

K-ONE 기술 문서 #4
ETSI문서에 따른
VNF Manager 구성 기능 현황 분석

Document No. K-ONE #4

Version 1.0

Date 2016-04-28

Author(s) 양현식

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.5	2016. 03. 01	양현식	
1.0	2016. 04. 28	양현식	

본 문서는 2015년도 정부(미래창조과학부)의 재원으로 정보통신
기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-15-2012, 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. B0190-15-2012, Global SDN/NFV
OpenSource Software Core Module/Function Development)

기술문서 요약

현재 Network Function Virtualization (NFV) 환경의 표준화를 주도 하고 있는 European Telecommunications Standards Institute(ETSI) 에서는 NFV의 기본적인 환경에 대한 표준화부터 다양한 인터페이스 및 서비스 제공을 위한 방법 까지 고려하여 표준화를 진행중이다. 한편 OPENSTACK 및 OPNFV 오픈 소스 단체에서는 표준화된 레퍼런스 아키텍처를 기반으로 실제적인 기능 구현을 위한 다양한 프로젝트가 진행중이다. 특히 오픈 스택은 인프라 환경의 기본이 되는 플랫폼으로써 기본 기능 제공을 위한 다양한 프로젝트들이 진행 중이다. ETSI의 레퍼런스 아키텍처는 크게 세 부분으로 분류가 가능하다. 먼저 Network Function Virtualization Infrastructure(NFVI)레벨이다. 이 부분은 NFV을 위한 Infrastructure 제공을 위한 부분으로 대다수의 OPENSTACK 프로젝트가 이에 해당한다. 따라서 기본 적인 기능들은 NFVI 레벨의 기능들을 통해 기능 구현이 가능하다. Virtual Network Function Manager(VNFM) 레벨에서는 NFVI레벨을 통해 가상화된 리소스를 바탕으로 Virtual Network Function(VNF)를 구성하고 관리하는 역할들이 주어진다. 이 부분은 실제적인 서비스와 가장 밀접한 부분으로 다양한 서비스 제공 및 VNF 관리를 위해 중요한 요소이다. 마지막으로 Network Function Virtualization Orchestration(NFVO) 레벨이다. 이 레벨에서는 전체적인 조율 및 관리에 대한 기능이 주어진다.

Openplatform for NFV (OPNFV)는 ETSI에서 정의한 레퍼런스 모델을 기반으로 통신사업자가 필요로 하는 요구 사항들을 정의하고 구현하여 반영하는 오픈소스 플랫폼이다. 따라서 많은 부분이 레퍼런스 모델을 기반으로 구현이 진행중이다. 특히 VNF Manager는 실제 Virtual Network Function(VNF) 동작시 밀접한 관련이 있는 엔티티로 전체적인 VNF의 인스턴스 라이프 사이클 관리를 한다. 각 VNF 인스턴스는 매니저에 의해 관리되며 VNF 관리를 위해 필요한 많은 기능들이 제공된다. 즉, VNF매니저는 실제서비스를 제공하는 VNF 의 어플리케이션 구성 및 관련 기능들 제공을 위해 핵심이 되는 서비스이다. 따라서 본 기술문서에서의 VM/VNF의 관리를 위해 개발되고 있는 VNFM 관련 표준화 현황을 분석하고 대표적인 오픈프로젝트를 통해 개발상황을 분석 하고자 한다.

Contents

K-ONE #4. ETSI문서에 따른 VNF Manager 구성 기능 현황 분석

1. Introduction	4
2. ETSI에서의 NFV용 프레임 워크 정의	4
3. ETSI 구조에서의 NFV용 VNFM의 정의	6
4. Tacker Overview	7
5. Tacker Function Analysis	9
5.1. 템플릿 기반의 VNF 생성 및 관리	9
5.2. 모니터링 드라이버	9
5.3. Service Function Chaining (SFC)	9
5.4. 추가 기능	10
6. Tacker implementation	12
6.1. Tacker Installation	12
6.2. Monitoring driver Function	14
6.3. Service Function Chaining integration	16
6.4. Service Function Chaining implementation	19
7. 결론	21
8. Reference	22

그림 목차

<그림 차례>

그림 1	5
그림 2	7
그림 3	8
그림 4	9
그림 5	10
그림 6	12
그림 7	13
그림 8	13
그림 9	13
그림 10	14
그림 11	14
그림 12	15
그림 13	15
그림 14	16
그림 15	17
그림 16	17
그림 17	18
그림 18	19
그림 19	19
그림 20	20
그림 21	20

1. Introduction

현재 Network Function Virtualization(NFV) 표준화는 ETSI를 중심으로 진행되고 있다. 2010년 10월 조직이 구성되어 2013년부터 Phase 1을 시작으로 현재 Phase 2 까지 표준화가 진행 중이다. Phase1 에서는 NFV 개념 정립, 유즈케이스, 기본 요구사항과 같은 구조를 중심으로 진행이 되었으며 주로 인프라 및 인터페이스에 대한 표준화가 중심이었다. 그러나 Phase 2 에서는 상호 호환성을 중심으로 한 규격설정 및 네트워크 관리 기능의 자동화 및 간소화, NFV 기능의 오류, 동작 모니터링, 오류 방지 기능이 포함되어 있는 규격OpenPlatform NFV (OPNFV)에서 진행되고 있는 사항들에 대한 요구사항 반영, NFV 구조에서의 보안관련 규격 개발등이 진행되고 있다. 이러한 구조는 현재 오픈소스 프로젝트인 Openstack 및 OPNFV와 밀접한 관련성을 가지고 있으며 많은 개발자들이 표준화된 ETSI의 레퍼런스 아키텍처를 기반으로 실제 코드 구현을 진행 되었으며, 인프라 모델부터 VNF Manager, NFVO 부분까지 많은 부분이 실제 모델에 반영되어 설계가 진행 중이다. 그중 VNF Manager는 실제 Virtual Network Function(VNF) 동작시 밀접한 관련이 있는 엔티티로 전체적인 VNF의 인스턴스 라이프 사이클 관리를 한다. 각 VNF 인스턴스는 매니저에 의해 관리되며 VNF 관리를 위해 필요한 많은 기능들이 제공된다. 즉, VNF매니저는 실제서비스를 제공하는 VNF 의 어플리케이션 구성 및 관련 기능들 제공을 위해 핵심이 되는 서비스이다. 따라서 본 기술문서에서의 VM/VNF의 관리를 위해 개발되고 있는 VNFM 관련 표준화 현황을 분석하고 대표적인 오픈프로젝트를 통해 개발상황을 분석 하고자 한다.

2. ETSI에서의 NFV용 프레임 워크 정의

NFV Architectural Framework는 기능 확장이 용이하도록 운영자 네트워크의 가상화로 발생할 새로운 기능과 인터페이스를 중심으로 설계가 진행되고 있다. 그림1은 ETSI에서 진행하고 있는 프레임워크를 보여준다. 가장 쉬운 예로 Firewall, DPI와 같은 기능들이 있다. VNF는 위에서 언급한 내용 이외에도 다양한 기능으로 동작 가능하며 물리적 네트워크 기능의 특성과 인터페이스는 동일한 형태를 가진다. EM은 Element Management 의 약자로 VNF 관련 설정 및 모니터링 기능을 하고 그에 대한 로그를 가지고 있는 엔티티이다. EM을 통해서 VNF의 결합 감지, 보안, 성능 측정, 설정등의 역할을 하게 된다. OSS/BSS는 네트워크 사업자의 서비스 운영 및 사업 지원 시스템을 의미한다. VIM 은 Virtualised Infrastructure Manager의 약자로 VNF 가 사용하는 물리적 자원인 컴퓨팅, 스토리지, 네트워크 등을 관리하고 제어 하는 부분을 의미한다. 물리적 자원을 바탕으로 가상화된 자원들을 배치하고 관리하는 역할을 하며, VIM을 통해 NFVI의 가시화를 통한 관리 기능 및 NFVI로 부터의 결합 정보 및 자원 상태 수집을 통한

NFVI 오류 감시 및 관리기능도 수행한다. VNFM은 VNF Manager의 약어로서 VNF의 라이프 사이클 관리 및 VNF 간 상호 운용을 위한 관리 및 제어기능을 가진다.

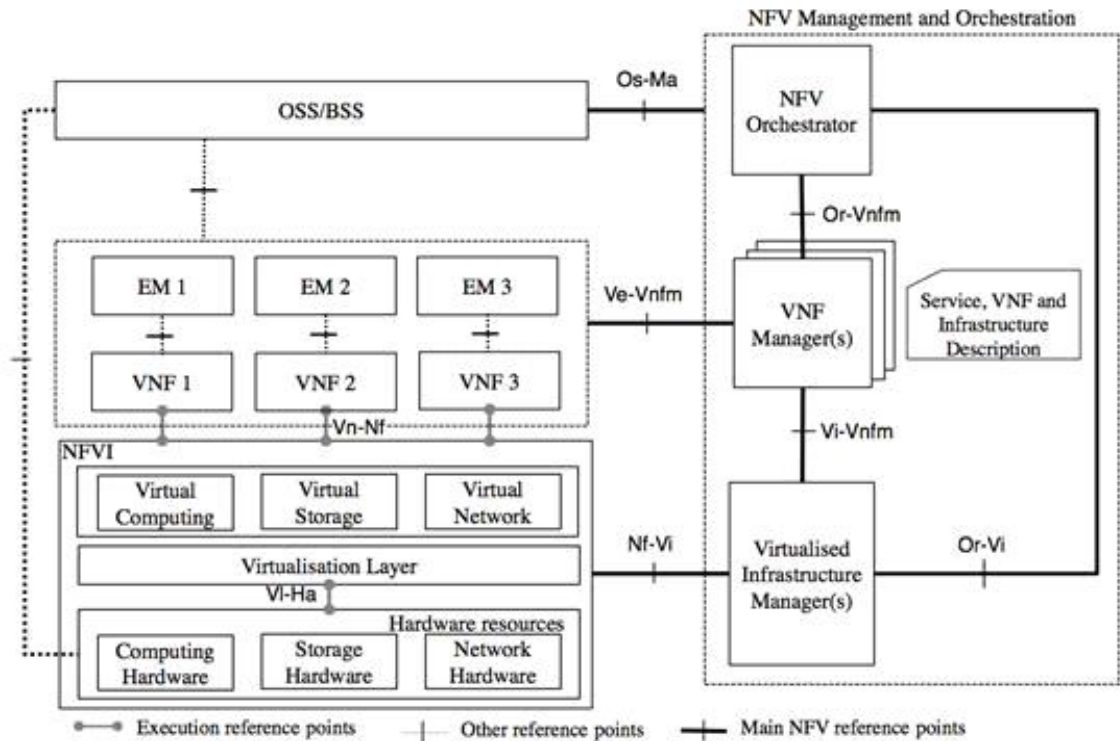


그림 1 . NFV reference architecture framework

위 그림을 보면 크게 기능 블록과 레퍼런스 포인트로 구분 가능하다. 먼저 기능 블록은 VNF, EM, NFVI, VIM, NFV Orchestrator, VNF Manager, Service, VNF and Infrastructure Description, OSS/BSS등으로 구성되어 있으며, 레퍼런스 포인트로는 VI-ha / Vn-Nf / Or-Vnfm / Vi-Vnfm / Or-Vi / Nf-Vi / Os-Ma / Ve-Vnfm 등으로 구성된다. NFVI(NFV Infrastructure)는 VNF를 배치하는 시점부터 VNF에 대한 관리와 실행을 위한 모든 물리적 논리적 요소를 말한다. 즉 물리적인 요소들을 가지고 가상화 단계를 거쳐 자원을 제공하는 개체를 말하는 것이다. VNF 관점에서는 결국 가상화된 자원을 사용하기 때문에 물리적인 요소와 논리적인 요소를 나눌 필요는 없다. NFVI를 구성하는 요소는 크게 두가지이다. 첫 번째로는 하드웨어 자원이다. 즉 가상화 계층을 통해서 VNF가 사용하게 될 실제 물리적 자원을 말한다. 둘째로는 가상화 계층이다. 가상화 계층을 통해 물리 자원을 추상화 시켜서 제약없이 자원을 유동적으로 사용할 수 있게 하는 역할을 담당한다. VNF는 Virtualised Network Function의 약자로 가상화된 네트워크 기능을 말한다. NFVO는 NFV Orchestrator의 약어로, 자원과 서비스에 대한 오케스트레이션 기능을 가진다.

자원에 대해서는 VNF 관련 자원 사용 정보 수집등을 관리하며 서비스 측면에서는 라이프사이클 및 자동화를 관리하는 역할을 한다. NFV 레퍼런스 포인트는 앞서 이야기한 구성 요소들간의 인터페이스를 의미하며 8개의 인터페이스로 구성되어 있다.

Vi-Ha는 가상화 계층과 하드웨어 자원간의 사용되는 인터페이스로 VNF를 위한 자원을 제공하는 역할을 하며 하드웨어에 대한 상태정보를 수집하는 역할을 한다. Vn-Nf는 VNF와 NFV Infrastructure 사이에서 사용되는 인터페이스로 NFVI가 VNF에게 자원을 제공하기 위한 용도로 사용된다. Or-Vnfm은 NFV Orchestrator와 VNF 매니저간의 사용되는 인터페이스로 VNFM이 NFVO에게 자원에 관련된 요청(사용자 인증, 유효성, 예약, 할당)을 할 때 혹은 NFVO가 VNFM에게 설정 정보를 보낼 때 사용된다. 또한 네트워크 서비스 라이프 사이클 관리를 위해서 VNF 정보를 수집할때도 사용된다. Vi-Vnfm은 VIM과 VNFM간에 사용되는 인터페이스로서, VNFM이 VIM에게 자원 할당을 요청하거나 가상화된 하드웨어 자원 설정 및 상태정보 교환시 사용된다. OR-Vi는 NFVO가 VIM에게 자원 예약을 하거나 자원에 대한 할당을 요청할 때 사용되는 인터페이스이다. 이 인터페이스 역시 가상화된 하드웨어 자원설정 및 상태 정보 교환할 때 사용된다. Nf-Vi는 VIM과 NFVI 사이에 연결에 정의된 인터페이스로서 자원할당 요청에 응답하거나 자원상태 정보를 전달하거나 하드웨어 자원을 설정하는 경우 사용되는 인터페이스이다. Ve-Vnfm은 VNF/EM과 VNF 매니저 사이에 정의된 인터페이스로서 VNF 라이프 사이클 관리에 관련된 요청이 필요하거나, 설정 및 이벤트 관련 정보를 전달하거나, 네트워크 서비스 라이프 사이클 관리를 위한 상태 정보를 교환하는 경우에 사용된다. Os-Ma는 OSS/BSS와 MFV Orchestrator 사이에 정의된 인터페이스로서 네트워크 서비스 라이프 사이클 관레이 관련된 요청이나 VNF 라이프 사이클 관리에 관련된 요청, NFV에 관련된 상태 정보 전달, 관리 정책 전달, 데이터 분석 교환, NFV에 관련된 과금 및 사용량 기록 전달, NFVI 기능 및 재고 관련 정보 교환등에 사용된다.[1]

3. ETSI 구조에서의 NFV용 VNFM의 정의

VNF 매니저는 VNF의 인스턴스의 라이프 사이클 관리를 위한 엔티티이다. 각 풀 인스턴스는 VNF 매니저에 의해서 관리 된다. VNF 매니저는 최소 한 개 혹은 여러개의 VNF인스턴스를 관리하며 이때 VNF 인스턴스의 종류는 같은 타입이거나 혹은 다른 타입의 인스턴스 일 수 있다. VNF 매니저는 기본적으로 모든 유형의 VNF에 적용되는 일반적인 공통기능을 제공하는데 주요 기능은 다음과 같다. VNF 인스턴스화, VNF 인스턴스화 가능성 검사, VNF 인스턴스 소프트웨어 업데이트 및 업그레이드, VNF 인스턴스 변경, VNF 인스턴스 Scaling out/in, up/down,

풀 인스턴스와 관련된 성능 측정 및 이벤트 관리, VNF 인스턴스 지원 및 자동 복구, VNF 인스턴스 종료, VNF Lifecycle Management 변경 통지, VNF 인스턴스의 무결성 관리, VIM과 EM사이의 configuration과 Event Reporting 에 대한 조정 및 적용등이다. 각 VNF의 배치와 동작은 VNFD(Virtualised Network Function Descriptor(VNFD)) 템플릿을 이용하며 VNFD는 VNF 카탈로그에 저장되어 있다. NFV-MANO는 VNFD를 사용해서 VNF 인스턴스를 생성하고 이러한 인스턴스의 라이프 사이클을 관리한다.

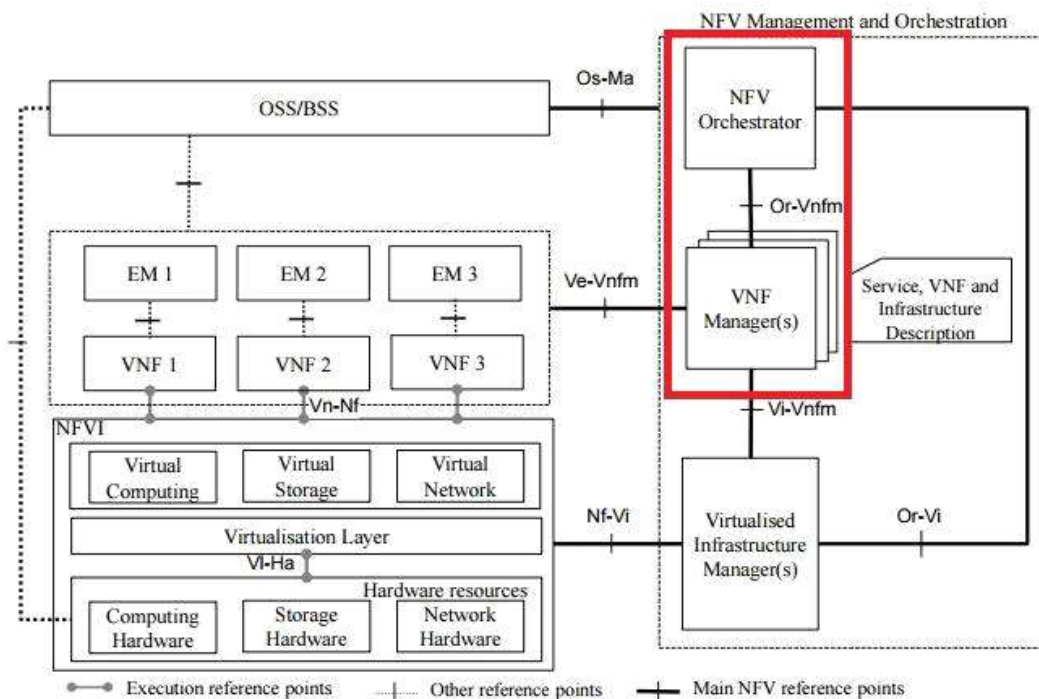


그림 2 . ETSI 레퍼런스 구조에서의 tacker scope

4. Tacker Overview

Tacker는 오픈스택 프로젝트로써 현재 구현되고 있는 대표적인 VNFM 이다.[2] 그림2와같이 현재의 모든 구조들은 VNFM과 동작시 필요한 VNFO부분을 중심으로 기능들이 구현이 되고 있다. NFVO는 VNFM 동작시 많은 부분을 함께 동작하기 때문에 VNFM과 주로 함께 논의가 된다. 앞서 이야기한 NFVM이 가지는 기능들은 Tacker에서도 고려하고 있다. 특히 VNFM의 경우에는 벤더의 특성에 따라서 구성되는 경우가 많을 것으로 예상하고 있지만, Tacker 에서는 ETSI에서 논의되는 표준을 따라 구성 하기 위해 노력하고 있다. VNFM은 VNF 인스턴스의 생성 및 제거 EMS(element management System)을 이용한 구성, VNF 모니터링, 셀프 힐링 및 오토힐링, VNF 이미지 업데이트 관리. VNF 지원등의 기능을 고려

하고 있으며, NFVO에서는 포워딩그래프와 VNF들의 수집을 활용한 네트워크 서비스 오케스트레이션, 다중 VIM환경에서의 VNF 배치를 위한 VNFM, 리소스 체크 및 할당, PNF(Physical Network Function) 와 VNF의 확장, SDN 컨트롤러 혹은 SFC API를 활용한 VNF 포워딩 그래프 관리등이 고려되고 있다. Tacker 에서는 ETSI에서 논의 되고 있는 기본 형태를 유지하며 제안하고 있는데 이 때문에 다음과 같은 이점들을 얻을 수 가 있다. 첫째로는 Vendor Lock-in이다. 기본적으로 Manager는 벤더에 종속적으로 구현이 되는경우가 많은데 Tacker의 경우에는 이 부분을 피할 수 있다. 그렇게 때문에 멀티 벤더로부터 VNF를 배치하는 것도 가능하다.

또한 NSD(Network Service Descriptor) 와 VNFD 템플릿 사용을 통한 데모를 통해 표준화된 형태의 실제 동작을 구현 함으로써 표준화에도 많은 부분 영향을 미치고 있다.

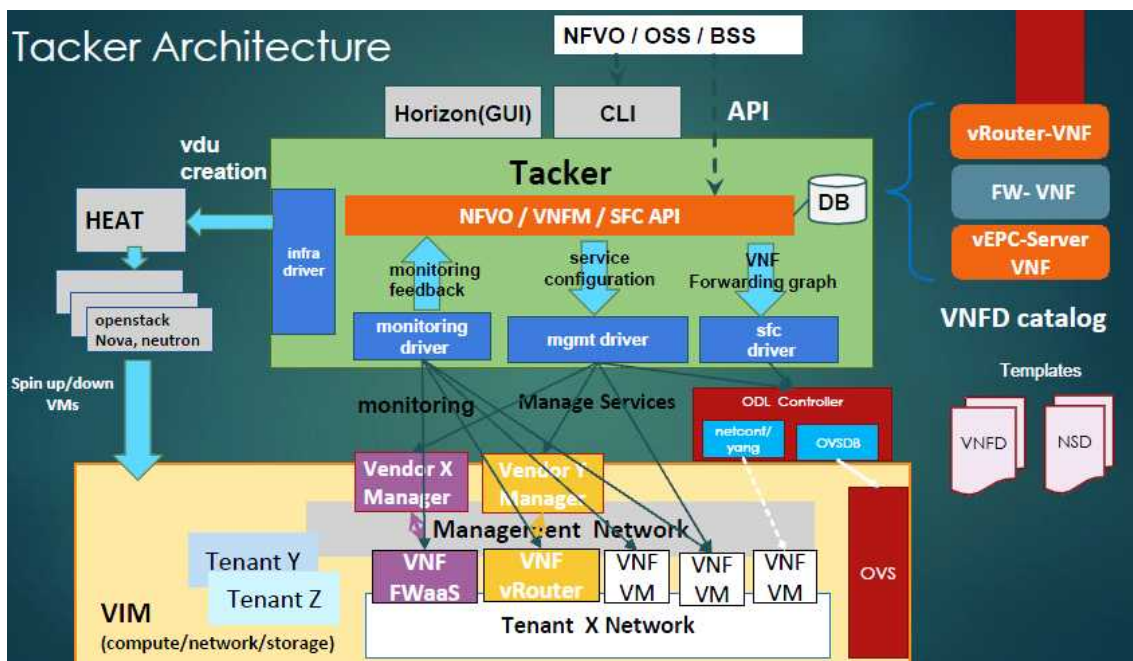


그림 3 . Tacker Architecture

그림3과 같이 tacker는 VIM부분의 VNF 생성을 위한 부분과 관리를 위한 드라이버 모니터링을 위한 드라이버 등이 존재하며, 위로는 Horizon 이나 NFVO와의 연동을 위한 부분이 정의 되어 있다. 현재 tacker가 가지고 있는 기본 기능들을 살펴보면 다음과 같다.

5. Tacker Function Analysis

5.1. 템플릿 기반의 VNF 생성 및 관리

Tacker VNFM에서는 정의된 템플릿을 통해 VNFD를 생성하고 이를 통해 VNF를 생성 가능하다. 정의된 VNFD를 등록하고 나면 오픈스택의 호라이즌을 통해 같은 성격을 가지는 VNF를 쉽게 생성 할 수 있다. 템플릿은 VNF의 생성뿐만 아니라 생성이후에 필요한 관리 기능에 대한 정의 및 모니터링 이후의 액션에 대한 정의도 포함 할 수 있어 유용하며, 앞으로 표준으로 정의된 템플릿을 사용되어질 예정이다. VNFD에는 기본적으로 VNFD의 ID 와 기본정보, vdu 정보 virtual link 정보, 모니터링 파라미터, 오토 스케일링 관련 파라미터 등이 포함된다. 현재 tacker에서는 yaml 파일의 형태를 사용하고 있지만 추후 TOSCA 템플릿을 기반으로 전체적인 수정이 진행될 것이다. [3]

5.2. 모니터링 드라이버

먼저 모니터링 기능이다. tacker는 VNF의 상태 관리를 위한 모니터링 기능을 제공한다. 현재까지 공식적으로는 ping과 HTTPping 두가지를 지원한다. 이 모니터링 드라이버는 VNF가 생성되는 경우 VNF가 없어질 때 까지 계속 동작을 한다. ping을 보내면서 실제로 VNF가 동작하고 있는지 꺼졌는지에 대한 판단이 가능하다. 현재 드라이버는 VNF 가 생성되는 경우 사용된 템플릿의 내용을 기반으로 동작하게 되며 VNF가 동작하는 동안 정해진 주기에 따라 모니터링 드라이버를 동작시켜 VNF 에 대한 검사를 실시하게 된다. 이와 같은 값들은 템플릿을 통해 정의하고 변경가능하다. 모니터링 드라이버를 통해 더 많은 부분을 모니터링 할 수 있지만 현재까지 공식적으로는 두가지 기능을 제공하고 있으며 추가적인 부분에 대해서는 사용자가 원하는 대로 구성이 가능하다.

5.3. Service Function Chaining (SFC)

VNFM는 전반적으로 VNF의 생성 및 관리를 책임지는 만큼 VNF간의 Service function chaining 에 대한 부분도 고려하고 있다. 특히 tacker 에서는 형성된 VNF에 대한 정보를 기반으로 SFC를 형성하는 부분에 대한 명령을 내리기 위한 드라이버도 고려하고 있다. 이는 SFC를 지원하는 여러 종류의 SDN컨트롤러와의 연동을 통해 지원 될 수 있다. 아래 그림과 같이 SFC을 지원하기 위해 Neutron 에서 SDN컨트롤러와 연동하는 구조 또한 하나의 방법으로 논의가 진행 되고 있다. [4]

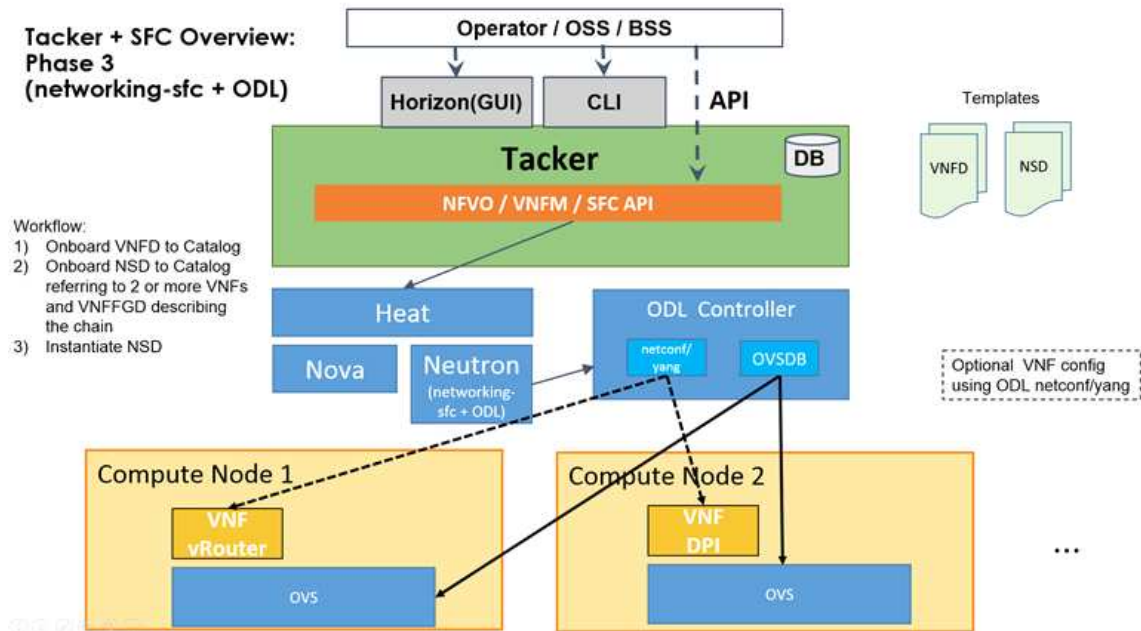


그림 4 . Tacker Architecture for SFC

5.4. 추가 기능

VNFM로서의 Tacker는 현재 제공하는 기능을 바탕으로 다양한 추가적인 기능에 대한 블루프린트를 가지고 있다. 먼저 멀티 환경에서의 VIM 에 대한 지원방안을 고려하고 있으며 앞서 이야기한 SFC 에 대한 확장에 대해서도 현재 논의중이다. 또한 VNF의 배치에 대한 관리부분 및 오토 스케일링에 대한 부분도 함께 논의가 진행되고 있다.

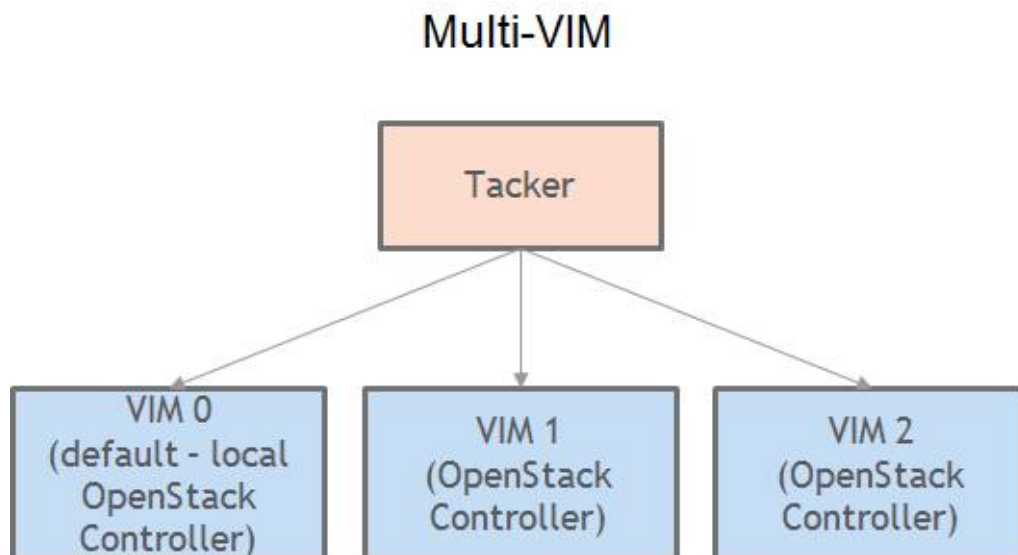


그림 5 . Multi VIM Support Architecture

위 그림과 같이 하나의 오퍼레이터가 멀티 오픈스택의 Virtual Infrastructure Manager(VIM)를 사용하는 경우를 고려한 구조를 현재 고려 중이다. SFC 에 대해서는 SFC 을 위한 API 고려하고 있으며 이를 통해 flow classification rules을 정의하고 flow chain을 설정하는 것을 계획 중이다. 이 외에도 SR-IOV, vhost, NUMA 등의 부가적인 사항도 함께 고려가 되고 있다.

6. Tacker implementation

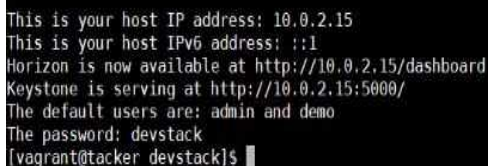
6.1. Tacker Installation

Tacker는 Openstack의 VNFM에 속하는 부분으로 현재 설치 하는 방법은 크게 두가지로 분류 할 수 있다. 먼저 오픈스택과 함께 전체 인프라 및 VNFM에 대하여 설치가 한번에 동작하는 devstack을 이용한 설치 방법 혹은 Openstack을 컴포넌트별로 설치하고 VNFM을 추가적으로 설치하는 방법이 있다. Tacker는 현재 Openstack에서 제공하는 호라이즌에 추가적인 기능을 구현하기 위해 추가적인 메뉴를 생성하였다. 따라서 설치가 성공적으로 끝나고 나면 아래와 같은 메뉴를 호라이즌에서 확인 할 수 있다.

Devstack을 통해 Tacker를 설치하는 경우는 다음과 같다. 먼저 devstack을 다운 받은 다음 네트워크 설정을 위해 local.conf를 작성한다. 이때 마지막 부분에 tacker를 설치 하기 위한 플러그인 형태의 아래와 같은 문구를 추가한다.

```
enable_plugin tacker https://git.openstack.org/openstack/tacker stable/liberty
```

그리고 파일을 저장후, stack.sh 파일을 실행함으로서 설치가 완료된다. 설치 도중 발생하는 많은 오류의원인은 네트워크 환경의 문제 일 수 있으므로, local.conf 파일을 작성시 네트워크 환경을 고려하여 작성 해야 한다. 설치가 완료 되고 나면 오픈스택의 호라이즌 화면을 통해서 설치가 된 것을 확인 할 수가 있다. tacker를 devstack으로 설치하는 경우, 머신이 재부팅되면 전체 환경의 오류가 발생한다. 따라서 재부팅을 하고 난 이후에는 rejoin.sh 라는 파일을 실행시키면 정상적으로 동작하는 것을 확인 할 수가 있다. 수동으로 설치하는 경우에는 기본적인 엔티티와 함께 테커 소스를 다운받아 설치하면 된다. [5]



```
This is your host IP address: 10.0.2.15
This is your host IPv6 address: ::1
Horizon is now available at http://10.0.2.15/dashboard
Keystone is serving at http://10.0.2.15:5000/
The default users are: admin and demo
The password: devstack
[vagrant@tacker devstack]$
```

그림 6 . Horizon dashboard with tacker

설치가 완료 되고 나면 위 화면처럼 오픈스택의 호라이즌 주소가 나타난다. 이를 통해 오픈스택 환경 확인 및 설정이 가능하다. 설치가 완료 되고 나면 아래와 같은 환경을 확인 할 수가 있다.

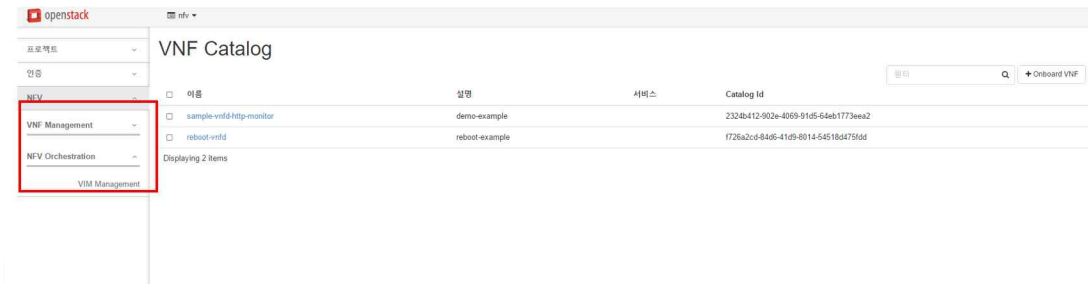


그림 7 . Horizon dashboard with tacker

VNF Management 부분은 위에서 언급했던 VNFD를 등록하고 관리하는 부분으로 VNFD를 등록하면 VNF Catalog 화면에서 확인하고 수정 할 수 있다. 이를 통해 파라미터 값 및 그 외에 값들도 수정 및 추가 가능하다. 이 부분은 Command Line을 통해서도 추가 가능하다. 먼저 tacker vnfd-list를 통해 현재 등록되어 있는 Vnfd-list 에 대한 정보를 확인 가능하다. 그림 7과 같이 리스트를 통해 VNFD의 이름과 ID 값을 확인 할 수가 있다.

```
root@controller:/usr/local/lib/python2.7/dist-packages/tacker/vm/monitor_drivers/port# ^C
root@controller:/usr/local/lib/python2.7/dist-packages/tacker/vm/monitor_drivers/port# ^C
root@controller:/usr/local/lib/python2.7/dist-packages/tacker/vm/monitor_drivers/port# tacker vnfd-list
+-----+-----+-----+-----+-----+
| id | name | description | infra_driver | mgmt_driver |
+-----+-----+-----+-----+-----+
| 2324b412-902e-4069-91d5-64eb1773eea2 | sample-vnfd-http-monitor | demo-example | heat | noop |
| f726a2cd-84d6-41d9-8014-54518d475fdd | reboot-vnfd | reboot-example | heat | noop |
+-----+-----+-----+-----+-----+
```

그림 8 . VNFD List

아래의 명령어를 통해 VNFD를 등록 할 수 있다.

tacker vnfd-create --vnfd-file sample-vnfd-http-monitor.yaml

```
Created a new vnfd:
+-----+-----+
| Field | Value |
+-----+-----+
| description | firewall-example |
| id | b277b857-c00f-4fa4-a0cb-5ef818cefd2 |
| infra_driver | heat |
| mgmt_driver | noop |
| name | test-vnfd |
| service_types | {"service_type": "firewall", "id": "77f8b3f0-91a6-429d-8f47-32d9af84f74a"} |
| | {"service_type": "vnfd", "id": "fa3a34e7-9032-4f4d-ad01-b99a15cdd728"} |
| tenant_id | 8502b27cb8c249f5bf92752ed195778c |
+-----+-----+
```

그림 9 . VNFD registration

등록된 VNFD를 이용하여 아래의 명령어를 통해 원하는 VNF를 생성할 수 있다.

tacker vnf-create --name testVNF1 --vnfd-name test-vnfd

Field	Value
description	firewall-example
id	67c6c726-a061-478e-8a9a-11b114ee59ed
instance_id	852a5d6f-0332-4db2-8415-6d27b90a15d2
mgmt_url	
name	testVNF1
status	PENDING_CREATE
tenant_id	8502b27Cb8c249f5bf92752ed195778c
vnfd_id	b277b857-c00f-4fa4-a0cb-5ef818cefdc2

그림 10 . VNF Creation

6.2. Monitoring driver Function

현재의 Tacker 환경에서는 두가지의 모니터링 기능을 제공한다. 먼저 첫 번째는 Ping 드라이버이다. Ping 드라이버는 VNF의 인터페이스에게 핑 메시지를 보내서 리플라이를 확인하면서 VNF의 동작에 대한 상황을 확인하는 형태이다. 만약 VNF로부터 응답 메시지를 받지 못하는 경우에는 특정 액션을 정의하여 실행되게 한다. Ping 드라이버는 VNF 안에 동작하는 어플리케이션과 무관하게 VNF의 인터페이스를 기반으로 동작한다.

```

placement_policy:
  availability_zone: nova

auto-scaling: noop
monitoring_policy:
  ping:
    monitoring_params:
      monitoring_delay: 45
      count: 3
      interval: .5
      timeout: 2
    actions:
      failure: respawn

config:
  param0: key0
  param1: key1

```

그림 11 . ping monitoring VNFD

그림 11과 같이 모니터링을 위해 필요한 파라미터들을 Ping 이라는 변수 아래 정의하였다. VNF가 생성되고 동작되는 시점부터 모니터링 주기를 정해주는 변수와 인터벌 변수 그리고 타임아웃 변수등이 이에 해당 한다. 최종적인 모니터링 과정 이후에는 액션 필드에 있는 액션 형태에 따라 다음 과정을 수행하게 된다. 현재 Tacker에서는 VNF가 발생하는 경우 자동적으로 VNF를 재 생성하는 형태의 액션인 Respawn을 정의하여 동작하게 하였다.

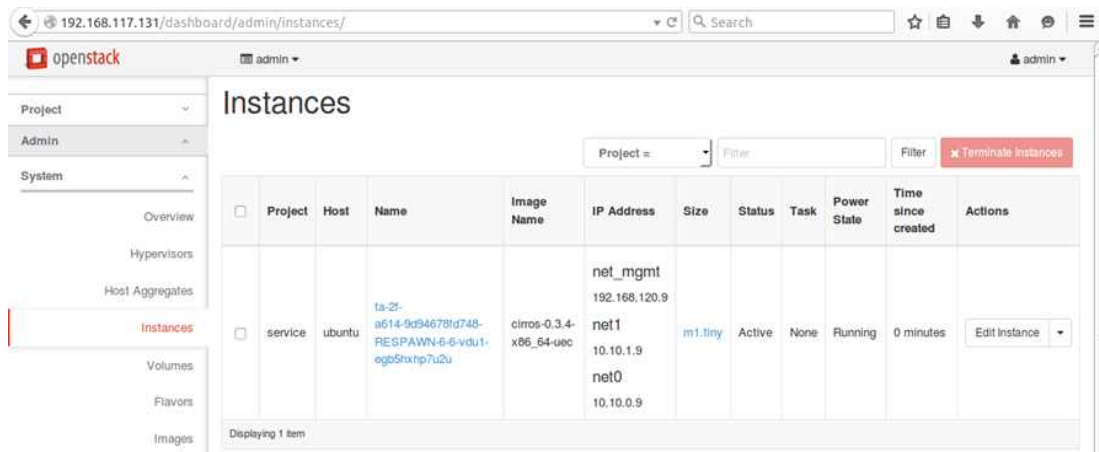


그림 12 . example of monitoring driver

그림 12과 같이 VNF가 재생성되게 되면 Name에 있는 VNF의 이름이 변경되고 재생성된 횟수에 따라 숫자가 증가한다. 이때 VNF의 내부 정보는 보존되지 않으며 IP 주소 또한 변경된다.

또 하나의 모니터링 드라이버는 HTTP ping 드라이버 이다. HTTP-ping 드라이버도 앞서 확인한 ping 드라이버와 유사한 형태로 동작한다. 하지만 파라미터 값은 기존의 드라이버와는 다르다. 그림 13와 같이 HTTP-ping 드라이버 경우에는 port 정보를 통해 모니터링 드라이버가 수행되게 된다.

```
placement_policy:
  availability_zone: nova

auto-scaling: noop
monitoring_policy:
  http_ping:
    monitoring_params:
      retry: 5
      timeout: 10
      port: 8000
    actions:
      failure: respawn

config:
  param0: key0
  param1: key1
```

그림 13 . example of HTTP-ping driver

6.3. Service Function Chaining integration

VNFM는 전반적으로 VNF의 생성 및 관리를 책임지는 만큼 VNF 간의 SFC 형성에 대한 부분도 고려하고 있다. Openstack 환경에서 SFC를 생성하는 방법은 크게 두가지로 나눌 수 있다. 첫 번째는 Openstack 환경에서 직접적으로 SFC를 생성하는 방법이다. 이를 위해 현재 Neutron networking 프로젝트에서 이를 위해 여러 가지 방안에 대해 논의중이다. 두 번째는 다른 SDN 컨트롤러와의 연동을 통한 SFC 형성이다. Tacker의 ODL에 대한 언급을 보면 짐작 가능하듯이 SFC를 형성하기 위한 방법으로 다른 컨트롤러를 이용한 방법도 논의가 되고 있다. 특히, 이전 Tacker 버전에서 ODL과의 데모를 보여줌으로써 전체적인 방법에 대한 아웃라인을 확인 할 수 있었다. 이 데모에서는 VNF들이 형성된 후 형성된 정보를 기반으로 SFC 형성에 대한 명령을 SDN 컨트롤러로 전달하면 컨트롤러에 의해 SFC에 필요한 엔티티들이 형성되는 형태로 구성되어 있었다. 특히 Tacker에서의 SFC에 대한 방안은 위에서 언급한 방안을 고려하여 크게 4가지 정도의 방법이 논의되고 있다. [6][7]

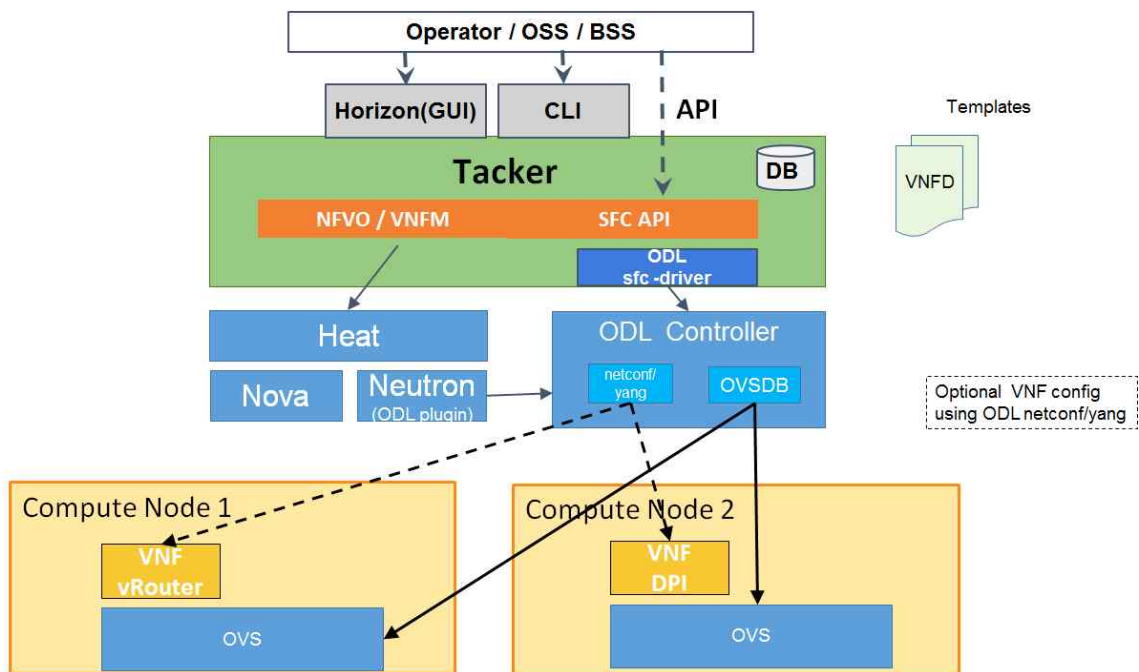


그림 14 . Tacker + SFC Overview(ODL API)

그림 14는 Tacker에서 SFC 구성을 위한 방법으로 ODL API를 통한 직접적인 명령 전달 방법을 고려한 케이스 이다. Tacker에서는 ODL API를 통해서 ODL 컨트롤러에게 직접적인 명령을 전달 하게 된다. 이 명령을 바탕으로 ODL은 OVS를 이용하여 SFC에 필요한 엔티티들을 구성하게 된다.

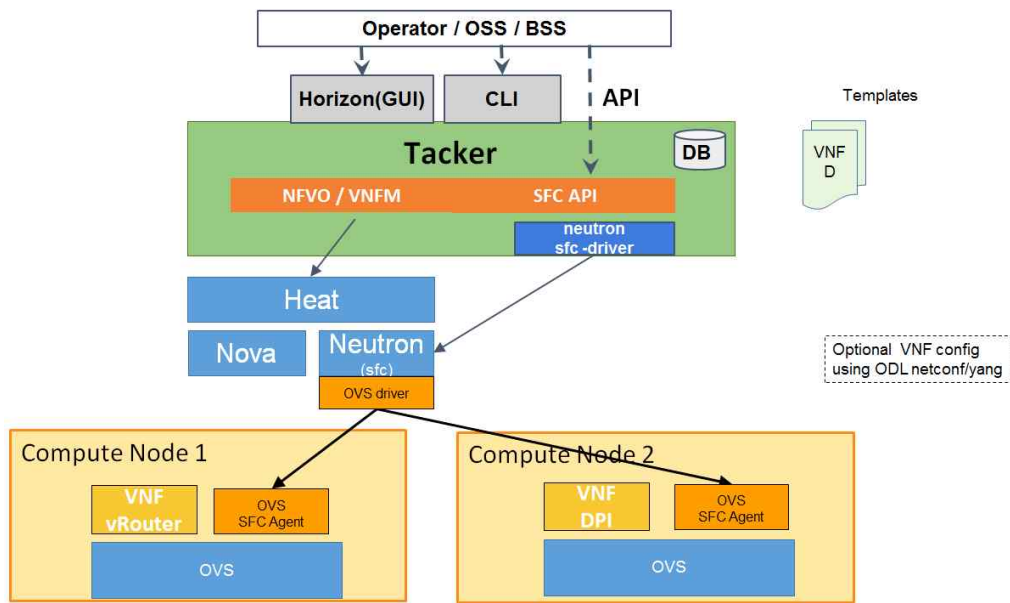


그림 15 . Tacker + SFC Overview(Neutron-sfc-API)

그림15는 다른 SDN 컨트롤러를 사용하지 않고 Neutron SFC-driver를 통해 Neutron에게 SFC 관련 정보를 전달하여 SFC를 생성하는 경우이다. 이 경우 각각의 컴퓨터 노드에는 OVS SFC agent가 존재하며 이는 Neutron의 OVS driver와의 통신을 통해 SFC 관련 정보들을 전달 받게 된다.

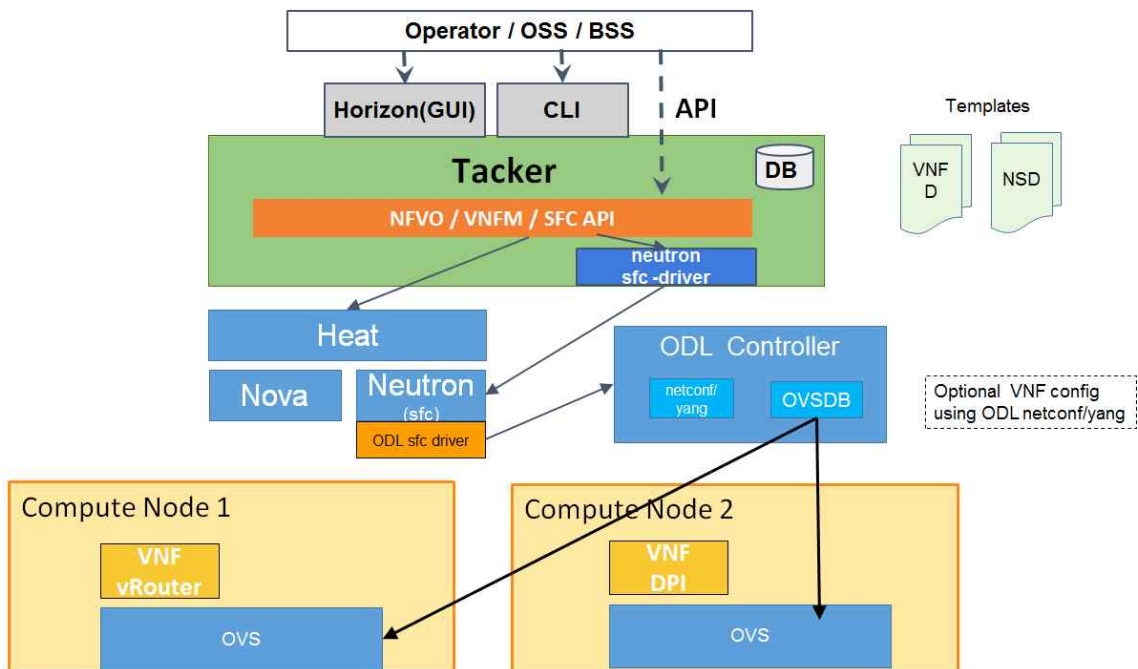


그림 16 . Tacker + SFC Overview(Neutron-sfc+ODL)

그림 16은 Tacker에서 Neutron-sfc-driver를 이용하여 SFC 구성에 대한 정보를 전달하는 방식은 그림15와 동일하지만 그 이후 SFC 구성을 위해 OVS agent를 사용하지않고 기존의 사용되는 SDN 컨트롤러와의 연동을 위한 구조이다. ODL sfc driver 와 ODL과의 통신을 통해 받은 정보를 기반으로 ODL이 SFC를 구성한다.

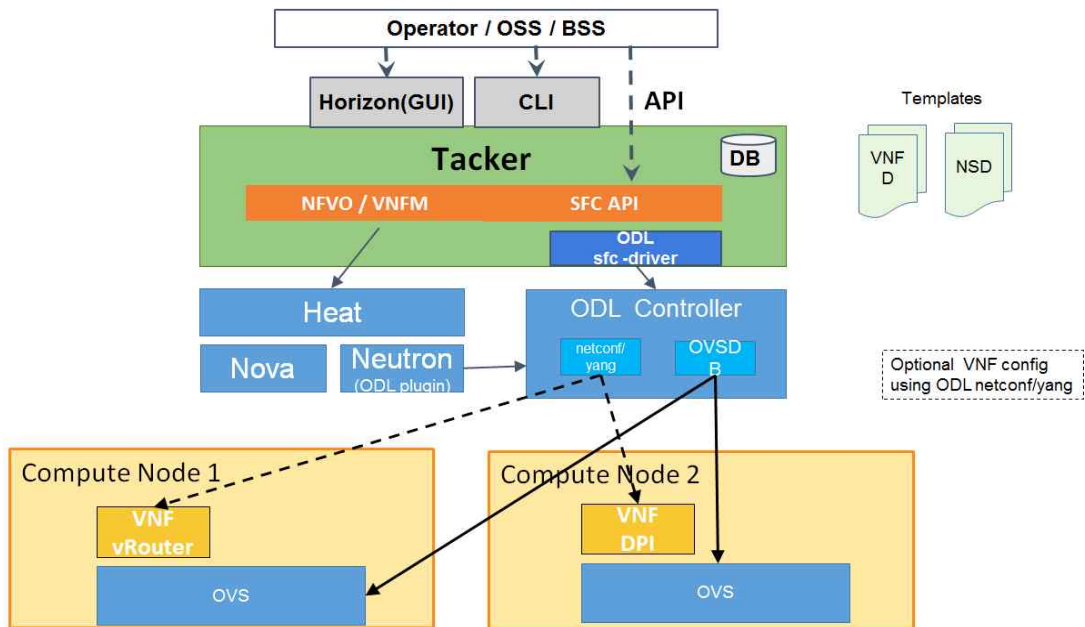


그림 17 . VNFFG+ NSD model

그림 17에서는 ODL-sfc driver를 생성하여 ODL에게 NSD 및 VNFFGD에서의 정보를 기반으로 Service Function Chaining을 구성하는 형태의 모델이다. 이때 Neutron 의 ODL Plugin 을 통해서 VNF 관련 Configuration 정보들도 전달 하게 된다.

6.4. Service Function Chaining implementation

그림18와 같이 tacker vnf-list를 통해 형성된 vnf 에 대한 리스트를 확인 할 수 있다. 처음 vnf 가 생성되고 나면 Pending 상태를 유지 하지만 시간이 흐르면 active 상태로 변경되는 것을 확인 할 수 있다.

```
[vagrant@tacker devstack]$ tacker vnf-list
```

id	name	description	mgmt_url	status
67c6c726-a061-478e-8a9a-11b114ee59ed	testVNF1	firewall-example		PENDING_CREATE
a8733c9a-9eef-421f-8652-21d73929e6f3	testVNF2	firewall-example		PENDING_CREATE

```
[vagrant@tacker devstack]$ tacker vnf-list
```

id	name	description	mgmt_url	status
67c6c726-a061-478e-8a9a-11b114ee59ed	testVNF1	firewall-example	{"vdu1": "11.0.0.3"}	ACTIVE
a8733c9a-9eef-421f-8652-21d73929e6f3	testVNF2	firewall-example	{"vdu1": "11.0.0.4"}	ACTIVE

그림 18 . tacker VNF-list

sfc-creation 이라는 명령어를 통해 특정 VNF 간의 체인을 구성할 수 있다. 명령어는 다음과 같이 수행된다. name 의 인자는 체인의 이름이 들어가고 chain 다음에는 사용되어지는 VNF의 이름이 삽입된다. 마지막으로 chain의 특징이 symmetrical 인지 아닌지에 대한 정보를 넣고 나면 이를 기반으로 체인이 형성되게 된다.

sfc-create --name *mychain* --chain *testVNF2, testVNF1* --symmetrical True

```
[vagrant@tacker devstack]$ tacker sfc-create --name mychain --chain testVNF2,testVNF1 --symmetrical True
```

Created a new sfc:

Field	Value
attributes	{}
chain	a8733c9a-9eef-421f-8652-21d73929e6f3 67c6c726-a061-478e-8a9a-11b114ee59ed
description	
id	4ac5267e-b558-4dde-87ed-5a138926700b
infra_driver	opendaylight
instance_id	Path-mychain-Path-20
name	mychain
status	PENDING_CREATE
symmetrical	True
tenant_id	8502b27cb8c249f5bf92752ed195778c

그림 19 . Service Function Chaining configuration

```
[vagrant@tacker devstack]$ tacker sfc-show mychain
```

Field	Value
attributes	{}
chain	a8733c9a-9eef-421f-8652-21d73929e6f3 67c6c726-a061-478e-8a9a-11b114ee59ed
description	
id	4ac5267e-b558-4dde-87ed-5a138926700b
infra_driver	opendaylight
instance_id	Path-mychain-Path-20
name	mychain
status	ACTIVE
symmetrical	True
tenant_id	8502b27cb8c249f5bf92752ed195778c

```
[vagrant@tacker devstack]$
```

그림 20 . tacker 에서의 SFC 정보 확인

만들어진 체인에 대한 정보는 tacker-sfc-show mychain 명령어를 통해 그림20과 같이 확인이 가능하다.

```
[vagrant@tacker devstack]$ sudo ovs-ofctl -O OpenFlow13 dump-flows br-int
OFFST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=121.823s, table=0, n_packets=37, n_bytes=4156, in_port=1,dl_src=fa:16:3e:0f:4b:6b actions=set
t_field:0x3ed->tun_id,load:0x1->NXM_NX_REG0[],goto_table:10
cookie=0x0, duration=83.213s, table=0, n_packets=30, n_bytes=2700, in_port=2,dl_src=fa:16:3e:be:65:0b actions=set
field:0x3ed->tun_id,load:0x1->NXM_NX_REG0[],goto_table:10
cookie=0x0, duration=71.598s, table=0, n_packets=30, n_bytes=2700, in_port=3,dl_src=fa:16:3e:0a:ff:be actions=set
field:0x3ed->tun_id,load:0x1->NXM_NX_REG0[],goto_table:10
cookie=0x0, duration=387.095s, table=0, n_packets=0, n_bytes=0, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b00000000000001, duration=387.573s, table=0, n_packets=0, n_bytes=0, priority=100,dl_type=0x88cc action
s=CONTROLLER:65535
cookie=0x0, duration=121.842s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=1 actions=drop
cookie=0x0, duration=83.138s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=2 actions=drop
cookie=0x0, duration=71.564s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=3 actions=drop
cookie=0x0, duration=387.109s, table=0, n_packets=1, n_bytes=0, priority=0 actions=goto_table:10
```

그림 21 . SFC 에 사용되는 패킷 헤더

사용자는 ovs-ofctl -o Openflow13을 통해 실제 사용되는 패킷의 형태를 그림21과 같이 확인이 가능하다.

7. 결론

본 기술문서에서의 VM/VNF의 관리를 위해 개발되고 있는 VNFM 관련 표준화 현황을 분석하고 대표적인 오픈프로젝트를 통해 개발상황을 확인하고자 하였다. 이를위해 ETSI에서 정의하고 있는 VNFM의 기능들을 나열하고 분석하였으며 이를 현재 대표적인 오픈소스 프로젝트인 오픈스택에서의 실제 구현되고 있는 Tacker 라는 VNFM을 통해 실제 개발된 VNFM의 기능을 확인하였다. 또한 tacker 프로젝트의 이론적인 배경과 Blue Print로 고려되고 있는 내용을 분석하고 실제 설치를 통해 다양한 기능들의 동작 유무도 확인 할 수 있었다.

8. Reference

- [1] ETSI GS NFV-MAN 001 V1.1.1 “Network Functions Virtualization (NFV) Management and Orchestration.
- [2] <https://wiki.openstack.org/wiki/Tacker>
- [3] TOSCA Simple Profile in YAML Version 1.0
- [4] Halpern, J. and C. Pignataro, “Service Function Chaining (SFC)Architecture“, RFC 7665, October 2015.
- [5] <https://github.com/openstack/tacker/tree/master/devstack>
- [6] <https://blueprints.launchpad.net/tacker/+spec/tacker-sfc>
- [7] <https://www.opendaylight.org/>