10,192.168.2.200
enable-tftp
tftp-root=/tftpboot
dhcp-boot=pxelinux.0
dhcp-sequential-ip
```

这里的 interface 是提供 DHCP 服务的网口，也是 tftp 的网口， 我们需要给 eth0 配置一个同网段的 IP。

```
$ cat /etc/sysconfig/network-scripts/ifcfg-eth0

TYPE=Ethernet
BOOTPROTO=static
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=192.168.2.2
NETMASK=255.255.255.0
```

11.2.2 Tftp 配置

inspector 和 provision 使用的是同一组 deploy 内核镜像 这里假设 tftp 服务器已经配置好了，我们这里只添加 default 文件，文件内容如下：

```
default introspect

label introspect
kernel deploy.vmlinuz
append initrd=deploy.initrd ipa-inspection-callback-url=http://192.168.2.2:5050/v1/
↪continue ipa-inspection-collectors=default ipa-collect-lldp=1 systemd.journald.
↪forward_to_console=no selinux=0

ipappend 3
```

在 default 文件中，确认如下几个配置：

- `ipa-inspection-callback-url`，这个 IP 填写 tftp 的 IP 地址，裸机需要访问这个 IP;
- `ipa-collect-lldp=1` 是让 IPA 收集 lldp 报文;
- `selinux=0` 防止某些情况下无法登陆 initramfs;

11.2.3 Ironic 配置

要在 ironic 里使用 inspector，需要先在 ironic 配置文件里使能 inspector， 配置如下：

```
$ cat /etc/ironic/ironic.conf

[inspector]
enabled = true
service_url = http://10.43.210.23:5050
```

这里的 `service_url` 也可以不写，ironic 会根据注册的 endpoint 来获取。

11.3 组网说明

由于 inspector 的 DHCP 服务是不区分 mac 地址的，如果在 flat 网络中使用，跟 neutron-dhcp-agent 有冲突。因此如果 是 flat 网络，建议分开进行 inspector 和 provision。 如果是 vlan 网络，把 inspector 和 provision 放到不同的 vlan 即可。

11.4 说明

如果把 ironic-inspector 和 ironic-conductor 放到同一个节点， 那么 provision流程和 inspector 流程是公用一个 tftp 服务器， 然后监听不同的网口。在正常情况下是没有冲突的，但是如果部署 流程失败了，导致 tftp 数据有残留，那么后续可能进行 inspector 流程时，会下到 deploy 的镜像和配置文件，从而导致 inspector 失败。

CHAPTER 12

4.1 Inspector 兼容 SDN lldp 报文

部分 SDN 交换机使用的 lldp 报文跟正常的 lldp 报文有些差异。要兼容这部分交换机，我们需要修改 inspector 的实现代码。

12.1 lldp 介绍

这里先简单介绍一下 lldp 报文结构， lldp 采用 TLV(type length value)方式存储数据。报文结构如下图所示:

TLV 的类型域的定义及分配如下图所示:

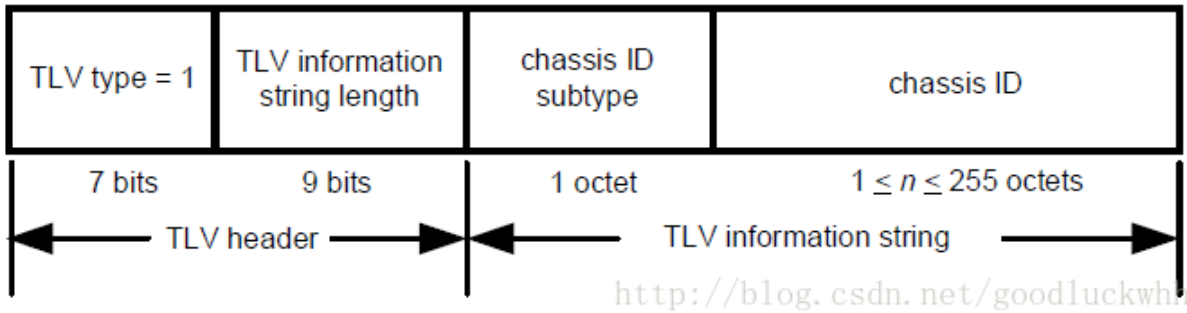
TLV type *	TLV name	Usage in LLDPDU
0	End Of LLDPDU	Mandatory
1	Chassis ID	Mandatory
2	Port ID	Mandatory
3	Time To Live	Mandatory
4	Port Description	Optional
5	System Name	Optional
6	System Description	Optional
7	System Capabilities	Optional
8	Management Address	Optional
9–126	reserved	—
127	Organizationally Specific TLVs	Optional

我们在 ironic 中使用的 local_link_connection 和报文的对应关系如下表：

TLV TYPE	TLV NAME	local link connection name
1	chassis ID	switch_id
2	port_id	port_id

再来介绍下 SDN 的 lldp 报文和普通的 lldp 报文有啥差异，首先不管是 SDN 的 lldp 还是普通交换的 lldp，我们要收集的都是 chassis ID 和 port id，那有什么差异呢？这是因为 chassis ID 和 port id，还存在 subtype，通过 subtype 可以定义不同的 chassis ID 和 port id 值。

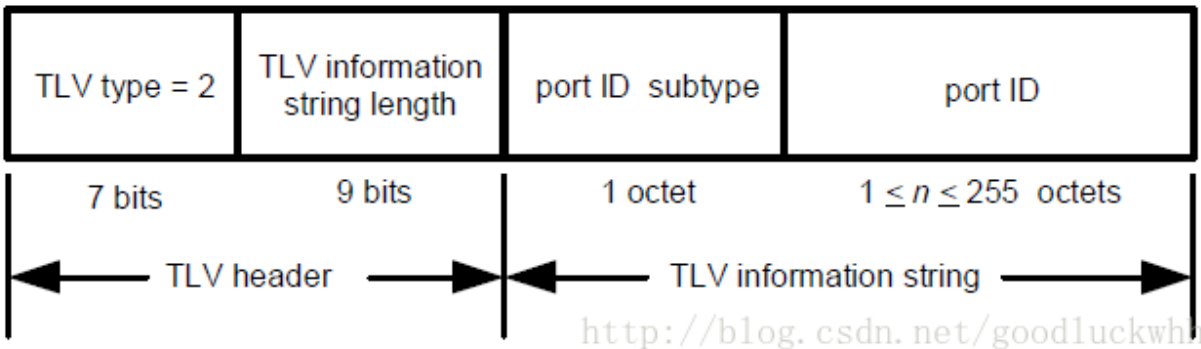
先看看 chassis ID 报文结构：



chassis子类型所可能的取值如图所示：

ID subtype	ID basis
0	Reserved
1	Chassis component
2	Interface alias
3	Port component
4	MAC address
5	Network address
6	Interface name
7	Locally assigned
8–255	Reserved

再看看 port id 值



其子类型的可能取值如下图所示:

ID subtype	ID basis
0	Reserved
1	Interface alias
2	Port component
3	MAC address
4	Network address
5	Interface name
6	Agent circuit ID
7	Locally assigned
8–255	Reserved

SDN 的 lldp 和普通 lldp 报文差异如下表所示:

SWITCH	TLV TYPE	TLV NAME	subtype	subtype value
SDN	1	chassis id	7	Locally assigned
normal	1	chassis id	4	Mac Address
SDN	2	port id	2	Port component
normal	2	port id	3	Mac Address
			5	Interface name
			7	Locally assigned

CHAPTER 13

4.3 Inspector 源码分析

ironic-inspector 使用 flask 框架来编写的。flask 是一个轻量级 的 python web 框架， 详细资料可以查看: http://flask.pocoo.org/

13.1 Ironic 处理阶段

我们首先通过 cli 来触发 ironic inspector 流程。

```
ironic node-set-provision-state <node_uuid> manage
ironic node-set-provision-state <node_uuid> inspect
```

我们知道 openstack 组件之前的调用都是通过 restful 接口来完成的， 而组件内部是通过 rpc 调用来完成的。

上面的 cli 会发送 PUT 请求到 `/v1/nodes/{node_ident}/provision` , ironic-api 收到这个请求后会根据 body 的 `target` 字段做处理:

```
# ironic/api/controllers/v1/node.py
def provision():
...
    elif target == ir_state.VERBS['inspect']:
        pecan.request.rpcapi.inspect_hardware()
```

然后 ironic 通过 rpc 调用 inspect_hardware 方法。然后通过发送 http 请求 到 ironic-inspector。inspect 的具体实现是跟 driver 有关，在 driver.inspect.inspect_hardware 中。

```
# ironic/drivers/modules/inspector.py
def _start_inspection(node_uuid, context):
    try:
        _call_inspector(client.introspect, node_uuid, context)
        ...

# ironic_inspector_client/client.py
def introspect(uuid, base_url=None, auth_token=None,
```

(continues on next page)

(续上页)

```
                new_ipmi_password=None, new_ipmi_username=None,
                api_version=DEFAULT_API_VERSION, session=None, **kwargs):
    c = v1.ClientV1(api_version=api_version, auth_token=auth_token,
                    inspector_url=base_url, session=session, **kwargs)
    return c.introspect(uuid, new_ipmi_username=new_ipmi_username,
                        new_ipmi_password=new_ipmi_password)

# ironic_inspector_client/v1.py
def introspect(self, uuid, new_ipmi_password=None, new_ipmi_username=None):
    ...
    params = {'new_ipmi_username': new_ipmi_username,
              'new_ipmi_password': new_ipmi_password}
    self.request('post', '/introspection/%s' % uuid, params=params)
```

通过上面的代码，我们可以看到 ironic 发送了 post 请求到 `/introspection/<uuid>` 。 下面流程就到了 inspector 了。

13.2 Inspector处理阶段

ironic-inspector 的 restful api 实现在 `main.py` 中，我们首先根据 url 找到对应的函数:

```
# main.py
@app.route('/v1/introspection/<uuid>')
@convert_exceptions
def api_introspection(uuid):
    ...
    introspect.introspect(uuid,
                          new_ipmi_credentials=new_ipmi_credentails,
                          token=flask.request.headers.get('X-Auth-Token'))
    return '', 202
```

再看看introspect的具体实现:

```
# introspect.py
def introspect():
node_info = node_cache.add_node(node.uuid,
                                bmc_address=bmc_address,
                                ironic=ironic)
future = utils.executor().submit(_background_introspect, ironic, node_info)
...
```

introspect函数先是更新了ipmi信息，然后在inspector的node表里添加一条记录， 另外在attributes表里添加bmc_address信息。最终后台调用 _background_introspect做主机发现。

```
# introspect.py
def _background_introspect_locked(ironic, node_info):
    try:
        ironic.node.set_boot_device(node_info.uuid, 'pxe',
                                    persistent=False)
    try:
        ironic.node.set_power_state(node_info.uuid, 'reboot')
```

我们在已经配置好了裸机，并在/tftpboot/pxelinux.cfg/default设置了如下信息:

```
default introspect
label introspect
kernel ironic-agent.vmlinuz
append initrd=ironic-agent.initramfs ipa-inspection-callback-url=http://192.168.2.
↪2:5050/v1/continue ipa-inspection-collectors=default,logs systemd.journald.forward_
↪to_console=no
ipappend 3
```

13.3 IPA 阶段

裸机从小系统启动之后，会启动ironic-python-agent服务。 该服务会收集裸机的硬件信息，并发送到 ipa-inspection-callback-url指定的url。

13.4 Inspector主机上报阶段

先看看inspector怎么处理ipa上报的数据:

```
# main.py
@app.route('/v1/continue', methods=[post])
@convert_exceptions
def api_continue():
    data = flask.request.get_json(force=True)
    return flask.jsonify(process.process(data))
# process.py
def process(introspection_data):
    """Process data from the ramdisk.

    This function heavily relies on the hooks to do the actual data processing.
    """
    hooks = plugins_base.processing_hooks_manager()
    failures = []
    for hook_ext in hooks:
        try:
            hook_ext.obj.before_processing(introspection_data)
        except utils.Error as exc:
            ...
    # 根据ipmi_address和macs获取inpsector node
    node_info = _finde_node_info(introspection_data, failures)
    try:
        node = node_info.node()
        ...

    try:
        return _process_node(node, introspection_data, node_info)
```

我们可以看到这里数据是交由process出来，而process函数又调用各种钩子来出来ipa数据。 接着根据ipmi_address查找对应的inspector node，再根据获取到的uuid来得到 ironic node，交由_process_node()函数处理。

```
# process.py
def _process_node(node, introspection_data, node_info):
    ir_utils.check_provision_state(node)
```

(continues on next page)

(续上页)

```
    node_info.create_ports(introspection_data.get('macs') or ())
    _run_post_hooks(node_inof, introspection_data)

    if CONF.processing.store_data == 'switf':
        stored_data = {k: v for k, v in introspection_data.items()
                       if k not in _STORAGE_EXCLUDE_KEYS}
        swift_object_name = switf.store.store_introspection_data(stored_data,
                                                                 node_info.uuid)
    ironic = ir_utils.get_client()
    firewall.update_filters(ironic)

    node_info.invalidate_cache()
    rules.apply(node_info, introspection_data)
    ...
    utils.executor().submit(_finish, ironic, node_info, introspection_data)

def _finish(ironic, node_info, introspection_data):
    try:
        ironic.node.set_power_state(node_info.uuid, 'off')
    node_info.finished()
```

我们可以看到，如果配置了store_data=swift，inspector会把ipa上报的数据存储到swift中。最后的node_info.finished()是删除inspector数据库中已完成的数据。

CHAPTER 14

5.0 IPA 介绍

IPA 是 ironic python agent 的缩写。IPA 是一组服务， 驻留在 deploy 映像中。主要完成裸机硬件信息的获取及 信息上报。

CHAPTER 15

5.2 IPA Hardware Manager

15.1 获取当前主机 bmc 信息

```
# 加载相关驱动
$ modprobe ipmi_msghandler
$ modprobe ipmi_devintf
$ modprobe ipmi_si

# 查看 ipmi 信息
$ ipmitool lan print
IP Address Source       : Static Address
IP Address              : 129.0.0.10
Subnet Mask             : 255.0.0.0
MAC Address             : 0c:12:62:e4:c1:19
SNMP Community String   : Public
Default Gateway IP      : 129.0.1.1
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
Cipher Suite Priv Max   : aaaaaaaaaaaaaaa
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
                        :     u=USER
                        :     o=OPERATOR
                        :     a=ADMIN
                        :     O=OEM
```

15.2 获取硬盘信息

```
$ lsblk -Pbdi -o KNAME,MODEL,SIZE,ROTA,TYPE
KNAME="sda" MODEL="Logical Volume  " SIZE="298999349248" ROTA="1" TYPE="disk"
KNAME="sdb" MODEL="Logical Volume  " SIZE="19999998607360" ROTA="1" TYPE="disk"
```

(continues on next page)

(续上页)

```
KNAME="sdc" MODEL="TOSHIBA MG04ACA4" SIZE="4000787030016" ROTA="1" TYPE="disk"
KNAME="loop0" MODEL="" SIZE="2158016512" ROTA="1" TYPE="loop"
KNAME="loop1" MODEL="" SIZE="1163403264" ROTA="1" TYPE="loop"
KNAME="loop2" MODEL="" SIZE="1163403264" ROTA="1" TYPE="loop"
KNAME="loop3" MODEL="" SIZE="1163403264" ROTA="1" TYPE="loop"
KNAME="loop4" MODEL="" SIZE="1163403264" ROTA="1" TYPE="loop"
```

15.3 获取 CPU 信息

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Stepping:              2
CPU MHz:               1396.593
BogoMIPS:              4809.94
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0-7,16-23
NUMA node1 CPU(s):     8-15,24-31
```

15.4 获取内存信息

内存计算稍微复杂一点，这里分为两部分 `total` 和 `physical_mb`

1. `total` 计算

   total, free, buffers, shared 几个字段是调用 python 的 `_psutil_linux` 库来获取的。

```
import _psutil_linux
from _psutil_linux import *

total, free, buffers, shared, _, _ = _psutil_linux.get_sysinfo()
```

```
$ cat /proc/meminfo | grep "^Cached:"
Cached:          1089344 kB

$ cat /proc/meminfo | grep "^Active:"
```

(continues on next page)

(续上页)

```
Active:          2491184 kB

$ cat /proc/meminfo | grep "^Active:"
Inactive:         977736 kB
```

```
avail = free + buffers + cached
used = total - free
percent = usage_percent((total - avail), total, _round=1)
```

最终返回的 total 就是 get_sysinfo() 的第一个值。

2. physical_mb 计算

   physical_mb 的计算是先通过 dmidecode 获取所有内存条的大小， 然后把结果求和。

```
$ dmidecode --type 17 | grep Size
  Size: 4096 MB
  Size: No Module Installed
```

15.5 网口信息

```
# 获取所有网卡名
ls /sys/class/net

# 获取所有网卡设备（去掉虚拟网卡）
# 所有实际网卡都会链接到一个 PCI 设备
ls /sys/class/net/eth0/device

# mac 地址
cat /sys/class/net/eth0/address

# carrier
cat /sys/class/net/eth0/carrier

# vendor
cat /sys/class/net/eth0/device/vendor

# device
cat /sys/class/net/eth0/device/device
```

15.6 启动方式

如果存在 /sys/firmware/efi 认为是 uefi 启动方式， 否则认为是 BIOS 启动。

15.7 制造商信息

```
$ dmidecode --type system
```

CHAPTER 16

6.0 Ironic 映像

ironic 整个部署流程中有两组映像，分别是 deploy 映像和 user 映像， 其中 deploy 映像用在 inspector 和 部署阶段， user 映像是用户需要安装的操作系统映像。

16.1 Deploy 映像

制作ironic deploy镜像其实就是在普通镜像中添加一个ipa服务，用来裸机和ironic通信。 官方推荐制作镜像的工具有两个，分别是CoreOS tools和disk-image-builder 具体链接如下: https://docs.openstack.org/project-install-guide/baremetal/ocata/deploy-ramdisk.html

16.1.1 coreos 映像

coreos 是一个 docker 镜像， 你可以自己构建，也可以直接下载社区 构建好的: http://tarballs.openstack.org/ironic-python-agent/coreos/files/

映像密码

添加内核启动参数:

```
coreos.autologin
```

16.1.2 dib 映像

映像密码

有时候，部署会卡很长时间，我们希望能登录到裸机，查看原因。 这个时候需要有密码可以或者是 ssh 能免密码登录。

对应 dib 添加密码，是通过 dynamic-login element 来完成的。 首先制作带 dynamic-login 的映像:

```
disk-image-create ironic-agent centos7 dynamic-login -o ironic-deploy
```

dynamic-login 的原理是在系统里起一个 dynamic-login 服务，在系统 上电时，解析 /proc/cmdline 里的参数，如果用户传了 rootpwd 或者 sshkey， 则写到对应的文件中，这样用户就可以登录系统了。

dynamic-login 使用的是密文，我们可以使用 openssl 生产密码:

```
$ openssl passwd

Password:
Verifying - Password:
mNw2hVHmny2Ho
```

然后我们把在 /etc/ironic/ironic.conf 添加我们的密码。

```
$ cat /etc/ironic/ironic.conf

[pxe]
pxe_append_params = rootpwd="mNw2hVHmny2Ho"
```

如果使用 ssh 方式登录，则添加 sshkey

```
$ cat ~/.ssh/id_rsa.pub

# 添加 sshkey="<your_sshkey>"
$ cat /etc/ironic/ironic.conf

[pxe]
pxe_append_params = sshkey=""
```

16.2 User 映像

user 映像又分为 partition 映像和 whole disk 映像，两者的区别是 whole disk 映像包含分区表和 boot。目前 partition 映像已经很少 使用了，现在基本都使用 whole disk 映像。

16.2.1 镜像驱动问题

我们使用虚机制作的镜像安装在物理机上，很可能缺少驱动，而导致用户 系统起不来。这里我们以 CentOS 为例，说明如何重新制作驱动。

```
mount -o loop CentOS.iso /mnt
cd /mnt/isolinux
lsinitrd initrd.img | grep "\.ko" | awk -F / '{print $NF}' | tr "\n" " "

# 将如上命令获得的ko列表拷贝到 /etc/dracut.conf 中
add_drivers+=""

rm -rf /boot/*kdump.img
dracut --force
```

CHAPTER 17

7.0 测试环境搭建

在平时学习中，我们身边可能没有具有 IPMI 的裸机， 我们希望最好能用虚拟模拟。但是普通的 kvm 虚拟并 没有提供 IPMI 功能。

我们知道社区提交代码都会有测试用例来保证代码的可靠 性，社区应该不会全部都用裸机来测试。通过分析社区 tempest 测试，我们发现有个叫 virtualbmc 的插件，该 插件可以给 kvm 虚机添加带外功能。关于 virtualbmc 的 使用参考后面章节。

CHAPTER 18

7.1 devstack

设置的 CI 都是用 devstack 来搭建 openstack 环境的，我们这里也使用 devstack 来搭建 ironic 环境。具体步骤如下：

1. 下载 devstack 代码:

```
git clone https://git.openstack.org/openstack-dev/devstack -b stable/ocata
```

2. 配置 stack 用户

```
# 创建用户
devstack/tools/create-stack-user.sh

# 设置权限
mv devstack /opt/stack
chown -R stack:stack /opt/stack/devstack

# 切换用户
su -- stack
cd devstack
```

3. 编写运行配置文件

```
$ cat /opt/stack/devstack/local.conf
[[local|localrc]]

# Configure ironic from ironic devstack plugin.
enable_plugin ironic https://git.openstack.org/openstack/ironic stable/ocata

# Install networking-generic-switch Neutron ML2 driver that interacts with OVS
enable_plugin networking-generic-switch https://git.openstack.org/openstack/
↪networking-generic-switch stable/ocata
#Q_PLUGIN_EXTRA_CONF_PATH=/etc/neutron/plugins/ml2
#Q_PLUGIN_EXTRA_CONF_FILES['networking-generic-switch']=ml2_conf_genericswitch.ini
```

(continues on next page)

(续上页)

```
# Add link local info when registering Ironic node
IRONIC_USE_LINK_LOCAL=True

IRONIC_ENABLED_NETWORK_INTERFACES=flat,neutron
IRONIC_NETWORK_INTERFACE=neutron

#Networking configuration
OVS_PHYSICAL_BRIDGE=brbm
PHYSICAL_NETWORK=mynetwork
IRONIC_PROVISION_NETWORK_NAME=ironic-provision
IRONIC_PROVISION_SUBNET_PREFIX=10.0.5.0/24
IRONIC_PROVISION_SUBNET_GATEWAY=10.0.5.1

Q_PLUGIN=ml2
ENABLE_TENANT_VLANS=True
Q_ML2_TENANT_NETWORK_TYPE=vlan
TENANT_VLAN_RANGE=100:150
# Neutron public network type was changed to flat by default
# in neutron commit 1554adef26bd3bd184ddab668660428bdf392232
Q_USE_PROVIDERNET_FOR_PUBLIC=False

# Credentials
ADMIN_PASSWORD=password
RABBIT_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=password
SWIFT_HASH=password
SWIFT_TEMPURL_KEY=password

# Enable Ironic API and Ironic Conductor
enable_service ironic
enable_service ir-api
enable_service ir-cond

# Enable Neutron which is required by Ironic and disable nova-network.
disable_service n-net
disable_service n-novnc
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron

# Enable Swift for agent_* drivers
enable_service s-proxy
enable_service s-object
enable_service s-container
enable_service s-account

# Disable Horizon
disable_service horizon

# Disable Heat
disable_service heat h-api h-api-cfn h-api-cw h-eng
```

(continues on next page)

(续上页)

```
# Disable Cinder
disable_service cinder c-sch c-api c-vol

# Disable Tempest
disable_service tempest

# Swift temp URL's are required for agent_* drivers.
SWIFT_ENABLE_TEMPURLS=True

# Create 3 virtual machines to pose as Ironic's baremetal nodes.
IRONIC_VM_COUNT=3
IRONIC_BAREMETAL_BASIC_OPS=True

# Enable Ironic drivers.
IRONIC_ENABLED_DRIVERS=fake,agent_ipmitool,pxe_ipmitool

# Change this to alter the default driver for nodes created by devstack.
# This driver should be in the enabled list above.
IRONIC_DEPLOY_DRIVER=agent_ipmitool

# The parameters below represent the minimum possible values to create
# functional nodes.
IRONIC_VM_SPECS_RAM=1024
IRONIC_VM_SPECS_DISK=10

# Size of the ephemeral partition in GB. Use 0 for no ephemeral partition.
IRONIC_VM_EPHEMERAL_DISK=0

# To build your own IPA ramdisk from source, set this to True
IRONIC_BUILD_DEPLOY_RAMDISK=False

VIRT_DRIVER=ironic

# By default, DevStack creates a 10.0.0.0/24 network for instances.
# If this overlaps with the hosts network, you may adjust with the
# following.
NETWORK_GATEWAY=10.1.0.1
FIXED_RANGE=10.1.0.0/24
FIXED_NETWORK_SIZE=256

# Log all output to files
LOGFILE=$HOME/devstack.log
LOGDIR=$HOME/logs
IRONIC_VM_LOG_DIR=$HOME/ironic-bm-logs

GIT_BASE=http://git.trystack.cn
```

4. 开始部署

```
$ ./stack.sh
```

18.1 配置

18.1.1 加速

1. 由于 openstack 的源在国内下载比较慢，所有把 git 源设置成了 `trystack` 的地址；
2. pip 可以提前配置成国内源；
3. yum 或 deb 源提前设置成国内源；

18.1.2 离线模式

local.conf 文件中添加:

```
OFFLINE=True
RECLONE=no
```

18.2 说明

ocata 版本的虚机是挂在 linux 桥上的，实测这种方式跟 `brbm` 桥是不通，不知道哪里配置不对， 后来把虚机直接挂到 `brbm` 上可以了。

CHAPTER 19

7.2 Virtualbmc使用

通常情况下，我们要使用 IPMI必须使用有带外管理功能的物理机。但是在很多测试环境，我们使用的是虚拟机。virtualbmc是一个可以使用 IPMI命令来控制虚机的openstack 组件。

19.1 Virtualbmc 安装

```
pip install virtualbmc
```

如果 pip 安装失败，可能是你环境中依赖包没有安装。

```
sudo yum install gcc libvirt-devel python-devel
```

19.2 Virtualbmc 使用

1. 查看环境中的虚拟机

```
$ virsh list --all
Id    Name                           State
----------------------------------------------------
12    centos7.0-3                    running
```

2. 给虚机添加 vmbc

```
vbmc add centos7.0-3 --port 6230
```

3. 查看 vmbc 信息

```
$ vbmc list
+--------------+--------+---------+------+
| Domain name | Status | Address | Port |
```

(continues on next page)

(续上页)

```
+-------------+--------+---------+------+
| centos7.0-3 |  down  |    ::   | 6233 |
+-------------+--------+---------+------+
```

4. 启动vbmc

```
$ vbmc start centos7.0-3
```

5. ipmi 控制虚机

   这里 ipmi 的默认用户名和密码分别为 admin 和 password， 用户可以通过—username 和 —password 来指定自己的用户名和密码。

```
$ ipmitool -I lanplus -H 127.0.0.1 -U admin -P password -p 6233 power␣
↪status
Chassis Power is on
```

19.3 常用命令

```
# 查看帮助
$ vbmc --help

# 添加vbmc
$ vbmc add node-0

# 启动vbmc
$ vbmc start node-0

# 停止vmbc
$ vbmc stop node-0

# 查看vmbc 列表
$ vbmc list

# 查看某个虚机vmbc 信息
$ vbmc show node-0
```

19.4 说明

- vmbc 使用不同的端口号来映射到不同的虚机；
- 使用vbmc add 命令时，是在用户的$HOME/.vbmc/node_name/config 里记录 vbmc 的映射信息，vbmc list 也是查看当前用户的 vbmc信息。虽然不同用户记录文件在不同的地方，但是端口号不能重复，ipmitool 命令本身不区分
- vmbc 支持大部分的 IPMI 命令，但仍然有部分命令不支持， 例如 sol；

CHAPTER 20

8.0 裸机系统配置

单独的用户镜像部署到物理机之后，大部分情况下并不能直接使用， 我们需要对系统做一些配置，例如网络配置，分区修改等。

本章我们介绍使用 `config drive` + `cloud-init` 来配置系统。

注解： 如果是 windows 系列的系统，使用 `cloudbase-init`

CHAPTER 21

8.1 Config drive 使用

21.1 Config drive 介绍

Config drive 本质上是 base64 编码之后的数据。 Config drive 只能是 ISO 9600 或者是 VFAT 文件系统，这个跟 nova 的配置有关。 我们看看 Config driver 解压之后的目录结构：

```
$ tree configdrive
/configdrive
├── ec2
│   ├── 2009-04-04
│   │   ├── meta-data.json
│   │   └── user-data
│   └── latest
│       ├── meta-data.json
│       └── user-data
└── openstack
    ├── 2012-08-10
    │   ├── meta_data.json
    │   └── user_data
    ├── 2013-04-04
    │   ├── meta_data.json
    │   └── user_data
    ├── 2013-10-17
    │   ├── meta_data.json
    │   ├── user_data
    │   └── vendor_data.json
    ├── 2015-10-15
    │   ├── meta_data.json
    │   ├── network_data.json
    │   ├── user_data
    │   └── vendor_data.json
    └── latest
        ├── meta_data.json
        ├── network_data.json
```

(continues on next page)

(续上页)

```
        ├── user_data
        └── vendor_data.json
```

Config drive 分区了两个部分，分别是 ec2 和 openstack, 其中 ec2 是亚马逊云使用的，我们这里只介绍 openstack.

Openstack 组下有分了 2012-08-10, 2013-10-17, 2015-10-15 和 latest, 这些代表这 Openstack 的版本，对应如下：

- 2012-08-10: Essex
- 2013-10-17: HAVANA
- 2015-10-15: LIBERTY

每个版本下面对应如下文件：

- meta_data.json: 存储 nova instance 相关信息;
- network_data.json: 存储租户网络相关信息;
- user_data: 用户自定义脚本;
- vendor_data.json: 一般为空;

我们比较关注的是 network_data.json 和 user_data, 通过 network_data.json 我们知道该怎么配置网络。 而 user_data 是用户自己编写的脚本。

21.2 Config drive 信息保存

Config drive 本身并不区分虚拟机和物理机，但是虚拟机和物理机 Config drive 存储的方式有些差异。

21.2.1 虚拟机

对于虚拟机比较简单，在虚拟机所在计算节点生成 disk.config 文件，修改虚机的 xml 文件， 添加一个 disk 标签。这样虚机启动时就能看到这个分区了。

既然已经有了分区，那么程序怎么知道 config drive 数据存放在哪呢？ Nova 在生成 config driver 时会加上一个 config-2 的 label。 也就是说不管 config drive 在哪个分区，盘符是什么， 都能通过 /dev/disk/by-label/config-2 找到它（仅 linux 系统）。

21.2.2 物理机

物理机并不存在 xml 文件，没有办法直接挂载。目前的做法是 Nova 把 config drive 数据传给 ironic, ironic 在部署的时候把 config drive 写到物理机磁盘的最后。

注解： Config drive 最大是 64M，ironic 会在物理机磁盘的最后创建一个 64M 的分区， 然后把数据 dd 到该分区。

21.3 使用 config drive

使用 config drive 需要在命令行指定 `--config-drive true`

```
nova boot --config-drive true --flavor baremetal --image test-image instance-1
```

或者通过 nova 配置文件，强制所有 instance 使用 config drive:

```
[DEFAULT]
...

force_config_drive=True
```

CHAPTER 22

8.3 cloud-init 介绍

cloud-init 是一个系统配置工具，当系统起来时， cloud-init 会从固定分区读取数据，然后执行定制化 操作。
cloud-init 有四个执行阶段：

- local: cloud-init-local.service
- init: cloud-init.service
- config: cloud-config.service
- final: cloud-final.service

22.1 配置

cloud-init 各阶段完成哪些工作可以在 `/etc/cloud/cloud.cfg` 中查看

1. local 阶段
   作为 cloud-init 的初始阶段，此时主要完成网口的配置
   `/usr/bin/cloud-init init --local`
2. init 阶段
3. config 阶段
4. final 阶段

22.2 cloud-init 脚本生成

社区提供了 *write-mime-multipart* 工具来生成 cloud-init 脚本。 使用方法如下：

```
$ cat my-boothook.txt
#!/bin/sh
echo "Hello World!"
echo "This will run as soon as possible in the boot sequence"

$ cat my-user-script.txt
#!/usr/bin/perl
print "This is a user script (rc.local)\n"

$ cat my-include.txt
# these urls will be read pulled in if they were part of user-data
# comments are allowed.  The format is one url per line
http://www.ubuntu.com/robots.txt
http://www.w3schools.com/html/lastpage.htm

$ cat my-upstart-job.txt
description "a test upstart job"
start on stopped rc RUNLEVEL=[2345]
console output
task
script
echo "====BEGIN======="
echo "HELLO From an Upstart Job"
echo "=====END========"
end script

$ cat my-cloudconfig.txt
#cloud-config
ssh_import_id: [smoser]
apt_sources:
 - source: "ppa:smoser/ppa"

$ write-mime-multipart --output=combined-userdata.txt \
    my-boothook.txt:text/cloud-boothook \
    my-include.txt:text/x-include-url \
    my-upstart-job.txt:text/upstart-job \
    my-user-script.txt:text/x-shellscript \
    my-cloudconfig.txt
```

22.3 User Data 输入格式

1. Gzip Compressed Content
2. Mime Multi Part archive
3. User-Data Script

   以 `#!` 或 `Content-Type:  text/x-shellscript` 开头。

   存放用户脚本，可以是 python, shell, perl 等。`user data` 里的脚本执行比较晚，类似 rc.local， 而且只在第一次启动时 执行。

4. Include File

   以 `#include` 或 `Content-Type:  text/x-include-url` 开头。

   `includ file` 的内容是一串 url, 每行一个，cloud-init 启动时会 读取 url 链接的内容。

5. Cloud Config Data

   以 `#cloud-config` 或 `Content-Type: text/cloud-config` 开头，内容按规定格式编写。

6. Upstart Job

   以 `#upstart-job` 或 `Content-Type: text/upstart-job` 开头，

   文件内容会存放在 */etc/init*，以供其它 job 使用。

7. Cloud Boothook

   以 `#cloud-boothook` 或 `Content-Type: text/cloud-boothook` 开头。

   `boothook` 数据会保存到 */var/lib/cloud*，然后立刻执行。`boothook` 每次上电都会执行，没有机制指定只运行一次。

8. Part Handler

22.4 测试

如果要测试 cloud-init 脚本，每次通过 `nova boot` 来操作有点麻烦。这里提供两种便捷的方式：

22.4.1 修改 xml 文件

1. 准备好你的 configdrive 文件，这里假设是 disk.cfg;
2. 虚机 xml 文件中添加一个 disk 标签，内容如下:

```
<disk type="file" device="cdrom">
    <driver name="qemu" type="raw" cache="none"/>
    <source file="/root/disk.cfg"/>
    <target bus="ide" dev="hdd"/>
</disk>
```

3. 通过 virsh 启动虚机

当我们需要修改 disk.cfg 内容时，进行如下操作：

```
$ sudo mount /root/disk.cfg /mnt
$ mkdir configdrive
$ cp -r /mnt/* configdrive
$ genisoimage -o disk.cfg -ldots -allow-lowercase -allow-multidot -l -quiet \
  -J -r -V 'config-2' configdrive
```

22.4.2 使用 config-2 分区

1. 启动虚机;
2. 创建一个 64M 的分区;
3. 把 configdrive 文件 dd 到该分区，eg:

```
$ sudo dd if=/root/disk.cfg of=/dev/sda4
```

4. 重启系统

以上两种方式如果要多次运行，需要在下次运行前把 cloud-init 记录删除:

```
$ sudo rm -rf /var/lib/cloud/*
```

CHAPTER 23

9.0 Ironic 常见故障

23.1 Nova 返回 “No valid host was found” 错误

有时部署的时候，在 “nova-conductor.log” 或者 http 返回中有如下错误:

```
NoValidHost: No valid host was found. There are not enough hosts available.
```

“No valid host was found” 是指 Nova Scheduler 没有找到合适的裸机来部署新的实例。

出现这个错误时，做如下检查:

1. 确认有足够的裸机在 available 状态，且没有进入维护模式，没有和 instance 关联 可以通过如下命令检查:

   ```
   ironic node-list --provision-state available --maintenance false --associated␣
   ↪false
   ```

   如果上面的命令没有显示 node, 使用 ironic node-list 来检查其它节状态点. 例如, manageable 状态的节点。 通过如下命令切换到 available:

   ```
   ironic node-set-provision-state <IRONIC NODE> provide
   ```

   当裸机发生异常时，例如电源同步失败，ironic 会自动把 node 设置成维护模式。 这时候要检查电源认证信息(eg: ipmi_address, ipmi_username 和 ipmi_password) 同时检查带外网络是否正常。如果恢复，通过下面的命令移除维护模式:

   ```
   ironic node-set-maintenance <IRONIC NODE> off
   ```

   ironic node-validate 用来检查各个字段是否符合要求, 执行如下命令，正常情况下不应该有返回:

   ```
   ironic node-validate baremetal-0 | grep -E '(power|management)\W*False'
   ```

   如果自动清理失败了，ironic 也会把 node 设置为维护模式。

2. 确保每个 available 状态的 node 的 properties 属性中包含： cpus, cpu_arch, memory_mb 和 local_gb. 如果没有这些信息，你需要手动输入或者通过 Inspection 流程来收集。 正常情况下，看到的信息如下:

```
$ ironic node-show <IRONIC NODE> --fields properties
+------------+--------------------------------------------------------------------
↪----------------+
| Property   | Value                                                              ␣
↪                |
+------------+--------------------------------------------------------------------
↪----------------+
| properties | {u'memory_mb': u'8192', u'cpu_arch': u'x86_64', u'local_gb': u'41',
↪ u'cpus': u'4'} |
+------------+--------------------------------------------------------------------
↪----------------+
```

警告： 如果使用 Nova Scheduler 的 exact match filters，要确保上面看到的属性和 flavor 精确匹配。

3. Nova flavor 和 ironic node properties 不匹配 查看 flaover 信息:

```
openstack flavor show <FLAVOR NAME>
```

比较 Flavor capability: 和 Ironic node 的 node.properties['capabilities'].

注解： capabilities 的格式在 Nova 和 Ironic 中略有差异. E.g. in Nova flavor:

```
$ openstack flavor show <FLAVOR NAME> -c properties
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| properties | capabilities:boot_option='local' |
+------------+----------------------------------+
```

Ironic node:

```
$ ironic node-show <IRONIC NODE> --fields properties
+------------+----------------------------------------+
| Property   | Value                                  |
+------------+----------------------------------------+
| properties | {u'capabilities': u'boot_option:local'} |
+------------+----------------------------------------+
```

---

4. 当 Ironic node 信息发生变化时，Nova 同步信息需要一段时间， 大概是 1min，可以通过如下命令查看资源信息

```
openstack hypervisor stats show
```

查看具体节点的资源信息，使用 openstack hypervisor show <IRONIC NODE>

5. 查看是哪个 Nova Scheduler filter 没通过, 可以在 nova-scheduler 日志中搜索:

```
Filter ComputeCapabilitiesFilter returned 0 hosts
```

找到哪个 filter 把节点都过滤了，关于 fileter 的作用可以参考: Nova filters documentation

6. 如果上面的检查都没发现什么问题，检查 Ironic conductor 日志，看看有么有 相关的错误，导致 “No valid host was found”.

23.2 Patching the Deploy Ramdisk

当调试部署或者 inspection 问题时，为了方便定位，你可能想快速修改 ramdisk 的内容。 一种是制作 ramdisk 的时候注入脚本，更通用的做法是直接解压。

创建一个空目录，解压 ramdisk 的内容到该目录:

```
mkdir unpack
cd unpack
gzip -dc /path/to/the/ramdisk | cpio -id
```

修改完 ramdisk 文件之后，重新打包 ramdisk:

```
find . | cpio -H newc -o > /path/to/the/new/ramdisk
```

注解: 不要修改 kernel 部分(e.g. `tinyipa-master.vmlinuz`), 仅修改 ramdisk 的内容.

注解: CentOS 系列的 ramdisk 需要解压 ramdisk 里的 squashfs.

23.3 Retrieving logs from the deploy ramdisk

当部署失败是，分析 ramdisk 的日志往往很有帮助。当部署失败时， Ironic 会默认保存 ramdisk 日志到 `/var/log/ironic/deploy` 目录。

`/etc/ironic/ironic.conf` 文件的 `[agent]` 组:

- `deploy_logs_collect`: Ironic 是否收集部署阶段的日志，有效配置项:
  - `on_failure` (**default**): 部署失败时收集。
  - `always`: 所有情况都收集。
  - `never`: 不收集。
- `deploy_logs_storage_backend`: 部署日志存储后端。
  - `local` (**default**): 存放在本地文件系统。
  - `swift`: 存放在 Swift.
- `deploy_logs_local_path`: 部署日志存放路径，只有配置 `deploy_logs_storage_backend` 为 local, 才有效。默认存放在 **/var/log/ironic/deploy**.

23.4 PXE 或 iPXE DHCP 不正确或地址要不到

这可能是由某些交换机上的生成树协议延迟引起的。 该延迟阻止交换机端口尝试PXE， 因此数据包不会将其发送到DHCP服务器。 解决这个问题你应该设置连接到你的裸金属节点的交换机端口作为边缘或PortFast类型端口。 以这种方式配置交换机端口一旦建立链接，就转到转发模式。

Cisco Nexus交换机配置如下：

```
$ config terminal
$ (config) interface eth1/11
$ (config-if) spanning-tree port type edge
```

CHAPTER 24

9.1 Inspect 常见问题

24.1 Introspection 开始时出错

- *Invalid provision state "available"*

  进行Introspection 时，node 的初始状态必须为 `manageable`, 如果是 `available` 状态，可通过如下命令切换:

```
ironic node-set-provision-state <IRONIC NODE> manage
```

24.2 Introspection 超时

Introspection 超时有三种可能（默认超时时间是 60min, 可以通过配置项 timeout 来更改）：

1. 处理数据错误. 参考 *Troubleshooting data processing*.
2. 下载镜像错误。参考 *Troubleshooting PXE boot* .
3. 运行错误. 参考 *Troubleshooting ramdisk run*.

24.2.1 Troubleshooting data processing

主要检查 **ironic-inspector** 的日志:

```
sudo journalctl -u openstack-ironic-inspector
```

(use `openstack-ironic-discoverd` for version < 2.0.0).

如果配置了 `ramdisk_error` 和 `ramdisk_logs_dir`, **ironic-inspector** 会接收裸机的日志，并保存到 `ramdisk_logs_dir` 目录。 implementation.

24.2.2 Troubleshooting PXE boot

Introspection 大部分问题都是 PXE 启动失败，如果带外网络可以连接，登录 KVM 排查问题。

查看 DHCP 和 TFTP 日志:

```
$ sudo journalctl -u openstack-ironic-inspector-dnsmasq
```

(use `openstack-ironic-discoverd-dnsmasq` for version < 2.0.0).

使用 `tcpdump` 抓 DHCP 和 TFTP 报文

```
$ sudo tcpdump -i any port 67 or port 68 or port 69
```

把上面的 any 换成实际的 DHCP 口，并观察服务器是否收到 DHCP 和 TFTP 报文。

如果发现裸机没有从 PXE 启动，或者启动的网口不正确，请配置 BIOS 和交换机。

如果 PXE 启动失败，做如下检查:

1. 交换机配置正确，检查 VLAN 信息，DHCP 配置.
2. 检查是否有防火墙规则，拦截了 67 端口.

如果裸机要到了 DHCP IP， 但是下载内核镜像失败了，做如下检查:

1. TFTP 正常且可以访问，检查 xinet 服务(或者dnsmasq)，检查 SELinux,
2. 没有防火墙规则拦截 TFTP 报文,
3. DHCP options 中的 TFTP server 地址正确,
4. `pxelinux.cfg/default` 中的 kernel 和 ramdisk 路径正确。

注解: 如果使用的是 iPXE，检查 HTTP 服务器的日志以及 iPXE 的配置。

24.2.3 Troubleshooting ramdisk run

如果配置了接收日志，先查看日志，检查错误，具体配置参考： *Troubleshooting data processing* 章节。

如果 KVM 获取网络可以连接裸机，登录上去，检查服务状态，查看 journalctl 日志。

关于怎么动态登录裸机，可以参考文档:

CHAPTER 25

Indices and tables

- genindex
- modindex
- search