# neutron\_fwaas学习

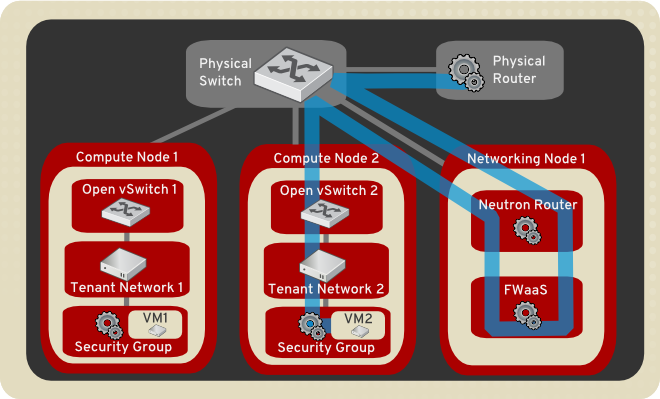
## 简介

Neutron\_fwaas是从neutron中独立出来的一个项目，提供Firewall-as-a-Service的功能，引用官方的介绍：

The Firewall-as-a-Service (FWaaS) plug-in adds perimeter firewall management to Networking. FWaaS uses iptables to apply firewall policy to all Networking routers within a project. FWaaS supports one firewall policy and logical firewall instance per project.

Whereas security groups operate at the instance-level, FWaaS operates at the perimeter to filter traffic at the neutron router.

可以理解，同样是iptables规则，security group是在计算节点上虚拟机的外部生效，而fwaas是在router上生效，官方文档的图上看得很清楚：



## 安装配置

这里简单介绍一下在centos7上的安装过程。

要使用fwaas，首先需要在网络节点安装fwaas的包：

yum install openstack-neutron-fwaas

安装完后需要修改一些配置项：

vi /etc/neutron/neutron.conf

service\_plugins = router,neutron\_fwaas.services.firewall.fwaas\_plugin.FirewallPlugin

[fwaas]

driver = neutron\_fwaas.services.firewall.drivers.linux.iptables\_fwaas.IptablesFwaasDriver

enabled = True

升级数据库：

neutron-db-manage --service fwaas upgrade head

修改horizon的配置文件local\_settings.py：

OPENSTACK\_NEUTRON\_NETWORK = {

...

'enable\_firewall' = True,

...

}

重启apache

systemctl restart httpd

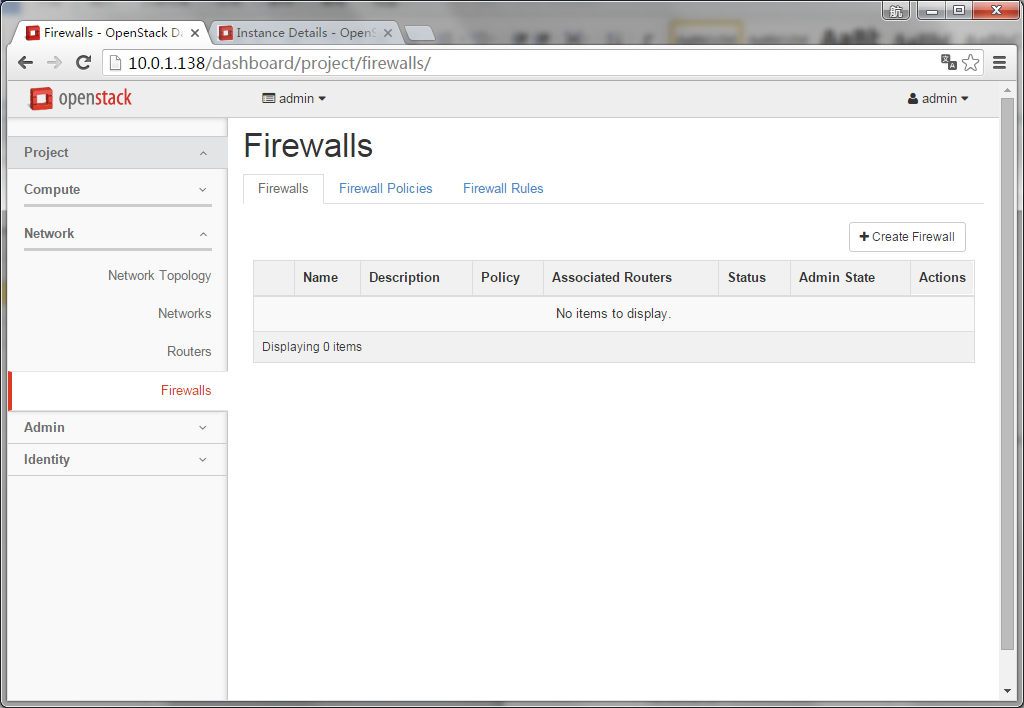
重启neutron-server和l3-agent

systemctl restart neutron-l3-agent

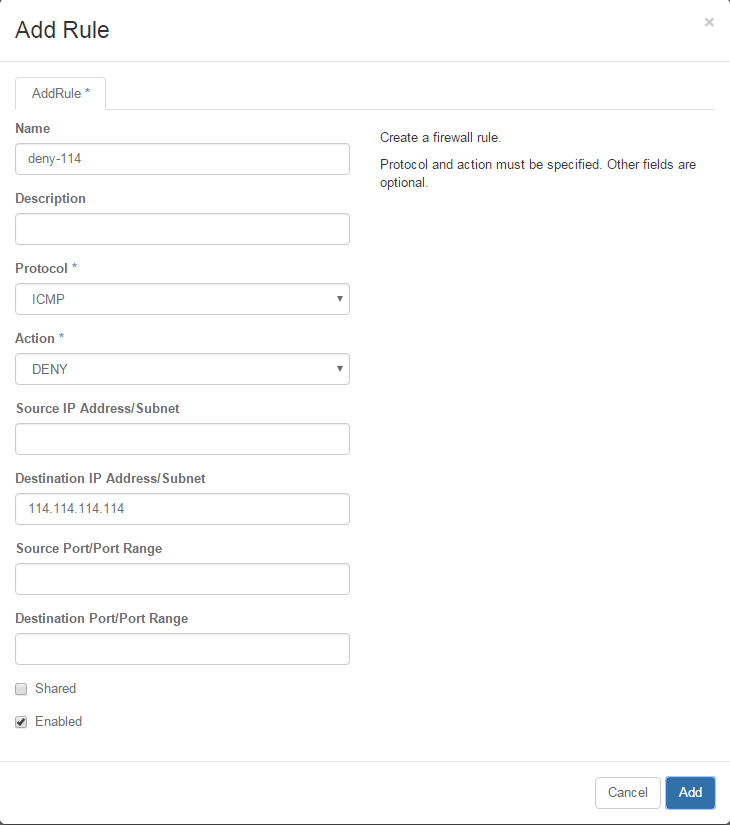
systemctl restart neutron-server

## 用户操作过程

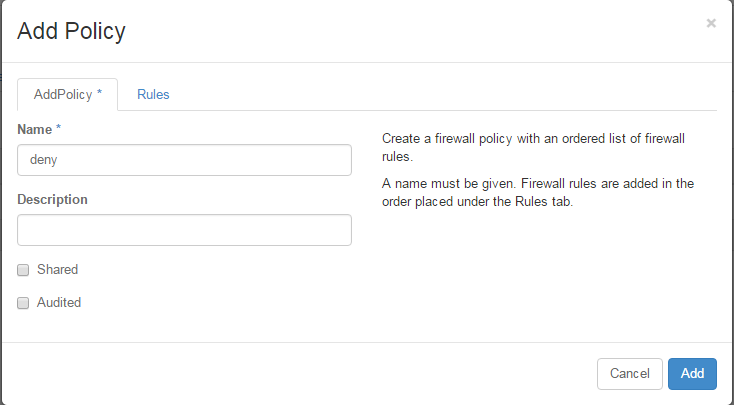
可以选择从horizon界面或者CLI创建防火墙，这里展示界面创建的过程：

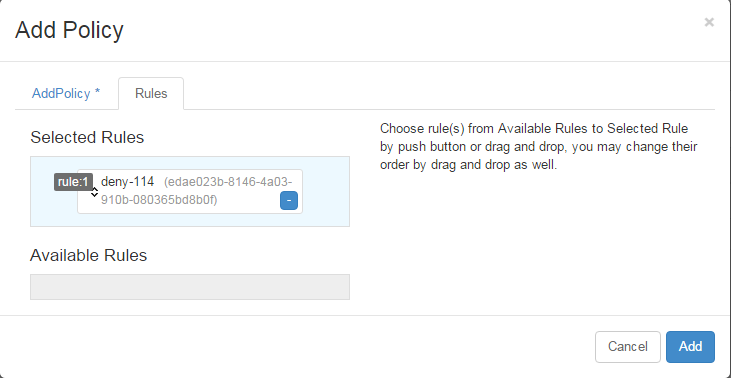


首先创建一条防火墙规则：

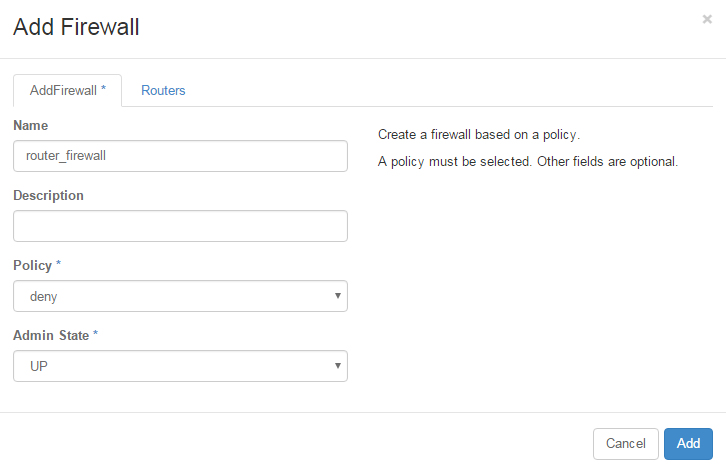


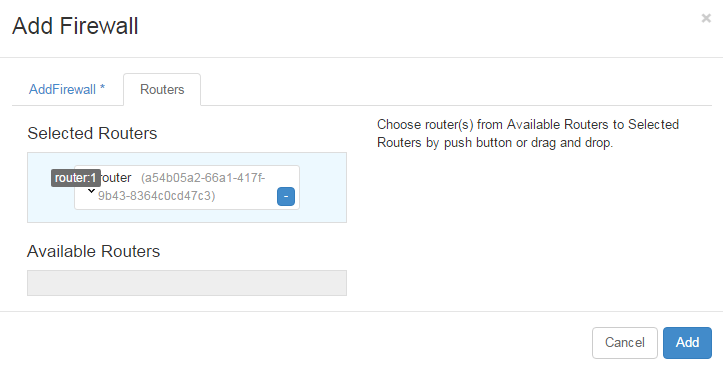
然后创建一个防火墙策略并插入刚才创建的规则：





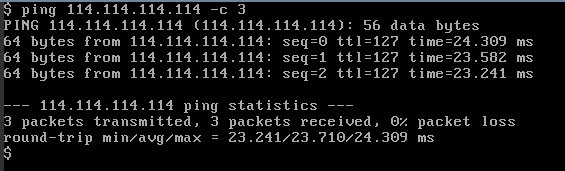
最后创建防火墙，并选择要关联的路由器：



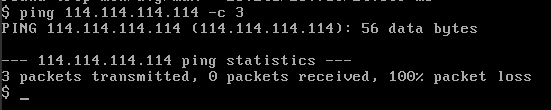


## 功能验证

防火墙未关联路由器时ping 114可以通：

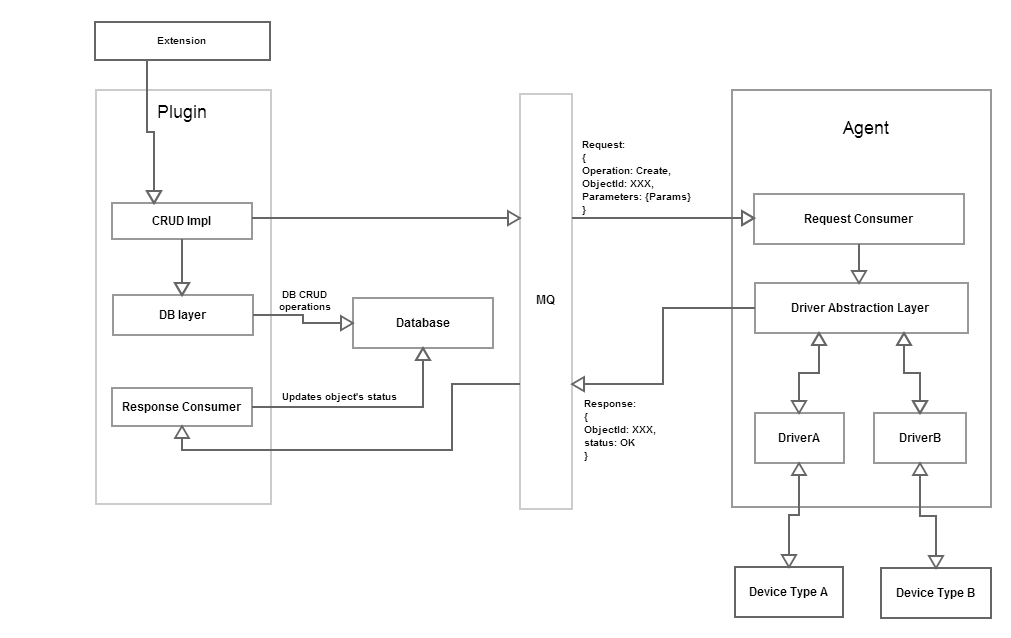


防火墙关联路由器后，ping 114不通：



## 代码逻辑简介

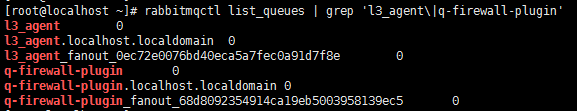
用一句话来概括Fwaas的架构：**plugin接收请求，先在数据库创建对象的记录，然后通过rpc调用agent，agent接收rpc请求后在driver上执行请求中对应的方法完成实际操作，并通过rpc将执行后的status返回给plugin，plugin更新数据库中的status。**引用一张lbaas wiki上的图，他们是类似的，虽然是三年前的，但架构没什么变化，清楚地描述了这个过程。



看过了大的框架，我们知道fwaas分为三部分**plugin,agent,driver**。

在plugin和agent之间的队列有两个：**l3\_agent**和**q-firewall-plugin。**

可以用rabbitmqctl list\_queues看到：



下面我们分别看一下这三部分关键的代码。

### plugin

Fwaas的API和lbaas，vpnaas一样，是作为neutron api的extension，neutron-server会在启动的时候根据service\_plugins的配置来加载extension，这部分的代码主要在neutron/api/extensions.py中。

先看一下这个配置项的内容：

service\_plugins = router,neutron\_fwaas.services.firewall.fwaas\_plugin.FirewallPlugin

也就是neutron\_fwaas/services/firewall/fwaas\_plugin.py中的FirewallPlugin类

这个FirewallPlugin类的对象会在neutron server启动的时候被创建

这个对象在创建时第一步会调用**start\_rpc\_listeners**，这个方法如其名，启动了plugin的RPC server，topic是**q-firewall-plugin**。

第二步创建一个rpc client对象， topic是**l3\_agent**。

当plugin接收到用户的请求时会发送rpc请求到**l3\_agent**这个队列里，监听这个队列的RPCserver就是neutron l3 agent。

class **FirewallPlugin**(

firewall\_db.Firewall\_db\_mixin,

firewall\_router\_insertion\_db.FirewallRouterInsertionDbMixin):

*"""Implementation of the Neutron Firewall Service Plugin.*

*This class manages the workflow of FWaaS request/response.*

*Most DB related works are implemented in class*

*firewall\_db.Firewall\_db\_mixin.*

*"""*

supported\_extension\_aliases = [*"fwaas"*, *"fwaasrouterinsertion"*]

path\_prefix = fw\_ext.FIREWALL\_PREFIX

def **\_\_init\_\_**(*self*):

*"""Do the initialization for the firewall service plugin here."""*

***self*.start\_rpc\_listeners()**

*self*.agent\_rpc = FirewallAgentApi(

f\_const.L3\_AGENT,

cfg.CONF.host

)

firewall\_db.subscribe()

def **start\_rpc\_listeners**(*self*):

***self*.endpoints = [FirewallCallbacks(*self*)]**

***self*.conn = n\_rpc.create\_connection()**

***self*.conn.create\_consumer(**

**f\_const.FIREWALL\_PLUGIN, *self*.endpoints, fanout=False)**

return *self*.conn.consume\_in\_threads()

class **FirewallCallbacks** (object):

target = oslo\_messaging.Target(version=*'1.0'*)

def **\_\_init\_\_**(*self*, plugin):

super(FirewallCallbacks, *self*).\_\_init\_\_()

*self*.plugin = plugin

def **set\_firewall\_status**(*self*, context, firewall\_id, status, \*\*kwargs):

*"""Agent uses this to set a firewall's status."""*

LOG.debug(*"Setting firewall %s to status: %s"* % (firewall\_id, status))

# Sanitize status first

if status in (n\_const.ACTIVE, n\_const.DOWN, n\_const.INACTIVE):

to\_update = status

else:

to\_update = n\_const.ERROR

# ignore changing status if firewall expects to be deleted

# That case means that while some pending operation has been

# performed on the backend, neutron server received delete request

# and changed firewall status to PENDING\_DELETE

updated = *self*.plugin.update\_firewall\_status(

context, firewall\_id, to\_update, not\_in=(n\_const.PENDING\_DELETE,))

if updated:

LOG.debug(*"firewall %s status set: %s"* % (firewall\_id, to\_update))

return updated and to\_update != n\_const.ERROR

def **firewall\_deleted**(*self*, context, firewall\_id, \*\*kwargs):

*"""Agent uses this to indicate firewall is deleted."""*

LOG.debug(*"firewall\_deleted() called"*)

with context.session.begin(subtransactions=True):

fw\_db = *self*.plugin.\_get\_firewall(context, firewall\_id)

# allow to delete firewalls in ERROR state

if fw\_db.status in (n\_const.PENDING\_DELETE, n\_const.ERROR):

*self*.plugin.delete\_db\_firewall\_object(context, firewall\_id)

return True

else:

LOG.warning(\_LW(*'Firewall %(fw)s unexpectedly deleted by '*

*'agent, status was %(status)s'*),

{*'fw'*: firewall\_id, *'status'*: fw\_db.status})

fw\_db.update({*"status"*: n\_const.ERROR})

return False

### agent

l3 agent 在启动的时候，会创建一个neutron.service.Service对象，它继承了neutron.common.rpc. Service，调用**create\_consumer**，topic是**l3\_agent**。

class **Service**(service.Service):

*"""Service object for binaries running on hosts.*

*A service enables rpc by listening to queues based on topic and host.*

*"""*

def **\_\_init\_\_**(*self*, host, topic, manager=None, serializer=None):

super(Service, *self*).\_\_init\_\_()

*self*.host = host

*self*.topic = topic

*self*.serializer = serializer

if manager is None:

*self*.manager = *self*

else:

*self*.manager = manager

def **start**(*self*):

super(Service, *self*).start()

*self*.conn = create\_connection(new=True)

LOG.debug(*"Creating Consumer connection for Service %s"*,

*self*.topic)

endpoints = [*self*.manager]

*self*.conn.create\_consumer(*self*.topic, endpoints)

# Hook to allow the manager to do other initializations after

# the rpc connection is created.

if callable(getattr(*self*.manager, *'initialize\_service\_hook'*, None)):

*self*.manager.initialize\_service\_hook(*self*)

# Consume from all consumers in threads

*self*.conn.consume\_in\_threads()

与此同时，l3\_agent创建时，默认的manager如下：

def **main**(manager=*'neutron.agent.l3.agent.L3NATAgentWithStateReport'*):

register\_opts(cfg.CONF)

common\_config.init(sys.argv[1:])

config.setup\_logging()

server = neutron\_service.Service.create(

binary=*'neutron-l3-agent'*,

topic=topics.L3\_AGENT,

report\_interval=cfg.CONF.AGENT.report\_interval,

manager=manager)

service.launch(cfg.CONF, server).wait()

下面的代码片段可以看到它的继承关系：

class **L3NATAgentWithStateReport**(L3NATAgent):

class **L3NATAgent**(firewall\_l3\_agent.FWaaSL3AgentRpcCallback,

ha.AgentMixin,

dvr.AgentMixin,

manager.Manager):

下方代码中的**红色**部分读取了安装时配置文件中的这两个参数：

**[fwaas]**

**driver = neutron\_fwaas.services.firewall.drivers.linux.iptables\_fwaas.IptablesFwaasDriver**

**enabled = True**

这部分的逻辑基本就是读取配置文件中fwaas的driver并创建driver的对象，l3 agent接收到plugin的RPC请求后会调用实际driver对象中对应的**create\_firewall，update\_firewall，delete\_firewall**方法，然后设置firewall的status并发送RPC请求到plugin在监听的**q-firewall-plugin**队列，这部分逻辑是下方代码中**绿色**的部分。

plugin收到请求后就会执行**set\_firewall\_status** 或者**firewall\_deleted**更新数据库中firewall的status。

这两个方法的代码在前文的plugin部分。

class **FWaaSL3AgentRpcCallback**(api.FWaaSAgentRpcCallbackMixin):

*"""FWaaS Agent support to be used by Neutron L3 agent."""*

def **\_\_init\_\_**(*self*, conf):

LOG.debug(*"Initializing firewall agent"*)

*self*.conf = conf

**fwaas\_driver\_class\_path = provconf.get\_provider\_driver\_class(**

**cfg.CONF.fwaas.driver)**

***self*.fwaas\_enabled = cfg.CONF.fwaas.enabled**

# None means l3-agent has no information on the server

# configuration due to the lack of RPC support.

if *self*.neutron\_service\_plugins is not None:

fwaas\_plugin\_configured = (constants.FIREWALL

in *self*.neutron\_service\_plugins)

if fwaas\_plugin\_configured and not *self*.fwaas\_enabled:

msg = \_(*"FWaaS plugin is configured in the server side, but "*

*"FWaaS is disabled in L3-agent."*)

LOG.error(msg)

raise SystemExit(1)

*self*.fwaas\_enabled = *self*.fwaas\_enabled and fwaas\_plugin\_configured

if *self*.fwaas\_enabled:

try:

*self*.fwaas\_driver = importutils.import\_object(

fwaas\_driver\_class\_path)

LOG.debug(*"FWaaS Driver Loaded: '%s'"*, fwaas\_driver\_class\_path)

except ImportError:

msg = \_(*'Error importing FWaaS device driver: %s'*)

raise ImportError(msg % fwaas\_driver\_class\_path)

*self*.services\_sync = False

# setup RPC to msg fwaas plugin

***self*.fwplugin\_rpc = FWaaSL3PluginApi(topics.FIREWALL\_PLUGIN,**

**conf.host)**

super(FWaaSL3AgentRpcCallback, *self*).\_\_init\_\_(host=conf.host)

.........

def **\_invoke\_driver\_for\_plugin\_api**(*self*, context, fw, func\_name):

*"""Invoke driver method for plugin API and provide status back."""*

LOG.debug(*"%(func\_name)s from agent for fw: %(fwid)s"*,

{*'func\_name'*: func\_name, *'fwid'*: fw[*'id'*]})

try:

routers = *self*.plugin\_rpc.get\_routers(context)

router\_info\_list = *self*.\_get\_router\_info\_list\_for\_tenant(

routers,

fw[*'tenant\_id'*])

if not router\_info\_list:

LOG.debug(*'No Routers on tenant: %s'*, fw[*'tenant\_id'*])

# fw was created before any routers were added, and if a

# delete is sent then we need to ack so that plugin can

# cleanup.

if func\_name == *'delete\_firewall'*:

*self*.fwplugin\_rpc.firewall\_deleted(context, fw[*'id'*])

return

LOG.debug(*"Apply fw on Router List: '%s'"*,

[ri.router[*'id'*] for ri in router\_info\_list])

# call into the driver

try:

*self*.fwaas\_driver.\_\_getattribute\_\_(func\_name)(

*self*.conf.agent\_mode,

router\_info\_list,

fw)

if fw[*'admin\_state\_up'*]:

status = constants.ACTIVE

else:

status = constants.DOWN

except nexception.FirewallInternalDriverError:

LOG.error(\_LE(*"Firewall Driver Error for %(func\_name)s "*

*"for fw: %(fwid)s"*),

{*'func\_name'*: func\_name, *'fwid'*: fw[*'id'*]})

status = constants.ERROR

# delete needs different handling

**if func\_name == *'delete\_firewall'*:**

**if status in [constants.ACTIVE, constants.DOWN]:**

***self*.fwplugin\_rpc.firewall\_deleted(context, fw[*'id'*])**

**else:**

***self*.fwplugin\_rpc.set\_firewall\_status(**

**context,**

**fw[*'id'*],**

**status)**

except Exception:

LOG.exception(

\_LE(*"FWaaS RPC failure in %(func\_name)s for fw: %(fwid)s"*),

{*'func\_name'*: func\_name, *'fwid'*: fw[*'id'*]})

*self*.services\_sync = True

return

.........

def **create\_firewall**(*self*, context, firewall, host):

*"""Handle Rpc from plugin to create a firewall."""*

return *self*.\_invoke\_driver\_for\_plugin\_api(

context,

firewall,

*'create\_firewall'*)

def **update\_firewall**(*self*, context, firewall, host):

*"""Handle Rpc from plugin to update a firewall."""*

return *self*.\_invoke\_driver\_for\_plugin\_api(

context,

firewall,

*'update\_firewall'*)

def **delete\_firewall**(*self*, context, firewall, host):

*"""Handle Rpc from plugin to delete a firewall."""*

return *self*.\_invoke\_driver\_for\_plugin\_api(

context,

firewall,

*'delete\_firewall'*)

### driver

从上面的配置文件可以看到driver配置如下

[fwaas]

driver = neutron\_fwaas.services.firewall.drivers.linux.iptables\_fwaas.IptablesFwaasDriver

这是社区默认的iptables的driver

如果要使用vyatta firewall的话这里要配成

driver = neutron\_fwaas.services.firewall.drivers.vyatta.vyatta\_fwaas.VyattaFirewallDriver

社区的iptables driver实现了三个方法**create\_firewall，update\_firewall，delete\_firewall。**

接收的参数是agent\_mode, apply\_list, firewall

agent\_mode是router的模式，可以在l3\_agent.ini中配置，默认router的是legacy

apply\_list是防火墙关联的router的列表，每个router元素都有属性iptables\_manager，用来在相应的router namespace中执行iptables命令

firewall是防火墙的对象，其中包含了防火墙的规则等各种信息

这里主要逻辑就是根据传入参数的防火墙规则，生成需要的iptables命令，然后通过iptables\_manager调用iptables\_table对象的各种方法，如add\_chain, remove\_chain, add\_rule, remove\_rule等，在相应的router namespace中执行相应的iptables命令，实现防火墙的功能。

class **IptablesFwaasDriver**(fwaas\_base.FwaasDriverBase):

*"""IPTables driver for Firewall As A Service."""*

def **\_\_init\_\_**(*self*):

LOG.debug(*"Initializing fwaas iptables driver"*)

def **create\_firewall**(*self*, agent\_mode, apply\_list, firewall):

LOG.debug(*'Creating firewall %(fw\_id)s for tenant %(tid)s'*,

{*'fw\_id'*: firewall[*'id'*], *'tid'*: firewall[*'tenant\_id'*]})

try:

if firewall[*'admin\_state\_up'*]:

*self*.\_setup\_firewall(agent\_mode, apply\_list, firewall)

else:

*self*.apply\_default\_policy(agent\_mode, apply\_list, firewall)

except (LookupError, RuntimeError):

# catch known library exceptions and raise Fwaas generic exception

LOG.exception(\_LE(*"Failed to create firewall: %s"*), firewall[*'id'*])

raise fw\_ext.FirewallInternalDriverError(driver=FWAAS\_DRIVER\_NAME)

def **delete\_firewall**(*self*, agent\_mode, apply\_list, firewall):

LOG.debug(*'Deleting firewall %(fw\_id)s for tenant %(tid)s'*,

{*'fw\_id'*: firewall[*'id'*], *'tid'*: firewall[*'tenant\_id'*]})

fwid = firewall[*'id'*]

try:

for router\_info in apply\_list:

ipt\_if\_prefix\_list = *self*.\_get\_ipt\_mgrs\_with\_if\_prefix(

agent\_mode, router\_info)

for ipt\_if\_prefix in ipt\_if\_prefix\_list:

ipt\_mgr = ipt\_if\_prefix[*'ipt'*]

*self*.\_remove\_chains(fwid, ipt\_mgr)

*self*.\_remove\_default\_chains(ipt\_mgr)

# apply the changes immediately (no defer in firewall path)

ipt\_mgr.defer\_apply\_off()

except (LookupError, RuntimeError):

# catch known library exceptions and raise Fwaas generic exception

LOG.exception(\_LE(*"Failed to delete firewall: %s"*), fwid)

raise fw\_ext.FirewallInternalDriverError(driver=FWAAS\_DRIVER\_NAME)

def **update\_firewall**(*self*, agent\_mode, apply\_list, firewall):

LOG.debug(*'Updating firewall %(fw\_id)s for tenant %(tid)s'*,

{*'fw\_id'*: firewall[*'id'*], *'tid'*: firewall[*'tenant\_id'*]})

try:

if firewall[*'admin\_state\_up'*]:

*self*.\_setup\_firewall(agent\_mode, apply\_list, firewall)

else:

*self*.apply\_default\_policy(agent\_mode, apply\_list, firewall)

except (LookupError, RuntimeError):

# catch known library exceptions and raise Fwaas generic exception

LOG.exception(\_LE(*"Failed to update firewall: %s"*), firewall[*'id'*])

raise fw\_ext.FirewallInternalDriverError(driver=FWAAS\_DRIVER\_NAME)