목차

Octavia LB 구현 및 분석 ... 3
- ***Octavia*** ... 3
- ***기본 객체 개념*** ... 3
- ***기본 사용 과정*** ... 4
- ***소프트웨어 아키텍처*** ... 11
  - 서비스 프로세스 목록 ... 12
  - 코드 구조 ... 12
- ***LoadBalaner 프로세스 분석*** ... 14
  - network_tasks.AllocateVIP ... 21
  - network_tasks.PlugVIP ... 22
- ***Amphora*** ... 24
  - Amphora Agent ... 25
  - AmphoraAPIClient ... 26
  - AmphoraePostVIPPlug ... 26
  - Keepalived 서비스 프로세스 시작 ... 28
- ***리스너 생성 프로세스 분석*** ... 31
- ***haproxy 서비스 프로세스 시작*** ... 33
- ***참조링크*** ... 33

Octavia LB 구현 및 분석

Octavia

`Octavia`는 `OpenStack`과 함께 작동하도록 설계된 오픈 소스, 운영자 규모 로드 밸런싱 솔루션입니다.

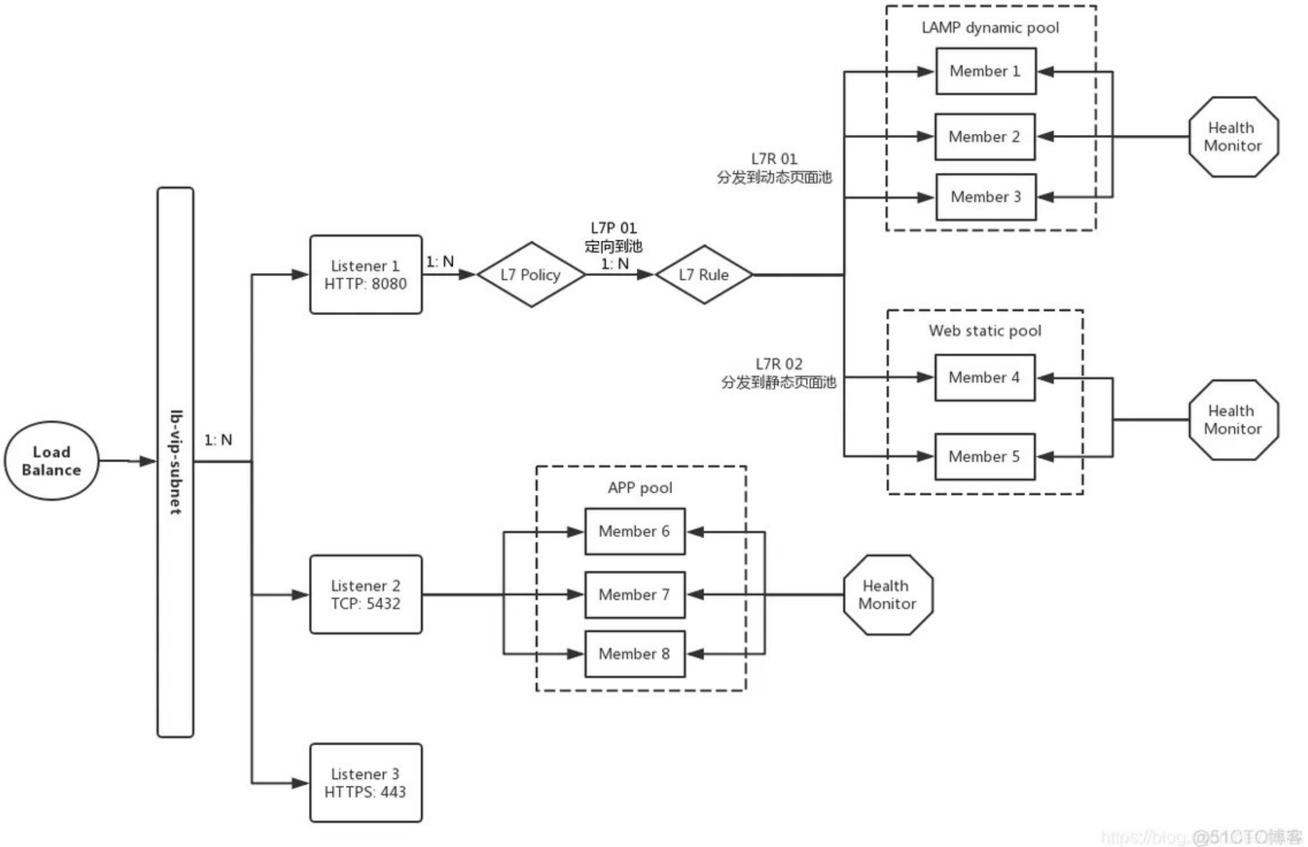
`Pike` 이후 `OpenStack`은 서비스로서의 로드 밸런싱을 위한 기본 솔루션으로 `neutron-lbaas Extension` 대신 `Octavia`를 사용할 것을 권장하고 `Queens`에서 `neutron-lbaas`를 더 이상 사용되지 않는 것으로 표시합니다. *Neutron-lbaas is now deprecated.*

커뮤니티에서 `Octavia`를 추천하는 데에는 `neutron-lbaas`가 남긴 역사적 문제를 해결하고 독립적이고 안정적인 API(Neutron/LBaaS/Deprecation)를 외부 세계에 제공할 수 있는 여러 가지 이유가 있습니다. 간단히 말해서, 커뮤니티는 `neutron-lbaas`가 `Neutron`의 프로젝트 관리를 장기화하고 `LBaaS`가 독립적인 프로젝트로 개발되어야 한다고 믿고 있으며 실제로 그렇습니다.

이 기사는 `Rocky`를 기반으로 `OpenStack LBaaS`로서의 `Octavia`의 추상 설계, 개발 설계 및 코드 구현을 기록하고 분석하며 Octavia에 대한 커뮤니티 개발자의 신뢰를 느낍니다.

기본 객체 개념

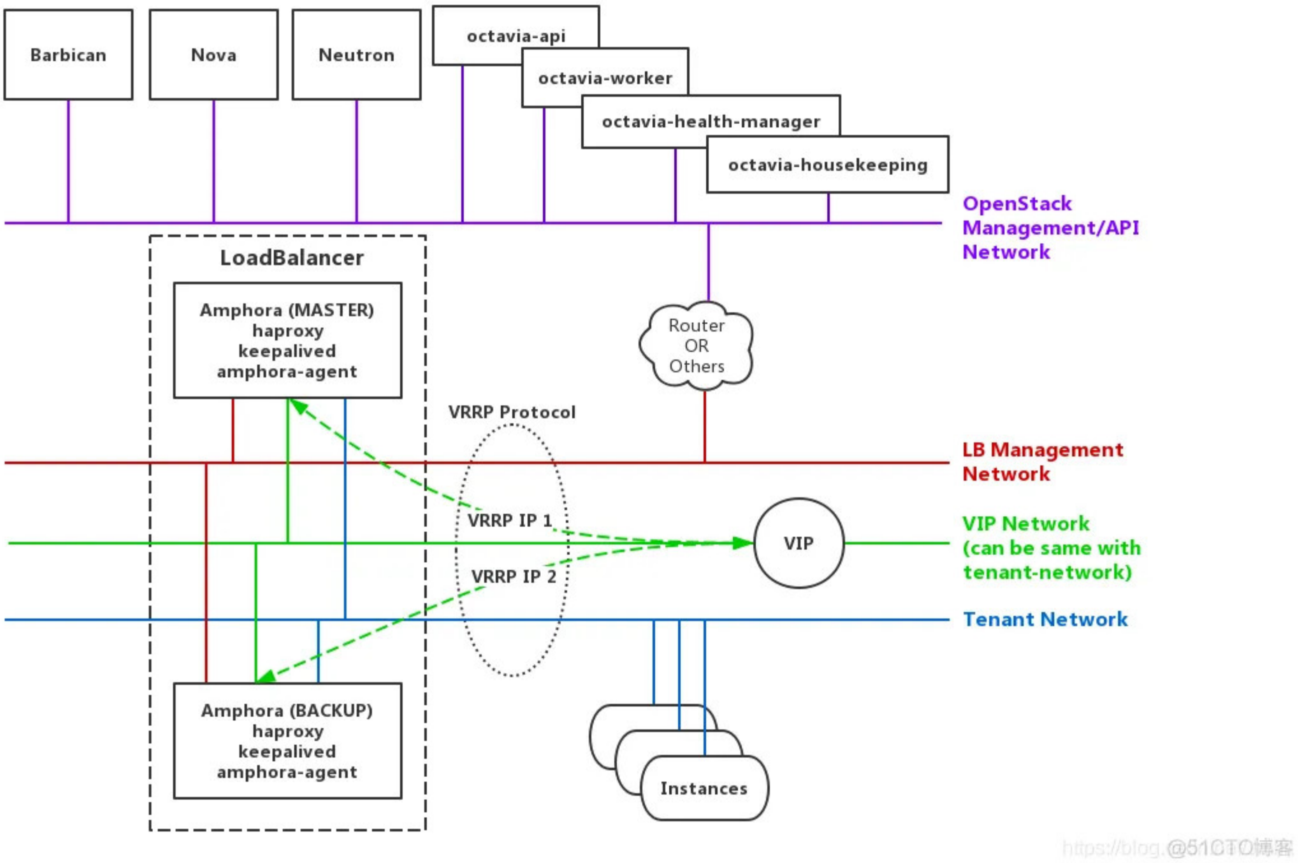
- `LBaaS` : `OpenStack` 플랫폼의 경우 LB(로드 밸런싱)가 사용자에게 서비스로 제공되며, 사용자는 필요에 따라 언제든지 구성 가능한 비즈니스 로드 밸런싱 솔루션을 얻을 수 있습니다. 이를 서비스형 로드 밸런싱이라고 합니다.
- `loadbalancer` : 로드 밸런싱 서비스의 루트 객체로, 이를 기반으로 로드 밸런싱에 대한 사용자 정의, 구성 및 동작이 이루어집니다.
- `VIP` : 로드 밸런서와 연결된 IP 주소입니다. 각 로드 밸런서에는 백엔드 비즈니스 클러스터에 대한 표준 외부 액세스 입구 역할을 하는 VIP가 하나 이상 있습니다.
- 리스너 : 로드밸런서의 하위 리스너로, VIP에 대한 외부 접속의 리스닝 유형(예: 프로토콜, 포트)을 구성할 수 있습니다.
- `Pool` : 백엔드의 실제 비즈니스 클라우드 호스트 클러스터 도메인. 일반적으로 사용자는 클라우드 호스트의 비즈니스 유형에 따라 구분합니다.
- `구성원(Member)` : `Pool`에 종속된 비즈니스 클라우드 호스트로서 기존 로드 밸런싱 시스템의 Real Server에 해당합니다.
- `Health Monitor` : `Pool`에 연결되어 `Pool` 내 멤버에 대한 Health Check를 주기적으로 수행합니다.
- `L7 정책` : 패킷 전달 작업을 설명하는 레이어 7 전달 정책(예: 풀로 전달, URL로 전달, 전달 거부)
- `L7 규칙` : `L7 정책`에 종속된 레이어 7 전달 규칙은 데이터 패킷 전달을 위한 일치 도메인을 설명합니다(예: 풀의 웹 서버로 시작하는 모든 구성원에게 전달됨).


LAMP dynamic pool
Member 1
Member 2
Member 3
Health Monitor
L7R 01
分发到动态页面池
L7P 01
定向到池
1: N
Listener 1
HTTP: 8080
1: N
L7 Policy
L7 Rule
Web static pool
L7R 02
分发到静态页面池
Member 4
Member 5
Health Monitor
Load Balance
lb-vip-subnet
1: N
APP pool
Member 6
Member 7
Member 8
Listener 2
TCP: 5432
Health Monitor
Listener 3
HTTPS: 443
https://blog.@51CTO博客

위 그림은 이러한 개념과 개인과 전체 간의 관계를 이해하는 데 도움이 되는 간단한 활성 및 정적 페이지 분리 로드 밸런싱 애플리케이션 아키텍처입니다.

이 시점에서 우리는 다음과 같은 질문을 하려고 합니다. 왜 이러한 객체를 추상화하는가?

기본 사용 과정

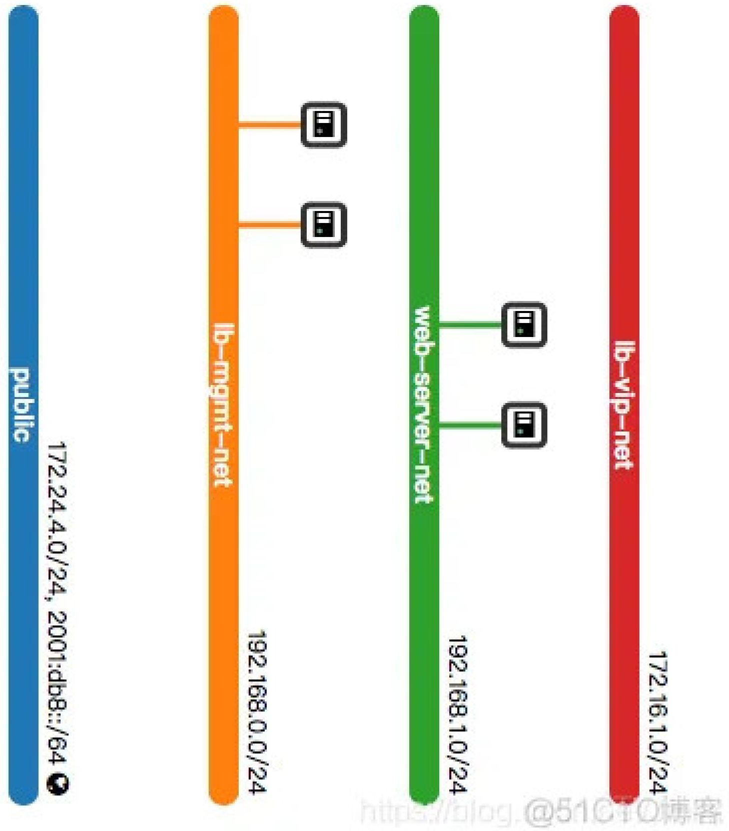

Barbican
Nova
Neutron
octavia-api
octavia-worker
octavia-health-manager
octavia-housekeeping
OpenStack Management/API Network
LoadBalancer
Amphora (MASTER)
haproxy
keepalived
amphora-agent
Router OR Others
VRRP Protocol
LB Management Network
VRRP IP 1
VRRP IP 2
VIP
VIP Network (can be same with tenant-network)
Tenant Network
Amphora (BACKUP)
haproxy
keepalived
amphora-agent
Instances
https://blog.@51CTO博客

사용의 관점에서 `Octavia`가 어떤 모습인지 계속해서 이해하세요.

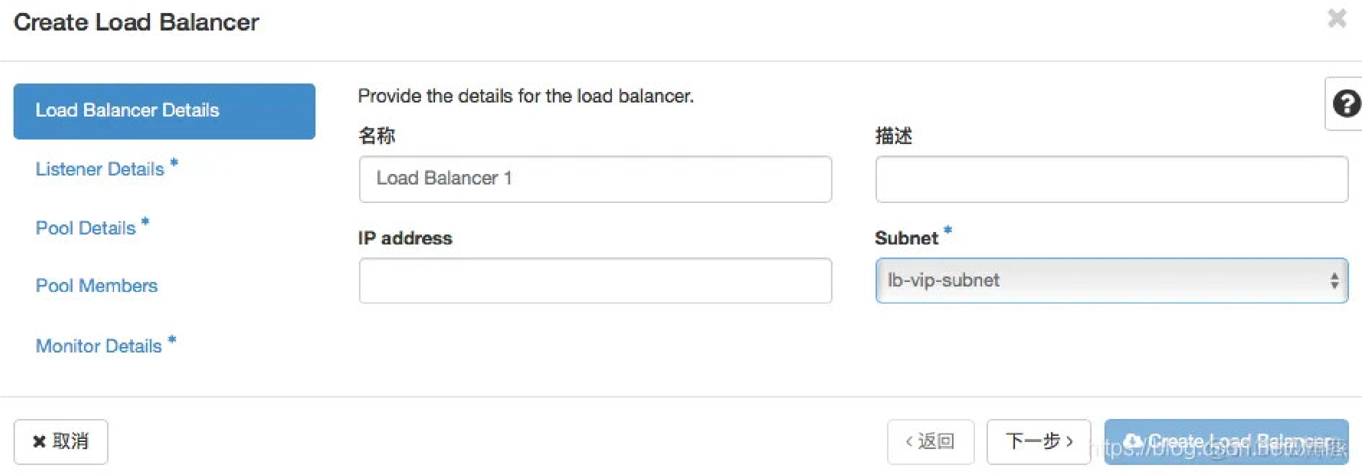
위 그림은 다음을 포함하는 **표준 Octavia 네트워크 아키텍처**입니다.

- `Amphora(e)` : 엔터티는 로드 밸런서의 캐리어 역할을 하며 Octavia의 기본 로드 밸런서 공급자이기도 한 클라우드 호스트입니다.
- `lb-mgmt-net` : OpenStack Management/API Network에 연결된 네트워크로 프로젝트 관리자에게 공개되며 동쪽은 Amphora 인스턴스에 연결되고 서쪽은 Octavia 서비스 프로세스에 연결됩니다.
- `tenant-net` : 비즈니스 클라우드 호스트가 위치한 네트워크
- `vip-net` : VIP 주소 풀을 제공하는 네트워크

참고: vip-net과 테넌트-넷은 동일한 네트워크일 수 있지만 프로덕션 환경에서는 보다 표적화된 방식으로 보안 정책을 적용하고 네트워크 보안 격리 도메인을 서로 다른 수준에서 나누기 위해 분리하는 것이 좋습니다.


public
172.24.4.0/24, 2001:db8::/64
lb-mgmt-net
192.168.0.0/24
web-server-net
192.168.1.0/24
lb-vip-net
172.16.1.0/24
https://blog.@51CTO博客

1단계. 로드밸런서의 VIP를 설정합니다. VIP는 직접 지정하거나 DHCP를 통해 할당할 수 있습니다.


Create Load Balancer

Provide the details for the load balancer.

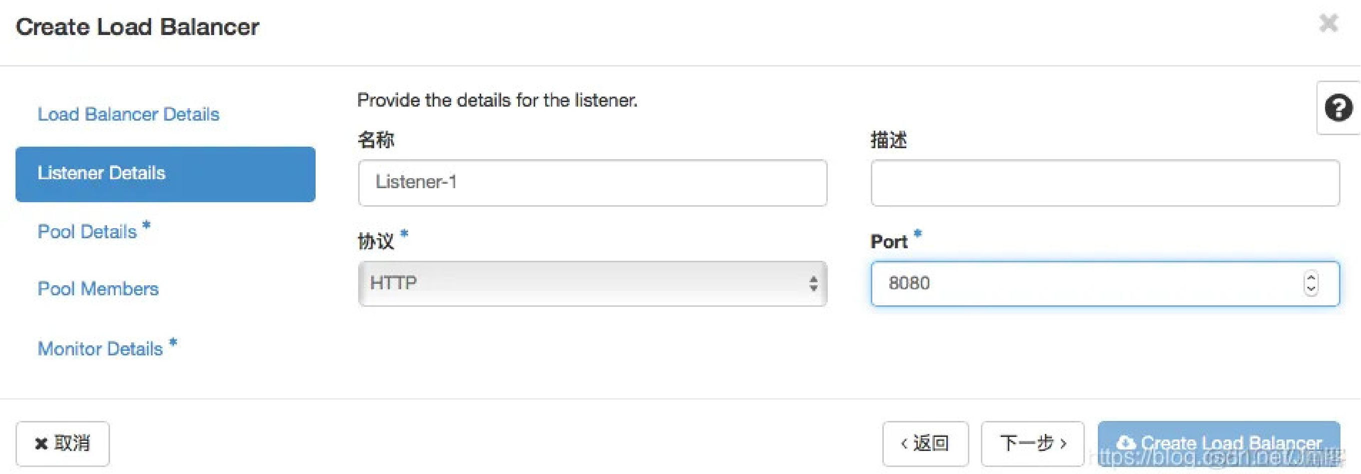
名称: Load Balancer 1

描述:

IP address:

Subnet *: lb-vip-subnet
Load Balancer Details
Listener Details *
Pool Details *
Pool Members
Monitor Details *
取消
‹ 返回
下一步 ›
Create Load Balancer

2단계. 리스너가 수신할 프로토콜과 포트를 설정합니다. 외부 액세스를 모니터링합니다 `http://<VIP>:8080/`.



Create Load Balancer

Provide the details for the listener.

名称: Listener-1

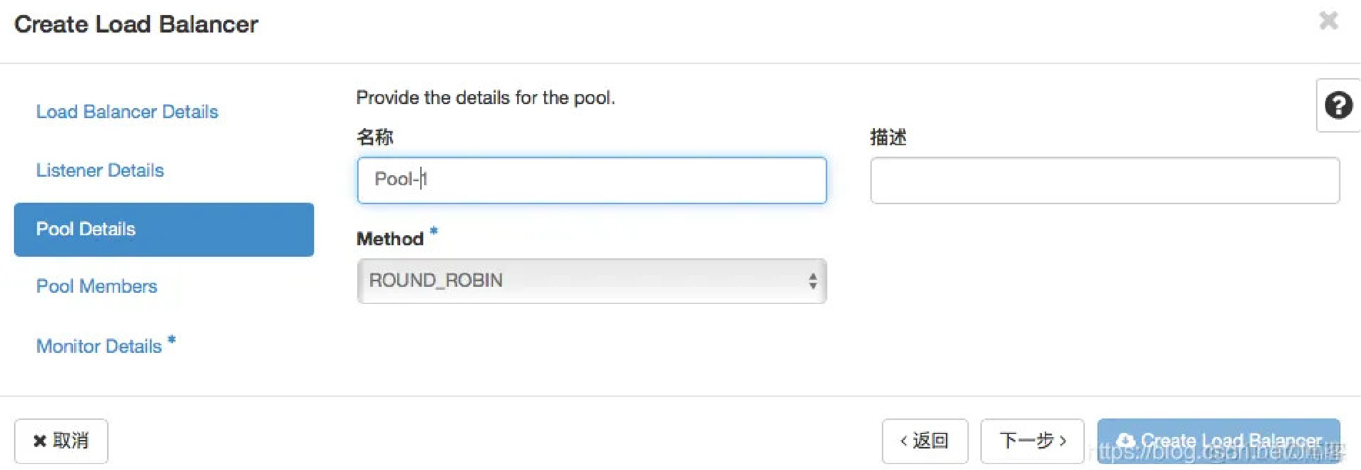
描述:

协议 *: HTTP

Port *: 8080

Load Balancer Details
Listener Details
Pool Details *
Pool Members
Monitor Details *
取消
返回
下一步
Create Load Balancer
https://blog.csdn.net/

3단계. 풀의 로드 밸런싱 알고리즘을 설정합니다. 여기서는 RR 폴링 분포 알고리즘이 선택됩니다.



Create Load Balancer

Provide the details for the pool.

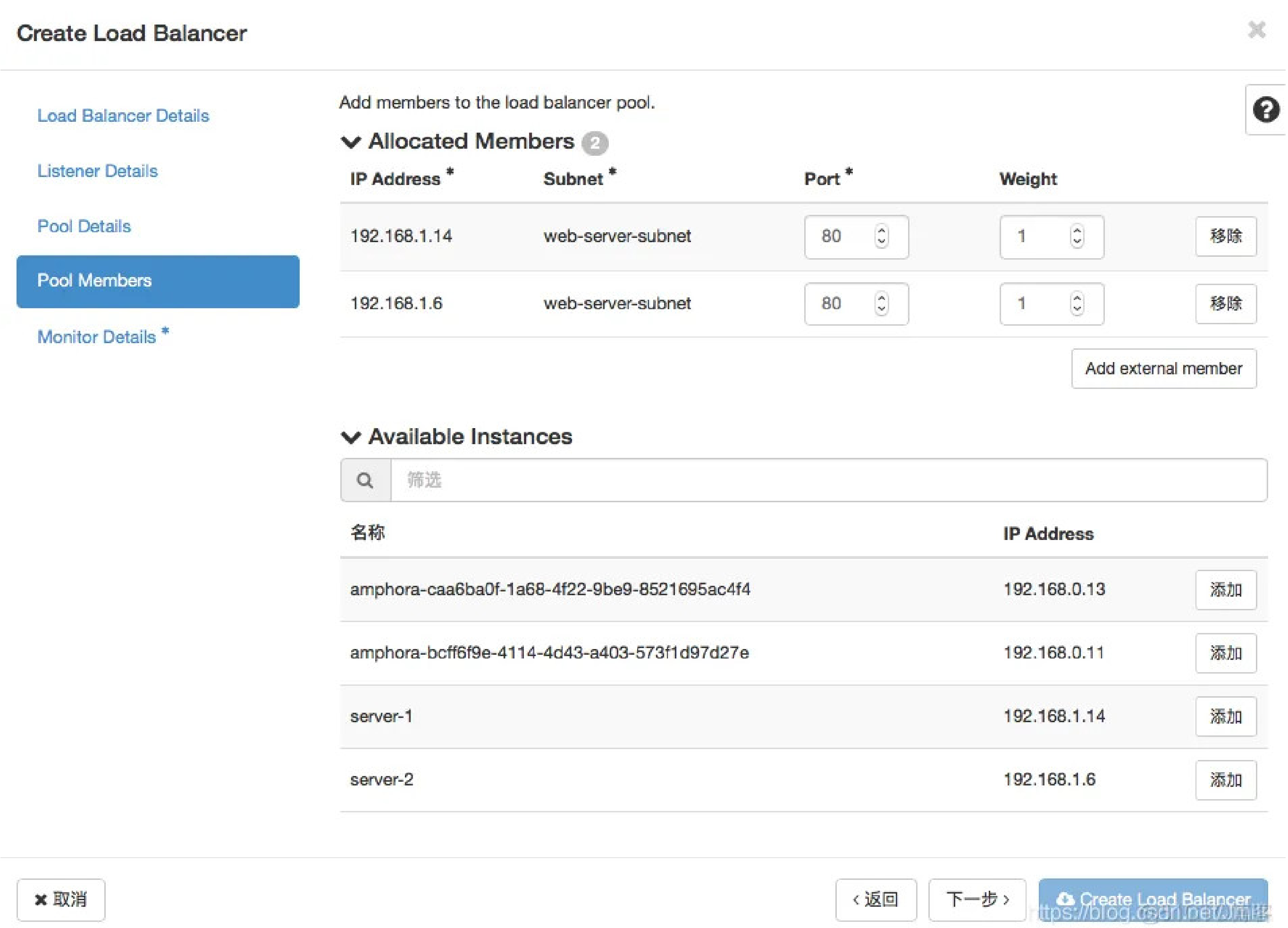
名称: Pool-1

描述:

Method *: ROUND_ROBIN

Load Balancer Details
Listener Details
Pool Details
Pool Members
Monitor Details *
取消
返回
下一步
Create Load Balancer
https://blog.csdn.net/

4단계. 풀의 구성원을 설정합니다. 멤버를 설정하려면 포트와 가중치를 지정해야 합니다. 전자는 데이터 전달을 허용하는 소켓을 나타내고 후자는 배포 우선 순위를 나타냅니다.



Create Load Balancer

Load Balancer Details
Listener Details
Pool Details
Pool Members
Monitor Details *

Add members to the load balancer pool.

Allocated Members 2

IP Address *	Subnet *	Port *	Weight	
192.168.1.14	web-server-subnet	80	1	移除
192.168.1.6	web-server-subnet	80	1	移除

Add external member

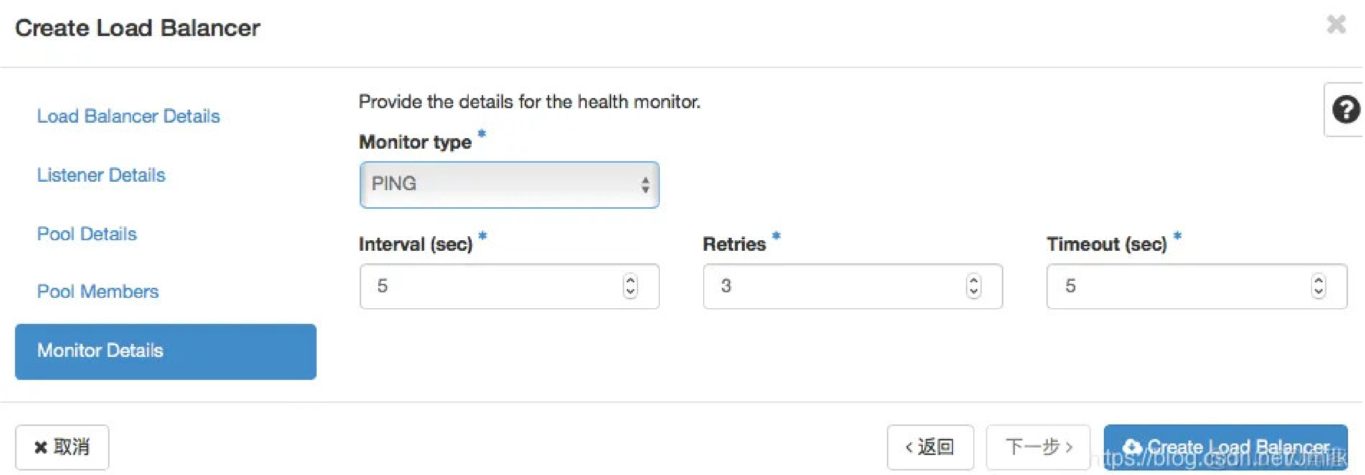
Available Instances

筛选

名称	IP Address	
amphora-caa6ba0f-1a68-4f22-9be9-8521695ac4f4	192.168.0.13	添加
amphora-bcff6f9e-4114-4d43-a403-573f1d97d27e	192.168.0.11	添加
server-1	192.168.1.14	添加
server-2	192.168.1.6	添加

✖ 取消
‹ 返回
下一步 ›
Create Load Balancer

5단계. 상태 모니터의 상태 확인 규칙을 설정합니다. 구성원의 PING이 다른 경우 결함이 있는 것으로 표시되고 더 이상 배포가 허용되지 않습니다.



Create Load Balancer

Load Balancer Details
Listener Details
Pool Details
Pool Members
Monitor Details

Provide the details for the health monitor.

Monitor type *: PING

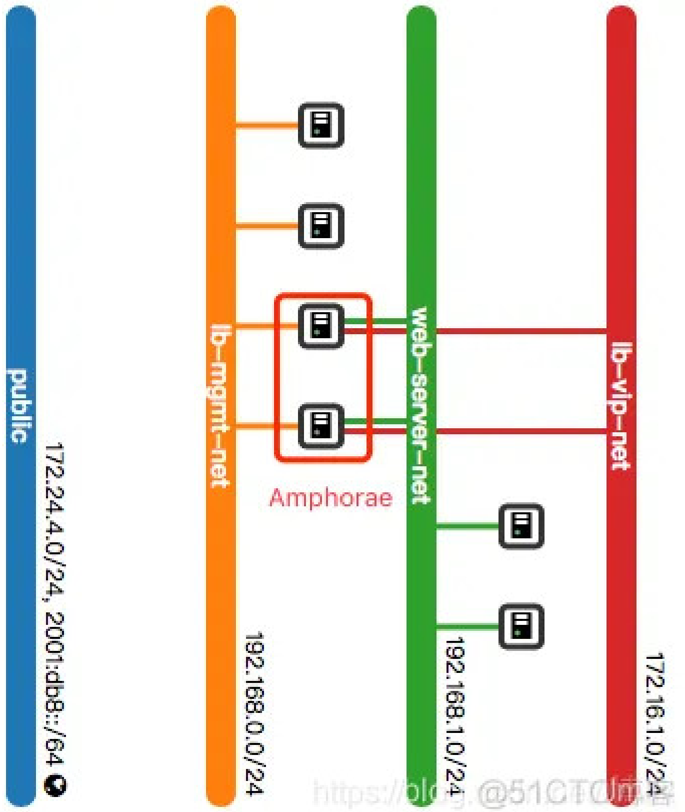
Interval (sec) *: 5

Retries *: 3

Timeout (sec) *: 5

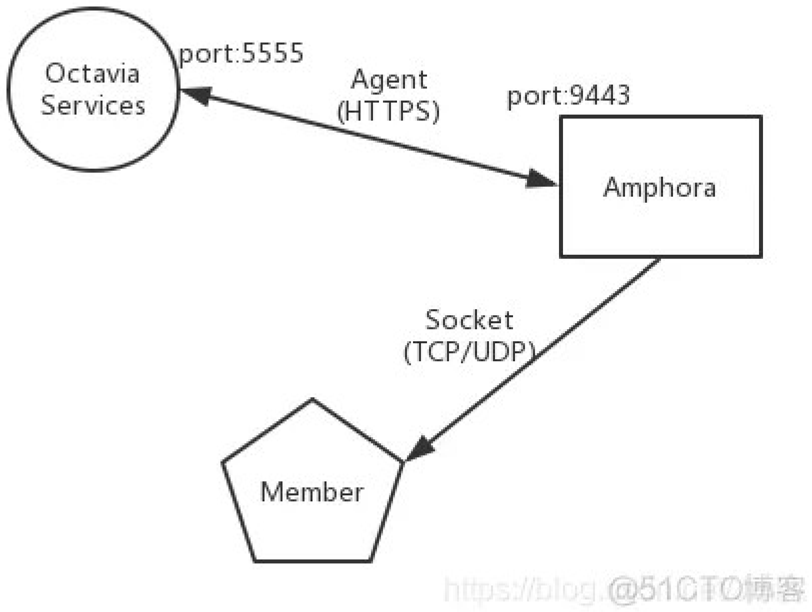
✖ 取消
‹ 返回
下一步 ›
Create Load Balancer

현재 네트워크 토폴로지 변경 사항은 다음과 같습니다. Amphorae가 포트 마운팅을 사용하여 세 가지 다른 네트워크에서 VIP, Member 및 Octava 서비스 프로세스를 연결하는 것을 볼 수 있습니다. 이름.


public
172.24.4.0/24, 2001:db8::/64
lb-mgmt-net
192.168.0.0/24
Amphorae
web-server-net
192.168.1.0/24
lb-vip-net
172.16.1.0/24
https://blog.@51CTO博客

이제 Octavia Amphora Provider의 디자인 아이디어를 간략하게 검토해 보겠습니다.

- Amphora는 로드 밸런서 소프트웨어(HAProxy) 및 고가용성 지원(Keepalived)을 실행하는 캐리어 역할을 하며 에이전트를 통해 Octavia 서비스 프로세스와 통신합니다.
- Octavia 서비스 프로세스는 사용자의 로드 밸런서 및 VIP 구성 매개변수를 수신하고 에이전트를 통해 haproxy 및 keepalived 구성 파일을 동적으로 수정합니다.
- Member가 속한 Subnet을 Amphora에 연결하면 Amphora는 Member Socket(IP, Port)을 통해 요청 패킷을 배포합니다.


Octavia Services
port:5555
Agent (HTTPS)
port:9443
Amphora
Socket (TCP/UDP)
Member
https://blog.@51CTO博客

여기에는 Octavia와 관련된 이미지 및 보안 그룹의 내용을 추가하겠습니다. Amphora 인스턴스는 특정

이미지를 사용하여 시작됩니다. Octavia는 centos 및 ubuntu 운영 체제를 지원하는 특수 이미지 생성 스크립트를 제공합니다. 그러나 프로덕션 환경에서 로그인하려면 키 쌍을 사용하는 것이 좋습니다. 보안 그룹의 경우, Amphora의 보안 그룹은 적어도 두 가지 ingress 규칙(UDP/5555 및 egress:TCP/9443)을 충족해야 함을 위 그림에서 볼 수 있습니다.

앰포라 이미지를 사용하는 단계 :

1단계. 앰포라 이미지 업로드

```
$ /opt/rocky/octavia/diskimage-create/diskimage-create.sh -i ubuntu

$ openstack image create amphora-x64-haproxy \
  --public \
  --container-format=bare \
  --disk-format qcow2 \
  --file /opt/rocky/octavia/diskimage-create/amphora-x64-haproxy.qcow2 \
  --tag amphora
```

2단계. 앰포라 이미지 구성 앰포라 이미지를 업로드한 후 용도를 지정 하도록 구성해야 합니다. `[controller_worker] amp_image_owner_id, amp_image_tag`

```
[controller_worker]
amp_image_owner_id = 9e4fe13a6d7645269dc69579c027fde4
amp_image_tag = amphora
...
```

amphora 보안 그룹을 사용하는 단계 :

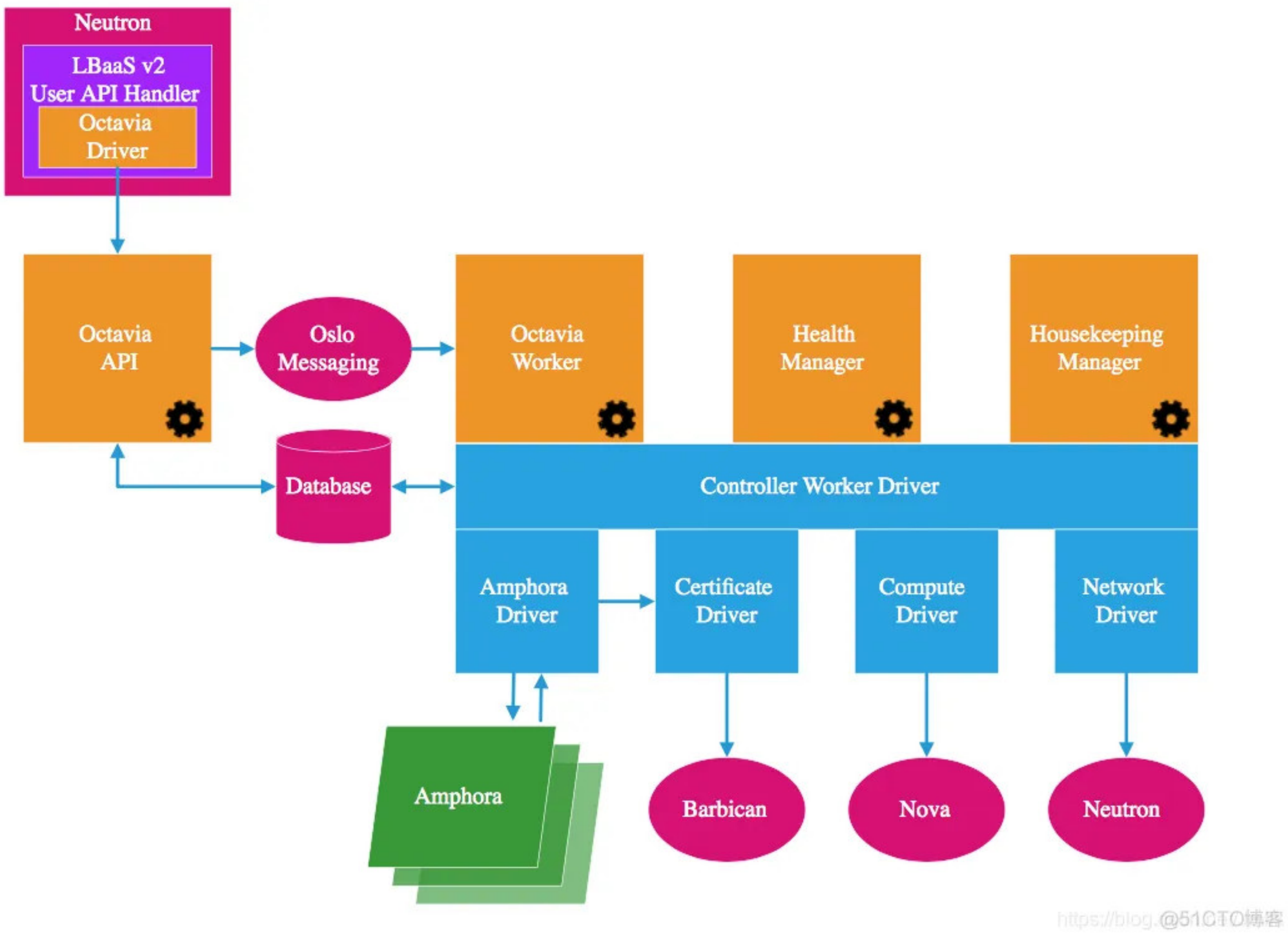
1단계. amphora에서 사용하는 보안 그룹 생성

```
$ openstack security group create amphora-sec-grp --project <admin project id>
$ openstack security group rule create --remote-ip "0.0.0.0/0" --dst-port 9443 --protocol tcp --ingress --ethertype IPv4 --project <admin project id> amphora-sec-grp
$ openstack security group rule create --remote-ip "0.0.0.0/0" --dst-port 5555 --protocol udp --egress --ethertype IPv4 --project <admin project id> amphora-sec-grp
```

2단계. amphora 보안 그룹 구성

```
[controller_worker]
amp_secgroup_list = <amphora-sec-grp id>
...
```

소프트웨어 아키텍처


Neutron
LBaaS v2
User API Handler
Octavia
Driver
Octavia
API
Oslo
Messaging
Octavia
Worker
Health
Manager
Housekeeping
Manager
Database
Controller Worker Driver
Amphora
Driver
Certificate
Driver
Compute
Driver
Network
Driver
Amphora
Barbican
Nova
Neutron
https://blog.@51CTO博客

(참고: 사진은 Octavia 공식 문서에서 가져온 것입니다 .)

Octavia의 소프트웨어 아키텍처 디자인은 여전히 일반적인 " 생산자-소비자 " 모델입니다. API는 작업자와 분리되어 있으며 MessageQueens를 통해 통신합니다.

- `Octavia API` : 표준 RESTful API, Octavia v2 API(기본적으로 활성화됨)는 LBaaS v2 API의 상위 집합이며 이전 버전과 완전히 호환됩니다. 따라서 뒤떨어진 버전의 OS 플랫폼도 Neutron Octavia Driver를 통해 통합할 수 있습니다.
- `Octavia Controller Worker` : Octavia의 핵심입니다. 하단 계층은 Driver & Plugin을 사용하여 OS 플랫폼의 개방성을 나타내며 상위 계층에서 구현되는 세 가지 구성 요소를 지원합니다.
  - `Octavia Worker` : API 요청 완료를 담당하며 Octavia의 주요 기능을 실행합니다.
  - `Health Manager` : 로드밸런서의 고가용성을 보장하는 역할을 담당합니다.
  - `Housekeeping Manager` : 옥타비아의 건전한 운영을 보장하는 진정한 하우스키핑 서비스입니다. SpaceAmphora, DatabaseCleanup 및 CertRotation을 구현했습니다.

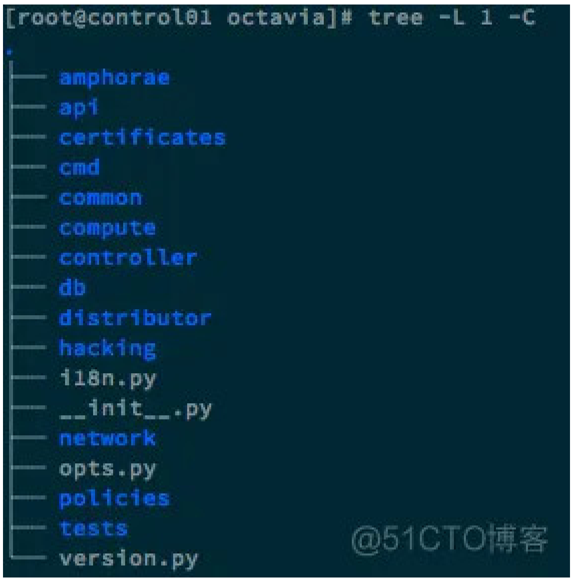
참고: 아키텍처 다이어그램에는 하나의 LB 공급자인 Amphora만 표시되어 있지만 Octavia의 드라이버 설계는 실제로 여러 LB 공급자(예: F5)를 지원합니다. 실제로 커뮤니티에서는 항상 openstack/neutron-lbaas 저장소에 구현된 드라이버를 Octavia로 마이그레이션할 계획을 세웠지만 이를 수행할 사람이 부족했습니다.

서비스 프로세스 목록

서비스 목록은 소프트웨어 아키텍처를 구체적으로 표현한 것입니다.

- Octavia API
- Octavia Worker
- Octavia Health Manager
- Octavia Housekeeping

코드 구조



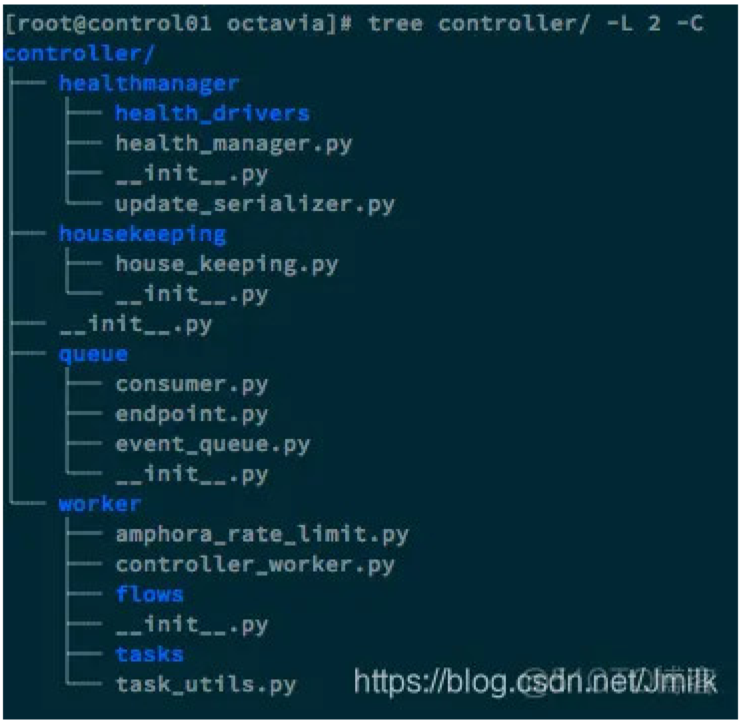
```
[root@control01 octavia]# tree -L 1 -C
.
├── amphorae
├── api
├── certificates
├── cmd
├── common
├── compute
├── controller
├── db
├── distributor
├── hacking
├── i18n.py
├── __init__.py
├── network
├── opts.py
├── policies
├── tests
└── version.py
```

@51CTO博客

다음은 몇 가지 주요 디렉터리입니다.

- `amphora` : amphora Rest API 및 amphora-agent 구현
- `api` : Octavia API 구현
- `certificates` : CA 인증 구현, amphora와 Octavia Worker 간 HTTPS 통신 및 TLS 기능 지원
- `Compute` : Compute Driver의 추상화와 novaclient의 캡슐화를 구현합니다.
- `network` : 네트워크 드라이버의 추상화와 neutronclient의 캡슐화를 구현합니다.
- `db` : ORM 구현
- `policies` : API 요청에 대한 인증 정책을 정의합니다.

계속해서 컨트롤러 디렉터리를 확장합니다.



```
[root@control01 octavia]# tree controller/ -L 2 -C
controller/
├── healthmanager
│   ├── health_drivers
│   ├── health_manager.py
│   ├── __init__.py
│   └── update_serializer.py
├── housekeeping
│   ├── house_keeping.py
│   └── __init__.py
├── __init__.py
├── queue
│   ├── consumer.py
│   ├── endpoint.py
│   ├── event_queue.py
│   └── __init__.py
└── worker
    ├── amphora_rate_limit.py
    ├── controller_worker.py
    ├── flows
    ├── __init__.py
    ├── tasks
    └── task_utils.py
```

https://blog.csdn.net/Jmilk

- `healthmanager` : Health Manager 구현
- `housekeeping` : HouseKeeping 구현
- `queue` : cotyledon 프레임워크 및 oslo_messaging을 사용하여 내부 RPC 통신 구현
  - `producer`:api/handlers/queue/producer.py
  - `consumer`:controller/queue/consumer.py
- `worker` : 작업 흐름 프레임워크를 사용하여 Octavia Worker 구현
  - `flow` : 작업 흐름을 캡슐화하여 각 작업을 흐름으로 정의합니다.
  - `task` : 태스크를 캡슐화하고 태스크 로직을 추상화하여 태스크의 재사용성을 높입니다.

추신: cotyledon은 oslo.service를 대체하기 위해 커뮤니티에서 개발한 타사 오픈 소스 라이브러리입니다.

> Cotyledon은 장기 실행 서비스 정의를 위한 프레임워크를 제공하며 Unix 신호 처리, 작업자 생성, 하위 프로세스 감독, 데몬 다시 로드, sd-notify, 작업자 생성 속도 제한 등을 제공합니다.
>
> 이 라이브러리는 주로 OpenStack Telemetry 프로젝트에서 oslo.service를 대체하여 사용됩니다. 그러나 oslo.service는 eventlet에 의존하므로 애플리케이션이 Python 표준 라이브러리를 Monkeypatch하지 않는 경우에는 다른 라이브러리가 필요합니다. 더 이상 Greenlet은 시기적절하지 않습니다. 이로 인해 Tooz 또는 oslo.messaging과 같은 다른 라이브러리가 하트비트 시스템과 함께 실패하게 되었습니다. 또한 Greenpipe가 처리되지 않기 때문에 프로세스가 예상대로 존재하지 않습니다. ——Cotyledon 공식 문서에서 발췌.

OpenStack의 독립 프로젝트인 Octavia의 아키텍처 설계를 요약하면 일관되게 우수한 개방형 설계 아이디어를 계승하고 있으며 Driver 클래스는 LB Provider, Certificates Driver, Compute Driver 및 Network Driver와 같은 외부 지원 노드에서 고도로 추상화되어 Vendor를 만듭니다. 사용자는 기존 인프라에 더 쉽게 연결할 수 있습니다. 이는 의심할 여지 없이 Octavia와 OpenStack이 인기를 끄는 이유 중 하나입니다. 또한 위에서 제기한 질문에 대한 한 가지 측면에 대한 답변이기도 합니다. 왜 이러한 객체를 추상화

해야 할까요?

LoadBalaner 프로세스 분석

가장 일반적인 Octavia 구현 사양은 로드 밸런서 생성 프로세스입니다. 우리는 이것을 시작점으로 사용하고 UML 다이어그램의 도움으로 Octavia의 코드 구현을 계속해서 탐구할 것입니다.

CLI:

```
$ openstack loadbalancer create --vip-subnet-id lb-vip-subnet --name lb1
```

API:

```
POST /v2.0/lbaas/loadbalancers
```

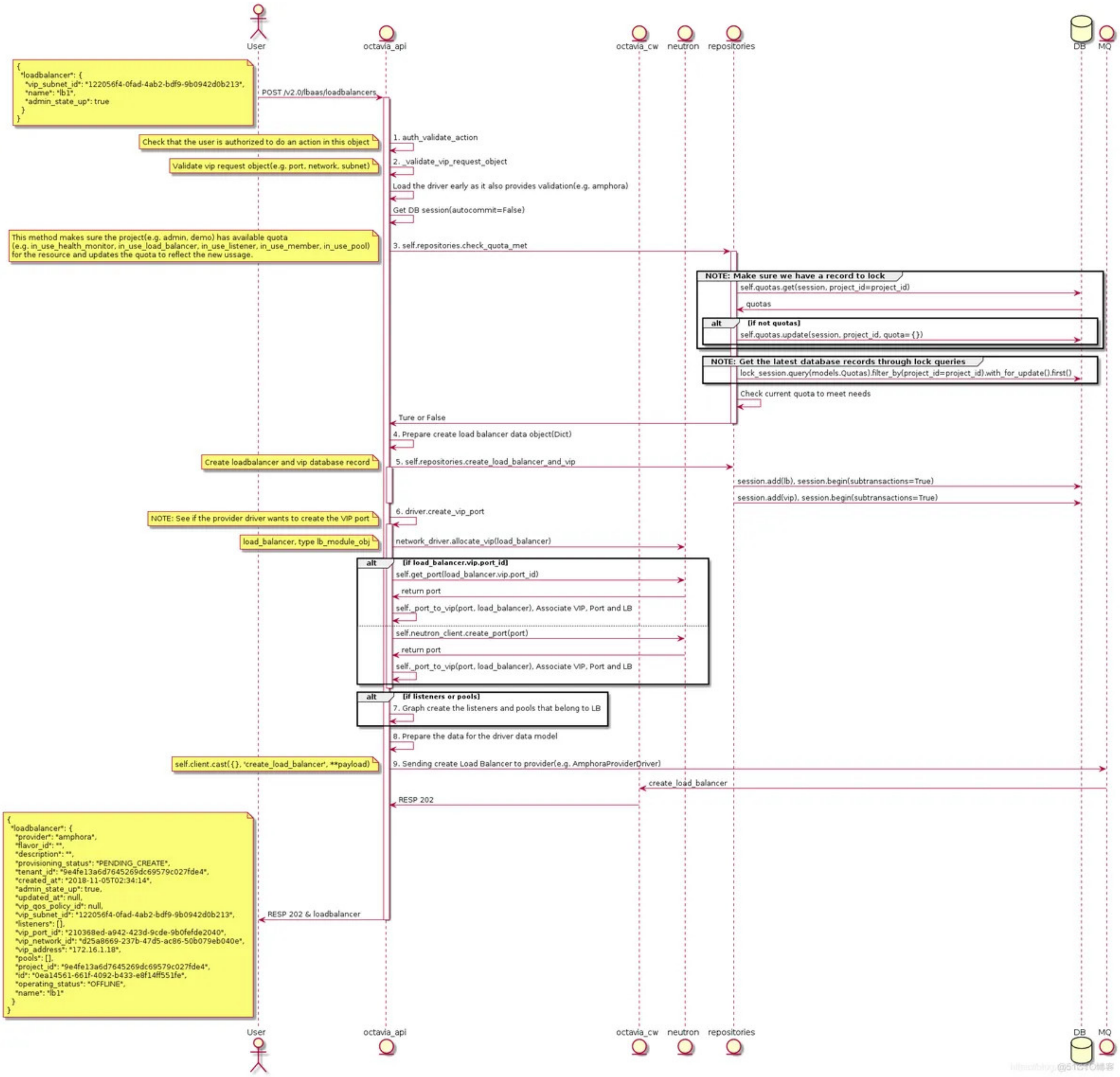
요청 본문:

```
{
  "loadbalancer": {
    "vip_subnet_id": "c55e7725-894c-400e-bd00-57a04ae1e676",
    "name": "lb1",
    "admin_state_up": true
  }
}
```

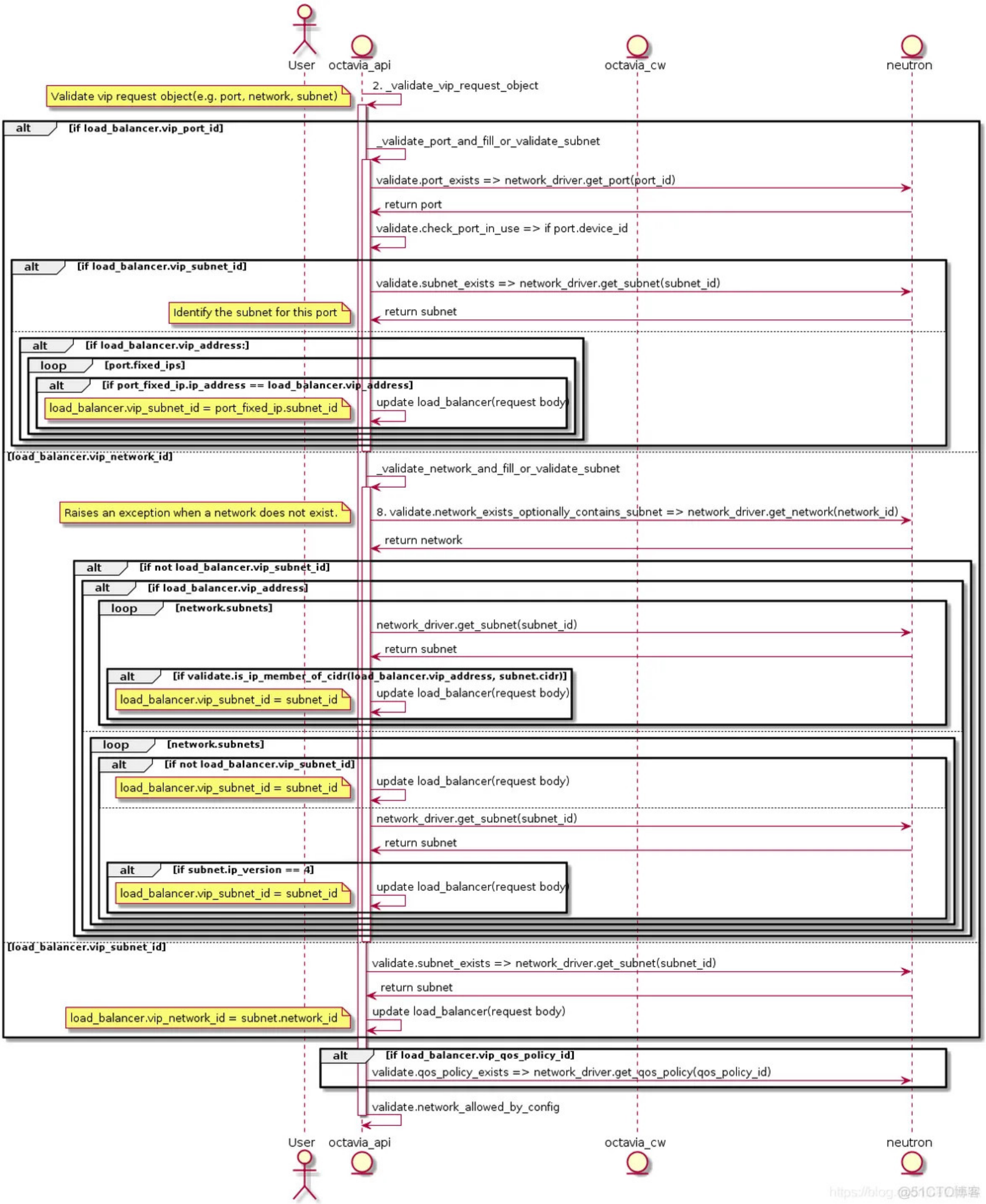
응답:

```
{
  "loadbalancer": {
    "provider": "octavia",
    "flavor_id": "",
    "description": "",
    "provisioning_status": "PENDING_CREATE",
    "created_at": "2018-10-22T02:52:04",
    "admin_state_up": true,
    "updated_at": null,
    "vip_subnet_id": "c55e7725-894c-400e-bd00-57a04ae1e676",
    "listeners": [],
    "vip_port_id": "6629fef4-fe14-4b41-9b73-8230105b2e36",
    "vip_network_id": "1078e169-61cb-49bc-a513-915305995be1",
    "vip_address": "10.0.1.7",
    "pools": [],
    "project_id": "2e560efadb704e639ee4bb3953d94afa",
    "id": "5bcf8e3d-9e58-4545-bf80-4c0b905a49ad",
    "operating_status": "OFFLINE",
    "name": "lb1"
  }
}
```

Create LB의 Octavia API UML 다이어그램 :


User
octavia_api
octavia_cw
neutron
repositories
DB
MQ
{
"loadbalancer": {
"vip_subnet_id": "122056f4-0fad-4ab2-bdf9-9b0942d0b213",
"name": "lb1",
"admin_state_up": true
}
}
POST /v2.0/lbaas/loadbalancers
Check that the user is authorized to do an action in this object
1. auth_validate_action
Validate vip request object(e.g. port, network, subnet)
2. _validate_vip_request_object
Load the driver early as it also provides validation(e.g. amphora)
Get DB session(autocommit=False)
This method makes sure the project(e.g. admin, demo) has available quota
(e.g. in_use_health_monitor, in_use_load_balancer, in_use_listener, in_use_member, in_use_pool)
for the resource and updates the quota to reflect the new usage.
3. self.repositories.check_quota_met
NOTE: Make sure we have a record to lock
self.quotas.get(session, project_id=project_id)
quotas
alt
[if not quotas]
self.quotas.update(session, project_id, quota={})
NOTE: Get the latest database records through lock queries
lock_session.query(models.Quotas).filter_by(project_id=project_id).with_for_update().first()
Check current quota to meet needs
Ture or False
4. Prepare create load balancer data object(Dict)
Create loadbalancer and vip database record
5. self.repositories.create_load_balancer_and_vip
session.add(lb), session.begin(subtransactions=True)
session.add(vip), session.begin(subtransactions=True)
6. driver.create_vip_port
NOTE: See if the provider driver wants to create the VIP port
load_balancer, type lb_module_obj
network_driver.allocate_vip(load_balancer)
alt
[if load_balancer.vip.port_id]
self.get_port(load_balancer.vip.port_id)
return port
self._port_to_vip(port, load_balancer), Associate VIP, Port and LB
self.neutron_client.create_port(port)
return port
self._port_to_vip(port, load_balancer), Associate VIP, Port and LB
alt
[if listeners or pools]
7. Graph create the listeners and pools that belong to LB
8. Prepare the data for the driver data model
self.client.cast({}, 'create_load_balancer', **payload)
9. Sending create Load Balancer to provider(e.g. AmphoraProviderDriver)
create_load_balancer
RESP 202
{
"loadbalancer": {
"provider": "amphora",
"flavor_id": "",
"description": "",
"provisioning_status": "PENDING_CREATE",
"tenant_id": "9e4fe13a6d7645269dc69579c027fde4",
"created_at": "2018-11-05T02:34:14",
"admin_state_up": true,
"updated_at": null,
"vip_qos_policy_id": null,
"vip_subnet_id": "122056f4-0fad-4ab2-bdf9-9b0942d0b213",
"listeners": [],
"vip_port_id": "210368ed-a942-423d-9cde-9b0fefde2040",
"vip_network_id": "d25a8669-237b-47d5-ac86-50b079eb040e",
"vip_address": "172.16.1.18",
"pools": [],
"project_id": "9e4fe13a6d7645269dc69579c027fde4",
"id": "0ea14561-661f-4092-b433-e8f14ff551fe",
"operating_status": "OFFLINE",
"name": "lb1"
}
}
RESP 202 & loadbalancer
User
octavia_api
octavia_cw
neutron
repositories
DB
MQ

2. _validate_vip_request_object의 UML 다이어그램을 확장합니다 .


User
octavia_api
octavia_cw
neutron
2. _validate_vip_request_object
Validate vip request object(e.g. port, network, subnet)
alt [if load_balancer.vip_port_id]
_validate_port_and_fill_or_validate_subnet
validate.port_exists => network_driver.get_port(port_id)
return port
validate.check_port_in_use => if port.device_id
alt [if load_balancer.vip_subnet_id]
validate.subnet_exists => network_driver.get_subnet(subnet_id)
Identify the subnet for this port
return subnet
alt [if load_balancer.vip_address:]
loop [port.fixed_ips]
alt [if port_fixed_ip.ip_address == load_balancer.vip_address]
load_balancer.vip_subnet_id = port_fixed_ip.subnet_id
update load_balancer(request body)
[load_balancer.vip_network_id]
_validate_network_and_fill_or_validate_subnet
Raises an exception when a network does not exist.
8. validate.network_exists_optionally_contains_subnet => network_driver.get_network(network_id)
return network
alt [if not load_balancer.vip_subnet_id]
alt [if load_balancer.vip_address]
loop [network.subnets]
network_driver.get_subnet(subnet_id)
return subnet
alt [if validate.is_ip_member_of_cidr(load_balancer.vip_address, subnet.cidr)]
load_balancer.vip_subnet_id = subnet_id
update load_balancer(request body)
loop [network.subnets]
alt [if not load_balancer.vip_subnet_id]
load_balancer.vip_subnet_id = subnet_id
update load_balancer(request body)
network_driver.get_subnet(subnet_id)
return subnet
alt [if subnet.ip_version == 4]
load_balancer.vip_subnet_id = subnet_id
update load_balancer(request body)
[load_balancer.vip_subnet_id]
validate.subnet_exists => network_driver.get_subnet(subnet_id)
return subnet
update load_balancer(request body)
load_balancer.vip_network_id = subnet.network_id
alt [if load_balancer.vip_qos_policy_id]
validate.qos_policy_exists => network_driver.get_qos_policy(qos_policy_id)
validate.network_allowed_by_config
User
octavia_api
octavia_cw
neutron
https://blog.@51CTO博客

요청을 받은 `POST /v2.0/lbaas/loadbalancers`후 `octavia-api` 서비스가 처리하는 작업 요약:

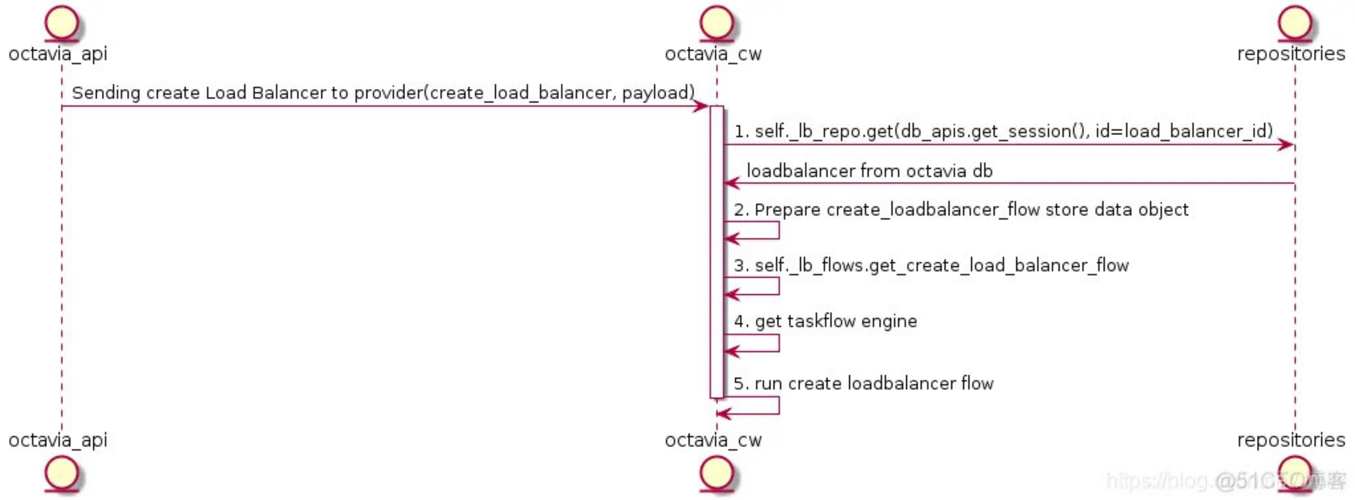
1. 사용자에게 로드밸런서를 생성할 수 있는 권한이 있는지 확인하기 위해 인증을 요청합니다.
2. VIP 및 관련 개체(예: 포트, 서브넷, 네트워크)를 사용할 수 있는지 확인합니다. 여기서 VIP를 생성할 때 특정 네트워크 개체 유형을 허용/허용하지 않도록 `config secition [networking]` 구성 할 수 있습니다.

3. 사용자 프로젝트의 LB 할당량을 확인합니다. `config section [quotas]`이를 통해 기본 할당량을 구성할 수 있습니다(예: Project1을 지정하면 로드 밸런서는 3개만 생성할 수 있습니다).
4. 테이블 `load_balancer` 및 `vip`에 대한 데이터베이스 레코드를 생성합니다.
5. `Amphora` 드라이버(기본 `lb` 공급자)를 호출하여 VIP에 해당하는 포트를 생성하고 Port, VIP 및 LB의 데이터베이스 레코드를 연결합니다.
6. 그래프 플로우 형태로 로드밸런서에 종속된 리스너와 풀을 생성합니다.
7. `create_loadbalancer_flow`에 전달된 저장소를 준비합니다.
8. `octavia-worker` 서비스를 비동기식으로 호출하여 `create_loadbalancer_flow`를 실행합니다.

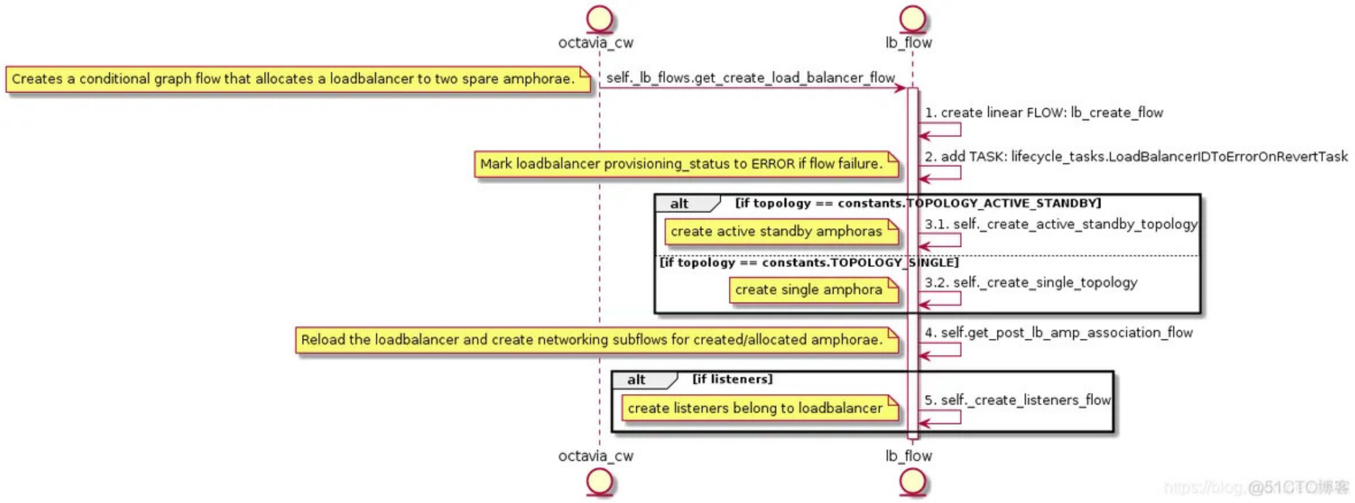
주목할 만한 몇 가지 사항이 있습니다.

- 로드 밸런서 할당량은 여전히 명령을 통해 설정됩니다`openstack quota set`.
- `openstack loadbalancer create` 지시어는 로드밸런서와 동시에 하위 리스너와 풀을 생성하는 `--listeners` 또는 `--pools`와 같은 옵션을 제공하지 않지만, POST `/v2.0/lbaas/loadbalancers`는 두 속성을 모두 받을 수 있습니다.따라서 대시보드의 UI/UX를 이에 맞게 최적화할 수 있습니다.
- 지정된 VIP 포트가 존재하지 않는 경우 `octavia-api` 서비스는 먼저 `neutronclient`를 호출하여 포트를 생성하고 이름을 `loadbalancer-<load_balancer_id>` 로 지정 하므로 `vip-net`에서 이러한 유형의 포트를 볼 수 있습니다.
- VIP는 네트워크, 서브넷, 포트 등 모든 방법을 통한 생성을 지원하며 `VIP QoS` 설정도 지원합니다.

Create LB의 Octavia Controller Worker UML 다이어그램 :


octavia_api
octavia_cw
repositories
Sending create Load Balancer to provider(create_load_balancer, payload)
1. self._lb_repo.get(db_apis.get_session(), id=load_balancer_id)
loadbalancer from octavia db
2. Prepare create_loadbalancer_flow store data object
3. self._lb_flows.get_create_load_balancer_flow
4. get taskflow engine
5. run create loadbalancer flow
octavia_api
octavia_cw
repositories
https://blog.@51CTO博客

3. get_create_load_balancer_flow의 UML 다이어그램을 확장합니다 .

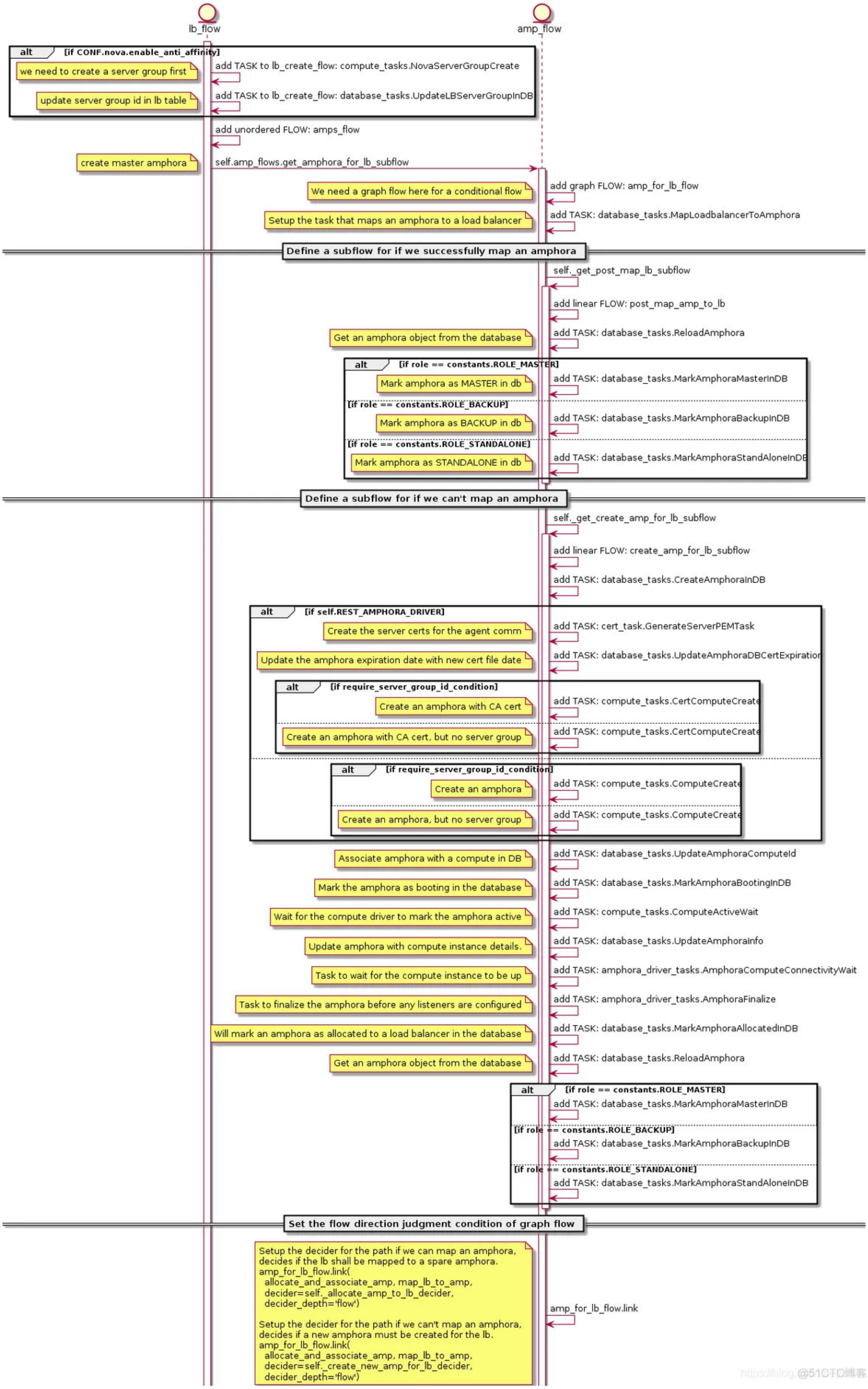

octavia_cw
lb_flow
Creates a conditional graph flow that allocates a loadbalancer to two spare amphorae.
self._lb_flows.get_create_load_balancer_flow
1. create linear FLOW: lb_create_flow
Mark loadbalancer provisioning_status to ERROR if flow failure.
2. add TASK: lifecycle_tasks.LoadBalancerIDToErrorOnRevertTask
alt
[if topology == constants.TOPOLOGY_ACTIVE_STANDBY]
create active standby amphoras
3.1. self._create_active_standby_topology
[if topology == constants.TOPOLOGY_SINGLE]
create single amphora
3.2. self._create_single_topology
Reload the loadbalancer and create networking subflows for created/allocated amphorae.
4. self.get_post_lb_amp_association_flow
alt
[if listeners]
create listeners belong to loadbalancer
5. self._create_listeners_flow
octavia_cw
lb_flow
https://blog.@51CTO博客

로드 밸런서 흐름 생성에는 두 가지 주요 사항이 있음을 알 수 있습니다.

- 로드 밸런서 토폴로지 생성
- amphora(e)에 대한 네트워킹 생성

먼저 첫 번째 요점을 설명합니다. 소위 로드 밸런서 토폴로지는 본질적으로 amphorae의 고가용성 토폴로지를 의미합니다. `SINGLE`과 `ACTIVE_STANDBY`의 두 가지 유형을 지원합니다. 이름에서 알 수 있듯이 `SINGLE`은 가용성이 높지 않으며 프로덕션 환경에서 사용하지 않는 것이 권장되는 단일 노드 앰포라입니다. 반면 `ACTIVE_STANDBY`는 `Keepalived` 마스터/백엔드 마스터-슬레이브 모드에 의존하는 이중 앰포라를 구현합니다. 따라서 이 문서에서는 `SINGLE` 토폴로지에 대해서는 다루지 않습니다.

Amphora 토폴로지 만들기의 UML 다이어그램 :


lb_flow
amp_flow
alt [if CONF.nova.enable_anti_affinity]
we need to create a server group first
add TASK to lb_create_flow: compute_tasks.NovaServerGroupCreate
update server group id in lb table
add TASK to lb_create_flow: database_tasks.UpdateLBServerGroupInDB
add unordered FLOW: amps_flow
create master amphora
self.amp_flows.get_amphora_for_lb_subflow
add graph FLOW: amp_for_lb_flow
We need a graph flow here for a conditional flow
Setup the task that maps an amphora to a load balancer
add TASK: database_tasks.MapLoadbalancerToAmphora
Define a subflow for if we successfully map an amphora
self._get_post_map_lb_subflow
add linear FLOW: post_map_amp_to_lb
Get an amphora object from the database
add TASK: database_tasks.ReloadAmphora
alt [if role == constants.ROLE_MASTER]
Mark amphora as MASTER in db
add TASK: database_tasks.MarkAmphoraMasterInDB
[if role == constants.ROLE_BACKUP]
Mark amphora as BACKUP in db
add TASK: database_tasks.MarkAmphoraBackupInDB
[if role == constants.ROLE_STANDALONE]
Mark amphora as STANDALONE in db
add TASK: database_tasks.MarkAmphoraStandAloneInDB
Define a subflow for if we can't map an amphora
self._get_create_amp_for_lb_subflow
add linear FLOW: create_amp_for_lb_subflow
add TASK: database_tasks.CreateAmphoraInDB
alt [if self.REST_AMPHORA_DRIVER]
Create the server certs for the agent comm
add TASK: cert_task.GenerateServerPEMTask
Update the amphora expiration date with new cert file date
add TASK: database_tasks.UpdateAmphoraDBCertExpiration
alt [if require_server_group_id_condition]
Create an amphora with CA cert
add TASK: compute_tasks.CertComputeCreate
Create an amphora with CA cert, but no server group
add TASK: compute_tasks.CertComputeCreate
alt [if require_server_group_id_condition]
Create an amphora
add TASK: compute_tasks.ComputeCreate
Create an amphora, but no server group
add TASK: compute_tasks.ComputeCreate
Associate amphora with a compute in DB
add TASK: database_tasks.UpdateAmphoraComputeId
Mark the amphora as booting in the database
add TASK: database_tasks.MarkAmphoraBootingInDB
Wait for the compute driver to mark the amphora active
add TASK: compute_tasks.ComputeActiveWait
Update amphora with compute instance details.
add TASK: database_tasks.UpdateAmphoraInfo
Task to wait for the compute instance to be up
add TASK: amphora_driver_tasks.AmphoraComputeConnectivityWait
Task to finalize the amphora before any listeners are configured
add TASK: amphora_driver_tasks.AmphoraFinalize
Will mark an amphora as allocated to a load balancer in the database
add TASK: database_tasks.MarkAmphoraAllocatedInDB
Get an amphora object from the database
add TASK: database_tasks.ReloadAmphora
alt [if role == constants.ROLE_MASTER]
add TASK: database_tasks.MarkAmphoraMasterInDB
[if role == constants.ROLE_BACKUP]
add TASK: database_tasks.MarkAmphoraBackupInDB
[if role == constants.ROLE_STANDALONE]
add TASK: database_tasks.MarkAmphoraStandAloneInDB
Set the flow direction judgment condition of graph flow
Setup the decider for the path if we can map an amphora,
decides if the lb shall be mapped to a spare amphora.
amp_for_lb_flow.link(
allocate_and_associate_amp, map_lb_to_amp,
decider=self._allocate_amp_to_lb_decider,
decider_depth='flow')
Setup the decider for the path if we can't map an amphora,
decides if a new amphora must be created for the lb.
amp_for_lb_flow.link(
allocate_and_associate_amp, map_lb_to_amp,
decider=self._create_new_amp_for_lb_decider,
decider_depth='flow')
amp_for_lb_flow.link
https://blog.@51CTO博客

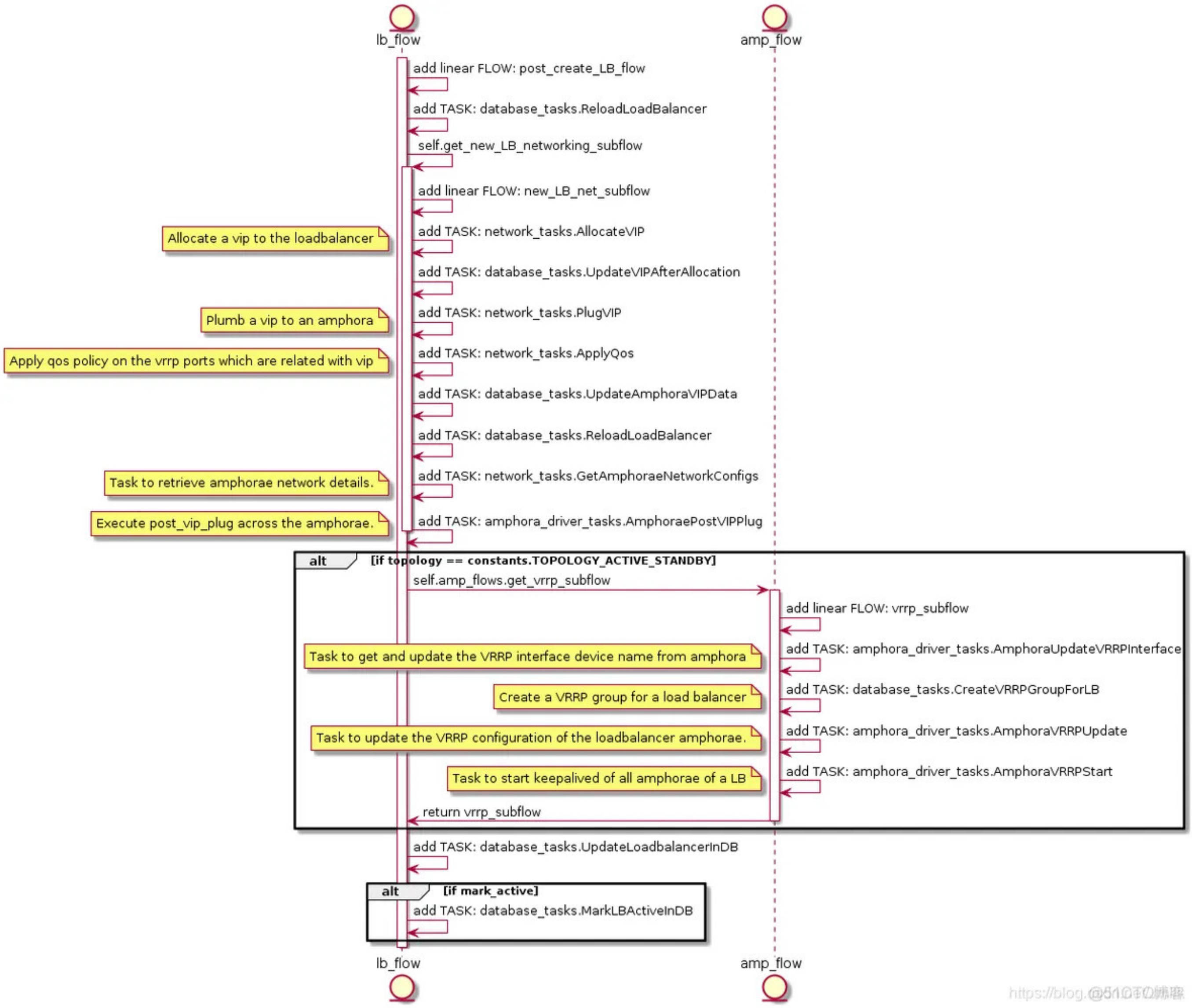
저는 특히 몇 가지 세부 사항을 강조합니다.

- 로드 밸런서 토폴로지가 ACTIVE_STANDBY인 경우 `[nova] enable_anti_affinity = True` 설정을 통해 Nova의 반친화성 메커니즘을 적용하여 고가용성을 더욱 향상시키도록 할 수도 있습니다.
- 로드밸런서에 대한 암포라 준비는 즉석에서 새로운 암포라를 생성하여 시간 낭비 없이 `space amphora pool`에서 직접 수행할 수 있습니다. `amphora for lb flow`는 먼저 `space amphora pool`에 로드밸런서에 매핑할 수 있는 여유 암포라가 있는지 확인하고, 있으면 직접 매핑합니다. `space amphora pool`은 `Housekeeping Manager` 메커니즘에 의해 유지 관리됩니다. 있는 경우 직접 매핑되며, 그렇지 않은 경우 새 앰포라 작업 흐름 생성을 활성화해야 합니다. `space amphora pool`은 `Housekeeping Manager` 메커니즘에 의해 유지되며 풀 크기는 구성에 따라 설정됩니다 `[house_keeping] spare_amphora_pool_ size=2`는 `pool size`를 설정합니다.
- `amphora for lb flow`는 그래프 흐름 방식을 사용하며, 이는 무방향성이며 흐름 방향을 사용자 정의할 수 있습니다. 개발자는 사용자 정의 판단 조건(`amp_for_lb_flow.link`)을 통해 작업 흐름 방향을 제어할 수 있습니다. 이 흐름에 정의된 판단 조건은 다음과 같습니다.

```
if loadbalancer mapping Amphora instance SUCCESS:
    Upload database associations for loadbalancer and amphora
else:
    Create amphora first
    Upload database associations for loadbalancer and amphora
```

두 번째는 `amphora`가 처음에는 `lb-mgmt-net`에만 연결되어 있다가 `loadbalancer`를 할당받은 후에는 `vip-net`에도 `amphora`를 연결해야 한다는 점입니다. 이때는 `octavia-api` 단계에서 `vip-net`에 생성한 `port:loadbalancer-<load_balancer_id>`가 이 시점에서 사용됩니다. 또한 `ACTIVE_STANDBY` 토폴로지를 사용하는 경우 `Keepalived VIP` 드리프트의 캐리어로 두 개의 암포라 각각에 마운트하기 위해 `vip-net`에 두 개의 `VRRP_port (octavia-lb-vrrp-<amphora_id>)`가 생성됩니다.

amphora(e)에 대한 네트워킹 생성을 위한 UML 다이어그램 :


lb_flow
amp_flow
add linear FLOW: post_create_LB_flow
add TASK: database_tasks.ReloadLoadBalancer
self.get_new_LB_networking_subflow
add linear FLOW: new_LB_net_subflow
Allocate a vip to the loadbalancer
add TASK: network_tasks.AllocateVIP
add TASK: database_tasks.UpdateVIPAfterAllocation
Plumb a vip to an amphora
add TASK: network_tasks.PlugVIP
Apply qos policy on the vrrp ports which are related with vip
add TASK: network_tasks.ApplyQos
add TASK: database_tasks.UpdateAmphoraVIPData
add TASK: database_tasks.ReloadLoadBalancer
Task to retrieve amphorae network details.
add TASK: network_tasks.GetAmphoraeNetworkConfigs
Execute post_vip_plug across the amphorae.
add TASK: amphora_driver_tasks.AmphoraePostVIPPlug
alt
[if topology == constants.TOPOLOGY_ACTIVE_STANDBY]
self.amp_flows.get_vrrp_subflow
add linear FLOW: vrrp_subflow
Task to get and update the VRRP interface device name from amphora
add TASK: amphora_driver_tasks.AmphoraUpdateVRRPInterface
Create a VRRP group for a load balancer
add TASK: database_tasks.CreateVRRPGroupForLB
Task to update the VRRP configuration of the loadbalancer amphorae.
add TASK: amphora_driver_tasks.AmphoraVRRPUpdate
Task to start keepalived of all amphorae of a LB
add TASK: amphora_driver_tasks.AmphoraVRRPStart
return vrrp_subflow
add TASK: database_tasks.UpdateLoadbalancerInDB
alt
[if mark_active]
add TASK: database_tasks.MarkLBActiveInDB
lb_flow
amp_flow
https://blog.@51CTO博客

Amphora 네트워킹과 관련된 몇 가지 주요 작업을 나열하십시오.

- `network_tasks.AllocateVIP`
- `network_tasks.PlugVIP`
- `amphora_driver_tasks.AmphoraePostVIPPlug`
- `amphora_driver_tasks.AmphoraVRRPUpdate`
- `amphora_driver_tasks.AmphoraVRRPStart`

실제로는 `Octavia Networking`이 구현의 초점이지만 문제 발생률이 높기 때문에 기본 구현을 마스터해야만 문제를 더 잘 파악할 수 있다고 생각하기 때문에 이러한 구현에 집중해야 합니다.

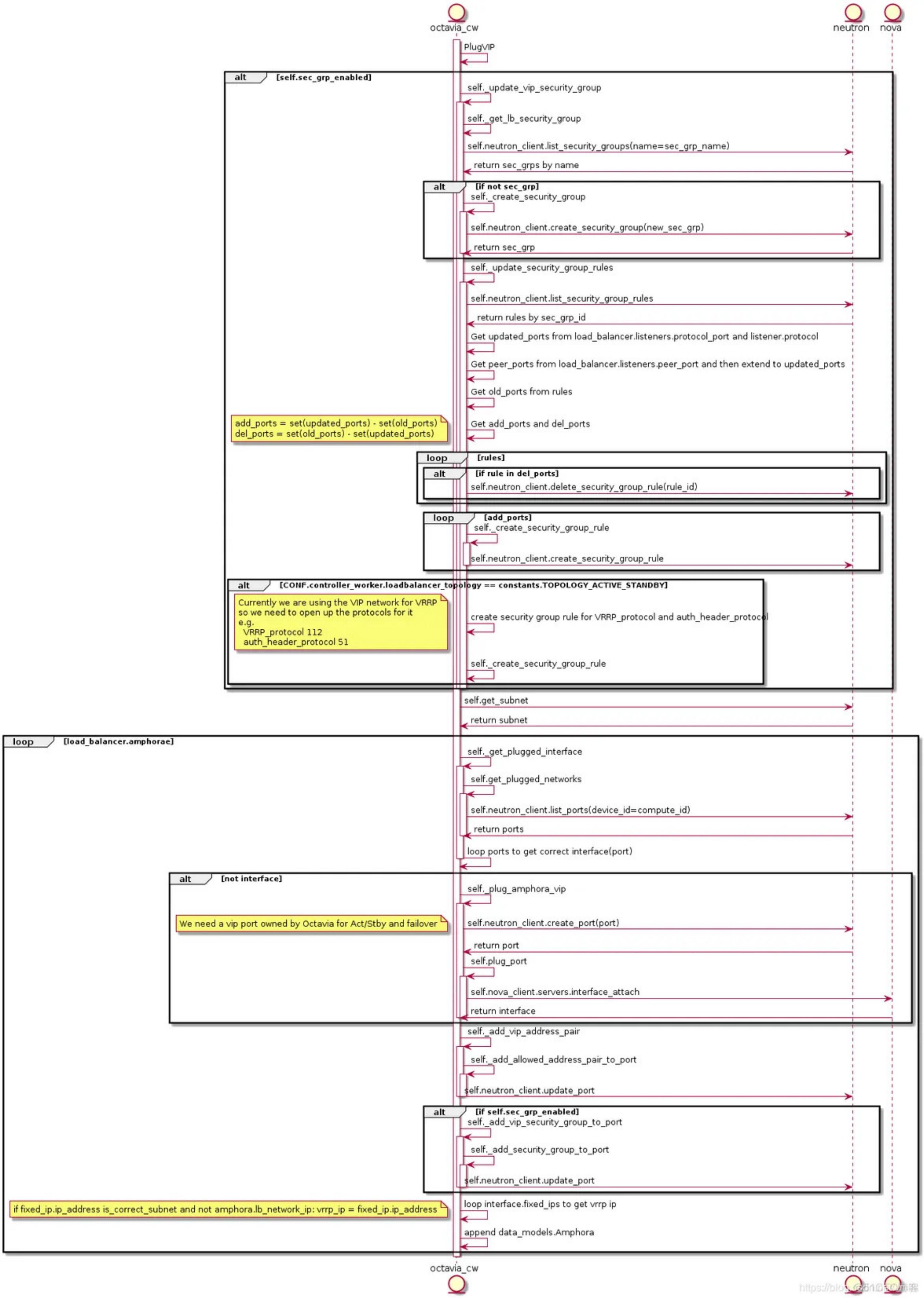
network_tasks.AllocateVIP

`AllocateVIP`는 `VIP`의 포트가 존재하는지 확인하고 `Port`, `VIP` 및 `LB`를 연결하는 `data_models.Vip` 객체를 반환하는 `Neutron`의 인터페이스 래퍼 `AllowedAddressPairsDriver.allocate_vip` 메서드를 호출합니다. 이 메서드는 `octavia-api`에서 한 번 호출되므로 애플리케이션이 `octavia-worker`로 이동할 때 `VIP`의 포트는 이미 생성되어 있고, `data_models.Vip` 라이브러리는 `Task:UpdateAmphoraVIPData`에 의해 유지됩니다.

network_tasks.PlugVIP

`AllocateVIP 로드는 Neutron에서 VIP를 할당하고 PlugVIP는 Amphora에 VIP를 삽입하는 역할을 담당합니다.`

PlugVIP의 UML 다이어그램


octavia_cw
neutron
nova
PlugVIP
alt [self.sec_grp_enabled]
self._update_vip_security_group
self._get_lb_security_group
self.neutron_client.list_security_groups(name=sec_grp_name)
return sec_grps by name
alt [if not sec_grp]
self._create_security_group
self.neutron_client.create_security_group(new_sec_grp)
return sec_grp
self._update_security_group_rules
self.neutron_client.list_security_group_rules
return rules by sec_grp_id
Get updated_ports from load_balancer.listeners.protocol_port and listener.protocol
Get peer_ports from load_balancer.listeners.peer_port and then extend to updated_ports
Get old_ports from rules
add_ports = set(updated_ports) - set(old_ports)
del_ports = set(old_ports) - set(updated_ports)
Get add_ports and del_ports
loop [rules]
alt [if rule in del_ports]
self.neutron_client.delete_security_group_rule(rule_id)
loop [add_ports]
self._create_security_group_rule
self.neutron_client.create_security_group_rule
alt [CONF.controller_worker.loadbalancer_topology == constants.TOPOLOGY_ACTIVE_STANDBY]
Currently we are using the VIP network for VRRP
so we need to open up the protocols for it
e.g.
VRRP_protocol 112
auth_header_protocol 51
create security group rule for VRRP_protocol and auth_header_protocol
self._create_security_group_rule
self.get_subnet
return subnet
loop [load_balancer.amphorae]
self._get_plugged_interface
self.get_plugged_networks
self.neutron_client.list_ports(device_id=compute_id)
return ports
loop ports to get correct interface(port)
alt [not interface]
self._plug_amphora_vip
We need a vip port owned by Octavia for Act/Stby and failover
self.neutron_client.create_port(port)
return port
self.plug_port
self.nova_client.servers.interface_attach
return interface
self._add_vip_address_pair
self._add_allowed_address_pair_to_port
self.neutron_client.update_port
alt [if self.sec_grp_enabled]
self._add_vip_security_group_to_port
self._add_security_group_to_port
self.neutron_client.update_port
if fixed_ip.ip_address is_correct_subnet and not amphora.lb_network_ip: vrrp_ip = fixed_ip.ip_address
loop interface.fixed_ips to get vrrp ip
append data_models.Amphora
octavia_cw
neutron
nova

PlugVIP의 로직 구현에는 크게 두 가지 영역이 있습니다.

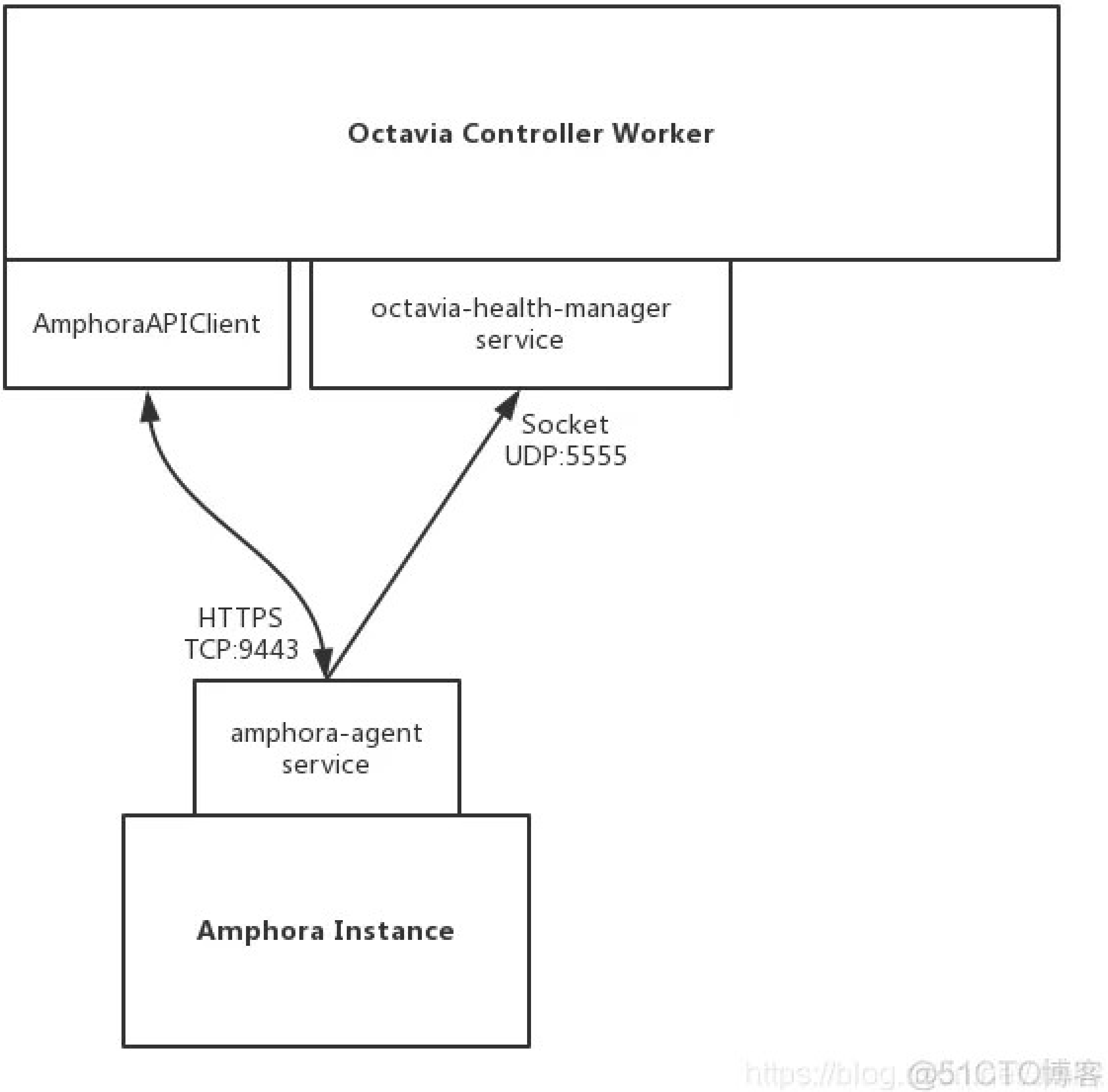
1. VIP 포트의 `security_group_rules`을 업데이트합니다. 서비스에 대한 외부 액세스는 VIP를 통해 이루어지고 리스너도 VIP에 종속되기 때문에 실제로 VIP의 보안 그룹 규칙은 동적입니다. 예를 들어 로드밸런서에 `HTTP:8080` 리스너를 추가하면 해당 VIP에 `HTTP:8080` 규칙이 업로드됩니다.
2. 모든 암포라에 대해 로드밸런서를 폴링하고, 암포라에 필요한 모든 포트가 있는지 확인하고, 없는 경우 Neutron API를 호출하여 포트를 생성한 다음 Nova API를 조사하여 암포라 인스턴스에 마운트합니다.

`TASK:AllocateVIP` 및 `TASK:PlugVIP` 이후의 `create lb flow`은 기본적으로 Amphora의 외부 리소스를 준비하는 것으로 완료되며, 다음 흐름은 Amphora의 내부 구현으로 이동합니다. 흐름의 다음 단계는 Amphora의 내부 구현이 될 것인데, `Octavia Controller Worker`와 Amphora 에이전트가 어떻게 서로 안전하게 통신할 수 있는지에 대한 문제가 남아 있기 때문입니다. 따라서 `Amphora Agent`와 `AmphoraAPIClient` 간의 통신 구현에 대해 논의한 다음, 우리가 이야기하지 않은 나머지 세 가지 작업으로 돌아가 보겠습니다.

Amphora

위에서 언급했듯이 Amphora는 본질적으로 `HAProxy`와 `Keepalived`의 런타임 캐리어 역할을 하는 인스턴스입니다. 제 생각에 Amphora는 매우 고전적인 프록시와 유사한 구현으로, '프록시가 어떻게 옥타비아 컨트롤러 워커와 안전하게 통신해야 하는가'라는 문제를 해결하는 데 매우 유용합니다. 하트비트 프로토콜을 사용자 정의하는 방법은 무엇인가요? 호스트의 운영 환경에 미치는 영향을 줄이는 방법은 무엇인가요? 이는 이러한 문제에 대한 훌륭한 데모이며 배우고 연구할 가치가 있습니다!

`amphora-agent`와 `Octavia Controller Worker` 간의 통신 모델 다이어그램


Octavia Controller Worker
AmphoraAPIClient
octavia-health-manager service
Socket UDP:5555
HTTPS TCP:9443
amphora-agent service
Amphora Instance
https://blog.@51CTO博客

먼저, `amphora-agent`가 `AmphoraAPIClient`와 통신을 설정하는 방법을 살펴보겠습니다.

Amphora Agent

`amphora-agent` 서비스 프로세스는 `Launch Amphora`과 함께 시작되며, 전자는 웹 애플리케이션을 제공하고 후자는 `WSGI HTTP` 서버 역할을 하는 `Flask & gunicorn` 구현을 사용합니다. 서비스 프로세스의 주요 기능은 `from octavia.cmd.agent import main`에서 가져옵니다.

```
# file:
/opt/rocky/octavia/octavia/amphorae/backends/agent/api_server/server.py

class Server(object):
    def __init__(self):
        self.app = flask.Flask(__name__)
        ...
        self.app.add_url_rule(rule=PATH_PREFIX +
'/listeners/<amphora_id>/<listener_id>/haproxy',
                                      view_func=self.upload_haproxy_config,
                                      methods=['PUT'])
        ...
```

위의 서버 클래스는 경량 Flask 프레임워크 래퍼 구현인 `amphora-agent API`의 경로 정의와 보기 기능을 완성하며, 앱 객체는 결국 `gunicorn`에 의해 로드되어 실행됩니다. 각 `route_url`의 의미를 이해할 수 있는 공식 문서 Octavia HAProxy Amphora API를 참조하시기 바라며, 여기서는 반복하지 않겠습니다.

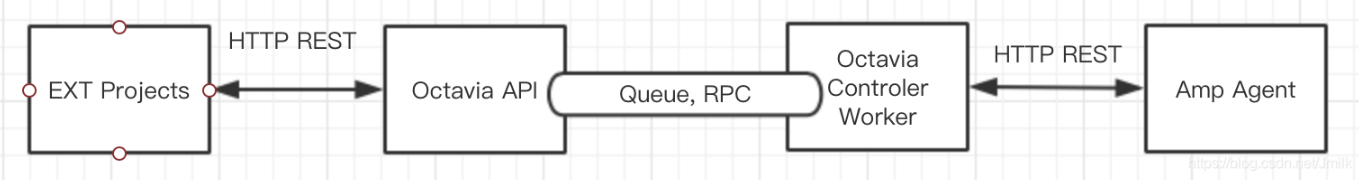
AmphoraAPIClient

`AmphoraAPIClient`는 `amphora-agent REST API`의 클라이언트 측 구현으로, 상위 계층 서비스 호출을 위한 `Octavia HAProxy Amphora API`에 대한 모든 URL 요청을 캡슐화합니다.

```
# file:
/opt/rocky/octavia/octavia/amphorae/drivers/haproxy/rest_api_driver.py

class AmphoraAPIClient(object):
    def __init__(self):
        super(AmphoraAPIClient, self).__init__()
        self.secure = False
        ...
```

옥타비아의 커뮤니케이션 아키텍처를 돌아보기

- `Octavia API`: 외부 REST API 통신을 제공합니다.
- `Queue`: 내부 RPC 통신을 제공합니다.
- `Amphora agent`: `Amphora`와 `Octavia Controller Worker` 간의 `REST API` 통신을 제공합니다.



AmphoraePostVIPPlug

`TASK:AmphoraePostVIPPlug` 구현으로 돌아가서, `AmphoraePostVIPPlug`는 모든 `Amphorae`를 개별적으로 폴링하여 `AmphoraAPIClient`를 호출하여 `PUT plug/vip/{vip}` 요청을 `amphora-agent`에 전송하여 VM의NIC 구성 파일을 업데이트하고 라우팅 규칙을 추가합니다. 네트워크의 주소 덮어쓰기를 방지하고 Amphora 운영 체제를 깨끗하게 유지하기 위해 `AmphoraePostVIPPlug`는 `Amphora` 액세스 `lb-mgmt-net`을 제외한 모든 NIC가 나뉘어져 있는 네트워크 네임스페이스를 생성합니다.

보시다시피 `AmphoraePostVIPPlug`의 의미는 VIP용 NIC 장치 파일을 생성하고 여기에 `vip-net` 포트의 네트워크 정보를 주입하는 것입니다. 구현은 `Plug:plug_vip` 메서드이며, 다음은 이 작업이 수행되

는 방법을 보여줍니다.

초기 상태의 Amphora에는 `lb-mgmt-net`과 통신할 수 있는 포트가 하나만 있습니다.

```
root@amphora-cd444019-ce8f-4f89-be6b-0edf76f41b77:~# ifconfig
ens3      Link encap:Ethernet  HWaddr fa:16:3e:b6:8f:a5
          inet addr:192.168.0.9  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:feb6:8fa5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:19462 errors:14099 dropped:0 overruns:0 frame:14099
          TX packets:70317 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1350041 (1.3 MB)  TX bytes:15533572 (15.5 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Amphora가 로드밸런서에 할당되면 `vrrp_port` 유형의 포트가 추가됩니다. `vrrp_port`는 `Keepalived` 가상 경로의 NIC 역할을 하며 네임스페이스(일반적으로 eth1)에 주입됩니다.

```
root@amphora-cd444019-ce8f-4f89-be6b-0edf76f41b77:~# ip netns exec amphora-
haproxy ifconfig
eth1      Link encap:Ethernet  HWaddr fa:16:3e:f4:69:4b
          inet addr:172.16.1.3  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fef4:694b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:12705 errors:0 dropped:0 overruns:0 frame:0
          TX packets:613211 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:762300 (762.3 KB)  TX bytes:36792968 (36.7 MB)

eth1:0    Link encap:Ethernet  HWaddr fa:16:3e:f4:69:4b
          inet addr:172.16.1.10  Bcast:172.16.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
```

`VRRP IP: 172.16.1.3` 및 `VIP: 172.16.1.10`은 모두 `lb-vip-network`의 DHCP에 의해 할당되며 포트 `octavia-lb-vrrp-<amphora_uuid>` 및 `octavia-lb-<loadbalancer_uuid>`에 해당합니다. 여기서 인터페이스 eth1은 다음과 같이 구성됩니다.

```
root@amphora-cd444019-ce8f-4f89-be6b-0edf76f41b77:~# ip netns exec amphora-
haproxy cat /etc/network/interfaces.d/eth1
```

```
auto eth1
iface eth1 inet dhcp
root@amphora-cd444019-ce8f-4f89-be6b-0edf76f41b77:~# ip netns exec amphora-
haproxy cat /etc/network/interfaces.d/eth1.cfg

# Generated by Octavia agent
auto eth1 eth1:0
iface eth1 inet static
address 172.16.1.3
broadcast 172.16.1.255
netmask 255.255.255.0
gateway 172.16.1.1
mtu 1450

iface eth1:0 inet static
address 172.16.1.10
broadcast 172.16.1.255
netmask 255.255.255.0
# Add a source routing table to allow members to access the VIP
post-up /sbin/ip route add 172.16.1.0/24 dev eth1 src 172.16.1.10 scope link
table 1
post-up /sbin/ip route add default via 172.16.1.1 dev eth1 onlink table 1
post-down /sbin/ip route del default via 172.16.1.1 dev eth1 onlink table 1
post-down /sbin/ip route del 172.16.1.0/24 dev eth1 src 172.16.1.10 scope
link table 1
post-up /sbin/ip rule add from 172.16.1.10/32 table 1 priority 100
post-down /sbin/ip rule del from 172.16.1.10/32 table 1 priority 100
post-up /sbin/iptables -t nat -A POSTROUTING -p udp -o eth1 -j MASQUERADE
post-down /sbin/iptables -t nat -D POSTROUTING -p udp -o eth1 -j MASQUERADE
```

Keepalived 서비스 프로세스 시작

고가용성 서비스 제공을 위해 `loadbalancer_topology = ACTIVE_STANDBY`일 때만 Keepalived 시작 프로세스가 수행되며, `TASK:AmphoraVRRPUpdate`와 `TASK:AmphoraVRRPStart`는 각각 Keepalived 구성 파일의 내용 편집과 Keepalived 서비스 프로세스 시작을 담당합니다.

`TASK:AmphoraVRRPUpdate`의 로직은 비교적 간단한데, `amphora topology`의 `VIP port`, `VRRP_ports`의 네트워크 정보를 `keepalived.conf` 설정 파일의 Jinja 템플릿에 렌더링한 후, `AmphoraAPIClient`를 통해 `amphora-agent`에 `PUT vrrp/upload` 요청을 보내 Keepalived 설정 파일의 내용을 업데이트하는 것입니다.

`TASK:AmphoraVRRPStart`는 `AmphoraAPIClient`에서 `PUT vrrp/start` 요청을 전송하여 `amphora-agent`의 `view_func:manage_service_vrrp(action=start)`를 실행합니다.

```

# file:
/opt/rocky/octavia/octavia/amphorae/backends/agent/api_server/keepalived.py

    def manager_keepalived_service(self, action):
        action = action.lower()
        if action not in [consts.AMP_ACTION_START,
                          consts.AMP_ACTION_STOP,
                          consts.AMP_ACTION_RELOAD]:
            return webob.Response(json=dict(
                message='Invalid Request',
                details="Unknown action: {0}".format(action)), status=400)

        if action == consts.AMP_ACTION_START:
            keepalived_pid_path = util.keepalived_pid_path()
            try:
                # Is there a pid file for keepalived?
                with open(keepalived_pid_path, 'r') as pid_file:
                    pid = int(pid_file.readline())
                os.kill(pid, 0)

                # If we got here, it means the keepalived process is
running.
                # We should reload it instead of trying to start it again.
                action = consts.AMP_ACTION_RELOAD
            except (IOError, OSError):
                pass

        cmd = ("/usr/sbin/service octavia-keepalived {action}".format(
            action=action))

        try:
            subprocess.check_output(cmd.split(), stderr=subprocess.STDOUT)
        except subprocess.CalledProcessError as e:
            LOG.debug('Failed to %s octavia-keepalived service: %s %s',
                      action, e, e.output)
            return webob.Response(json=dict(
                message="Failed to {0} octavia-keepalived service".format(
                    action), details=e.output), status=500)

        return webob.Response(
            json=dict(message='OK',
                      details='keepalived
{action}ed'.format(action=action)),
            status=202)
```

분명히 amphora-agent는 `/usr/sbin/service octavia-keepalived start` 명령을 실행하여 `keepalived`서비스 프로세스를 시작합니다. `octavia-keepalived.service`의 내용을 살펴보세요:

```
# file: /usr/lib/systemd/system/octavia-keepalived.service

[Unit]
Description=Keepalive Daemon (LVS and VRRP)
After=network-online.target .service
```

```
Wants=network-online.target
Requires=.service

[Service]
# Force context as we start keepalived under "ip netns exec"
SELinuxContext=system_u:system_r:keepalived_t:s0
Type=forking
KillMode=process

ExecStart=/sbin/ip netns exec amphora-haproxy /usr/sbin/keepalived  -D -d -f
/var/lib/octavia/vrrp/octavia-keepalived.conf -p
/var/lib/octavia/vrrp/octavia-keepalived.pid

ExecReload=/bin/kill -HUP $MAINPID
PIDFile=/var/lib/octavia/vrrp/octavia-keepalived.pid

[Install]
WantedBy=multi-user.target
```

위 내용에 따르면

- 실제 keepalived 서비스 프로세스는 namespace amphora-haproxy에서 시작됩니다.
- keepalived 구성 파일은 `/var/lib/octavia/vrrp/octavia-keepalived.conf`입니다.

view_func:manage_service_vrrp는 시작 외에도 중지 및 다시 로드 작업을 지원하며, keepalived 구성 파일에 대한 업데이트는 view_func:upload_keepalived_config에서 처리합니다.

keepalived 설정 파일의 내용으로 넘어가 보겠습니다.

```

# file: /var/lib/octavia/vrrp/octavia-keepalived.conf

vrrp_script check_script {
  script /var/lib/octavia/vrrp/check_script.sh   # VRRP check
  interval 5
  fall 2
  rise 2
}

vrrp_instance 01197be798d5440da846cd70f52dc503 { # VRRP instance name is
loadbalancer UUID
  state MASTER                                   # Master router
  interface eth1                                 # VRRP IP device
  virtual_router_id 1                            # VRID
  priority 100
  nopreempt
  garp_master_refresh 5
  garp_master_refresh_repeat 2
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass b76d77e
  }

  unicast_src_ip 172.16.1.3                      # VRRP IP
  unicast_peer {
    172.16.1.7                                   # Backup router VRRP IP
  }

  virtual_ipaddress {
    172.16.1.10                                  # VIP address
  }
  track_script {
    check_script
  }
}
```

보시다시피, `keepalived`는 `eth1`을 `VRRP IP`와 `VIP`를 위한 인터페이스로 사용하며, 또한 `eth1`은 이미 `TASK:AmphoraePostVIPPlug`에서 `namespace amphora`에 준비되어 있습니다.

`check_script.sh` 스크립트는 VIP 드리프트를 결정하기 위한 기준으로 각 Amphorae의 HAProxy의 상태를 확인하는 데 사용됩니다.

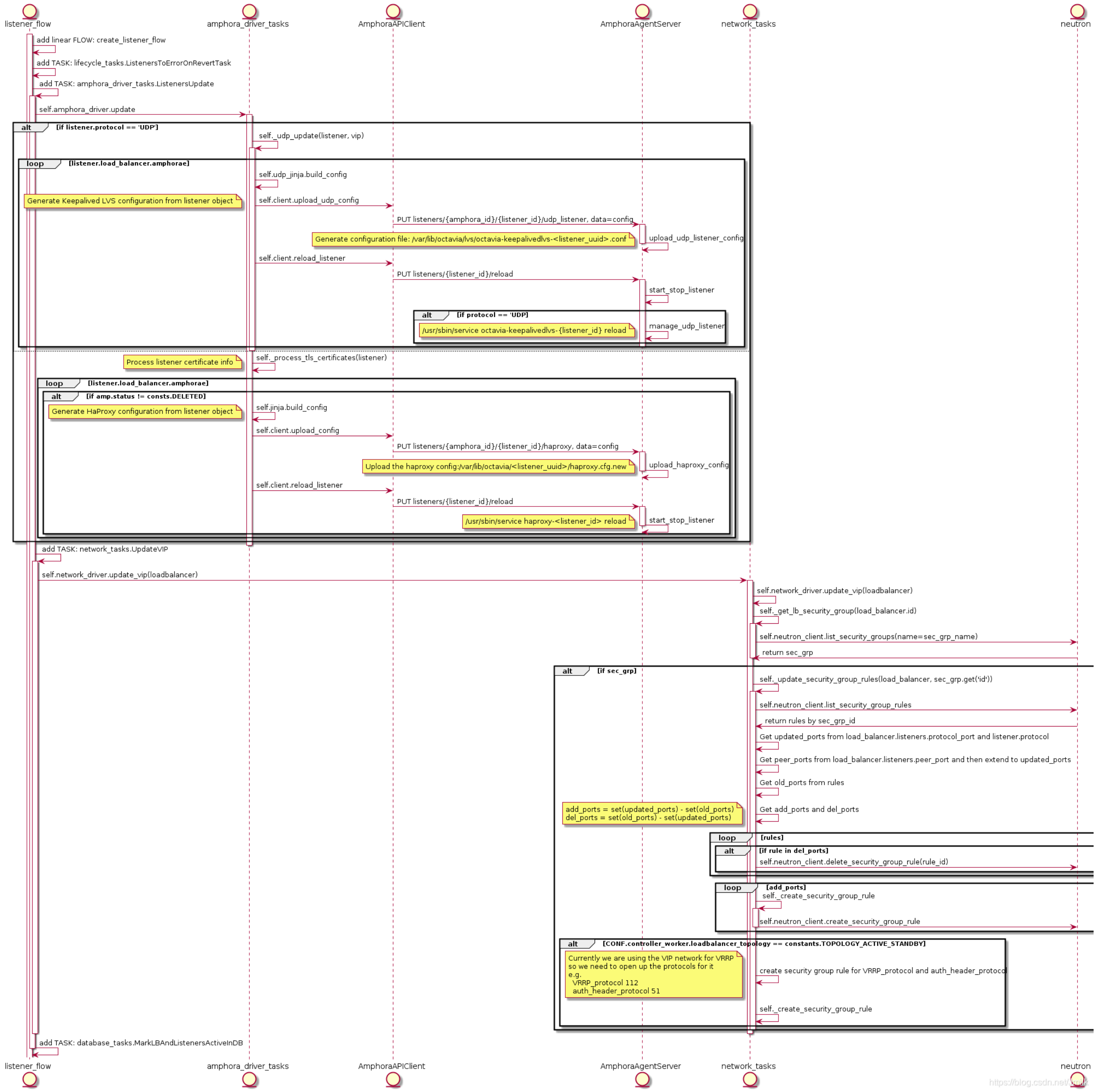
```

root@amphora-caa6ba0f-1a68-4f22-9be9-8521695ac4f4:~# cat
/var/lib/octavia/vrrp/check_scripts/haproxy_check_script.sh
haproxy-vrrp-check /var/lib/octavia/d367b5ec-24dd-44b3-b947-
e0ff72c75e66.sock; exit $?
```

`Amphora Instance`는 `amphora-agent`와 `keepalived`도 실행하지만 리스너가 생성될 때만 시작되는 haproxy도 실행합니다. haproxy는 리스너가 생성된 후에야 시작되므로 리스너 웨어러 프로세스를 분석할 때까지 기다리겠습니다.

로드밸런서를 만드는 과정을 분석해보니, 간단히 말해 `amphorae`를 준비하고 `amphorae`를 `vip-net`에 연결하는 것이지만 그 사이에는 음미할 만한 세부 사항이 많이 있습니다.

리스너 생성 프로세스 분석

리스너 흐름 생성 UML 도표


listener_flow
amphora_driver_tasks
AmphoraAPIClient
AmphoraAgentServer
network_tasks
neutron
add linear FLOW: create_listener_flow
add TASK: lifecycle_tasks.ListenersToErrorOnRevertTask
add TASK: amphora_driver_tasks.ListenersUpdate
self.amphora_driver.update
alt [if listener.protocol == 'UDP']
self._udp_update(listener, vip)
loop [listener.load_balancer.amphorae]
self.udp_jinja.build_config
Generate Keepalived LVS configuration from listener object
self.client.upload_udp_config
PUT listeners/{amphora_id}/{listener_id}/udp_listener, data=config
Generate configuration file: /var/lib/octavia/lvs/octavia-keepalivedlvs-<listener_uuid>.conf
upload_udp_listener_config
self.client.reload_listener
PUT listeners/{listener_id}/reload
start_stop_listener
alt [if protocol == 'UDP']
/usr/sbin/service octavia-keepalivedlvs-{listener_id} reload
manage_udp_listener
self._process_tls_certificates(listener)
Process listener certificate info
loop [listener.load_balancer.amphorae]
alt [if amp.status != consts.DELETED]
self.jinja.build_config
Generate HaProxy configuration from listener object
self.client.upload_config
PUT listeners/{amphora_id}/{listener_id}/haproxy, data=config
Upload the haproxy config:/var/lib/octavia/<listener_uuid>/haproxy.cfg.new
upload_haproxy_config
self.client.reload_listener
PUT listeners/{listener_id}/reload
/usr/sbin/service haproxy-<listener_id> reload
start_stop_listener
add TASK: network_tasks.UpdateVIP
self.network_driver.update_vip(loadbalancer)
self.network_driver.update_vip(loadbalancer)
self._get_lb_security_group(load_balancer.id)
self.neutron_client.list_security_groups(name=sec_grp_name)
return sec_grp
alt [if sec_grp]
self._update_security_group_rules(load_balancer, sec_grp.get('id'))
self.neutron_client.list_security_group_rules
return rules by sec_grp_id
Get updated_ports from load_balancer.listeners.protocol_port and listener.protocol
Get peer_ports from load_balancer.listeners.peer_port and then extend to updated_ports
Get old_ports from rules
add_ports = set(updated_ports) - set(old_ports)
del_ports = set(old_ports) - set(updated_ports)
Get add_ports and del_ports
loop [rules]
alt [if rule in del_ports]
self.neutron_client.delete_security_group_rule(rule_id)
loop [add_ports]
self._create_security_group_rule
self.neutron_client.create_security_group_rule
alt [CONF.controller_worker.loadbalancer_topology == constants.TOPOLOGY_ACTIVE_STANDBY]
Currently we are using the VIP network for VRRP
so we need to open up the protocols for it
e.g.
VRRP_protocol 112
auth_header_protocol 51
create security group rule for VRRP_protocol and auth_header_protocol
self._create_security_group_rule
add TASK: database_tasks.MarkLBAndListenersActiveInDB
listener_flow
amphora_driver_tasks
AmphoraAPIClient
AmphoraAgentServer
network_tasks
neutron
https://blog.csdn.net/Jmilk

위 그림에서 볼 수 있듯이, `openstack loadbalancer listener create –protocol HTTP –protocol-port 8080 lb-1` 명령을 실행하여 리스너를 생성하면 `Task:ListenersUpdate`로 실행되며, 여기서 `AmphoraAPIClient`가 호출됩니다:

- `PUT listeners/{amphora_id}/{listener_id}/haproxy`: `haproxy` 구성 파일 업데이트
- `PUT listeners/{listener_id}/reload` : `haproxy` 서비스 프로세스 재시작

따라서 `haproxy` 서비스 프로세스는 로드밸런서에 대한 리스너가 생성될 때만 시작됩니다. 또한 리스너가 생성될 때 Listener에 포함된 프로토콜 및 포트 정보를 VIP의 보안 그룹 규칙에서 업데이트해야 하기 때문에 `Task:UpdateVIP`도 실행됩니다.

haproxy 서비스 프로세스 시작

amphora에 로그인하여 haproxy 구성 파일을 확인합니다.

```
# file: /var/lib/octavia/1385d3c4-615e-4a92-aea1-c4fa51a75557/haproxy.cfg, Listener UUID: 1385d3c4-615e-4a92-aea1-c4fa51a75557

# Configuration for loadbalancer 01197be7-98d5-440d-a846-cd70f52dc503
global
    daemon
    user nobody
    log /dev/log local0
    log /dev/log local1 notice
    stats socket /var/lib/octavia/1385d3c4-615e-4a92-aea1-c4fa51a75557.sock mode 0666 level user
    maxconn 1000000

defaults
    log global
    retries 3
    option redispatch

peers 1385d3c4615e4a92aea1c4fa51a75557_peers
    peer l_Ustq0qE-h-_Q1dlXLXBAiWR8U 172.16.1.7:1025
    peer O08zAgUhIv9TEXhyYZf2iHdxOkA 172.16.1.3:1025


frontend 1385d3c4-615e-4a92-aea1-c4fa51a75557
    option httplog
    maxconn 1000000
    bind 172.16.1.10:8080
    mode http
    timeout client 50000
```

참조링크

- https://www.cnblogs.com/jmilkfan-fanguiju/p/10589749.html
- https://blog.51cto.com/u_15301988/3126511

From:
https://atl.kr/dokuwiki/ - **AllThatLinux!**

Permanent link:
https://atl.kr/dokuwiki/doku.php/octavia_lb_%EA%B5%AC%ED%98%84_%EB%B0%8F_%EB%B6%84%EC%84%9D?rev=1728534437

Last update: **2024/10/10 04:27**