

# K-ONE 기술 문서 #35

## Redundancy VNFFG module in tacker

Document No. K-ONE #35

Version 1.0

Date 2018-12-27

Author(s) 백호성

■ 문서의 연혁

버전	날짜	작성자	내용
1.0	18.12.27	백호성	

본 문서는 2018년도 정부(미래창조과학부)의 재원으로 정보통신  
기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-16-2012, 글로벌  
SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information &  
communications Technology Promotion(IITP) grant funded by the  
Korea government(MSIP) (No. B0190-16-2012, Global SDN/NFV  
OpenSource Software Core Module/Function Development)

## 기술문서 요약

본 고는 에 대한 기술문서이다. 본 고는 다음과 같이 구성된다. 1장은 VNF Forwarding Graph 구성을 위한 오픈 소스 프로젝트로 Tacker, Ceilometer 그리고 Gnocchi 프로젝트에 대해 분석한다. 2장은 OpenStack Tacker의 기능에 대해 설명하고 3장에서는 기존의 VNFFG 구성 방식과 Redundancy VNFFG module 개발 방안  
에 대해 기술한다.

## Contents

### K-ONE #35. Redundancy VNFFG module in tacker

1. VNF Forwarding Graph를 위한 오픈 소스 프로젝트 .....	5
1.1. OpenStack Tacker .....	6
1.2. OpenStack Ceilometer .....	8
1.3. OpenStack Gnocchi .....	10
2. OpenStack Tacker의 NFV Orchestration 기능 .....	10
2.1. VNF Manager .....	11
2.2. VNFFG Manager .....	11
3. OpenStack Tacker 기반의 Redandacy VNFFG module 제안 .....	24
3.1. 기존의 VNFFG 구성 module .....	24
3.2. Redundancy VNFFG module .....	28
4. 결론 .....	30

## 그림 목차

그림 1. OpenStack 개념적 구조.....	6
그림 2. OpenStack Tacker target 기능.....	7
그림 3. Tacker의 구조.....	8
그림 4. Ceilometer의 구조.....	8
그림 5. Ceilometer의 데이터 처리 기능.....	9
그림 6. http monitoring VNFD template (1/2).....	12
그림 7. http monitoring VNFD template (2/2).....	13
그림 8. networking-sfc 드라이버가 추가된 tacker 구조.....	14
그림 9. VNFM NSH TOSCA template.....	15
그림 10. VNFFGD TOSCA template exmaple.....	17
그림 11. VNFFGD 예제 코드 (1/2).....	18
그림 12. VNFFGD 예제 코드 (2/2).....	18
그림 13. VNFFG 결과 예제.....	19
그림 14. chain-show 결과 예제.....	19
그림 15. classifier-show 결과 예제.....	20
그림 16. VNFFGD REST Call.....	21
그림 17. VNFFGD 특성.....	21
그림 18. vnffg_db.py 소스코드 (1/3).....	26
그림 19 . vnffg_db.py 소스코드 (2/3).....	27
그림 20. vnffg_db.py 소스코드 (3/3).....	28

## K-ONE #35. Redundancy VNFFG module in tacker

## 1. VNF Forwarding Graph를 위한 오픈 소스 프로젝트

최근 네트워크가 다양한 서비스를 제공하면서 보안 유지나 성능 향상 등을 위해 Firewall, Deep Packet Inspection (DPI), Network Address Translation (NAT) 등과 같은 미들박스 (Middlebox), 즉 서비스 기능에 대한 의존성이 높아지고 있다. 또한, 특정 플로우 (flow)가 여러 네트워크 서비스 기능을 필요로 하는 경우에는, 서비스들이 해당 플로우에 대하여 논리적인 순서에 따라 적용될 수 있는 합리적인 방법이 필요하다. 즉, 여러 요구사항에 따라 통신 사업자의 목적과 정책에 맞추어 서비스 기능들을 하나의 논리적인 연결로 순서화하는 서비스 기능 체이닝 기술이 주목받고 있으며, 이에 대한 구조 및 프로토콜 표준화, 실제 구현에 관련된 다양한 연구가 학계, 산업계를 불문하고 활발하게 이루어지고 있다.

이러한 서비스 기능 체이닝은 Software Defined Networking (SDN) 기술 및 Network Function Virtualization (NFV) 기술을 통하여 더욱 효과적이고 유연하게 제공될 수 있다. SDN은 기존 네트워크 장비의 제어 평면과 데이터 평면을 분리시켜 논리적으로 중앙 집중된 컨트롤러를 통해 네트워크 제어 기능을 제공한다. 따라서, SDN 기술을 통해 전체 네트워크 뷰를 유지하며, 요구 사항 및 가변적인 네트워크 상황에 따라 동적으로 서비스 기능 체인을 구성할 수 있다. 한편, NFV 기술이란 고가의 네트워크 장비에 대한 투자비용 (CAPEX) 및 운용비용 (OPEX) 문제를 해결하기 위하여, 다양한 네트워크 서비스 기능들을 범용의 표준 서버 하드웨어에서 실행될 수 있는 소프트웨어로 구현하여 운용하는 것을 의미한다. 관련 표준화 노력의 일환으로, 통신 분야 표준화 단체인 ETSI (European Telecommunications Standards Institute)는 전 세계 주요 통신사업자들과 NFV ISG (Industry Specification Group)을 출범하고, 활발한 표준화 활동을 진행하고 있다.

현재 오픈소스 커뮤니티 기반의 SDN 컨트롤러 개발 프로젝트인 OpenDaylight [1]에서는 서비스 체이닝을 제공하기 위하여, 하위 프로젝트로 SFC 프로젝트를 유지하며 활발한 개발을 진행 중에 있다. OpenDaylight SFC 프로젝트는 IETF SFC [2] 작업 그룹에서 제안한 SFC 구조를 기반으로 개발을 진행중에 있다. 또한, ETSI NFV ISG [3]에서 제안한 NFV 구조를 기반으로 NFV 레퍼런스 플랫폼을 개발하고 있는 프로젝트인 OPNFV (Open Platform for NFV) [4]에서도 하위 프로젝트로 SFC 프로젝트가 존재하며 서비스 체인이 기술 개발을 진행하고 있다. 한편, 기존의 OpenDaylight SFC 프로젝트에서는 OPNFV 플랫폼 상에서의 가상화된 네트워크 기능들 (즉, Virtual Network Function (VNF))에 대한 서비스 기능 체이닝을 고려하지 않고 있기 때문에, OPNFV SFC 프로젝트에서는 OpenDaylight SFC 프로젝트를 업스트림 프로젝트로 하여 추가적으로 필요한 인터페이스 및 기능들에 대한 정의 및 개발을 진행 중이다.

OpenStack은 클라우드 컴퓨팅 오픈 소스 프로젝트로, 2010년 미 항공우주국과 랙스페이스사가 오픈스택을 설립하였고, 2012년 9월 비영리 단체인 오픈스택 파운데

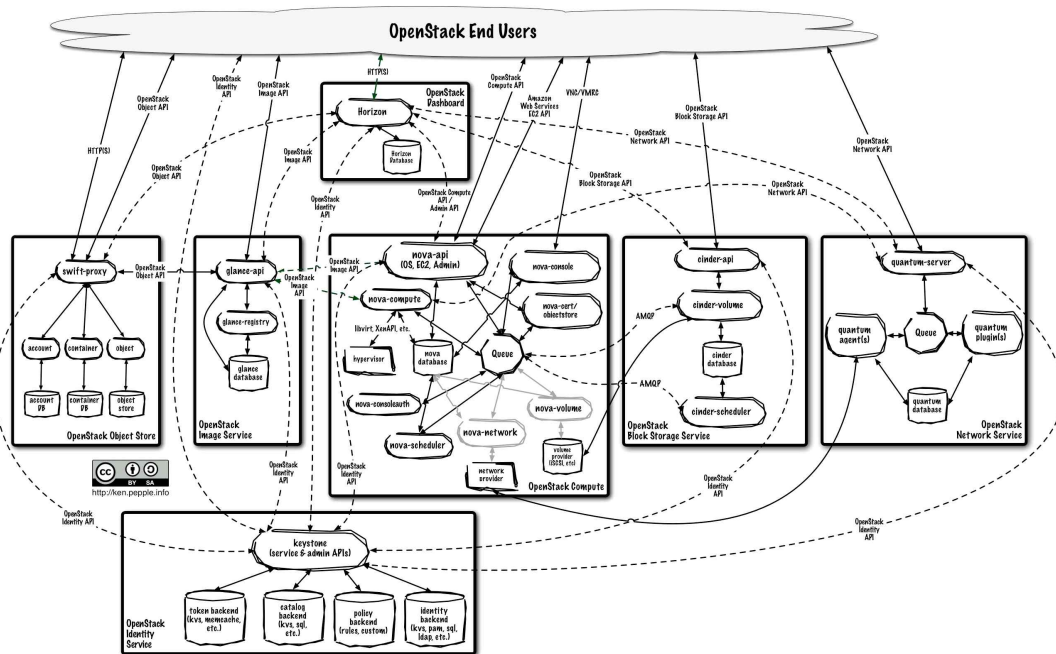


그림 1. OpenStack 개념적 구조

### 1.1. OpenStack Tacker 프로젝트

본 장에서는 OpenStack의 Kilo 버전부터 새롭게 추가된 Tacker 프로젝트에 대해 분석한다. Tacker 프로젝트는 NFV 환경에서 VNF를 관리하는 VNF Manager 기능과 NFV 서비스들을 관장하고 관리하는 NFV Orchestrator 기능을 지원하기 위한 프로젝트이다. 즉, Tacker 프로젝트에서 제공하는 기능은 아래 그림에서 빨간색 박스 부분이다. 가상화된 리소스를 바탕으로 VNF를 관리하고 구성하는 역할을 담당하는 VNFM과 NFV 환경에서 네트워크의 전체적인 조율 및 관리에 대한 기능을 수



행하는 NFVO로 구성된다. VNFM과 NFVO에서 VNF의 배치와 네트워크 서비스를 관리하는 데는 Network Service Descriptor (NSD)와 VNF 카탈로그에 저장되어 있는 Virtualized Network Function Descriptor (VNFD)를 기반으로 이루어진다.

Tacker의 구조는 아래 그림과 같다. VIM 부분의 VNF 생성을 위한 부분과 관리를 위한 드라이버 모니터링을 위한 드라이버가 존재하며, 현재 ping과 HTTPping 그리고 alarm-based monitoring이 있다. NFVO에서는 포워딩그래프와 네트워크 서비스 오케스트레이션, 다중 VIM 환경에서의 VNF 배치, 리소스 체크 및 할당 위한 VNFM으로 구성되어 있다. Tacker는 ETSI에서 논의되고 있는 기본 형태를 기반으로 제안하고 있기 때문에 특정 벤더에 종속되지 않고 멀티 벤더로부터 VNF를 배치하는 것이 가능하다.

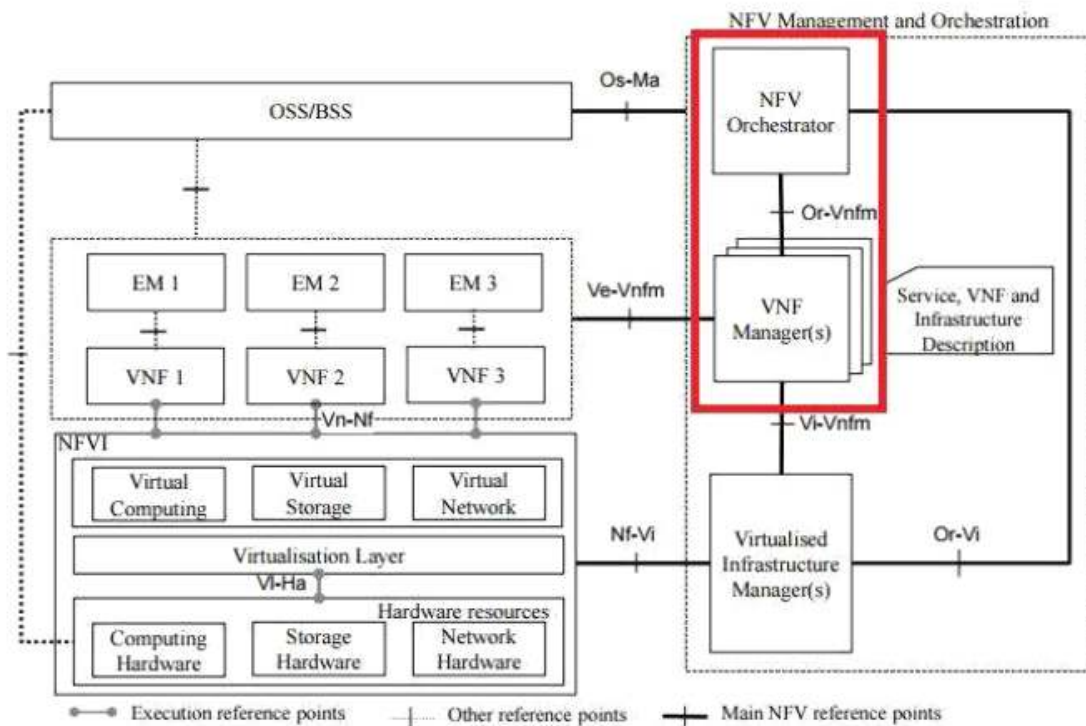


그림 2. OpenStack Tacker target 기능

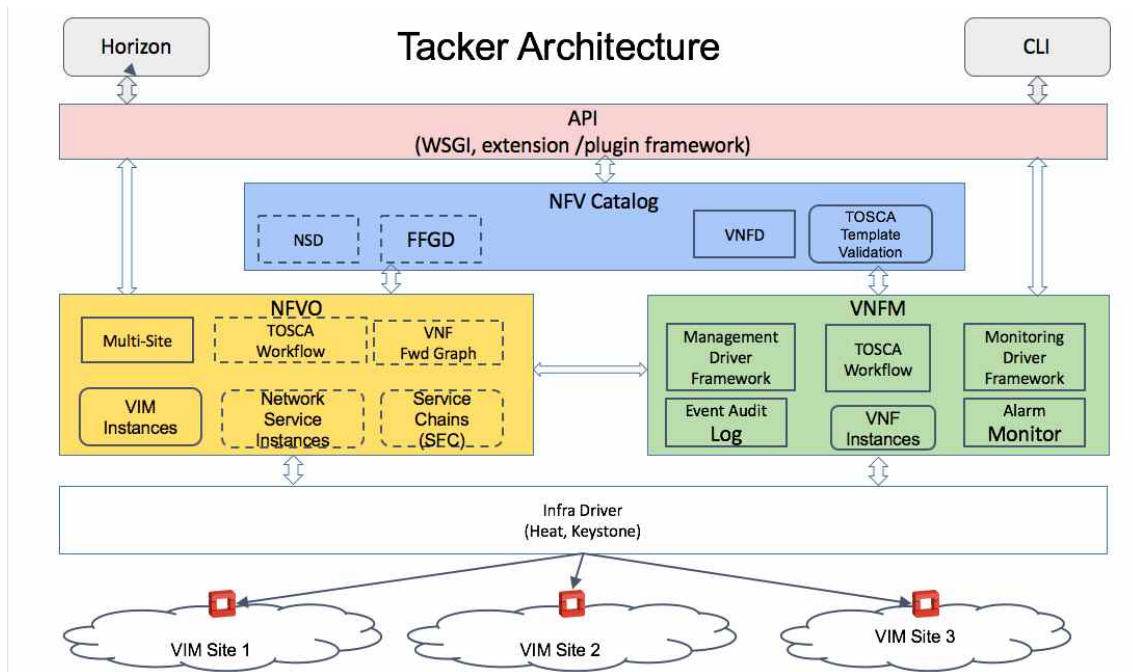


그림 3. Tacker의 구조

## 1.2. OpenStack Ceilometer 프로젝트

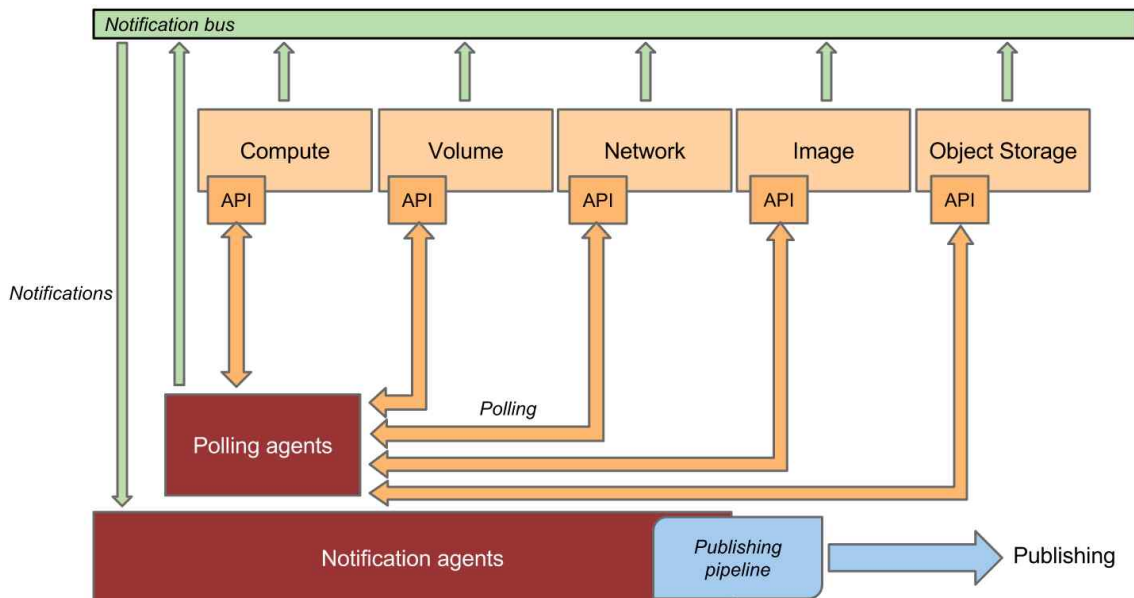


그림 4. Ceilometer 구조

본 장에서는 OpenStack에서 미터링 서비스를 담당하는 Ceilometer 프로젝트에 대해 분석한다. Ceilometer 프로젝트는 이후의 OpenStack 구성 요소를 지원하기 위해

현재의 모든 OpenStack 구성 프로젝트에서 데이터를 정규화하고 변환하는 기능을 제공하는 것을 타겟으로 하는 데이터 수집 서비스이다. 이 데이터는 모든 OpenStack 핵심 구성 요소에 대해 비용 청구, 리소스 트래킹 및 알람 기능을 제공하는 데 사용할 수 있다.

위의 그림은 Ceilometer의 구조이다. 각 구성요소의 역할로는 다음과 같다. polling agent는 OpenStack 서비스들을 폴링하고 Meter를 빌드하는 기능을 수행한다. notification agent는 메시지 큐로부터 notifications을 듣고 있다가 notification이 오면 이를 이벤트와 샘플로 변환시켜주는 기능을 수행하고 이에 대한 데이터는 미리 정의된 특정 타겟으로 publish된다. 현재 디폴트로는 gnocchi 프로젝트로 push된다. 다음으로 collector daemon은 이벤트와 미터링 데이터를 모으고 이를 기록하는 기능을 수행한다. api service는 collector service로부터 기록된 데이터를 보고 query를 하는 기능을 수행한다. 마지막으로, alarming은 특정 기준을 갖고 그 기준에 따라 알람을 하는 기능을 수행한다. 즉, Ceilometer가 제공하는 기능으로는 크게 3가지로, OpenStack 서비스들과 연관된 데이터를 효율적으로 미터링, 서비스들로부터 보내진 모니터링 알람을 통해 미터링 데이터와 이벤트를 수집하는 것, 그리고 메시지 큐와 데이터 스토어를 포함한 여러 타겟으로 수집된 데이터를 publish하는 것이다. Ceilometer로부터 작성된 데이터는 publisher를 사용하여 원하는 타겟들로 푸시할 수 있다. 이 경우, 사용하지 않는 데이터 미터링으로 인해 데이터 베이스가 비효율적으로 사용되는 것을 방지하기 위해, 사용하고자 하는 특정 데이터만 선택하여 미터링하는 것이 권장된다.

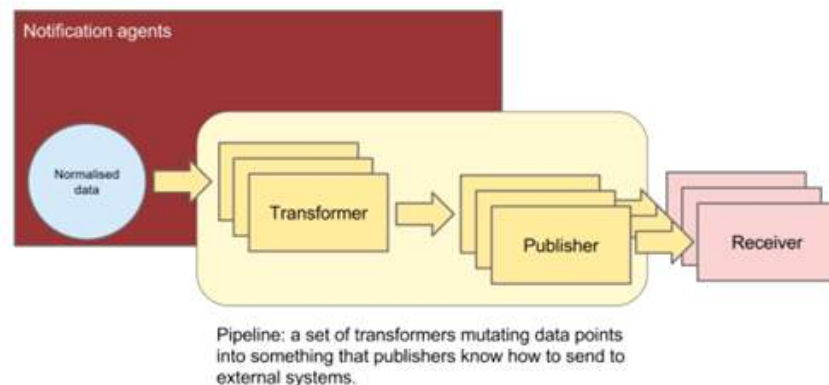


그림 5. Ceilometer의 데이터 처리 기능

현재 Ceilometer 프로젝트에서 지원되는 미터링 데이터는 대표적으로 memory, memory.usage, memory.resident, cpu, cpu.delta, cpu.util 등이 있다. 그리고 버전마다 지원되는 데이터가 추가되고 있다.

### 1.3. OpenStack Gnocchi 프로젝트

Gnocchi 프로젝트는 Ceilometer 프로젝트와 직접적으로 연관되어 개발되는 오픈소스 프로젝트로, 멀티 테넌트를 지원하는 서비스로서의 time series 데이터베이스이다. Gnocchi 프로젝트가 개발되는 motivation은 다음과 같다. Ceilometer 프로젝트의 초기 목표는 수집된 time-series 데이터를 저장하는 것이었고, 프로젝트의 초기 단계에서 이 time-series 데이터가 무엇을 어떻게 처리하고 조작하고 query하는지 명확하지 않았기 때문에 Ceilometer에서 사용되는 데이터 모델은 매우 유연하였다. 하지만 performance 측면에서는 결과가 좋지 않았다. 즉, 몇 주 동안 많은 양의 메트릭을 저장하기 위해서는 데이터 저장소 백엔드에 문제가 발생하였다. 이를 방지하기 위해서, Ceilometer가 생성하고 저장해야하는 데이터 및 측정 항목의 양을 고려할 때 새로운 방법이 필요했고, 이에 대한 solution이 Gnocchi 프로젝트이다.

Ceilometer에서 지원되는 publisher로는 7가지로, Gnocchi, notifier, udp, http, kafka, file, database 가 있다. 현재 Ceilometer의 디폴트 publisher로 Gnocchi가 설정되어 있다. Gnocchi는 데이터를 생성하고 조작하기 위해 HTTP REST 인터페이스를 제공한다. 따라서 Gnocchi를 실행하기 위해서는 HTTP 서버와 metric daemon이 요구된다. Gnocchi에서 archive policy는 metric을 수집하는 방법과 데이터의 lifetime을 정의한다. 즉, Gnocchi는 metric 객체 유형을 제공하고, metric은 측정 할 수 있는 모든 것(e.g., 서버의 cpu 사용량)을 지정한다. 또한 수많은 resource type 중에서 속성 value 기반의 필터링을 통해 특정 resource type을 찾아낼 수 있다.

## 2. OpenStack Tacker의 NFV Orchestration 기능

### 2.1. VNF Manager

Tacker 프로젝트는 NFV Orchestration 기능을 타겟으로 한 프로젝트로 대표적으로 VNF manager 기능이 있다. VNF manager의 기능은 다음과 같다. OpenStack과 같은 VIM 위에서 VNF의 lifetime을 관리한다. 즉, VNF의 생성 및 삭제, 모니터링, 스케일링을 통해 전반적으로 VNF를 관리한다. VNF를 배치하는 과정은 다음과 같다. Tacker 프로젝트는 template 기반으로 VNF를 생성 및 관리하기 때문에, Horizon 혹은 Command line을 통해 Tacker VNFD catalog에 TOSCA VNFD template을 온보딩 시켜야 한다. VNFD에는 기본적으로 VNFD의 id, 기본정보, vdu 정보, virtual link 정보, 모니터링 및 오토 스케일링 관련 파라미터 등이 포함되어 있다. 이를 통해, VNF의 생성뿐만 아니라, 생성이후에 필요한 관리 기능에 대한 정의 및 모니터링 이후의 액션에 대한 정의도 포함되어 있어 유용하다.

VNFD template을 온보딩하는 명령어는 다음과 같다.

```
tacker vnfd-create --vnfd-file <yaml file path> <VNFD-NAME>
```

VNFD template 의 간단한 예제로 다음과 같다. 아래 코드는 http monitoring 기능을 하는 VNFD template이다. VNFD template을 온보딩 시킨 후 VNF를 deploy 시키는 방법은 두가지로 다음과 같다.

1. Tacker에서 가장 일반적으로 쓰이는 방법으로 Tacker Catalog에 온보딩되어 있는 VNFD template을 이용하여 VNF를 생성한다. 이 방법을 통해 VNF를 생성시키는 명령어는 다음과 같다.

```
tacker vnf-create --vnfd-name <VNFD-FILE-NAME> <VNF-NAME>
```

즉, 미리 온보딩 되어있는 VNFD template을 이용하여 VNF를 생성한다.

2. TOSCA template으로 Tacker Catalog에 온보딩 시키지 않고 직접적으로 VNF를 생성한다. 이 방법은 NFV Catalog가 Tacker에 밖에 위치하여 Tacker가 단지 NFV workflow engine으로만 사용될 때 주로 쓰이는 방법이다. 이 방법을 통해 VNF를 생성하는 명령어는 다음과 같다.

```
tacker vnf-create --vnfd-template <VNFD-FILE-NAME> <VNF-NAME>
```

즉, VNF를 생성할 때 참조할 VNFD template을 NFV Catalog에서 직접 가져온다.

```
1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2
3  description: Demo example
4
5  metadata:
6    template_name: sample-tosca-vnfd
7
8  topology_template:
9    node_templates:
10      VDU1:
11        type: tosca.nodes.nfv.VDU.Tacker
12        capabilities:
13          nfv_compute:
14            properties:
15              num_cpus: 1
16              mem_size: 2 GB
17              disk_size: 20 GB
18        properties:
19          image: ubuntu
20          availability_zone: nova
21          mgmt_driver: noop
22          config: |
23            param0: key1
24            param1: key2
25          monitoring_policy:
26            name: http_ping
27            parameters:
28              retry: 5
29              timeout: 10
30              port: 8000
31          actions:
32            failure: respawn
33
34      CP1:
35        type: tosca.nodes.nfv.CP.Tacker
36        properties:
37          management: true
38          order: 0
39          anti_spoofing_protection: false
40        requirements:
```

그림 6. http monitoring VNFD template (1/2)

현재 VNF Manager에 구현되어 있는 명령어는 다음과 같다.

vnfd-list는 tacker의 Catalog에 온보딩 되어있는 vnfd의 리스트를 나타낸다. Onboarding 되어있지 않고 직접 VNF를 생성하는 데 사용된 VNFD list를 보기 위해서는 '-template-source all' flag를 사용하여야 한다.

다음으로, VNFM의 상태를 나타내는 명령어로는 vim-list, vnf-list, vnf-show,

```
46     CP2:
47         type: toska.nodes.nfv.CP.Tacker
48         properties:
49             order: 1
50             anti_spoofing_protection: false
51         requirements:
52             - virtualLink:
53                 node: VL2
54             - virtualBinding:
55                 node: VDU1
56
57     CP3:
58         type: toska.nodes.nfv.CP.Tacker
59         properties:
60             order: 2
61             anti_spoofing_protection: false
62         requirements:
63             - virtualLink:
64                 node: VL3
65             - virtualBinding:
66                 node: VDU1
67
68     VL1:
69         type: toska.nodes.nfv.VL
70         properties:
71             network_name: net_mgmt
72             vendor: Tacker
73
74     VL2:
75         type: toska.nodes.nfv.VL
76         properties:
77             network_name: net0
78             vendor: Tacker
79
80     VL3:
81         type: toska.nodes.nfv.VL
82         properties:
83             network_name: net1
84             vendor: Tacker
```

---

그림 7. http monitoring VNFD template (2/2)

vnfd-show가 있다. 그리고 생성된 VNF 및 VNFD를 제거할 수 있고 command는 다음과 같다.



```
tacker vnf-delete <VNF_ID/NAME>  
tacker vnfd-delete <VNFD_ID/NAME>
```

## 2.2. VNFFG Manager

VNFFG Manager는 Tacker project에 새롭게 추가된 기능이다. NFV 진영에서는 단순히 VNF를 생성하고 관리하는 단계를 넘어, 생성된 VNF를 SFC 기술을 이용하여 플로우의 요구사항에 따라 VNF를 논리적으로 연결하여 그래프를 구성하는 단계까지를 필요로 한다. 최종적으로, 단순히 논리적이고 추상적으로 구성된 그래프를 오버레이 네트워크 아래의 물리적인 네트워크에 실질적으로 렌더링하는 것이다.

현재 Tacker project에서는 단일 경로의 SFC만 지원 가능한 상태이다. 이 문제를 해결하기 위해서, 다중 경로 그래프를 여러 개의 단일 경로 SFC로 파싱하여 더 확장된 VNFFG를 생성할 수 있도록 한다. 이 feature는 체인 최적화 문제로 고려할 사항이 많아서 다음 버전에서 다루어질 것이다.

이 기능은 Tacker project에서 새롭게 추가되었기 때문에, Tacker Client, Horizon에도 추가된 기능에 맞게 추가되는 부분이 있다. 수정되는 부분은 다음과 같다.

1. tacker-horizon에 VNFFG 탭이 추가되었다. 이 탭을 통해서 미리 생성된 VNF를 이용하여 그래프를 생성할 수 있다. 이 부분은 TOSCA VNFFGD를 이용한다.
2. tacker client에서 tacker server에 CRUD VNFFG call을 통과하도록 하여야 한다.
3. tacker server는 NFVO 확장 부분과 VNFFG 기능과 자원을 통합하기 위한 플러그인 부분의 업데이트가 필요하다.
4. VNFFG를 위한 드라이버가 필요하다. 현재 SFC를 생성하기 위한 드라이버로는 뉴트론 기반의 networking-sfc와 OpenDaylight SFC가 있다. 이후에 VNFFG 플러그인을 지원할 드라이버는 networking-sfc 드라이버가 될 것이다. OpenDaylight SFC 드라이버는 networking-sfc의 별도의 spec으로 networking-sfc로의 드라이버로 처리될 것이다. 구조는 아래 그림과 같다.

데이터 모델에서 추가되는 부분은 다음과 같다. 'vnffgd', 'vnffgd\_nfp', 'vnffg', 'vnffg\_nfp', 'vnffg\_chain', 'vnffg\_classifier' 테이블들의 생성이 추가된다. 각 테이블의 기능을 간단히 설명하면 다음과 같다. 'vnffgd' 테이블은 VNFFG 인스턴스 생성 관련 속성과 'vnffg\_nfp' 테이블에 저장된 관련 NFP 및 'vnffg\_chain' 및 'vnffg\_classifier' 테이블에서 생성된 연관된 SFC 및 분류 자에 대한 참조를 갖는

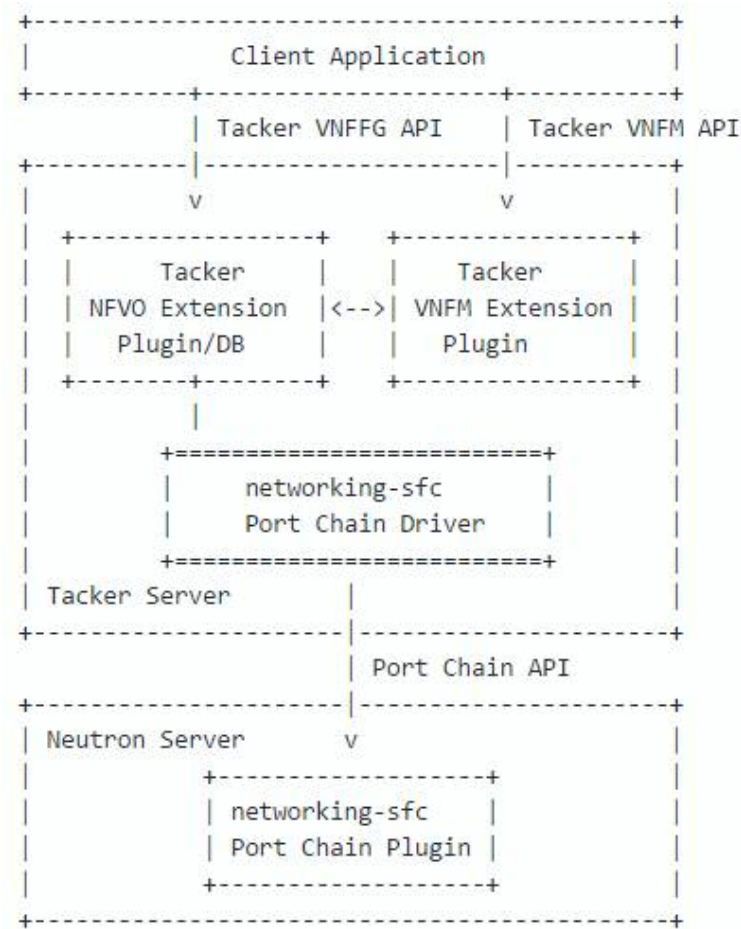


그림 8. networking-sfc 드라이버가 추가된 tacker 구조다.

또한 VNFD template은 VNFM에서 아래 코드와 같이, TOSCA 속성을 추가해야 한다. 아래의 속성은 VNFFG가 SFC provider에게 VNF가 Network Service Header (NSH) 를 지원하는지 그리고 패킷을 전송할 때, 어떤 encapsulation을 사용하는지에 대해 나타낼 수 있다. 여기서, NSH는 홑 단위로 SFC 홑에 대한 정보를 전달하는 IETF 프로토콜이다. 즉, NSH 헤더를 이용하여 체인을 통과하는 각 패킷에 추가되고 해당 체인에 대한 속성을 보유하여 패킷이 체인의 다음 VNF에 도착하면 VNF는 패킷이 속한 체인을 결정할 수 있고 패킷이 이전에 몇 개의 노드들을 통과하였는지에 대해 알 수 있다.

다음으로, REST API의 변화 부분은 다음과 같다. VNFFG를 생성하기 위해서는 VNFFGD가 생성되어야 한다. VNFFGD를 생성하는 방법은 TOSCA template scheme을 따른다. 포맷은 그룹에 정의되어 있는 하나 혹은 그 이상의 VNFFG가 요구된다.

VNFFG의 생성의 기본적인 방법은 미리 생성된 VNFFGD를 인스턴스화하여 수행

```
tosca.nodes.nfv.VNF:
  properties:
    nsh_aware:
      type: boolean
      required: false
      description: Does this VNF support IETF NSH

tosca.nodes.nfv.CP:
  properties:
    sfc_encapsulation:
      type: string
      required: false
      description: Identifies the method of encapsulation for NSH/SFC
      constraints:
        - [vxlan_gpe, ethernet, mpls]
```

그림 9. VNFM NSH TOSCA template

된다. VNFFG 생성의 기본 동작은 추상화된 VNF type을 선택하는 것이다. VNFFGD에는 경로에 사용된 forwarder 리스트를 포함하는 하나 이상의 Network Forwarding Path (NFP)가 존재한다. VNFFG 생성시 NFVO 플러그인은 VNFM을 쿼리하여 각 NFP에 해당하는 VNFD에서 존재하는 사용 가능한 VNF 인스턴스를 찾는다. 현재는 VNFD에 둘 이상의 같은 type의 VNF가 존재할 경우 어떤 VNF를 사용할지는 랜덤하게 선택된다. VNF는 다중 경로로는 선택될 수 있지만 여러 VNFFG에 중복되게는 선택되지 못한다. 그리고 '--vnf-mapping'를 이용하여 그래프에서 사용할 VNFM을 통해 이미 생성된 VNF 인스턴스를 지정하는 기능을 수행할 수 있다. 이것은 <VNFD>:<VNF Instance ID/NAME>로 매핑이 된다. 예를 들어, 아래 VNFFGD TOSCA template 예제 코드를 보면 "Forwarding\_path1"을 사용하는 경우 VNF1 및 VNF3 VNFD가 포함된다. 따라서 해당 VNFD, VNF1Test 및 VNF3Test에서 인스턴스가 두 개 이상 생성된 경우 해당 VNF 인스턴스를 구체적으로 사용하도록 NFVO에 알리기 위해 '--vnf-mapping VNF1 : VNF1Test, VNF3 : VNF3Test'로 매핑이 된다.

가상 포트에 매핑되어야 하는 요구 사항으로 CP는 특정 forwarder에 대해 VNFD에 정의되어야 한다. CP는 체인의 일부로 '포워딩'기능을 갖는 것으로 정의되어야 한다. VNFD의 논리적 CP는 VNF 인스턴스의 neutron 포트로 매핑되고, VNFFG는 주어진 CP에 대한 neutron port id를 얻기 위해, VNFM에게 query한다. 그리고 VNFM은 Heat 드라이버를 호출하여 정보를 찾고, 이것은 VNFM에 새롭게 구현되어야 할 기능이다. 추가적으로, '--symmetrical'를 이용하여 VNFFG에서 대상으로 나열된 경로에 대해 역방향 경로를 자동으로 구성할 수 있다. 다른 방법으로, VNFFGD에서도 정의될 수 있지만, 편의상 '--symmetrical'을 통해 구성한다.

```
Forwarding_path1:
  type: toska.nodes.nfv.FP
  id: 51
  description: creates path (CP11->CP12->CP32)
  properties:
    policy:
      type: ACL
      criteria:
        - neutron_net_name: tenant1_net
        - dest_port_range: 80-1024
        - ip_proto: tcp
        - ip_dest: 192.168.1.2
  requirements:
    - forwarder: VNF1
      capability: CP11
    - forwarder: VNF1
      capability: CP12
    - forwarder: VNF3
      capability: CP32

groups:
  VNFFG1:
    type: toska.groups.nfv.VNFFG
    description: HTTP to Corporate Net
    properties:
      vendor: tacker
      version: 1.0
      number_of_endpoints: 5
      dependent_virtual_link: [VL1,VL2,VL3]
      connection_point: [CP11,CP12,CP32]
      constituent_vnfs: [VNF1,VNF3]
    members: [Forwarding_path1]
```

그림 10. VNFFGD TOSCA template exmaple

VNFFG의 사용법은 다음과 같다. 먼저 Tacker를 이용하여 VNFFG를 생성하기 위해서는 networking-sfc 프로젝트를 설치하여야 한다. OpenStack에서 VNFFG는 SFC와 Classifier를 생성하기 위해 Neutron networking-sfc를 이용하기 때문이다. VNFFG의 생성법은 VNFD를 통해 VNF를 생성하는 것과 비슷하다. VNFFGD를 이용하여 VNFFG를 생성한다. 즉, VNFFGD를 생성한 후, Tacker 명령어를 이용하여 VNFFG를 생성한다. VNFFG에 대한 명령어는 다음과 같다.

VNFFGD를 생성하기 위한 명령어는 다음과 같다. VNFFGD를 생성하기 위해서는 VNFFGD에 리스트업 되어 있는 VNFD type에 대해 VNF 인스턴스가 생성되어 있어야 한다.

```
tacker vnffgd-create --vnffgd-file <vnffgd file> <vnffgd name>
```

VNFFGD의 예제코드는 아래 그림과 같다.

```
1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2
3  description: Sample VNFFG template
4
5  topology_template:
6    description: Sample VNFFG template
7
8    node_templates:
9
10     Forwarding_path1:
11       type: tosca.nodes.nfv.FP.Tacker
12       description: creates path (CP12->CP22)
13       properties:
14         id: 51
15         policy:
16           type: ACL
17           criteria:
18             - network_src_port_id: 640dfd77-c92b-45a3-b8fc-22712de480e1
19             - destination_port_range: 80-1024
20             - ip_proto: 6
21             - ip_dst_prefix: 192.168.1.2/24
22         path:
23           - forwarder: VNFD1
24             capability: CP12
25           - forwarder: VNFD2
26             capability: CP22
```

그림 11. VNFFGD 예제 코드 (1/2)

VNFFG를 생성하는 명령어는 다음과 같다. 미리 인스턴스화 되어 있는 vnffgd를 이용하여 VNFFG를 생성하는 것이다.

```
tacker vnffg-create --vnffgd-name <vnffgd name> \
  --vnf-mapping <vnf mapping> --symmetrical <boolean>
```

```

28     groups:
29         VNFFG1:
30             type: tosca.groups.nfv.VNFFG
31             description: HTTP to Corporate Net
32             properties:
33                 vendor: tacker
34                 version: 1.0
35                 number_of_endpoints: 5
36                 dependent_virtual_link: [VL12,VL22]
37                 connection_point: [CP12,CP22]
38                 constituent_vnfs: [VNFD1,VNFD2]
39             members: [Forwarding_path1]

```

그림 12. VNFFGD 예제 코드 (2/2)

```

tacker vnffg-create --vnffgd-name <vnffgd name> \
  --param-file <param file> --vnf-mapping <vnf mapping> \
  --symmetrical <boolean>

```

VNFFG를 나타내는 명령어는 다음과 같다. 이 명령어를 통해 나온 결과의 예제는 아래 그림과 같다.

```

tacker vnffg-show <vnffg name>

```

Field	Value
forwarding_paths	Forwarding_path1
id	19233232-d3e2-4c47-a94d-d1b1ab9889e5
name	myvnffg
tenant_id	0b324885958c42ad939e7d636abe2352
vnffgd_id	5279690a-2153-11e6-b67b-9e71128cae77
vnf_mapping	[{VNFD1:testVNFD1}, {VNFD2:testVNFD2}]
status	ACTIVE

그림 13. VNFFG 결과 예제

생성된 체인을 보는 명령어는 다음과 같다. 이를 통해 나타나는 특성들은 아래 그림과 같다.

```

tacker chain-show <chain id>

```



Field	Value
chain_id	b8ad61b1-5fac-48ab-9231-dc7d5de6ad4d
classifier_id	0a52a0d9-2a1f-4019-94c3-5401c4af5d36
id	19233232-d3e2-4c47-a94d-d1b1ab9889e5
name	Forwarding-path1
tenant_id	0b324885958c42ad939e7d636abe2352
path_id	200
symmetrical	false
vnffg_id	19233232-d3e2-4c47-a94d-d1b1ab9889e5

그림 14. chain-show 결과 예제

Classifier 자체를 나타내는 명령어는 다음과 같다. 이를 통해 나타나는 특성들에 대한 예제는 아래 그림과 같다.

```
tacker classifier-show <classifier id>
```

Field	Value
acl_match_criteria	{"source_port": 2005, "protocol": 6, "dest_port": 80}
chain_id	b8ad61b1-5fac-48ab-9231-dc7d5de6ad4d
id	0a52a0d9-2a1f-4019-94c3-5401c4af5d36
nfp_id	19233232-d3e2-4c47-a94d-d1b1ab9889e5
status	PENDING_CREATE
tenant_id	0b324885958c42ad939e7d636abe2352

그림 15. classifier-show 결과 예제

REST Calls	Type	Expected Response	Body Data Schema	Description
create_vnffgd	post	200 OK	schema 1	Creates VNFFGD
delete_vnffgd	delete	200 OK	None	Deletes VNFFG by name or ID
show_vnffgd	get	200 OK	None	Returns output of specific VNFFG ID, including associated chains and classifiers
list_vnffgds	get	200 OK	None	Returns list of configured VNFFGD Names/IDs

그림 16. VNFFGD REST Call

VNFFGD에 정의되어 있는 특성들은 아래 그림과 같다. 즉, id, name, VNFFGD에 대한 설명이 나타나 있는 description, VNFFGD template이 나타나 있는 attributes 그리고 VNFFGD를 실행할 프로젝트 id인 tenant\_id로 구성되어 있다.

Attribute Name	Type	Access	Default Value	Validation/Conversion	Description
id	string (UUID)	RO, All	generated	N/A	identity
name	string	RW, All	None (required)	string	human+readable name
description	string	RW, All	''	string	description of VNFFGD
attributes	dict	RW, All	None	template/dict	VNFFGD template
tenant_id	string	RW, All	None (required)	string	project id to launch VNFFGD

그림 17. VNFFGD 특성

VNFFGD에 대한 REST call은 아래 그림과 같다. VNFFGD를 생성하고 삭제하는 create\_vnffgd, delete\_vnffgd와 특정 VNFFG id를 return하는 show\_vnffgd 그리고



VNFFGD의 리스트를 나타내는 list\_vnffgds가 있다.

## 3. OpenStack Tacker 기반의 Redandacy VNFFG module 제안

### 3.1. 기존의 VNFFG 구성 module

앞서 기술한 바와 같이 Tacker의 VNFFG 기능은 트래픽이 지나야하는 VNF들을 정의하고, 해당 트래픽을 요구하는 VNF들을 지나도록 조율하고 관리하는 것을 목적으로 한다. 즉, Tacker 서버로부터 추상화된 VNFFG TOSCA에서 정의된 VNF목록 등의 VNFFG 관련된 정보를 서비스 기능 체인 (SFC) 및 트래픽을 분류하는 Classifier에 제공해야한다. 정보를 받은 SFC는 트래픽에 요구하는 VNF들의 목록을 구성하고, Classifier은 다수의 트래픽을 분류하는 역할을 한다.

VNF가 VNFD로 정의되는 것과 같이 VNFFG는 VNFFG Descriptor로 정의된다. VNFFGD를 생성한 후에 VNFFG는 Tacker 서버에 의해 인스턴스화 된다.

VNFFG 생성 관련된 명령어는 다음과 같다.

- VNFFGD 생성

```
tacker vnffgd-create --vnffgd-file <vnffgd file> <vnffgd name>
```

미리 작성된 vnffgd file을 참조하여 vnffgd name의 이름으로 VNFFGD를 온보드 하는 명령어이다.

- VNFFG 생성

```
tacker vnffg-create --vnffgd-name <vnffgd name> --vnf-mapping <vnf mapping> --symmetrical <boolean> <vnffg name>
```

온보딩된 VNFFGD를 참조하여 vnffg name의 이름으로 VNFFG를 생성하는 명령어이다. --vnf-mapping 옵션을 통해 특정 VNF를 지칭하여 VNFFG를 생성 가능하며, 해당 옵션을 사용하지 않으면 임의의 VNF를 선택하여 VNFFG를 생성한다.

- SFC 관련 명령어

- **tacker nfp-list**

- networking forwarding path (nfp)의 리스트를 보여준다.

- **tacker nfp-show <nfp id>**

- nfp id를 갖는 특정 nfp의 속성을 보여준다.

- **tacker chain-list**

- 생성된 chain의 리스트를 보여준다.

- **tacker chain-show <chain id>**

- chain id를 갖는 특정 chain의 속성을 보여준다.

- **tacker classifier-list**

- classifier의 리스트를 보여준다.
- **tacker classifier-show <classifier id>**
  - classifier id를 갖는 특정 classifier의 속성을 보여준다.

앞서 기술한 명령어들이 어떻게 코드상으로 구현되었는지 확인하기 위해 Tacker 내의 VNFFG 생성 소스코드를 분석한다. 소스 코드는 VNFFG 생성 관련 함수들이 정의된 vnffg\_db.py 파일을 분석한다. 해당 파일은 tackер/db/nfvo 폴더 내에 존재하며, 분석하는 vnffg\_db.py 파일은 4월 1일 기준의 master 브랜치의 파일이다.

아래의 그림은 VnffgPluginDbMixin 클래스 중 일부를 나타낸다. 해당 클래스에서는 VNFFGD 와 VNFFG 관련 tackер CLI 명령어에 따른 호출 함수들을 정의하고 있다. VnffgPluginDbMixin 클래스의 퍼블릭 함수는 다음과 같다.

- VNFFG 관련 함수
  - def create\_vnffg
  - def get\_vnffg
  - def get\_vnffgs
  - def update\_vnffg
  - def delete\_vnffg
- VNFFGD 관련 함수
  - def create\_vnffgd
  - def get\_vnffgd
  - def get\_vnffgds
  - def delete\_vnffgd
- SFC 관련 함수
  - def get\_classifier
  - def get\_classifiers
  - def get\_nfp
  - def get\_nfps
  - def get\_sfc
  - def get\_sfcs

본 소스 코드 분석에서는 VNFFG 생성시 호출되는 create\_vnffg 함수를 중점적으로 분석한다. create\_vnffg 함수 분석을 통해 새로운 VNFFG 알고리즘을 구상할 예정이다.

```
188 class VnffgPluginDbMixin(vnffg.VNFFGPluginBase, db_base.CommonDbMixin):
189
190     def __init__(self):
191         super(VnffgPluginDbMixin, self).__init__()
192
193     def create_vnffg(self, context, vnffg):
194         vnffg_dict = self._create_vnffg_pre(context, vnffg)
195         sfc_instance = str(uuid.uuid4())
196         fc_instance = str(uuid.uuid4())
197         self._create_vnffg_post(context, sfc_instance,
198                                 fc_instance, vnffg_dict)
199         self._create_vnffg_status(context, vnffg_dict)
200         return vnffg_dict
201
202     def get_vnffg(self, context, vnffg_id, fields=None):
203         vnffg_db = self._get_resource(context, Vnffg, vnffg_id)
204         return self._make_vnffg_dict(vnffg_db, fields)
205
206     def get_vnffgs(self, context, filters=None, fields=None):
207         return self._get_collection(context, Vnffg, self._make_vnffg_dict,
208                                     filters=filters, fields=fields)
209
210     def update_vnffg(self, context, vnffg_id, vnffg):
211         vnffg_dict = self._update_vnffg_pre(context, vnffg_id)
212         self._update_vnffg_post(context, vnffg_id, constants.ACTIVE, vnffg)
213         return vnffg_dict
214
215     def delete_vnffg(self, context, vnffg_id):
216         self._delete_vnffg_pre(context, vnffg_id)
217         self._delete_vnffg_post(context, vnffg_id, False)
218
```

그림 18. vnffg\_db.py 소스코드 (1/3)

```

219     def create_vnffgd(self, context, vnffgd):
220         template = vnffgd['vnffgd']
221         LOG.debug(_('template %s'), template)
222         tenant_id = self._get_tenant_id_for_create(context, template)
223
224         with context.session.begin(subtransactions=True):
225             template_id = str(uuid.uuid4())
226             template_db = VnffgTemplate(
227                 id=template_id,
228                 tenant_id=tenant_id,
229                 name=template.get('name'),
230                 description=template.get('description'),
231                 template=template.get('template'))
232             context.session.add(template_db)
233
234             LOG.debug(_('template_db %(template_db)s'),
235                       {'template_db': template_db})
236             return self._make_template_dict(template_db)
237
238     def get_vnffgd(self, context, vnffgd_id, fields=None):
239         template_db = self._get_resource(context, VnffgTemplate,
240                                         vnffgd_id)
241         return self._make_template_dict(template_db, fields)
242
243     def get_vnffgds(self, context, filters=None, fields=None):
244         return self._get_collection(context, VnffgTemplate,
245                                     self._make_template_dict,
246                                     filters=filters, fields=fields)
247
248     def delete_vnffgd(self, context, vnffgd_id):
249         with context.session.begin(subtransactions=True):
250             vnffg_db = context.session.query(Vnffg).filter_by(
251                 vnffgd_id=vnffgd_id).first()
252             if vnffg_db is not None:
253                 raise nfvo.VnffgdInUse(vnffgd_id=vnffgd_id)
254
255             template_db = self._get_resource(context, VnffgTemplate,
256                                             vnffgd_id)
257             context.session.delete(template_db)

```

그림 19 . vnffg\_db.py 소스코드 (2/3)

```

259     def get_classifier(self, context, classifier_id, fields=None):
260         classifier_db = self._get_resource(context, VnffgClassifier,
261                                           classifier_id)
262         return self._make_classifier_dict(classifier_db, fields)
263
264     def get_classifiers(self, context, filters=None, fields=None):
265         return self._get_collection(context, VnffgClassifier,
266                                     self._make_classifier_dict,
267                                     filters=filters, fields=fields)
268
269     def get_nfp(self, context, nfp_id, fields=None):
270         nfp_db = self._get_resource(context, VnffgNfp, nfp_id)
271         return self._make_nfp_dict(nfp_db, fields)
272
273     def get_nfps(self, context, filters=None, fields=None):
274         return self._get_collection(context, VnffgNfp,
275                                     self._make_nfp_dict,
276                                     filters=filters, fields=fields)
277
278     def get_sfc(self, context, sfc_id, fields=None):
279         chain_db = self._get_resource(context, VnffgChain, sfc_id)
280         return self._make_chain_dict(chain_db, fields)
281
282     def get_sfcs(self, context, filters=None, fields=None):
283         return self._get_collection(context, VnffgChain,
284                                     self._make_chain_dict,
285                                     filters=filters, fields=fields)
286

```

그림 20. vnffg\_db.py 소스코드 (3/3)

## 3.2. Redundancy VNFFG module

기존의 VNFFG를 구성하는 방식은 다음과 같다. VNF Descriptor들의 순서 집합을 참조하는 VNFFG Descriptor로부터 실제 VNF 인스턴스들을 참조하는 VNFFG 인스턴스를 생성할 시, 하나의 VNF Descriptor에 대해 여러 VNF 인스턴스가 존재할 경우, 특정 알고리즘을 통해 VNF 인스턴스를 선택하게 된다. 즉, 플로우의 요구사항에 따라 VNF를 논리적으로 연결하여 그래프를 구성하는데, 단순히 논리적이고 추상적으로 구성된 그래프를 오버레이 네트워크 아래의 물리적인 네트워크에 실질적으로 렌더링하는 것이다.

현재 tacker 프로젝트에서 지원하는 VNFFG 구성은 VNFFG 생성시 NFVO 플러그인은 VNFM을 쿼리하여 각 NFP에 해당하는 VNFD에서 존재하는 사용 가능한 VNF 인스턴스를 찾고, 하나의 VNFFG가 구성된다. 현재는 VNFFG에서의 고가용성 보장을 위한 방안이 구현되어 있지 않고, 이와 같이 VNFFG를 구성하는 것은 네트워크 사업자의 다양한 요구사항을 충족시키기 어렵다. 5G/B5G에서는 데이터를 저지연으로 전달하고 전달 경로의 장애 발생시에도 데이터 손실이 없게 하기 위해 L2기반의 TSN (Time Sensitive Networks)을 L3로 확장한 저지연 및 저손실로 서비스를 제공하는 DetNet (Deterministic Network)에 대한 연구가 활발하게 진행되고 있다. DetNet에서 네트워크 서비스를 제공하기 위해 필요한 서비스 기능들을 채이닝하는 VNFFG (VNF Forwarding Graph) 기술이 필요할 것으로 보이지만, DetNet에서 VNFFG를 구성하기 위해서는 DetNet의 요구사항(즉, 저지연/저손실)에 맞게 VNFFG가 구성되어야 하는데, 현재의 VNFFG 구성 방법으로는 저손실에 대한 요구사항을 만족시키기 어렵다. 따라서, 네트워크 사업자의 요구사항이 저손실에 대한 것인 경우, Redundancy VNFFG module이 필요하다. 즉, 데이터의 손실을 방지하기 위해 tacker VNFFG feature에 redundancy VNFFG module을 추가하여 두개 이상의 disjoint한 VNFFG를 구성하고, 데이터를 구성된 VNFFG forwarding 경로로 중복 전송하는 기능이 필요하다.

이를 위해, 우선적으로 tacker 프로젝트 내부의 VNFFG feature에 redundancy module을 적용하여 저손실의 요구사항인 VNFFG에 대해 두개의 서로 다른 VNFFG를 구성할 수 있도록 한다. 즉, VNFFG를 구성할 때, Redundancy에 대한 사항을 option으로 두어, Redundancy가 option으로 선택된 경우, 같은 VNFFG Descriptor를 이용하여 두 개의 서로 다른 VNFFG 인스턴스를 생성하도록 구현하였다. 이 경우, 두개의 VNFFG에 대한 forwarding path를 disjoint하게 하기 위해, 서로 다른 compute node에 생성된 VNF instance를 참조하는 방식으로 구현이 진행되어야 한다. 또한 두개의 VNFFG로 데이터가 중복되어 전송되므로, 이를 handling 하기 위한 기능도 추가로 구현되어야 한다. 우선 서로 다른 path로 전송되는 중복된 데이터 제거를 위해, 해당 기능을 제공하는 VNFD를 구현하고 이를, 독립적인 compute node에 deploy함. VNF는 중복된 데이터에 대해서 처음 받은 데이터는 forwarding path 대로 forwarding하고, 그 다음 데이터에 대해서는 드랍시키는 방법이 고려 될 수 있다.

## 4. 결론

본 문서에서는 OPNFV/OpenDaylight SFC 및 OpenStack Tacker 기반의 VNFFG 구성 기능을 분석하였다. VNFFG 관련 소스 코드분석을 통해, 기존의 VNFFG가 어떻게 구성되는지 확인할 수 있었다. 현재 VNFFG가 구성되는 방식은 네트워크 사업자의 다양한 요구사항, 특히 저손실 서비스를 원하는 경우, 이에 대한 요구사항을 충족하기 어려울 것으로 고려된다. Redundancy VNFFG module이 저손실의 요구사항을 충족하기 위한 방안 중 하나로 고려 될 수 있으며, 이를 구현하기 위해서는 OpenStack tacker 프로젝트에서 VNFFG feature에 redundancy VNFFG module이 추가되어야 한다. 이를 위해, 우선적으로 데이터를 disjoint한 서로 다른 path로 중복 전송이 가능하도록 개발이 진행되어야 한다. 해당 모듈의 구현이 완료되고, 최종적으로 중복된 데이터를 제거하는 VNF가 구현이 되면, 저손실의 요구사항을 만족시킬 수 있는 VNFFG 구성 기능이 구현된다.



## References

- [1] OpenDaylight, [Online] Available : <https://www.opendaylight.org/>
- [2] Halpern, J., Ed. and C. Pignataro, Ed., “Service Function Chaining (SFC) Architecture,” RFC 7665, DOI 10.17487/RFC7665, October 2015, <<http://www.rfc-editor.org/info/rfc7665>>.
- [3] ETSI ISG NFV, <http://portal.etsi.org/portal/server.pt/community/NFV/367>
- [4] OPNFV, “Open Platform for Network Function Virtualization,” [Online] Available : <https://www.opnfv.org/>
- [5] Quinn, P. and U. Elzur, “Network Service Header“, draft- ietf-sfc-nsh-04, March 2016.
- [6] OpenStack, [Online] Available : <https://www.openstack.org/>

## *K-ONE* 기술 문서

- K-ONE 컨소시엄의 확인과 허가 없이 이 문서를 무단 수정하여 배포하는 것을 금지합니다.
- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.
- 본 문서와 관련된 문의 사항은 아래의 정보를 참조하시길 바랍니다.  
(Homepage: <http://opennetworking.kr/projects/k-one-collaboration-project/wiki>, E-mail: [k1@opennetworking.kr](mailto:k1@opennetworking.kr))

작성기관: K-ONE Consortium  
작성년월: 2018/12