

K-ONE 기술 문서 #32  
K-ONE 공용개발환경에서의 Network-aware  
Service Mesh 개념과 부분적 검증

Document No. K-ONE #32

Version 0.1

Date 2018-12-24

Author(s) 강문중

■ 문서의 연혁

버전	날짜	작성자	내용
초안 - 0.1	2018. 12. 24	강문중	초안 작성

본 문서는 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신  
기술진흥센터의 지원을 받아 수행된 연구임 (No. 2015-0-00575, 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(MSIT) (No. 2015-0-00575, Global SDN/NFV  
OpenSource Software Core Module/Function Development)

## 기술문서 요약

본 기술문서는 K-ONE 컨소시엄 참여기관인 광주과학기술원, 고려대학교, 숭실대학교, 포항공과대학교, 한국과학기술원, 총 5개 기관에서 각자 목표로 상정한 다양한 오픈소스 소프트웨어의 개발 및 검증환경을 공통된 하드웨어 묶음 형태로 제공하기 위하여 Edge-Cloud 패러다임에 대응하는 클러스터형 테스트베드인 K-Cluster 다수가 여러 사이트에 걸쳐 분산 배포된 상황에서 이 K-Cluster들을 네트워크로 연동한 멀티 사이트 K-ONE 공용개발환경(또는 공용개발환경)에 대해, 멀티 사이트를 아우르는 Kubernetes 기반의 컨테이너 환경에서 멀티 사이트 컨테이너 네트워킹 및 마이크로서비스 구조의 서비스에 대해 기능간 통신을 제어하기 위한 방안에 대해서 서술한다.

## Contents

### K-ONE #xx. Network-aware Service Mesh

1. 서론 .....	5
1.1. 목적 .....	5
1.2. K-Cluster .....	6
1.3. K-ONE 공용개발환경 .....	7
2. 관련 기술 배경 .....	9
2.1. 서비스 메쉬 .....	9
3. K-ONE 공용개발환경에서의 Network-aware Service Mesh 설계 .....	11
3.1. Network-aware Service Mesh 설계 .....	11
3.2. K-ONE 공용개발환경에서의 Network-aware Service Mesh의 부분적인 검증	13
4. 결론 .....	18

## 그림 목차

그림 1 K-Cluster 구성요소 .....	6
그림 2 K-Cluster Hardware 구성 .....	7
그림 3 소프트웨어 정의 인프라 패러다임의 MultiX Challenges .....	8
그림 4 K-ONE 공용개발환경의 구축 현황도 .....	8
그림 5 서비스 메쉬 .....	10
그림 6 기존 서비스 메쉬 프룻기 .....	12
그림 7 노드가 등록되고 가상화된 환경이 준비된 상태 .....	13
그림 8 서비스 메쉬의 대시보드 .....	14
그림 9 JSON 형태의 API 입출력 .....	14
그림 10 setup_manager.sh로 부분적인 검증 환경을 구성 중인 모습 .....	15
그림 11 예시 워크로드의 구성 .....	16
그림 12 setup_tsm.sh, setup_workload.sh로 서비스 메쉬 및 예시 워크로드를 구성 중인 모습 .....	16
그림 13 서비스 메쉬 및 예시 워크로드가 준비된 상태 .....	17

## K-ONE #32. K-ONE 공용개발환경에서의 Network-aware Service Mesh 개념과 부분적 검증

## 1. 서론

### 1.1. 목적

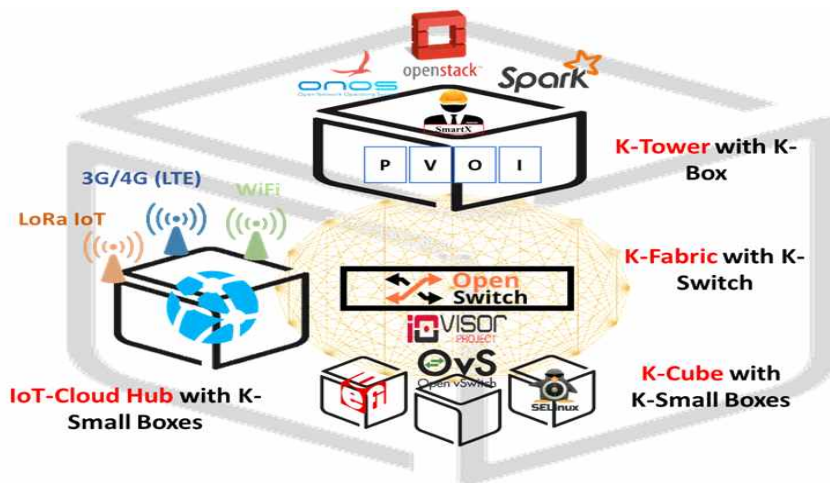
- o 본 기술문서는 K-ONE 컨소시엄 참여기관인 광주과학기술원, 고려대학교, 숭실대학교, 포항공과대학교, 한국과학기술원, 총 5개 기관에서 각자 목표로 상정한 다양한 오픈소스 소프트웨어의 개발 및 검증환경을 공통된 하드웨어 묶음 형태로 제공하기 위하여 Edge-Cloud 패러다임에 대응하는 클러스터형 테스트베드인 K-Cluster 다수가 여러 사이트에 걸쳐 분산 배포된 상황에서 이 K-Cluster들을 네트워크로 연동한 멀티 사이트 K-ONE 공용개발환경(또는 공용개발환경)에 대해, 멀티 사이트를 아우르는 Kubernetes 기반의 컨테이너 환경에서 멀티 사이트 컨테이너 네트워킹 및 마이크로서비스 구조의 서비스에 대해 기능간 통신을 제어하기 위한 방안에 대해서 서술한다.
- o 현재 클라우드 시장이 활성화되면서 마이크로서비스 구조의 어플리케이션들이 많이 나오고 있고, 이런 서비스들의 견실함을 보장하는 건 중요한 문제이다. 그런데 대역폭이 한정되어 있는 네트워크 자원 상에서 서비스를 구성하며 붙어 있는 기능들이 서로 끊어지지 않게 하려면 그 연결을 유지하기 위해서 해당 트래픽에 대한 QoS 제어와 모니터링이 필요한데, 이런 QoS 제어는 보통 네트워크 인프라 내의 스위치나 라우터 내에서 이루어진다. 이는 VM 등의 가상의 주체와 그에 대한 가상화된 네트워킹을 제공하는 OpenStack 클라우드, 그리고 이를 기반으로 하고 있는 K-ONE 공용개발환경 내에서도 마찬가지이다. 그런데 클라우드 환경처럼 여러 서비스가 인프라를 공유하는 환경에서는 트래픽 흐름의 관문 역할을 하는 스위치나 라우터 내에서 트래픽이 모두 뒤섞여서 트래픽이 어느 서비스의 어느 기능에 속하는지 알기 어렵다.
- o 기존의 QoS, 트래픽 모니터링 방법들을 비교해 보면 총 3가지가 있다. 스위치나 라우터에서는 트래픽 식별이 힘드므로 단순히 5-tuple (소스/데스티네이션 주소, 포트, 그리고 프로토콜 정보)로만 제어를 하는 방법이 있으나 이것만으로는 트래픽 식별이 어렵다. 여기에 더해 클라우드 환경에서는 NAT나 오버레이 네트워킹이 광범위하게 쓰이기 때문에 5-tuple 정보까지 왜곡되서 제약이 더 심해진다. 이로 인해 클러스터 내 기능 대 기능 간의 트래픽에 대한 가시성도 제공이 힘들다.
- o 그래서 DPI나 머신 러닝을 시도하는 방법도 있다. 그러나 머신 러닝은 연산 자원을 많이 소모하고 그러면서도 트래픽 식별이 완전하게 되지는 않는다. 여전히 스위치나 라우터 앞, 뒷단 정도에서 트래픽을 모니터링하기 때문에 클러스터 내부 간 기능 대 기능 간의 트래픽 가시성도 제공이 힘들다.

- o Application-specific Agent는 서비스 자체에서 아예 스스로 QoS 기능을 담당하거나, 서비스에 Agent를 별도로 심어서 해결하는 방법이다. 각 기능마다 Agent가 붙기 때문에 기능 단위의 가시성이나 제어도 가능할 수 있으나, 문제는 서비스가 스스로 QoS를 떠안거나, 기능이 담기는 컨테이너 안에 에이전트를 같이 심어야 하기 때문에 기능을 자체적으로 따로 변형해서 관리해야 하는 등의 문제가 생길 수 있다.
- o 그래서 견실한 Service를 위한 세밀한 QoS 제어를 위한 조건은 먼저 기능 단위의 세밀하고 정확한 트래픽 식별이 가능한 가시성이 있어야 한다. 그리고 이 가시성이 최소한의 자원 소모만으로 이루어져야 한다. 그리고 서비스 합성 자체를 수정할 필요도 없어야 한다. 그리고 이러한 세밀한 QoS를 통합 관리할 수 있어야 한다.
- o OpenStack Neutron은 소프트웨어-정의 네트워킹을 활용해 OpenStack 클러스터 내의 네트워킹을 자동화하는 프로젝트로, 클러스터의 사용자가 원하는 형태의 네트워크 구조를 소프트웨어적으로 구축할 수 있다. 그러나 OpenStack Neutron은 어디까지나 OpenStack 클러스터가 위치한 하나의 지역(예를 들어 데이터센터 하나)에 대해서만 고려해 설계되었다는 단점이 있다. 여러 지역에 걸친 OpenStack 클러스터를 구축하는 것이 불가능한 것은 아니나, 이렇게 구축된 OpenStack 클러스터는 내부적으로 멀티 사이트에 대한 구분이 불가능해 자원의 실제 인접성을 고려하지 않는 비효율적인 네트워킹이 발생할 수 있다.
- o OpenStack Tricircle은 OpenStack의 하위 프로젝트 중 하나로 앞서 언급한 멀티 사이트간 네트워킹 자동화의 한계점을 넘는 것을 목적으로 한다[1]. Tricircle은 OpenStack 외의 별도 설정 없이 Neutron의 API를 확장시켜 OpenStack의 다른 프로젝트들에서도 매끄럽게 멀티 사이트 네트워킹을 활용할 수 있고, 클러스터간 지역 구분을 통해 자원의 실제 인접성이 잘 고려된 효율적인 네트워킹을 구축할 수 있게 한다. 그러나 본질적으로 Tricircle 또한 스위치 및 라우터의 한계를 벗어나는데 있어 다소간 방향이 다르다.
- o K-ONE 공용개발환경은 단일 K-Cluster로는 실증이 어려운 멀티사이트/도메인 관련 실증을 통합적으로 제공하기 위하여 K-Cluster 시제품들을 K-ONE 컨소시엄의 참여 사이트에 각각 배포하고 이를 KREONET 연구망으로 연결함으로써 구축한 5개의 멀티사이트에 걸친 실증 테스트베드이다. 본 기술문서에서는 K-ONE 공용개발환경의 OpenStack Tricircle을 활용한 네트워킹 자동화 설계 및 구축 과정에 대해 상세히 기술한다.



## 1.2. K-Cluster

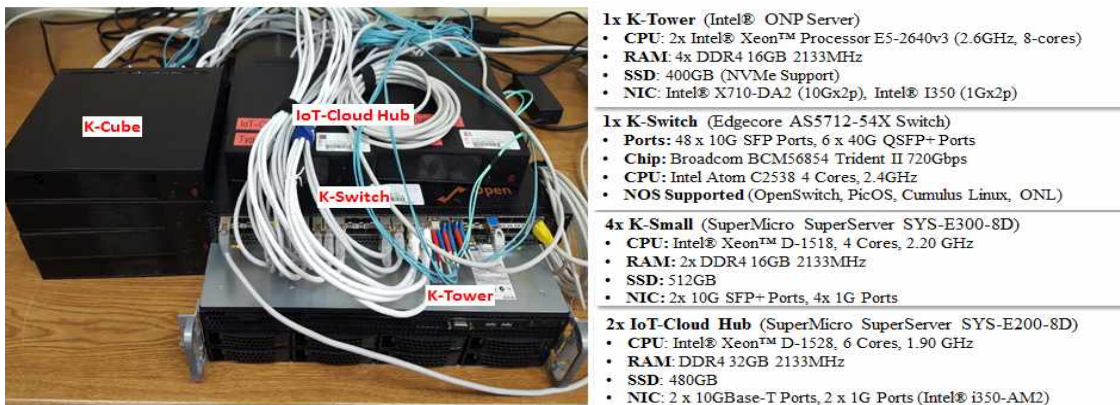
- o K-Cluster 및 K-ONE 공용개발환경의 자세한 설명은 'K-ONE 기술문서 16#\_멀티사이트 클라우드 실증환경에 대응하는 K-Cluster 중심의 K-ONE 공용개발환경 설계 및 활용 방안'을 참조하면 되고, 본 기술문서에는 간략하게 기술한다[2].



<그림 1: K-Cluster 구성요소>

- o <그림 1>과 같이 K-Cluster는 소프트웨어 정의 인프라 패러다임에 대응하는 다양한 SDN/NFV/Cloud 실증을 제공하기 위한, Edge-Cloud 모델에 대응하는 경제적인 소규모 클러스터 형태의 테스트베드로 정의한다. K-Cluster의 예상 활용처는 대규모의 ICT 서비스를 제공하는데 필요한 안정적이고 고성능의 인프라를 구성하는데 쓰이는 것이 아니라, Edge-Cloud와 연관된 다양한 SDI 실증을 및 연구개발을 위한 적정 규모의 테스트베드를 확보하기 어려운 연구자, 개발자를 위한 소규모 테스트베드 환경을 기본 활용처로 상정한다. 물론 K-Cluster도 안정적이고 고성능의 자원을 제공하는 것이 주요 요구사항 중 하나로 실제 Production 환경에 적용하는데 부족함이 없으나, 요구사항의 우선순위 측면에서 다양한 Edge-Cloud 기술을 하나의 테스트베드에 수용하기 위한 자원 측면의 유연성과 구축 경제성을 설계 시 가장 높은 우선순위로 한다.
- o 먼저 K-Tower는 K-Cluster 전체를 DevOps(개발운영병행체제) 관점에서 관리/운영의 자동화를 위한 단일 또는 복수의 K-Box들로 구성된 관제(monitor & control)을 전담한다. K-Tower는 내부적으로 오픈소스 DevOps 도구를 활용하여 자동화를 지원하는 Provisioning 센터, Visibility 센터, Orchestration센터, 그

리고 Intelligence 센터를 위한 지원역할을 담당하는 소프트웨어 적인 기능들을 포함한다. K-Cluster의 하단부에 위치한 K-Cube는 다수의 동종 K-Small 박스 자원들로 구성된 K-Cluster 내부의 클러스터로써 계산/저장/네트워킹을 담당하는 공유된 자원집합을 지칭한다. K-Fabric은 K-Cluster의 중앙에 위치하여 물리/가상 네트워킹 기능들을 활용하여 패브릭 형태의 밀결합된 네트워킹을 구성한다. 이를 통해 K-Cluster 구성 요소들인 K-Tower, K-Cube, IoT-Data Hub 간의 빠르고(fast) 유연하며(flexible) 안정된(reliable) 연결성을 제공한다.

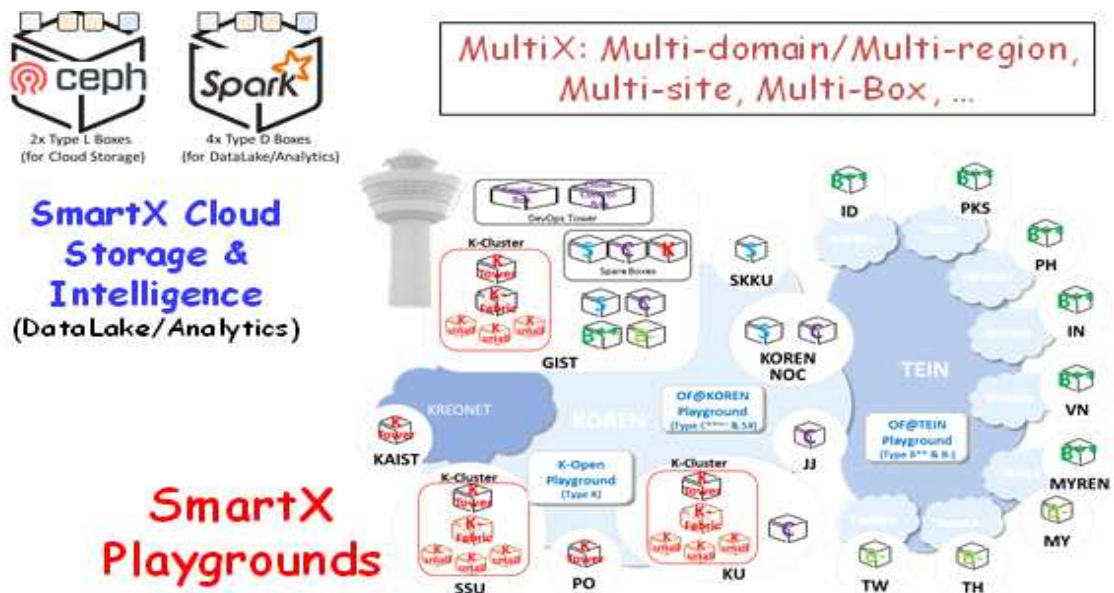


<그림 2: K-Cluster Hardware 구성>

- o K-Tower, K-Fabric, K-Cube, IoT-Data Hub등으로 구성된 K-Cluster 모델을 활용하면 에지 클라우드에서 요구하는 SDN/NFV/Cloud 통합과 FastData/BigData/ HPC 기반의 다양한 서비스를 저렴한 상용 화이트박스들을 활용하여 유연하게 대응할 수 있다.

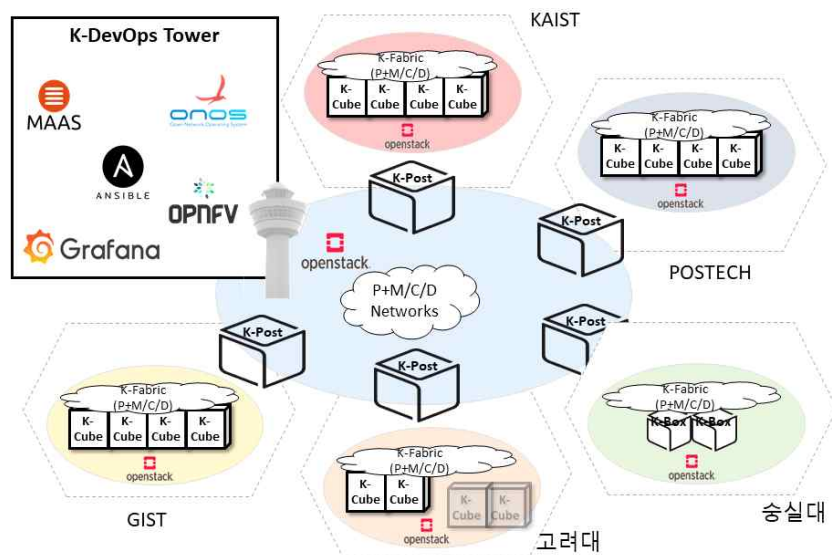
### 1.3. K-ONE 공용개발환경

- o 고립된 하나의 박스, 하나의 클러스터, 하나의 사이트를 고도화 하는 전략은 기술적으로 이미 많이 성숙되어 자원의 용량 및 성능 측면에서 큰 향상을 기대하는 것은 어려우며, 근본적으로 자원 활용의 유연성/확장성 측면의 커다란 단점을 내포하고 있다. 게다가 다수의 멀티 자원요소들 간의 연결성을 제공하는 클러스터링, 네트워킹 기술이 고도화 됨에 따라 다수의 요소를 연결함으로써 발생하는 병목현상, 성능저하 보다 자원의 클러스터링을 통한 유연하고 확장성 있는 활용이 큰 가치를 갖게 되었다. 따라서 최근에는 <그림 3>의 우측 상단과 같이 멀티박스를 넘어서 멀티리전, 멀티사이트, 심지어 멀티도메인과 같이 다양한 멀티 이슈에 대응하는 MultiX가 ICT 인프라의 연구 키워드의 하나로 부상하고 있다.



<그림 3: 소프트웨어 정의 인프라 패러다임의 MultiX Challenges>

- o K-ONE 컨소시엄 기관을 대상으로 멀티사이트 SDN/NFV/Cloud 통합 실증환경인 K-ONE 공용개발환경을 <그림 4>과 같이 구성하였으며, 이를 위해 다음과 같은 노력을 수행하였다.
- o K-Cluster 및 K-ONE 공용개발환경의 자세한 설명은 ‘K-ONE 기술문서 16#\_멀티사이트 클라우드 실증환경에 대응하는 K-Cluster 중심의 K-ONE 공용개발환경 설계 및 활용 방안’을 참조하면 되고, 본 기술문서에는 간략하게 기술한다.

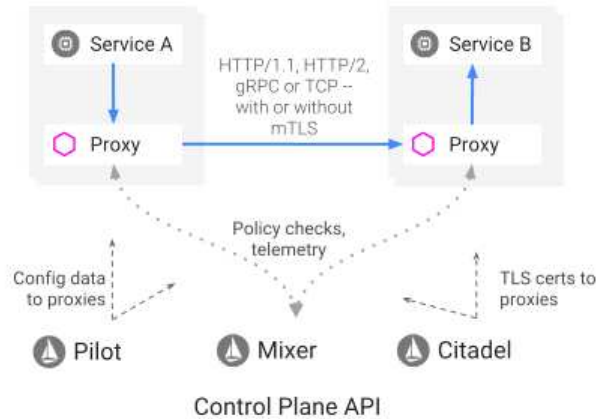


<그림 4: K-ONE 공용개발환경의 구축 현황도>

## 2. 관련 기술 배경

- o 2절은 멀티사이트 K-ONE 공용개발환경을 구축하기 위해 필요한 기술들과 관련 오픈소스 프로젝트들에 대해 간략하게 기술한다.

### 2.1. 서비스 메쉬



<그림 5: 서비스 메쉬>

- o 기존에는 어떤 어떤 서비스를 제공하기 위해 그 서비스의 모든 기능을 하나의 거대한 프로그램이 담당하는 방식인 모놀리식 구조가 일반적으로 사용되었다. 그러나 클라우드 시장이 활성화되고 클라우드 기반 서비스들이 증가함에 따라, 모놀리식 구조에서는 서비스를 구성하는 기능 중 하나만 오류가 생기더라도 모든 서비스가 영향을 받는 내결함성의 결여와, 확장성 등의 문제가 대두되었다. 이로 인해 개발용 이성과 유지보수의 이점으로 어플리케이션들도 하나의 서비스 속 여러 기능을 여러 개의 작은 어플리케이션으로 나눠 분담하고 이들이 네트워크를 통해 상호연동하여 하나의 서비스를 제공하는 형식인 마이크로서비스 구조로 그 트렌드가 넘어가고 있는 상황이다. 이러한 트렌드 전환으로 상호 기능간 네트워크를 통한 연계가 중요해지고 있지만 네트워크 인프라는 대역폭이 제한적이며, 항상 안정화된 연결을 보장하지도 않는다. 따라서 이러한 상호연계 보장을 전적으로 네트워크 인프라에 맡길 수 없으며 이러한 상호연계가 보장될 수 없는 상황을 고려해야하는 어플리케이션 자체의 부담이 커진다[4].
- o 이러한 네트워크 상에서의 응용간 상호연계에 대한 문제를 분리시켜 해결하고자 하는 기법이 서비스 메쉬(Service Mesh)이다[5]. 서비스 메쉬는 인프라 계층과 응용 계층의 사이에 위치하는 계층으로, 대역폭이 한정된, 불안정한 네트워크 인프라 상에서 제공해야할 견실한 QoS를 위한 모니터링 및 제어를 제공한다.

- o 일반적인 스위치, 라우터 등은 모두 인프라에 소속된 하드웨어로 특화 하드웨어에 의존하고 있기 때문에 인프라 수준 이상의 관제가 힘든 것이 보통이다. 서비스 메쉬는 이러한 문제점을 피하기 위해 범용 하드웨어와 가상화 네트워킹의 이점을 이용, 네트워크 구조 상에서 상호연동하는 응용간의 연결 사이에 에이전트를 삽입하여 응용들을 밀접하게 모니터링한다[6]. 이를 통해 단순한 자원 사용량이나 패킷 수준의 트래픽 모니터링을 벗어나 응용간 요청 단위를 인식하는 수준의 모니터링과 관제가 가능하다.

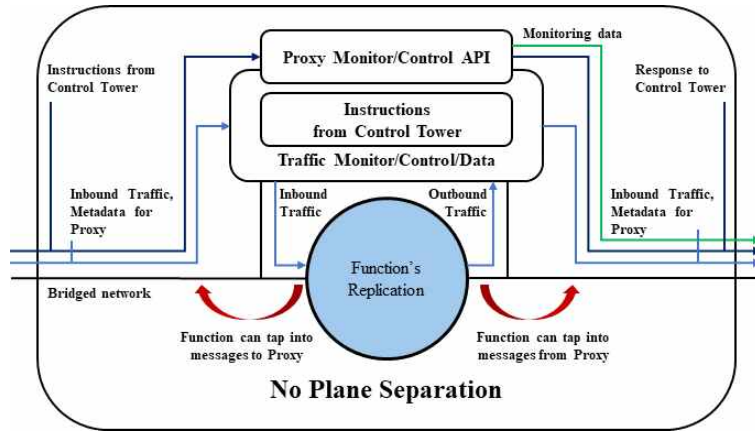
### 3. K-ONE 공용개발환경에서의 Network-aware Service Mesh 설계

- o 4절에서는 본격적으로 K-ONE 공용개발환경에 Istio에 기반한 Network-aware Service Mesh를 적용해 멀티사이트 클라우드 환경을 구축하는 방법에 대해서 논한다.

#### 3.1. Network-aware Service Mesh 설계

- o 기존 서비스 메쉬 기술에는 다음과 같은 한계가 있다. 일반적으로 서비스 메쉬 프록시는 HTTP 프록시에서 유래한 기술로, 당연히 HTTP 프로토콜에 크게 의존한다. 따라서 HTTP 프로토콜을 사용하는 경우, HTTP 헤더에 삽입된 정보에 대해서는 헤더를 파싱하여 손쉽게 정보를 모니터링할 수 있다. 그러나 HTTP가 아닌 형태의 프로토콜에 대해서는 이와 같이 트래픽에 대한 세밀한 모니터링이 어렵다.
- o Kubernetes의 경우 컨테이너에 대해 하나의 인터페이스만을 제공하며, 또한 하나의 Pod 내에서는 여러 개의 컨테이너가 하나의 네트워크 스택을 공유한다. 따라서 그리고 제어/모니터링 평면과 데이터 평면의 분리가 되지 않는다. 두 평면의 데이터가 하나의 인터페이스를 통해 흐른다. 따라서 공유 자원 상에서는 이용자의 응용이 자신을 지원하는 서비스 메쉬를 조작하여 공유 자원에 대한 무단접근/방해를 행할 수가 있다.





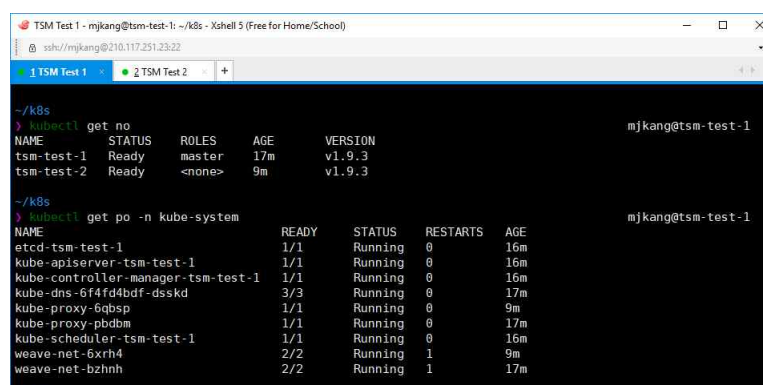
<그림 6: 기존 서비스 메쉬 프록시>

- o 또한 이런 프록시들은 CPU 연산에 전적으로 의존, 특정 인프라와의 매핑이 불가능하다. 예를 들어 Istio에서 사용하는 Envoy 등은 전적으로 CPU에서 해당 프록시의 처리를 담당하며, 네트워크 처리를 지원하는 특화된 하드웨어 등을 이용하거나 인프라를 나누는 네트워크 슬라이싱 등의 연계가 어렵다.
- o 따라서 프로토콜에 얽매이지 않고 공유 자원 상에서의 제어/모니터링 평면이 이용자가 보거나 조작할 수 없도록 데이터 평면과 완전히 분리되고, 특정 인프라와의 매핑을 통해 특화된 하드웨어 및 네트워크의 슬라이싱이 가능한 서비스 메쉬 기술이 필요하다.
- o P+V+C Functions를 총괄하여 제어 가능한 컨트롤 타워는 물리(Physical), 가상(Virtual), 컨테이너(Container) 형태의 기능들에 맞춰 대응하는 프록시 및 Security Enforcer를 총괄하여 제어할 수 있는 소프트웨어이다. 기능이 P/V/C 중 아무 것이라도 거기에 대응하는 프록시 및 Security Enforcer는 통일된 API로 대화하여 3가지 형태를 모두 총괄하여 제어할 수 있다. 프록시들이 자신이 배치된 노트에 대한 네트워크 상태 정보를 모두 타워에 보고하므로 타워에서는 네트워크 적응적 서비스 메쉬가 배치된 네트워크 상태를 파악할 수 있고, 이 정보를 바탕으로 프록시들에게 각 서비스를 위한 QoS 정책 설정을 쉽게 내릴 수 있는 기반을 제공한다.
- o 중간 기착지 노드에서의 트래픽의 네트워크 슬라이스 전환 및 중간처리를 한다. 트래픽이 거쳐가는 중간 노드들은 모두 인접 노드간 네트워크 상태 정보를 상시교환하면서 노드 간의 네트워크 상태를 사전에 파악하고 있다. 트래픽 중개 시 트래픽이 요구하는 QoS (ex. 지연률, 손실률, 전송 속도) 등이 얼마나 충족되고 있는지 평가, 다음 노드로의 트래픽 전송 시 QoS를 만족시킬 수 있는 최적의 네트워크 슬라이스를 골라 전송한다.

- o Security Enforcer는 중간 기착지 노드에서도 단순 중계만으로 그치지 않고 중간단계에 Security Enforcer가 개입하여 암호화 상태 및 ACL 필터링 수행, 문제 발생시 Delivery failure를 처리한다. 마이크로서비스에서 보안 상 우려가 될 수 있는 평문 전송 시 중간 단계에서 Security Enforcer가 개입하여 트래픽을 암호화하여 전송하고 다시 Recipient 쪽의 Security Enforcer가 암호화를 해제하여 평문으로 Recipient 마이크로서비스가 수신하므로 마이크로서비스 자체에서 암호화를 고려할 필요성을 줄인다. DNS 쿼리는 모두 평문 기반이며, 거의 모든 마이크로서비스도 이에 맞춰져 있으나, Security Enforcer가 기존의 평문 기반 DNS 쿼리를 중간 단계에서 암호화를 통해 DNS 서버에 안전하게 전송하여 위변조 및 감시를 차단하고, 암호화된 형태의 회신을 받아 다시 평문 상태로 마이크로서비스에 전달하여 마이크로서비스에 대한 호환성을 보장하면서 DNS 쿼리의 안전성을 보장한다. ACL 등을 통한 필터링으로 허용된 대상으로부터, 또는 허용된 대상으로만의 전송을 허용한다.

### 3.2. K-ONE 공용개발환경에서의 Network-aware Service Mesh의 부분적인 검증

- o K-ONE 공용개발환경 기반의 Network-aware Service Mesh는 Kubernetes 기반으로 가상화된 네트워크를 노드 하나 또는 다수에 걸쳐 형성하고, 그 위에 예시 서비스를 올린 후 이를 대상으로 트래픽에 대한 QoS 정책을 중앙제어할 수 있는 API 서버를 제공해 부분적인 검증을 수행한다.
- o 이를 위한 소프트웨어는 크게 다음과 같이 구성되어 있다. 검증 환경 구성은 부분적인 검증을 위해 가상화된 환경을 K-ONE 공용개발환경의 여러 노드에 걸쳐 생성한다.



```
TSM Test 1 - mjikang@tsm-test-1: ~/k8s - Xshell 5 (Free for Home/School)
ssh://mjikang@210.117.251.2322

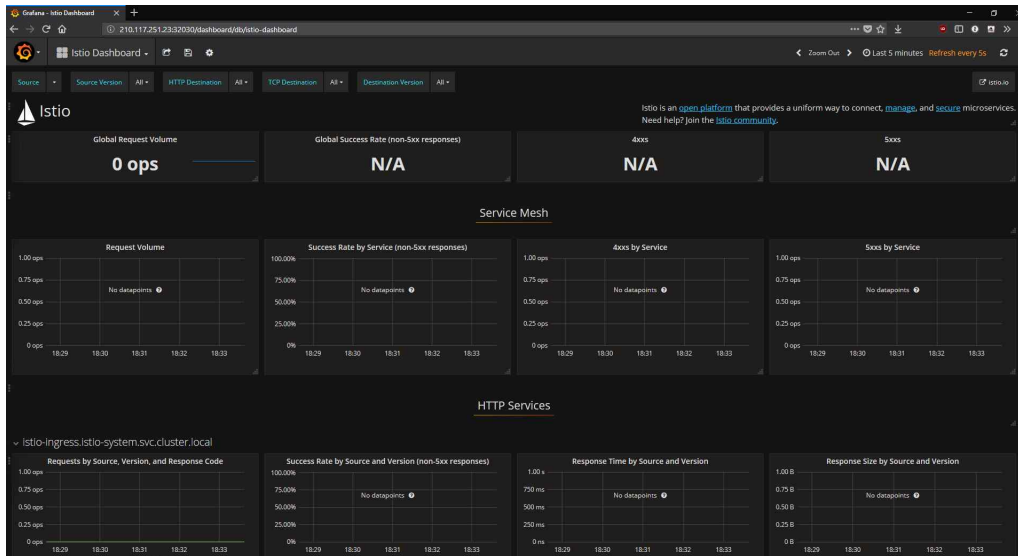
~/k8s
> kubectl get no
NAME          STATUS    ROLES    AGE     VERSION
tsm-test-1    Ready     master   17m     v1.9.3
tsm-test-2    Ready     <none>    9m      v1.9.3

~/k8s
> kubectl get po -n kube-system
NAME                                READY    STATUS    RESTARTS   AGE
etcd-tsm-test-1                     1/1      Running   0           16m
kube-apiserver-tsm-test-1            1/1      Running   0           16m
kube-controller-manager-tsm-test-1  1/1      Running   0           16m
kube-dns-6f4fd4bdf-dsskd            3/3      Running   0           17m
kube-proxy-6qbsp                     1/1      Running   0           9m
kube-proxy-pbdbm                     1/1      Running   0           17m
kube-scheduler-tsm-test-1            1/1      Running   0           16m
weave-net-6xrh4                     2/2      Running   1           9m
weave-net-bzhnh                     2/2      Running   1           17m
```

<그림 7: 노드가 등록되고 가상화된 환경이 준비된 상태>

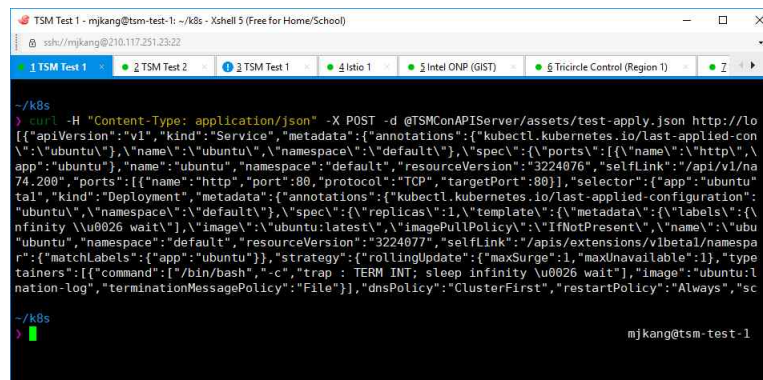
- o 그리고 서비스 메쉬 및 예시 워크로드들은 가상화된 환경에 트래픽에 대한 QoS 정책 관제를 위한 서비스 메쉬 계층을 구성하고, 서로 데이터를 주고받으며 하나의

집합으로 동작하는 워크로드들을 여러 노드에 분배해 형성한다.



<그림 8: 서비스 메쉬의 대시보드>

- o 마지막으로 API 서버는 서비스 메쉬 계층을 중앙제어할 수 있도록 호출가능한 HTTP, JSON 기반 API를 제공한다.

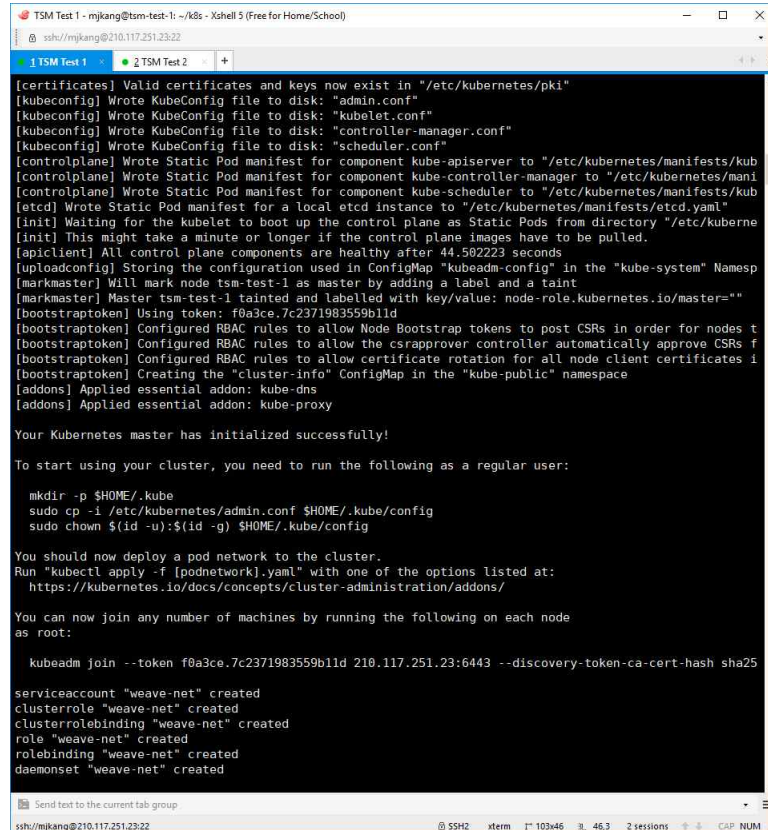


<그림 9: JSON 형태의 API 입출력>

- o 본 부분적인 검증에서는 가상화된 환경을 여러 노드에 걸쳐 생성하고, 사용자의 지시에 따라 트래픽에 대한 QoS 정책 관제를 위한 서비스 메쉬 계층을 구성하고, 서로 데이터를 주고받으며 하나의 집합으로 동작하는 워크로드들을 여러 노드에 분배해 형성한 뒤 서비스 메쉬 계층을 API를 통해 중앙제어하여 워크로드간 트래픽에 대해 QoS 정책 관제를 수행한다.
- o ● 검증 환경 구성을 위해 하나 또는 여럿의 노드가 필요하며, 하나의 대표 노드에



서 `setup_manager.sh`를 실행한 후 화면의 메시지 상의 명령어를 기록해줬다가(ex. `kubeadm join ...`), 나머지 노드에서 `setup_worker.sh`를 실행한 뒤 기록한 명령어를 실행해 대표 노드에 나머지 노드를 묶는다. 이 때 노드가 VM이 아닌 베어메탈인 경우 `patch_bm.sh`를 `setup_worker.sh` 이전에 실행한다.



```
[certificates] Valid certificates and keys now exist in "/etc/kubernetes/pki"
[kubeconfig] Wrote KubeConfig file to disk: "admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "scheduler.conf"
[controlplane] Wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] Wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] Wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etcd.yaml"
[init] Waiting for the kubelet to boot up the control plane as Static Pods from directory "/etc/kubernetes/manifests". This may take a minute or longer if the control plane images have to be pulled.
[apiclient] All control plane components are healthy after 44.582223 seconds
[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[markmaster] Will mark node tsm-test-1 as master by adding a label and a taint
[markmaster] Master tsm-test-1 tainted and labelled with key/value: node-role.kubernetes.io/master=""
[bootstraptoken] Using token: f0a3ce.7c2371983559b11d
[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get certificates
[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs for kubernetes.io/node-token
[bootstraptoken] Configured RBAC rules to allow certificate rotation for all node client certificates in the kube-public namespace
[bootstraptoken] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

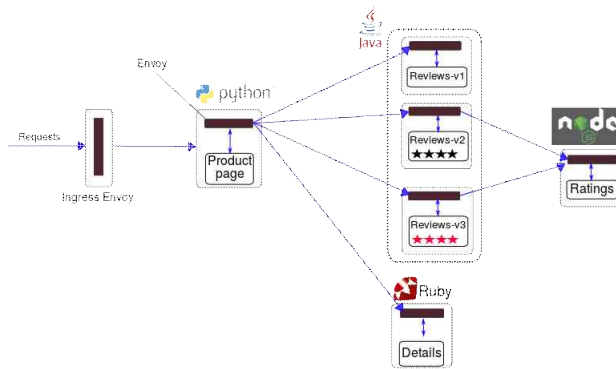
You can now join any number of machines by running the following on each node
as root:

kubeadm join --token f0a3ce.7c2371983559b11d 210.117.251.23:6443 --discovery-token-ca-cert-hash sha256:f0a3ce.7c2371983559b11d

serviceaccount "weave-net" created
clusterrole "weave-net" created
clusterrolebinding "weave-net" created
role "weave-net" created
rolebinding "weave-net" created
daemonset "weave-net" created
```

<그림 10: setup\_manager.sh로 부분적인 검증 환경을 구성 중인 모습>

- o 대표 노드에서 `setup_tsm.sh`를 실행하면 서비스 메쉬가 설치되며 “`kubectl get po -n istio-system`”을 통해 모든 항목이 Running 상태가 될 때까지 기다린 후 `setup_workload.sh`를 실행해 구성 및 예시를 위한 워크로드가 설치한다. 명령은 실행 후 금세 종료되지만 설치 완료까지는 수 분이 걸릴 수 있으며 설치가 완료되면 그림 7과 같은 명령어를 통해 확인했을 때 모든 항목의 STATUS가 Running으로 뜬다. 그리고 `check_tsm_example_ready.sh` 명령을 사용했을 때 200이란 회신을 받게 된다. 예시 워크로드는 그림 3과 같이 `producepage`, `reviews-v1`, `reviews-v2`, `reviews-v3`, `details`, `ratings` 등의 워크로드가 여러 노드에 분산 배치되어 트래픽을 주고받는다. 서비스 매쉬는 트래픽을 모니터링해 대시보드([http://\(대표노드 주소\):32030/dashboard/db/istio-dashboard](http://(대표노드 주소):32030/dashboard/db/istio-dashboard))로 가시화한다.



<그림 11: 예시 워크로드의 구성>

```

customresourcedefinition "quotas.config.istio.io" created
customresourcedefinition "reportnothings.config.istio.io" created
customresourcedefinition "servicecontrolreports.config.istio.io" created
customresourcedefinition "tracespans.config.istio.io" created
customresourcedefinition "servicerules.config.istio.io" created
customresourcedefinition "servicerolebindings.config.istio.io" created
configmap "istio" created
customresourcedefinition "destinationpolicies.config.istio.io" created
customresourcedefinition "egressrules.config.istio.io" created
customresourcedefinition "routerules.config.istio.io" created
service "istio-pilot" created
serviceaccount "istio-pilot-service-account" created
deployment "istio-pilot" created
service "istio-ingress" created
serviceaccount "istio-ingress-service-account" created
deployment "istio-ingress" created
serviceaccount "istio-ca-service-account" created
deployment "istio-ca" created
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
unable to recognize "istio-0.5.1/install/kubernetes/istio-nodeport.yaml": no matches for config.istio.i
service "details" created
deployment "details-v1" created
service "ratings" created
deployment "ratings-v1" created
service "reviews" created
deployment "reviews-v1" created
deployment "reviews-v2" created
deployment "reviews-v3" created
service "productpage" created
deployment "productpage-v1" created
ingress "gateway" created
    
```

<그림 12: setup\_tsm.sh, setup\_workload.sh로 서비스 메쉬 및 예시 워크로드를 구성 중인 모습>

```

TSM Test 1 - mjkgang@tsm-test-1: ~/k8s - Xshell 5 (Free for Home/School)
ssh://mjkgang@210.117.251.23:22

~ /k8s
> kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
details-v1-c9c6fd7b5-zt7b7         2/2     Running   0           7m
productpage-v1-57c7c76897-gm2f4    2/2     Running   0           7m
ratings-v1-5dc5dd6ccc-rvr4f        2/2     Running   0           7m
reviews-v1-7d9895b47c-zkqdq        2/2     Running   0           7m
reviews-v2-85c6cf8f97-rjd8k        2/2     Running   0           7m
reviews-v3-586dbcb589-s8gwf        2/2     Running   0           7m

~ /k8s
> kubectl get svc
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
details         ClusterIP   10.99.136.233    <none>        9080/TCP   7m
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP    56m
productpage     ClusterIP   10.187.220.205   <none>        9080/TCP   7m
ratings         ClusterIP   10.185.169.203   <none>        9080/TCP   7m
reviews         ClusterIP   10.96.185.11     <none>        9080/TCP   7m

~ /k8s
> kubectl get po -n istio-system
NAME                                READY   STATUS    RESTARTS   AGE
istio-ca-797dfb66c5-txtkt          1/1     Running   0           7m
istio-ingress-67ff757554-h6mzx      1/1     Running   0           7m
istio-mixer-5bf5b5b94c-td68b        3/3     Running   0           7m
istio-pilot-676d495bf8-zzghs        2/2     Running   0           7m

~ /k8s
> kubectl get svc -n istio-system
NAME            TYPE        AGE      CLUSTER-IP      EXTERNAL-IP   PORT(S)
istio-ingress   NodePort     7m       10.99.212.174    <none>        80:32000/TCP,443:32060/TCP
istio-mixer     ClusterIP    7m       10.101.36.10     <none>        9091/TCP,15004/TCP,9093/TCP,9094/TCP,9182/TCP
istio-pilot     ClusterIP    7m       10.182.200.224   <none>        15003/TCP,8080/TCP,9093/TCP,443/TCP

~ /k8s
>

```

<그림 13: 서비스 메쉬 및 예시 워크로드가 준비된 상태>

o API 서버 실행 후 API 호출을 위해선, 먼저 TSMConAPIServer/src/main.go에서 매니저 노드의 IP 주소, 사용자 ID, SSH 비밀번호 위치를 입력 후 빌드한 뒤 main 파일을 실행시켜 API 서버를 실행한 후, 다른 터미널에서 curl 명령을 이용해 JSON 형태로 API를 호출해 여러 노드에 걸쳐 있는 서비스 매쉬를 중앙제어할 수 있다. 서비스 메쉬 계층을 중앙제어할 수 있도록 호출가능한 HTTP, JSON 기반 API를 제공한다. 사용할 수 있는 API 호출 예제는 TSMConAPIServer/assets에 있으며, 터미널에서는 “curl -H "Content-Type: application/json" -X POST -d @TSMConAPIServer/assets/test-apply.json http://localhost:8080/k8s/apply” 형태로 API를 호출하고 응답을 받을 수 있다. API 콜을 받는 주소경로는 정책 생성의 경우 .../k8s/apply, 삭제의 경우 .../k8s/delete, 조회의 경우 .../k8s/get, 교체의 경우 .../k8s/replace, 그 외에 .../generic이다.

#### 4. 결론

- o 본 기술문서를 통해 소프트웨어 정의 인프라 패러다임에 준비하기 위한 Edge-Cloud에 대응하는 테스트베드 모델인 K-Cluster을 설계에 대해 상세히 설명하였다. 그리고 설계에 따라 구축한 K-Cluster 하드웨어 시제품의 구성 및 상세 스펙에 대해 기술하였다.
- o 본 기술문서를 통해 K-ONE 컨소시엄이 목표로 하는 멀티사이트 SDN/NFV/Cloud 실증을 유연하고 효율적으로 지원하기 위한 K-ONE 공용개발환경의 설계를 멀티사이트 관점에서 Istio와 이를 활용하는 Network-aware Service Mesh를 이용해 구축하는 방안을 제시하였다.
- o 제시한 멀티사이트 K-ONE 공용개발환경을 Istio와 이를 활용하는 Network-aware Service Mesh를 이용해 구축하는 방안을 K-ONE 공용개발환경을 활용해 부분적으로 검증하였다. 따라서 본 기술문서에서 제안한 K-Cluster/K-ONE 공용개발환경 모델에 따라 인프라 환경을 구축하면 현재 SDI 패러다임에 대응하는 다양한 기술을 적용할 수 있음을 확인할 수 있다.

## References

- [1] OpenStack Tricircle, <https://wiki.openstack.org/wiki/Tricircle>
- [2] K-ONE 기술문서 #16 '멀티사이트 K-ONE Playground' 활용을 위한 K-DevOps Tower의 구축 및 활용 방법
- [3] IEEE 802.1Q standard 2005 version  
<http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf>
- [4] Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks  
<https://tools.ietf.org/html/rfc7348>
- [5] VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks <http://dx.doi.org/10.17487%2FRFC7348>
- [6] Virtual Extensible LAN (VXLAN) Overview - Arista  
[https://www.arista.com/assets/data/pdf/Whitepapers/Arista\\_Networks\\_VXLAN\\_White\\_Paper.pdf](https://www.arista.com/assets/data/pdf/Whitepapers/Arista_Networks_VXLAN_White_Paper.pdf)
- [7] Enhanced VXLAN: Who needs Multicast?  
<https://adamraffe.com/2013/06/24/enhanced-vxlan-who-needs-multicast/>
- [8] Virtual Private Network: An Overview  
<https://technet.microsoft.com/en-us/library/bb742566.aspx>
- [9] IP Encapsulating Security Payload (ESP) <https://tools.ietf.org/html/rfc2406>
- [10] OpenVPN Open-source Project  
<https://openvpn.net/index.php/open-source.html>

## *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