

# KOREN SDI 고도화에 대응하는 오픈 플랫폼 Slicing/Visibility API

Document No. 12  
Version 0.1  
Date 2018-08-30  
Author(s) GIST Team

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2018. 08. 30	한정수, 김종원, 박선	
1.0			

본 연구는 한국정보화진흥원(NIA)의 미래네트워크연구시험망(KOREN)  
사업 지원과제의 연구결과로 수행되었음 (17-951-00-001)

This research was one of KOREN projects supported by National  
Information Society Agency (17-951-00-001)

## Contents

### #12. KOREN SDI 고도화에 대응하는 오픈 플랫폼 Slicing/Visibility API

#### Part A. API 및 사용방법

1. API 명세 및 리스트 .....	4
1.1. Slicing API .....	4
1.2. Visibility API .....	9
2. API 사용 예제 .....	17
2.1. Slicing API 사용 예제 .....	17
2.2. Visibility API 사용 예제 .....	19

#### Part B. 설계 및 구현/검증 (Working)

3. KOREN SmartX Platform .....	13
3.1. 목적 및 개요 .....	13
3.2. KOREN SmartX Playground .....	13
3.3. KOREN SmartX Platform으로의 확장 및 개선 .....	16
4. 개발자 맞춤형 인프라 슬라이싱 .....	17
4.1. 개발자 맞춤형 인프라 슬라이싱 개요 .....	17
4.2. 개발자 맞춤형 인프라 슬라이싱 요구사항 .....	17
5. 개발자 맞춤형 인프라 슬라이싱 설계 및 검증 .....	19
5.1. 개발자 맞춤형 인프라 슬라이싱 설계 .....	19
5.2. KOREN SmartX Platform 상의 인프라 슬라이싱 검증 및 활용 .....	25

## 그림 목차

그림 1 Visibility OpenAPI 리스트 .....	4
그림 2 Slicing API 리스트 .....	6
그림 3 Multiview User List Data API 반환값 .....	7
그림 4 Virtual Box List Data API 반환값 .....	8
그림 5 Slicing_Create 호출 화면 .....	9
그림 6 Cloud_Slicing API 수행 장면 .....	10
그림 7 VLAN 별 IP 대역 리스트 .....	11
그림 8 SD_Access_Slicing 호출 화면 .....	11
그림 9 KOREN SmartX Playground 및 관제타워 .....	13
그림 10 GIST KOREN Rack 및 구조 .....	15
그림 11 KOREN SmartX Open Platform 개념도 .....	16
그림 12 개발자 맞춤형 인프라 슬라이싱의 설계 .....	19
그림 13 인프라 슬라이싱을 위한 네트워킹 설계도 .....	20
그림 14 Type O'의 구조 및 Type S와의 네트워킹 연결 구성도 .....	21
그림 15 Type S 스위치-박스의 내부적인 네트워크 구성도 .....	21
그림 16 슬라이싱 ID 및 다른 엔티티와의 관계도 .....	22
그림 17 인슬라이싱 관리 DB 설계 .....	23
그림 18 VLAN 별 가용 네트워크 리스트 .....	23
그림 19 인프라 슬라이싱에 대한 전체적인 워크플로우 순서도 .....	24
그림 20 인프라 슬라이싱 수행을 위한 시나리오 구성 .....	25
그림 21 Slicing_Create 호출 화면 .....	26
그림 22 Cloud_Slicing 호출 화면 .....	26
그림 23 SD_Access_Slicing 호출 화면 .....	27
그림 24 Smart Air IoT-Cloud 서비스 통합 시연 데모 .....	28

## #12. KOREN SDI 고도화에 대응하는 오픈 플랫폼 Slicing/Visibility API

## 1. API 명세 및 리스트

### 1.1. Slicing API

- o KOREN SmartX Playground 환경에서 구성된 인프라를 모든 개발자들이 활용하는 공유 형태로 제공하고 있다. 하지만 개별적인 사용자들이 서로의 실험 및 개발에 영향을 주지 않도록 유지하려면, 사용자 별로 마치 자신만의 독립적인 인프라를 할당 받아 사용하는 것처럼 공유 중심의 전체 인프라의 자원을 나누고, 다른 환경과 격리시키면서 관리하는 기술이 필요하다. 이러한 인프라 즉, 클라우드 및 네트워크를 논리적으로 나누고, 할당하고 관리하는 기술 형태를 인프라 슬라이싱이라고 지칭하며, 사용자가 사용할 수 있도록 인터페이스 집합을 제공하는 것을 Slicing API라고 지칭한다.
- o KOREN SmartX Playground의 Slicing API는 작년의 CLI 형태의 스크립트에서 RESTful 인터페이스 형식으로 개선하면서 버전을 v1.0으로 업그레이드 하였다. Slicing API v1.0 버전에서는 다음 그림 2와 같이 총 9 종류의 API를 제공하고 있다.

SmartX Orchestration API: Example of API lists for supporting Slicing

API Calls	Description
Slicing Create [POST]	Create Slicing and return Slicing ID
Slicing Show [GET]	Show Slicing ID related with Tenant ID
Slicing Delete [DELETE]	Delete Slicing ID related with Tenant ID
Cloud Slicing Create [POST]	Create Cloud Slicing (with creating VM) related with Slicing ID
Cloud Slicing Show [GET]	Show Cloud Slicing related with Slicing ID
Cloud Slicing Delete [DELETE]	Delete the Cloud Slicing (with VM) related with Slicing ID
SD-Access Slicing Create [POST]	Create SD-access Slicing related with Slicing ID
SD-Access Slicing Show [GET]	Show SD-access Slicing related with Slicing ID
SD-Access Slicing Delete [DELETE]	Delete SD-access Slicing related with Slicing ID

그림 1 Slicing API 리스트

- o Slicing Create [POST]: 사용자가 자신의 ID에 슬라이싱 ID를 부여받기 위한 작업이다. 다음과 같이 사용자가 자신의 자원을 슬라이싱하기 위한, 슬라이싱 ID를 발급 받는 API이다. API를 수행하면, 슬라이싱 ID를 리턴 받는다. API 수행을 위한 예제는 다음과 같다.

**\$ curl -X POST http://IP\_ADDRESS:6126/slices -u USER:PASSWORD**

- o IP\_ADDRESS는 P+O Center의 IP를 입력하면 되고, USER와 PASSWORD는 OpenStack Keystone에서 할당받은 유저 ID와 비밀번호를 입력하면 된다.

- o Slicing Show [GET]: 사용자가 자신이 만든 슬라이싱 ID에 대한 정보들을 확인할 수 있는 작업이다. 자신의 만든 슬라이싱 ID와 이에 대한 정보들이 리스트 형태로 제공된다. API 수행을 위한 예제는 다음과 같다.

```
$ curl http://IP_ADDRESS:6126/slices -u USER:PASSWORD
```

- o Slicing Delete [DELETE]: 사용자가 부여받은 슬라이싱 ID를 삭제하는 작업이다. 슬라이싱 ID를 삭제하기 전에 해당 ID를 통해 수행한 Cloud Slicing 및 SD-Access Slicing은 사전에 삭제하여야 한다. API 수행을 위한 예제는 다음과 같다.

```
$ curl -X DELETE -H "Content-Type: application/json" -d '{"slice_id": "SLICE_ID"}' http://IP_ADDRESS:6126/slices -u USER:PASSWORD
```

- o 위의 예제에서 사용자의 입력을 json 형태로 함께 API 요청을 수행하여야 한다. JSON 파일에 들어가는 입력 사항은 다음과 같다.

```
{"slice_id": "SLICE_ID"}  
slice_id = 삭제할 슬라이싱 ID (ex., 970)
```

- o Cloud Slicing Create [POST]: 사용자의 슬라이싱 ID에 연결되는 Cloud Slicing을 생성하는 작업이다. Cloud Slicing을 만들 때, 슬라이싱 ID에 연결되는 가상 머신 인스턴스를 생성하는 과정이 추가된다. API를 수행하기 위한 예제는 다음과 같다.

```
$ curl -X POST -H "Content-Type: application/json" -d '{"slice_id": "SLICE ID",  
"region": "REGION", "flavor": "FLAVOR", "image": "IMAGE",  
"instance_name": "INSTANCE NAME", "key_name": "KEY NAME"}' http://IP_ADDRESS:6126/cloud_slices -u USER:PASSWORD
```

- o JSON 데이터에 들어가는 입력 사항은 다음과 같다.

```
{"slice_id": "SLICE ID", "region": "REGION", "flavor": "FLAVOR", "image":  
"IMAGE", "instance_name": "INSTANCE NAME", "key_name": "KEY NAME"}  
slice_id = 사용자가 보유한 슬라이싱 ID (ex., 970)  
region = 가상머신 인스턴스를 생성할 오픈스택의 리전 (ex., KR-GISTC)
```

flavor = 가상머신 인스턴스의 크기 (ex., m1.small)  
image = 가상머신 인스턴스의 이미지 (ex., cirros)  
instance\_name = 가상머신 인스턴스의 이름 (ex., VM\_test)  
key\_name = 가상머신 인스턴스를 접속하기 위한 ssh 키 페어 이름 (ex., ssh\_key)

- o API 수행을 요청하면, 슬라이싱 ID에 대응하는 네트워크가 없다면 생성하고, 이를 연결하는 가상머신 인스턴스를 생성한다. 생성이 완료가 되면 리턴값으로 Instance ID와 해당 가상머신의 IP 그리고 슬라이싱 ID를 리턴 받는다. 가상머신에 접속하기 위해서는 SSH 키를 통한 접속을 요구하며, 기본적으로 가상머신에는 Floating IP가 할당 되지 않았기 때문에 바로 접근을 할 수 없다. 따라서 가상머신이 생성된 후에는 대시보드를 통해서 Floating IP 할당받아 접속을 시도하면 된다.
- o Cloud Slicing Show [GET]: 사용자가 보유하고 있는 클라우드 슬라이싱에 대한 리스트를 보여주는 API 요청이다. API 요청 형식은 다음과 같다.

**\$ curl http://IP\_ADDRESS:6126/cloud\_slices -u USER:PASSWORD**

- o API 요청이 완료되면, 다음과 같은 JSON 형태의 데이터를 리턴 받는다.

```
[ {"slice_id": "SLICE_ID", "region": "REGION", "flavor": "FRAVOR", "image":  
  "IMAGE", "instance_name": "INSTANCE_NAME", "key_name": "SSH_NAME"},  
  ... ]
```

- o Cloud Slicing Show [DELETE]: 사용자가 보유하고 있는 클라우드 슬라이싱을 삭제하기 위한 API 요청이다. API 요청 형식은 다음과 같다.

**\$ curl -X DELETE -H "Content-type: application/json" -d '{"instance\_id":  
 "INSTANCE\_ID", "region": "REGION"}' http://IP\_ADDRESS:6126/cloud\_slices  
-u USER:PASSWORD**

- o SD-Access Slicing Create [CREATE]: 사용자의 슬라이싱 ID에 연결되는 IoT 디바이스를 등록하기 위한 SD-Access 슬라이싱을 생성하는 API이다. API 수행을 위한 요청 형식은 다음과 같다.

**\$ curl -X POST -H "Content-type: application/json" -d '{"slice\_id": "SLICE\_ID",  
 "mac": "DEVICE\_MAC", "location": "BOX\_LOCATION", "ip": "DEVICE\_IP"}'**



[http://IP\\_ADDRESS:6126/access\\_slices -u USER:PASSWORD](http://IP_ADDRESS:6126/access_slices -u USER:PASSWORD)

- o 입력사항으로 들어가는 JSON 데이터 형식은 다음과 같다.

```
{"slice_id": "SLICE_ID", "mac": "DEVICE_MAC", "location": "BOX_LOCATION",  
  "ip": "DEVICE_IP"}
```

slice\_id = 사용자의 슬라이싱 ID

mac = 등록할 IoT 디바이스의 MAC 주소

location = IoT 디바이스가 연결된 Type O 박스의 위치

ip = IOT 디바이스의 IP 주소

- o 중요한 포인트는 IoT 기기에 슬라이싱을 활용하기 위해서는 슬라이싱 ID에 따른 IP 주소 할당 규칙을 준수해야 한다. IP 주소 네트워크 대역은 아래 그림과 같으며, IP 주소는 IoT 기기의 경우 1~100번 이내의 IP를 할당하여야 한다.

```
951 : 192.168.51.0/24  
952 : 192.168.52.0/24  
953 : 192.168.53.0/24  
954 : 192.168.54.0/24  
955 : 192.168.55.0/24  
956 : 192.168.56.0/24  
957 : 192.168.57.0/24  
958 : 192.168.58.0/24  
959 : 192.168.59.0/24  
960 : 192.168.60.0/24  
961 : 192.168.61.0/24  
962 : 192.168.62.0/24  
963 : 192.168.63.0/24  
964 : 192.168.64.0/24  
965 : 192.168.65.0/24  
966 : 192.168.66.0/24  
967 : 192.168.67.0/24  
968 : 192.168.68.0/24  
969 : 192.168.69.0/24  
970 : 192.168.70.0/24  
971 : 192.168.71.0/24  
972 : 192.168.72.0/24  
973 : 192.168.73.0/24  
974 : 192.168.74.0/24  
975 : 192.168.75.0/24  
976 : 192.168.76.0/24  
977 : 192.168.77.0/24  
978 : 192.168.78.0/24  
979 : 192.168.79.0/24  
980 : 192.168.80.0/24
```

그림 2 슬라이싱 ID 별  
IP 대역 리스트

- o 예를 들어서 슬라이싱 ID를 951번을 부여받았다면, 슬라이싱 ID에 대응하는 IoT 기기들은 IP를 192.168.51.1 ~ 192.168.51.100 이하의 IP로 할당을 해야 한다.

- o SD-Access Slicing Show [GET]: 사용자가 보유하고 있는 SD-Access 슬라이싱을 삭제하기 위한 API 요청이다. API 요청 형식은 다음과 같다.

**\$ curl http://IP\_ADDRESS:6126/access\_slices -u USER:PASSWORD**

- o API 요청이 완료되면, 다음과 같은 JSON 형태의 데이터를 리턴 받는다.

```
[ {"MAC": "IoT_DEVICE_MAC", "IP": "IoT_DEVICE_IP", "Slicing_ID":  
  "SLICING_ID", "Intent": "ONOS_INTENT", "direction": "FLOW_DIRECTION"}, ...  
]
```

- o SD-Access Slicing Show [DELETE]: 사용자가 보유하고 있는 SD-Access 슬라이싱을 삭제하기 위한 API 요청이다. API 요청 형식은 다음과 같다.

**\$ curl -X DELETE -H "Content-type: application/json" -d '{"slice\_id":  
 "SLICE\_ID", "mac": "IoT\_DEVICE\_MAC", "location": "BOX\_LOCATION"}'  
http://IP\_ADDRESS:6126/cloud\_slices -u USER:PASSWORD**

- o 입력사항으로 들어가는 JSON 데이터 형식은 다음과 같다.

```
{"slice_id": "SLICE_ID", "mac": "IoT_DEVICE_MAC", "location":  
  "BOX_LOCATION"}
```

slice\_id = 사용자의 슬라이싱 ID

mac = 등록할 IoT 디바이스의 MAC 주소

location = IoT 디바이스가 연결된 Type O 박스의 위치

## 1.2. Visibility API

- 다양한 개발자들이 KOREN SmartX 오픈 플랫폼을 활용해 KOREN SmartX Playground 인프라 상의 자원들을 이용하여 다양한 어플리케이션을 효율적으로 개발 수 있도록 자원 집합의 토폴로지 및 상태 정보를 제공할 필요가 있다. 이를 위해 KOREN SmartX 하부 오픈 플랫폼에 위치하는 Visibility Center를 통해 어플리케이션 개발자가 자원간의 토폴로지 및 자원 상태를 어플리케이션 개발 단계에서 활용할 수 있도록 Visibility OpenAPI를 제공한다. 또한 수집된 Visibility Data는 가시화 Web UI 형태로 사용자에게 제공한다. 이를 통해 사용자는 인프라 운용 및 서비스 운용 단계에서도 시각화 정보를 활용하여 SmartX Playgorund에 위치한 자원들을 효과적으로 활용할 수 있다. 개발된 UI는 SmartX Playground 인프라를 구성하는 IoT, SD-Acess, Cloud, SD-WAN, SDN/Cloud 기반 관제타워 장비로 연결되는 SmartX Playgournd 인프라 상의 모든 계층의 네트워크 토폴로지 및 자원의 상태를 가시화 한다. 사용자는 종단에 위치한 IoT 장비를 포함하여 SmartX Playground 인프라에 연결된 모든 장비의 연결 상태를 실시간으로 감시하여, 효율적이고 신속하게 장비들의 이상 유무를 확인하여 조치할 수 있다. 또한 사용자는 모니터링 되는 정보들을 기반으로 새로운 서비스를 구현할 수 있다.
- 본 문서에서 제공하는 Visibility API는 웹 URL을 활용하여 호출하고 JSON 형태의 결과값을 반환받는 형태의 RESTful API를 제공한다. KOREN Repository에서 제공하는 Visibility API는 다음 그림 3, 4와 같이 Visibility Data API와 Visibility Visualization API로 두 가지 형태의 API를 지원한다.

API <sup>Ⓜ</sup>	Description <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3020/getcontrollersstatus">http://103.22.221.56:3020/getcontrollersstatus</a> <sup>Ⓜ</sup>	Returns list of platform controllers with associated status. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3020/getoverlaylatency">http://103.22.221.56:3020/getoverlaylatency</a> <sup>Ⓜ</sup>	Returns overlay network performance metrics e.g. latency. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3020/getboxesstatus">http://103.22.221.56:3020/getboxesstatus</a> <sup>Ⓜ</sup>	Returns a list of SmartX Boxes with current access status. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3020/getvminstancesstatus">http://103.22.221.56:3020/getvminstancesstatus</a> <sup>Ⓜ</sup>	Returns a list of OpenStack virtual machines with current access status. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3020/getiothostsstatus">http://103.22.221.56:3020/getiothostsstatus</a> <sup>Ⓜ</sup>	Returns a list of IoT Hosts connected with SmartX Boxes. <sup>Ⓜ</sup>

그림 3 Visibility Data API

API <sup>Ⓜ</sup>	Description <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/teinint">http://103.22.221.56:3011/teinint</a> <sup>Ⓜ</sup>	Returns two built onion-ring visualization. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/ren">http://103.22.221.56:3011/ren</a> <sup>Ⓜ</sup>	Returns three built onion-ring visualization. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/sites">http://103.22.221.56:3011/sites</a> <sup>Ⓜ</sup>	Returns four built onion-ring visualization. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/boxes">http://103.22.221.56:3011/boxes</a> <sup>Ⓜ</sup>	Returns five built onion-ring visualization. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/vms">http://103.22.221.56:3011/vms</a> <sup>Ⓜ</sup>	Returns six built onion-ring visualization. <sup>Ⓜ</sup>
<a href="http://103.22.221.56:3011/flows">http://103.22.221.56:3011/flows</a> <sup>Ⓜ</sup>	Returns seven built onion-ring visualization. <sup>Ⓜ</sup>

그림 4 Visibility Visualization API

- o Visibility API는 RESTful API의 GET Method를 활용하는 API들로 구성되어 있으며, KOREN SmartX Playground에서 수집되는 다양한 데이터를 반환해주는 역할을 하거나 가시화를 구성하기 위한 데이터 형태를 리턴하기도 한다. API 수행 예제는 다음과 같다.
- o Get Controller Status [GET]: KOREN SmartX Playground를 구성하기 위한 각종 컨트롤러의 정보들의 상태들을 리턴 하는 명령이다. API 수행 형태는 다음과 같다.

**\$ curl http://IP\_ADDRESS:3020/getcontrollersstatus**

- o 리턴 데이터의 형태는 다음과 같다.

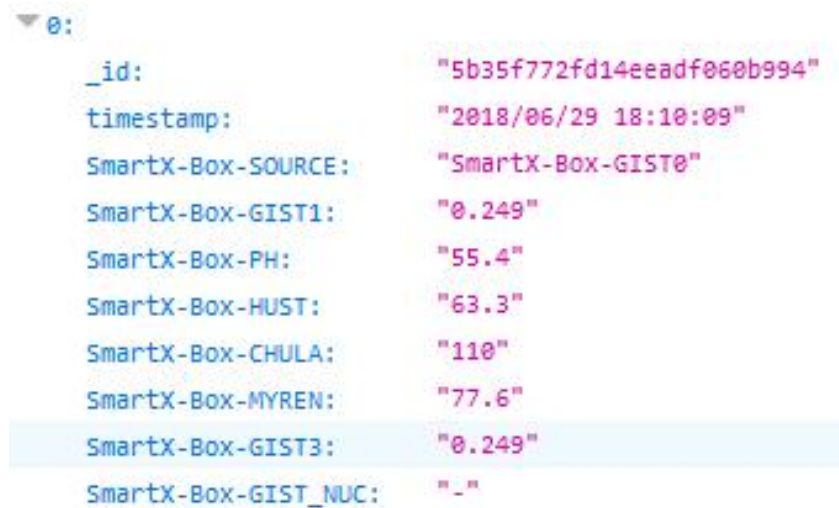
```
▼ 1:
  _id: "5989b5f9979044e26d6385d8"
  controllerIP: "103.22.221.152"
  controllerName: "SD-WAN Controller"
  controllerType: "SD-WAN"
  controllerSoftware: "ONOS"
  controllerStatus: "GREEN"
  controllerUser: "netcs"
  controllerPass: "netmedia"
```

그림 5 Get Controller Status API 리턴 JSON 데이터

- o Get Overlay Latency [GET]: KOREN SmartX Playground를 구성하고 있는 박스 사이 간의 지연성을 측정한 데이터를 리턴해주는 API이다. API 수행 형태는 다음과 같다.

**\$ curl http://IP\_ADDRESS:3020/getoverlaylatency**

- o 리턴 데이터의 형태는 다음과 같다.



```

0:
  _id: "5b35f772fd14eeadf060b994"
  timestamp: "2018/06/29 18:10:09"
  SmartX-Box-SOURCE: "SmartX-Box-GIST0"
  SmartX-Box-GIST1: "0.249"
  SmartX-Box-PH: "55.4"
  SmartX-Box-HUST: "63.3"
  SmartX-Box-CHULA: "110"
  SmartX-Box-MYREN: "77.6"
  SmartX-Box-GIST3: "0.249"
  SmartX-Box-GIST_NUC: "-"

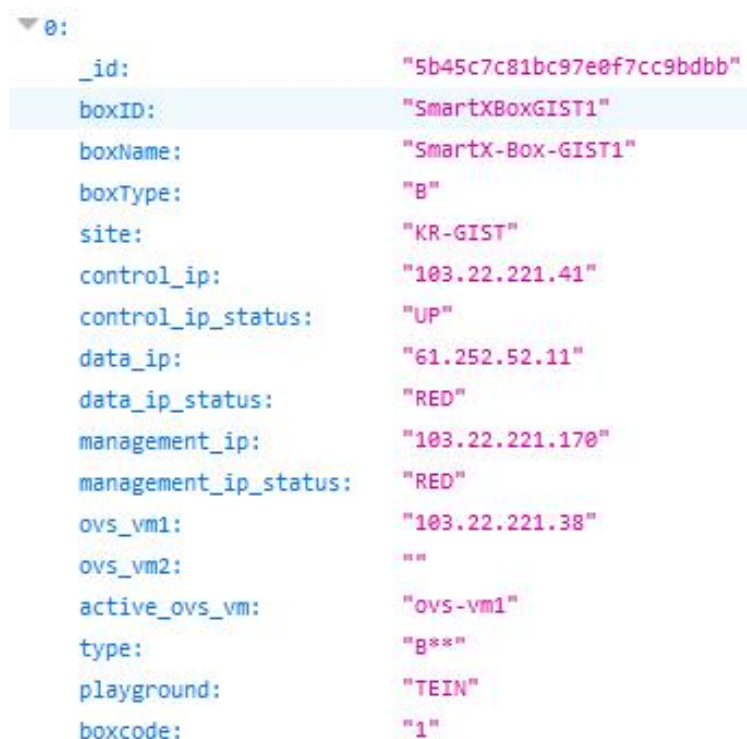
```

그림 6 Get Overlay Latency API 리턴 JSON 데이터

- o Get Boxes Status [GET]: KOREN SmartX Playground를 구성하고 있는 박스들의 상태 정보들을 반환하는 API이다. API 사용 예제는 다음과 같다.

\$ curl http://IP\_ADDRESS:3020/getboxesstatus

- o 리턴 데이터의 형태는 다음과 같다.



```

0:
  _id: "5b45c7c81bc97e0f7cc9bdbb"
  boxID: "SmartXBoxGIST1"
  boxName: "SmartX-Box-GIST1"
  boxType: "B"
  site: "KR-GIST"
  control_ip: "103.22.221.41"
  control_ip_status: "UP"
  data_ip: "61.252.52.11"
  data_ip_status: "RED"
  management_ip: "103.22.221.170"
  management_ip_status: "RED"
  ovs_vm1: "103.22.221.38"
  ovs_vm2: ""
  active_ovs_vm: "ovs-vm1"
  type: "B**"
  playground: "TEIN"
  boxcode: "1"

```

그림 7 Get Boxes Status API 리턴 JSON 데이터

- o Get VM Instance Status [GET]: KOREN SmartX Playground를 구성하고 있는 박스 내의 생성된 가상머신들의 상태 정보들을 반환하는 API이다. API 사용 예제는 다음과 같다.

**\$ curl http://IP\_ADDRESS:3020/getvminstancesstatus**

- o 리턴 데이터의 형태는 다음과 같다.

```
▼ 1:
  _id:      "5bf21cca38ced0c2d4609747"
  name:     "USCM-gist-vm"
  uuid:     "38a385cb-27ca-447b-8d81-48c1eb04335b"
  vlanid:   ""
  ostenantid: ""
  box:      "SmartX-Box-GIST3"
  state:    "SHUTOFF"
```

그림 8 Get VM Instance Status API 리턴 데이터

- o Get IoT Host Status [GET]: KOREN SmartX Playground에 연결된 IoT 디바이스들의 대한 상태 정보들을 반환하는 API이다. API 사용 예제는 다음과 같다.

**\$ curl http://IP\_ADDRESS:3020/getiothostsstatus**

- o 리턴 데이터의 형태는 다음과 같다.

```
▼ 0:
  _id:      "5a38846581ad15bc04fd600e"
  hostID:   "651"
  macaddress: "12:32:fa:a3:21:21:43:d2"
  vlanid:   "101"
  configured: "true"
  ipaddress: "192.168.101.1"
  box:      "GIST-Access-Box"
```

그림 9 Get IoT Host Status API 리턴 데이터

- o 2-level Onion Ring Visualization [GET]: KOREN SmartX Playground의 2-레벨 양파형태의 가시화를 리턴해주는 API이다. 해당 주소를 접근하면 양파형 가시화를 볼 수 있으며, API 요청을 진행하면 양파형 가시화를 그리기 위한 데이터 포맷을 리턴 받을 수 있다. 다음은 해당 URL에 접근한 화면이다.

\$ [http://IP\\_ADDRESS:3011/teinint](http://IP_ADDRESS:3011/teinint)

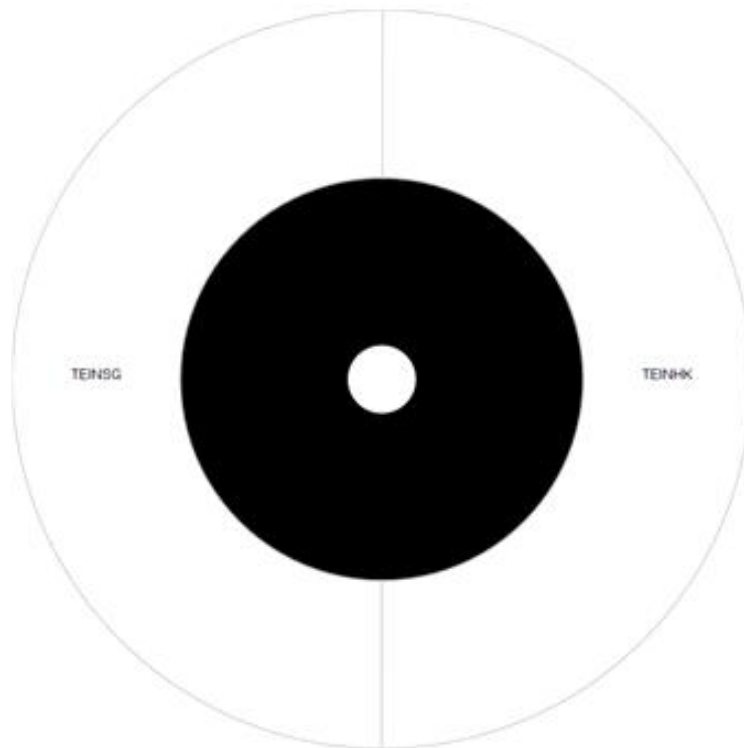


그림 10 2-레벨 양파형태의 가시화

- o 3-level Onion Ring Visualization [GET]: KOREN SmartX Playground의 3-레벨 양파형태의 가시화를 리턴해주는 API이다. 해당 주소를 접근하면 양파형 가시화를 볼 수 있으며, API 요청을 진행하면 양파형 가시화를 그리기 위한 데이터 포맷을 리턴 받을 수 있다. 다음은 해당 URL에 접근한 화면이다.

\$ [http://IP\\_ADDRESS:3011/en](http://IP_ADDRESS:3011/en)

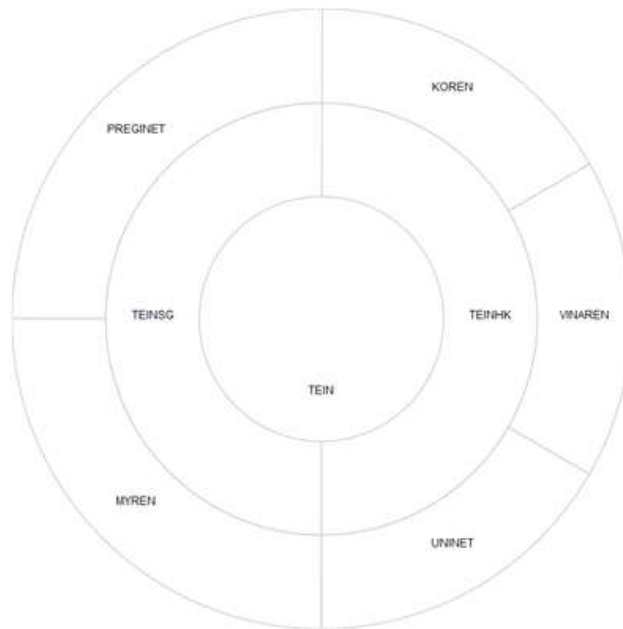


그림 11 3-레벨 양파형 가시화

- o 4-level Onion Ring Visualization [GET]: KOREN SmartX Playground의 4-레벨 양파형태의 가시화를 리턴해주는 API이다. 해당 주소를 접근하면 양파형 가시화를 볼 수 있으며, API 요청을 진행하면 양파형 가시화를 그리기 위한 데이터 포맷을 리턴 받을 수 있다. 다음은 해당 URL에 접근한 화면이다.

\$ [http://IP\\_ADDRESS:3011/sites](http://IP_ADDRESS:3011/sites)



그림 12 4-레벨 양파형 가시화



- o 5-level Onion Ring Visualization [GET]: KOREN SmartX Playground의 5-레벨 양파형태의 가시화를 리턴해주는 API이다. 해당 주소를 접근하면 양파형 가시화를 볼 수 있으며, API 요청을 진행하면 양파형 가시화를 그리기 위한 데이터 포맷을 리턴 받을 수 있다. 다음은 해당 URL에 접근한 화면이다.

\$ [http://IP\\_ADDRESS:3011/boxes](http://IP_ADDRESS:3011/boxes)

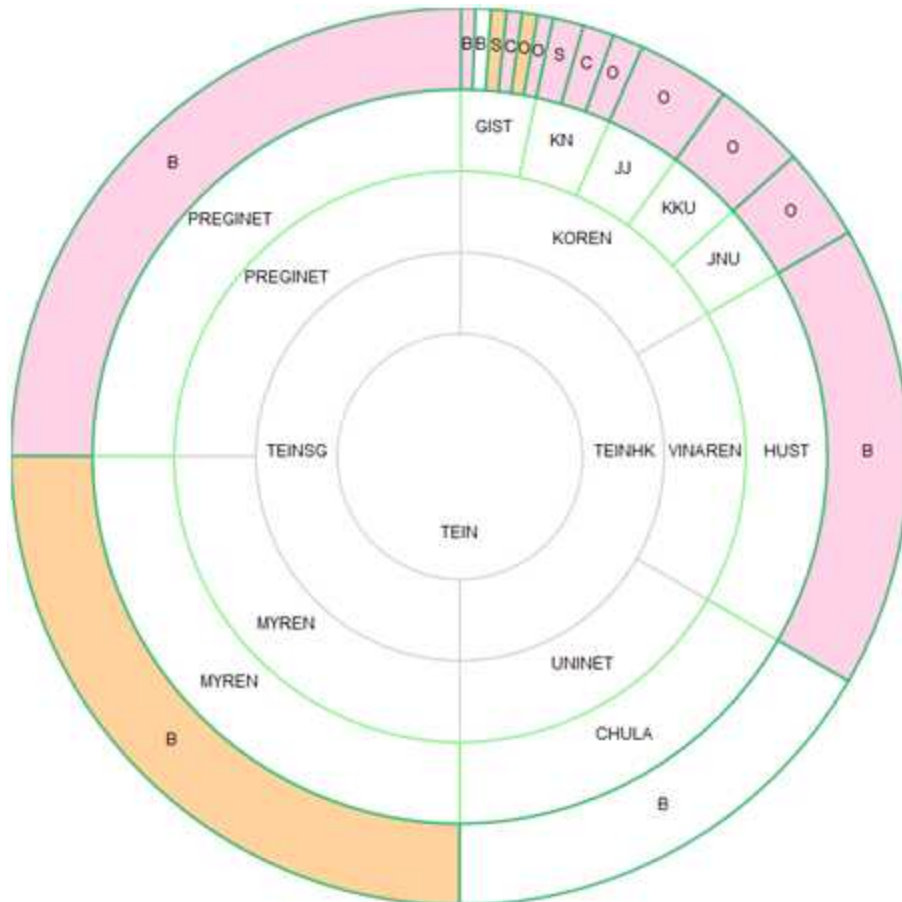
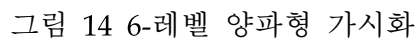


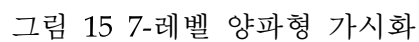
그림 13 5-레벨 양파형 가시화

- o 6-level Onion Ring Visualization [GET]: KOREN SmartX Playground의 6-레벨 양파형태의 가시화를 리턴해주는 API이다. 해당 주소를 접근하면 양파형 가시화를 볼 수 있으며, API 요청을 진행하면 양파형 가시화를 그리기 위한 데이터 포맷을 리턴 받을 수 있다. 다음은 해당 URL에 접근한 화면이다.

\$ [http://IP\\_ADDRESS:3011/vms](http://IP_ADDRESS:3011/vms)



- ```
$ http://IP_ADDRESS:3011/flows
```



## 2. API 사용 예제

### 2.1. Slicing API 사용 예제

- o 이번 절에서는 제공하고 있는 Slicing API를 통해 클라우드의 가상머신과 IoT 간 슬라이싱 연결에 대한 예제를 소개한다. 슬라이싱의 적용대상은 Type S, C, O를 구성하고 있는 특별한 인프라 환경인 KOREN SmartX Playground를 대상으로 하고 있다.
- o 사용자는 슬라이싱을 수행해 가상머신 인스턴스와 IoT 디바이스를 슬라이싱 중심으로 연결하기 위해서 필요한 API의 수행 순서는 다음과 같다.

1. Slicing Create
2. Cloud Slicing Create
3. SD-Access Slicing

- o 위의 순서대로 수행하기 위해서 먼저 Slicing Create API를 수행한다. 슬라이싱 ID를 부여받기 위해서 KOREN SmartX Playground에서 사용자 인증을 담당하고 있는 오픈스택 키스톤의 ID가 필요하다. 이번 예제에서는 demo ID와 임시적으로 secrete 패스워드를 가진 유저가 슬라이싱을 수행하는 것을 보여주고 있다. 다음과 같이 API 명령을 수행하면, 슬라이싱 ID가 생성이 되며, 생성 결과는 다음과 같다.

**\$ curl -X POST http://103.22.221.51:6126/slices -u demo:secrete**

```
root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs# curl -X POST http://103.2
2.221.51:6126/slices -u demo:secrete
{
  "Slice_ID": "954"
}
root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs#
```

그림 16 Slicing Create API 수행 결과

- o 두 번 째로 Cloud Slicing 생성을 수행한다. 이는 부여받은 슬라이싱 ID에 대응되는 클라우드 슬라이싱을 생성하는 부분이다. 이를 통해서 부가적으로 슬라이싱 ID에 연결되는 가상머신을 생성한다. 다음 명령어는 demo라는 유저가 GIST 내에 있는 Type C 박스에 m1.small이라는 flavor를 가지는 cirros 이미지를 활용한 test라는 이름을 갖고 hjs라는 ssh 키페어로 접속 가능한 가상머신 인스턴스를 생성하는 과정이다.

**\$ curl -X POST -H "Content-Type: application/json" -d '{"slice\_id": "954", "region": "KR-GISTC", "flavor": "m1.small", "image": "cirros", "instance\_name": "test", "key\_name": "hjs"}' http://103.22.221.51:6126/cloud\_slices -u demo:secrete**

```

root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs# curl -X POST -H "Content-Type: application/json" -d '{"slice_id": "954", "region": "KR-GISTC", "flavor": "ml.small", "image": "cirros", "instance_name": "test", "key_name": "hjs"}' http://103.22.221.51:6126/cloud_slices -u demo:secrete
{"Instance_ID": "6906a8d9-e448-49b2-b74b-12acc0d2d54c", "IP": "192.168.54.107", "Slicing_ID": "954"}
root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs#

```

그림 17 Cloud Slicing Create API 수행 결과

- o 위 그림에서 해당 요청에 대응되는 가상머신이 생성된 것을 확인할 수 있으며, 오픈스택 대시보드 URL을 접속해 해당 인스턴스가 정상적으로 생성된 것을 가시적으로 확인할 수 있다. 추가적으로 가상머신 인스턴스에 접속하기 위해서는 Floating IP 부여가 필수적이다. 슬라이싱 단계에서는 이러한 과정이 생략되므로 반드시 오픈스택 대시보드 URL를 통해 가상머신에 대한 Floating IP 부여를 수행한다.
- o 마지막 단계로 SD-Access Slicing을 수행한다. 이를 통해 사용자는 자신의 IoT 기기들의 네트워킹 연결 작업을 진행한다. 먼저, 사용자는 연결하고자 하는 IoT 기기를 Type O에 물리적으로 연결하는 작업을 진행해야 한다. 무선 혹은 유선을 통해 연결이 되었다면, IoT 기기에 접속을 시도하여 IP 대역을 수동으로 설정하여야 한다. 위의 예제에서는 Slicing ID가 954이므로 그림 2에서 954에 맞는 대역인 192.168.54.0/24 대역에서 192.168.54.1~192.168.54.100 범위에 맞는 IP로 할당하여야 한다.
- o Type O에 IoT 기기에 대한 물리적인 연결 및 네트워크 설정이 끝나면, SD-Access Create API 요청을 수행한다. 다음 예제에서는 MAC이 AA:BB:CC:DD:EE:FF를 가지고 IP가 192.168.54.50을 가진 광주에 위치한 Type O 박스의 붙은 IoT 기기에 대한 SD-Access 슬라이싱 생성을 수행한다.

```

$ curl -X POST -H "Content-Type: application/json" -d '{"slice_id": "954", "mac": "aa:bb:cc:dd:ee:ff", "location": "GJ", "ip": "192.168.54.50"}' http://103.22.221.51:6126/access_slices -u demo:secrete

```

```

root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs# curl -X POST -H "Content-Type: application/json" -d '{"slice_id": "954", "mac": "aa:bb:cc:dd:ee:ff", "location": "GJ", "ip": "192.168.54.50"}' http://103.22.221.51:6126/access_slices -u demo:secrete
[{"MAC": "aa:bb:cc:dd:ee:ff", "IP": "192.168.54.50", "Slicing_ID": "954", "Intent": "204", "direction": "cloud"}, {"MAC": "aa:bb:cc:dd:ee:ff", "IP": "192.168.54.50", "Slicing_ID": "954", "Intent": "207", "direction": "IoT"}]
root@POCenter:/home/netcs/SmartX_Open_Platform/Orchestration_OpenAPIs#

```

그림 18 SD-Access Slicing Create API 수행 결과

- o 그림 18은 SD-Access Slicing Create API의 수행 결과를 나타낸다. ONOS 컨트롤러에서 플로우를 관장하는 단위인 Intent는 한 기기 당 양방향의 쌍으로 진행

이 된다. IoT 기기에서 나가는 패킷의 경우, Slicing ID에 맞는 VLAN을 태깅하는 플로우 룰을 내려주는 Intent를 설치하며, IoT 기기로 들어가는 패킷에 대해서는 태깅된 VLAN ID를 벗겨내는 플로우 룰을 내려주는 Intent를 설치하게 된다. 이러한 작업이 끝나면, 사설 IP를 가진 IoT 기기들과 가상머신들이 서로 간의 통신이 가능하며, 사용자의 IoT 기기가 어떤 위치에 있든 간에, Type O 박스를 통해 연결되면, 서로간의 통신이 가능하다.

## 2.2. Visibility API 사용 예제

- 본 문서에서 제공하고 있는 Visibility API는 웹 URL 형태의 RESTful API로 구현하였기 때문에, KOREN 망을 접근할 수 있는 사용자들은 원격에서 바로 API를 호출 가능하다. 상황에 맞게 그림 3 과 4에 있는 적절한 API를 GET 메소드 형태로 호출한다.

- 리눅스 계열의 Ubuntu 환경에서는 다음과 같은 명령어를 통해 API 수행이 가능하다.

```
$ curl http://IP_ADDRESS:8181/API_이름
```

- 예를 들어 KOREN SmartX Playground를 관장하는 컨트롤러에 대한 정보를 얻고 싶으면 다음과 같이 명령을 입력하면 된다.

```
$ curl http://103.22.221.55:8181/getcontrollersstatus
```

- 위와 같이 수행하면 JSON 형태의 결과 값을 그림 19와 같이 확인할 수 있다.

```
1:
  _id: "5989b5f9979044e26d6385d8"
  controllerIP: "103.22.221.152"
  controllerName: "SD-WAN Controller"
  controllerType: "SD-WAN"
  controllerSoftware: "ONOS"
  controllerStatus: "GREEN"
  controllerUser: "netcs"
  controllerPass: "netmedia"
```

그림 19 API 수행 결과



### 3. KOREN SmartX Platform

#### 3.1. 목적 및 개요

- 본 문서에서는 여러 사이트에 분산되어 있는 박스들을 활용하여 구성하고 있는 KOREN SmartX Platform에서 여러 개발자들이 공유되는 환경 속에서 개발자 맞춤형 인프라를 제공하기 위해서 인프라 (클라우드 및 네트워크)를 논리적으로 할당 하고 고립시켜 제공해주는 슬라이싱에 대해 설계 및 구현하고 검증하는 과정을 다루고 있다. 이러한 슬라이싱은 상부 플랫폼의 관할 센터 중 하나인 Orchestration 센터에서 API 형태로 개발자에게 제공해 손 쉽게 자신만의 인프라를 KOREN SmartX Platform 환경 상에서 구성할 수 있도록 지원해준다. 본 절에서는 KOREN SmartX Platform에 대해 먼저 설명을 하고, 다음절부터는 인프라 슬라이싱 개념 및 요구 사항, 그리고 디자인 설계에서부터 검증까지 내용을 다루고자 한다.

#### 3.2. KOREN SmartX Playground

- Playground란 사용자들이 하고 싶은 실험 및 환경을 자유롭게 할 수 있게 제공되는 공간을 상징하고 사용자들이 Playground 위에서 하는 다양한 실증 실험들은 Play로 표현할 수 있다.

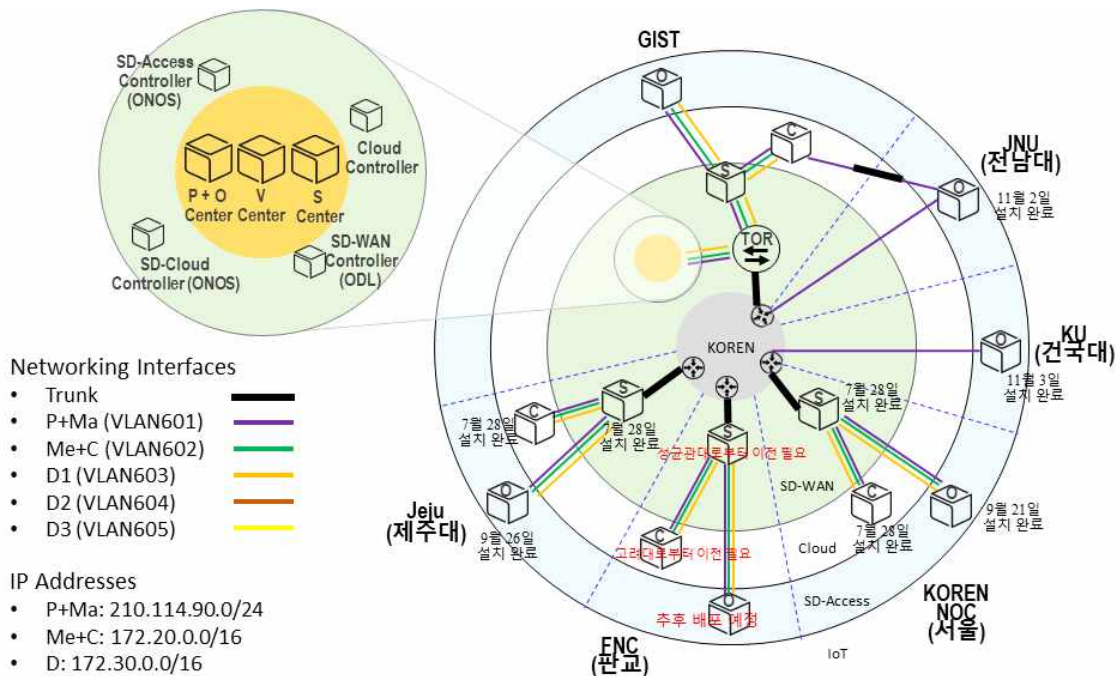


그림 20 KOREN SmartX Playground 및 관제타워

- o KOREN SmartX Playground는 사용자들이 원하는 실증 실험을 할 수 있도록 다양한 역할을 지원하는 박스들을 유연하게 엮어 컴퓨팅, 네트워킹, 스토리지로 대표되는 IT 자원들을 알맞게 제공할 수 있는 물리적인 인프라를 의미한다. 현재 KOREN SmartX Playground를 구성하는 인프라는 크게 그림 2과 같이 4가지 사이트로 구성되어 있다. GIST, KOREN NOC, 판교 FNC, Jeju 사이트로 구성된 인프라들을 구성하고 있는 Type O, Type C, Type S 박스들과 이를 관제하고 있는 센터 및 컨트롤러로 구성된다. 그리고 이러한 사이트들 사이의 내부형태를 보면, 각 벨트 별로 영역이 존재하며 (SD-WAN, Distributed Cloud, SD-Access) 자세한 설명은 아래와 같다.
- o SD-WAN (Software-defined Wide Area Network: 소프트웨어-정의 광역 통신망)은 단일 사용자 집단에 소속된 분산된 사이트들을 연결하는 WAN을 소프트웨어-정의 기반으로 관제하는 방식이며, 고가의 단일 MPLS (Multi Protocol Label Switching) WAN에 의존하지 않고 저렴한 다수 WAN들을 통합하여 분산된 사이트들을 연결하는 사설 오버레이 네트워킹을 지원하는 영역을 말한다.
- o Distributed Cloud (분산 클라우드)는 단일 사용자 집단에 소속된 분산된 사이트들을 연결하는 통합적으로 제공하는 클라우드 영역을 말한다.
- o SD-Access (Software-defined Access: 소프트웨어-정의 액세스)는 분산된 IoT 디바이스들을 연결하면서, 다양한 통신 인터페이스를 사용하는 유/무선 IoT 네트워크들을 소프트웨어-정의 기반으로 관제하는 방식을 말한다. IoT 디바이스들과 근접한 위치에서 IoT-Cloud 형태로 분산 클라우드와의 안정된 연결을 지원하면서, SD-Access 대응 서버-스위치들의 계산/저장 자원들을 활용하여 IoT 데이터에 대한 전처리를 담당하는 영역을 말한다.
- o 이러한 영역들은 각자의 역할을 수행하기 위해 필요한 기능들을 제공하고 있는 박스 (서버나 스위치가 어떠한 기능을 담고 있는 하드웨어적인 표현)가 필요하게 된다. 예를 들면, SD-WAN 영역에서는 분산된 사이트들을 연결하는 오버레이 네트워킹 및 스위칭 기능을 제공하는 박스가 필요하며, 분산 클라우드 영역에서는 클라우드 기능을 제공할 수 있는 박스가 필요하며, SD-Access 영역에서는 IoT 디바이스들이 연결될 수 있는 다양한 통신 인터페이스를 담고 있는 박스가 필요하다. 이렇게 각 영역에서 필요한 기능들을 담아서 제공하는 박스를 Type-S (SD-WAN 영역을 커버), Type-C (Cloud 영역을 커버), Type-O (SD-Access 영역을 커버)라고 명칭하며 이러한 영역에 대응하는 박스들을 구성하고 있다.

- o 이렇게 주어진 박스들을 한 단계 위에 레벨에서 관장할 수 있는 요소가 필요하다. 이러한 역할을 하기 위해 P+O (Provisioning & Orchestration) 센터와 Visibility 센터 그리고 Cloud 및 SDN 제어기들이 이러한 요소를 담당하고 있다. P+O Center에서는 분산된 사이트들에 박스들을 원격으로 세팅 및 준비하며 개발자들이 사용할 수 있는 여러 기능들을 제공하는 역할을 하며, Visibility 센터는 이러한 인프라들의 상태를 가시적으로 파악할 수 있도록 가시화 정보를 수집하고 이러한 데이터들을 가시화할 수 있는 데이터로 변환해 운영자에게 제공해주는 역할을 한다. 그리고 Cloud 및 SDN 제어기들은 실제적으로 이러한 센터들이 시키는 주요한 일들을 직접 처리하는 역할을 하며, Cloud 및 SDN에 관련된 전반적인 일을 센터들의 지시를 받아 수행하는 역할을 한다.

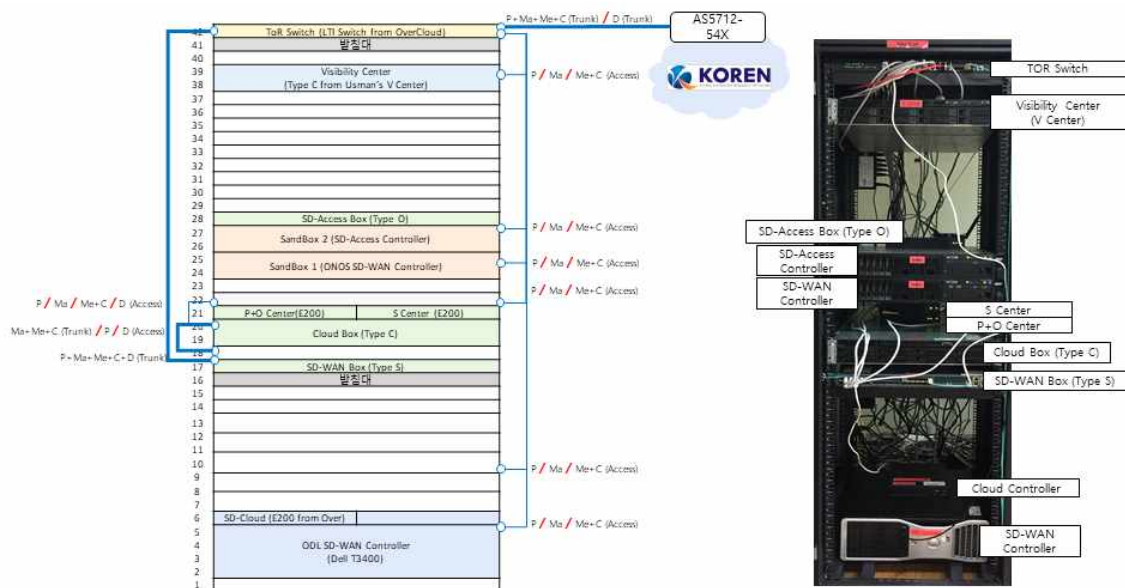


그림 21 GIST KOREN Rack 및 구조

- o GIST KOREN Rack에서는 GIST 내부의 관제타워 및 컨트롤 노드를 포함한 인프라들로 구성되어 있다. 위에서 제시한 여러 유형의 박스들을 잘 제어하기 위해서 Cloud 제어 노드, SD-WAN 네트워크 제어 노드, SD-Access 네트워크 제어기노드 등이 포함되어 있다. 또한 이런 제어기들을 관장하면서 직접 일을 시키는 센터들도 Rack에서 포함이 되어있다. Type S, C, O는 한 세트로 관리가 되면서 Type S가 스위치 역할을 할 수 있기 때문에, ToR 스위치 역할을 하면서 Type C와 Type O를 지원한다.
- o Rack에 구성된 박스들은 기본적으로 여러 개의 네트워크 인터페이스를 활용해 유기적으로 연결하고 제어할 수 있는 기반을 마련하고 있다. 원격지에서 박스들



을 대상으로 접속하고 제어하기 위한 P (Power) 네트워크, 박스들을 관리하거나 패키지들을 받는 용도의 M (Management) 네트워크, 박스 간의 제어 메시지를 위한 C (Control) 네트워크, 가상머신이나 서비스 간의 데이터 이동 트래픽을 위한 D (Data) 네트워크로 구성되어 있다. 다양한 용도에 따른 복수 개의 네트워크 인터페이스를 활용함으로써 보다 트래픽들을 분리시키고, 문제 발생 시 빠르고 즉각적인 대응이 가능할 수 있다.

### 3.3. KOREN SmartX platform으로의 확장 및 개선

- o KOREN SmartX Playground에서 다양한 개발자들이 자신의 실험들을 실증하기 위한 인프라는 구축이 됐으나, 개발자들이 손쉽게 KOREN SmartX Playground를 사용할 수 있도록 Platform 차원에서 인터페이싱 역할이 부족하다.

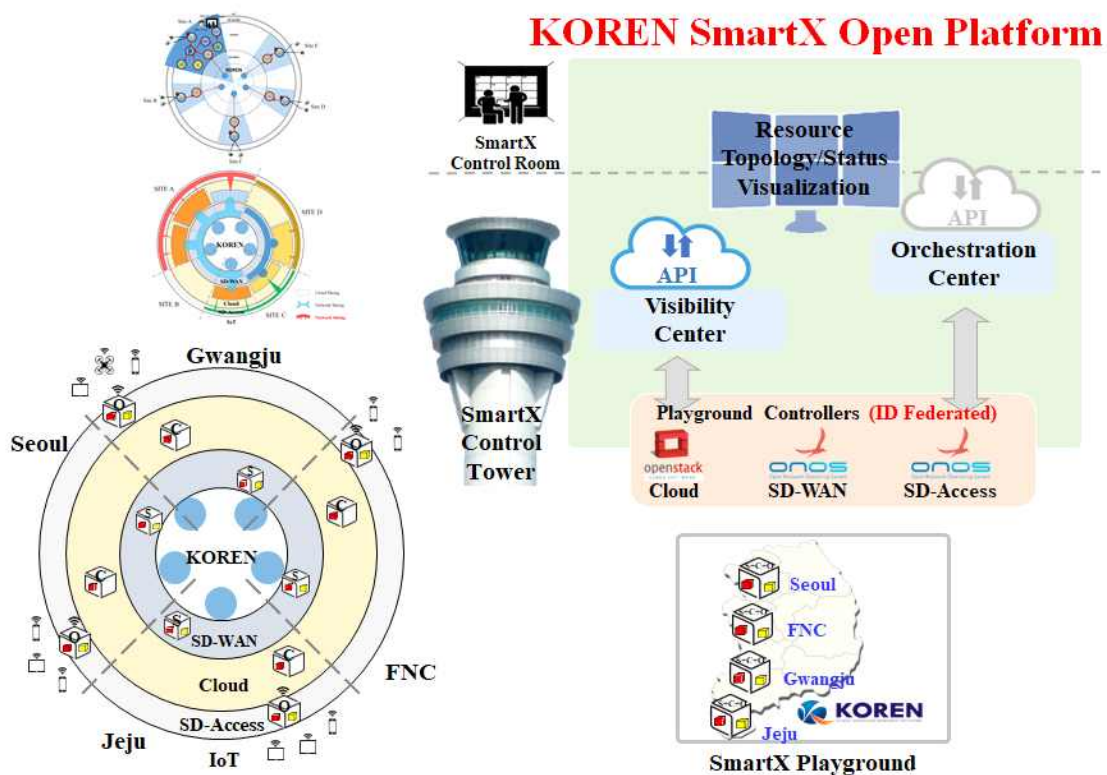


그림 22 KOREN SmartX Open Platform 개념도

- o 이를 위해 개발자가 API 형태로 사용가능 할 수 있도록 지원하는 센터들을 구축하고, 초기 단계에서는 개발자에게 Orchestration 기능과 Visibility 기능을 제공하는 센터들을 시작으로 향후에 점차 확장되는 형태로 개선하도록 한다. Visibility Center에서 제공하는 API는 현재 개발자가 가지고 있는 인프라에 대한 전체적인 상황을 파악할 수 있는 데이터들을 가져오고 시각화해주는 기능을 제공해주고,

P+O Center에서는 개발자들이 자신만의 독립적인 인프라를 가지기 위해서 클라우드 및 네트워크를 할당해주는 기능을 제공하도록 한다. 이러한 기능들을 통해 개발자들은 좀 더 손쉽게 KOREN SmartX Playground를 활용할 수 있도록 인터페이스 역할이 가능하게끔 개선 및 확장하는 형태로 KOREN SmartX Platform을 구축한다.

## 4. 개발자 맞춤형 인프라 슬라이싱

### 4.1. 인프라 슬라이싱 개요

- 국내 여러 사이트의 분산형 Box를 활용하여 구성된 KOREN SmartX Platform 환경에서 구성된 인프라를 모든 개발자들이 활용하는 공유 형태이지만, 사용자 별로 마치 자신만의 독립적인 인프라를 사용하는 것처럼 느낄 수 있도록 인프라를 나누고, 고립시키고, 할당하고 관리하는 기술이 필요하다. 이러한 인프라 즉, 클라우드 및 네트워크를 논리적으로 나누고, 할당하고 관리하는 기술 형태를 인프라 슬라이싱이라고 지칭한다.

### 4.2. 인프라 슬라이싱 요구사항

- 여러 사이트에 분산되어 있는 KOREN SmartX Platform 환경에서 인프라 슬라이싱은 개발자들이 Cloud 뿐만 아니라 IoT 기기들까지도 연결되는 서비스들을 고려한 IoT-Cloud 서비스에 대응하기 위해서는 다음과 같은 요구사항을 포함한다.
- 첫 번째로 인프라 슬라이싱의 범위는 클라우드에서 제공하는 컴퓨팅 스토리지 뿐만 아니라 IoT 기기 까지 확장된 네트워킹을 전제로 한다. 클라우드 같은 경우 컴퓨팅 및 스토리지 그리고 부분적인 네트워킹을 사용자 테넌트 별로 할당하고 관리할 수 있는 기술을 제공하고 있으나, 이는 클라우드로 묶인 인프라에 한정해서 다루고 있으며, IoT-Cloud 까지 확장하는 서비스에 대응할 경우에는 인프라를 제공하는 부분에 있어서 제한적일 수밖에 없다. 즉 Cloud 범위 관할을 벗어나는 IoT 기기들의 경우 네트워킹 인프라에 관해서 개발자에게 슬라이싱 형태로 제공해주는 것이 제한적이다. 따라서 IoT-Cloud 까지 확장된 인프라를 테넌트 별로 할당하고 관리할 수 있는 기술이 필요하다.
- 두 번째로, KOREN SmartX Platform을 사용하는 개발자들이 손쉽게 사용하면 자신의 서비스들을 실증할 수 있도록 API 형태로 제공해야 한다. API 형태로 제공되면, 개발자들은 내부적인 기술적인 수준을 몰라도 손쉽게 원하는 클라우드 인프라의 형태와 연결할 IoT 디바이스들을 파라미터로 입력해 클라우드 리소스를 제공받고 IoT 기기들과의 네트워킹 연결을 할 수 있게 된다.

- o 마지막으로 각 사용자들이 할당한 슬라이싱은 독립적이며, 다른 영역을 침범해선 안 되며, 고유의 할당 영역을 보장받아야 한다. 즉 사용자 별로 할당된 인프라 슬라이싱은 서로의 존재를 모르게끔 고립된 (Isolated) 형태로 존재해야 하며, 이들이 다른 영역을 침범해서는 안 되도록 보장해야 한다. 이를 위해서는 테넌트 별로 고유한 아이디와 이와 연결되는 클라우드 리소스 및 네트워크 리소스를 관리해야하며, 이러한 테넌트를 지원할 수 있는 기술적인 요소들을 활용해야 한다.

## 5. 개발자 맞춤형 인프라 슬라이싱 설계

### 5.1. 개발자 맞춤형 인프라 슬라이싱 설계

- o KOREN SmartX Platform 환경에서 개발자 맞춤형 인프라 슬라이싱을 위해 다음 그림과 같은 구조로 인프라 슬라이싱을 디자인 하였다.

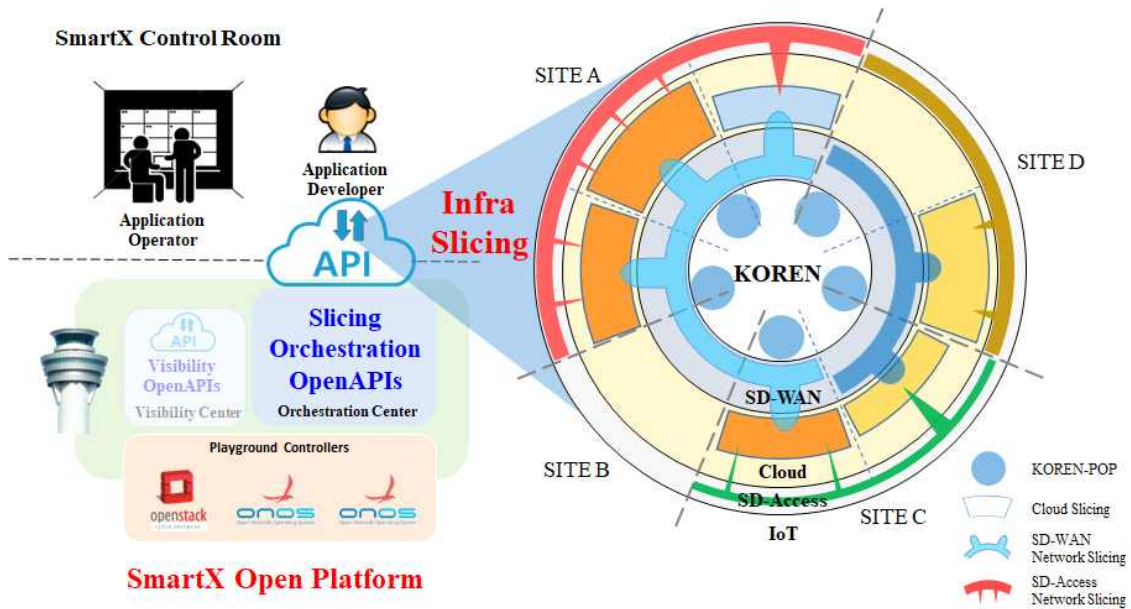


그림 23 개발자 맞춤형 인프라 슬라이싱의 설계

- o KOREN SmartX Platform의 Orchestration Center에서는 개발자들이 사용할 수 있는 슬라이싱 API가 지원된다. 인프라를 구성하고 있는 영역 벨트 별로 SD-WAN, Cloud, SD-Access와 같이 세분화된 영역에 대한 슬라이싱을 단계적으로 완성하고, 이들을 엮어서 개발자가 사용할 수 있는 최종적인 형태의 API를 제공하는 것을 목표로 하고 있다. 즉 SD-WAN 슬라이싱은 사이트와 사이트를 넘어가는 네트워킹을 중심으로 슬라이싱을 하는 것을 말하며, 클라우드 슬라이싱은 주로 클라우드 자원을 중심으로 슬라이싱 하는 기술을 말한다. 마지막으로 SD-Access 슬라이싱은 IoT 기기들과 클라우드 자원과의 네트워크 연결 중심의 슬라이싱을 말하며, 각자의 영역에서 필요한 클라우드 SDN 제어기들을 활용해 슬라이싱을 기술적으로 진행하는 역할을 한다.
- o 다음 그림은 2절에서 위와 같은 슬라이싱 구현을 위해 실제적인 인프라의 네트워크 구성을 어떻게 하였는지를 나타내는 그림이다.

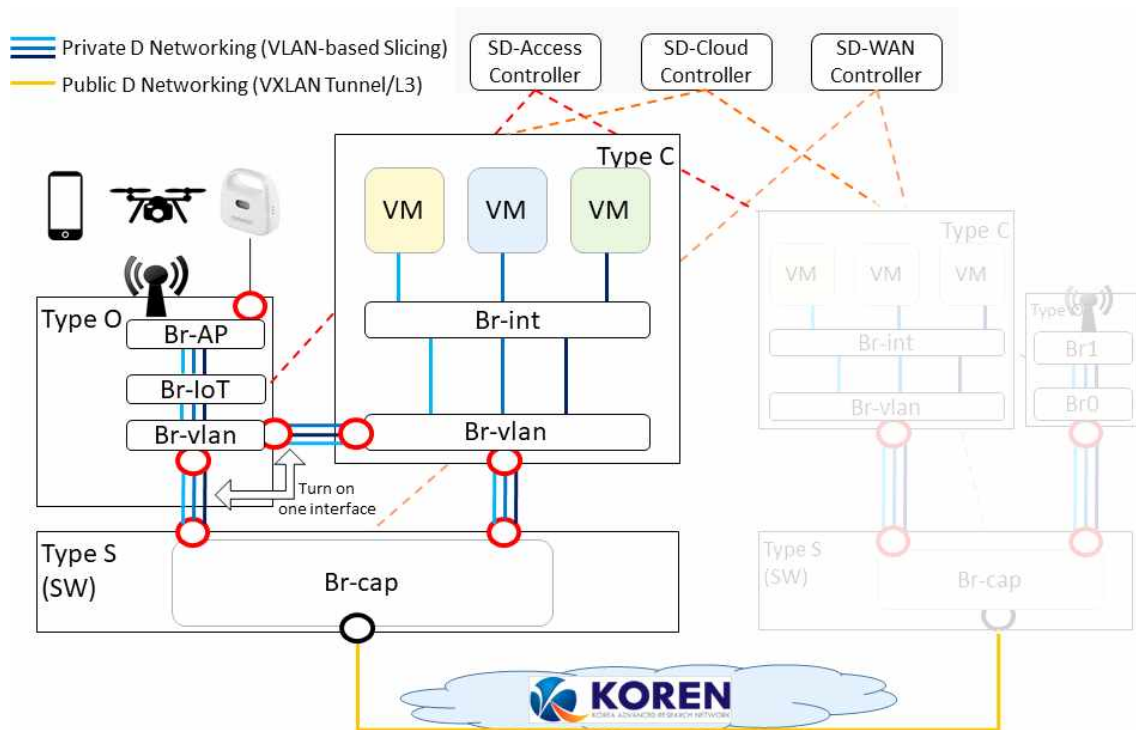


그림 24 인프라 슬라이싱을 위한 네트워킹 설계도

- o 각 사이트 별로 Type S, Type C, Type O에 대해서 내부적으로 네트워크를 어떻게 구성하는지에 대해서 설계하였다. Type C 박스 같은 경우 Cloud OS인 오픈스택을 활용하여 컴퓨팅 리소스와 부분적인 네트워킹에 대해서 지원을 받는 형태로 OVS를 구성하였다. 즉 여기서 만들어진 Br-int와 Br-vlan OVS 브릿지는 오픈스택의 Neutron이 관할하게 되며, Neutron에서 테넌트를 위한 네트워킹 형태는 VLAN 네트워크로 구성하였다. 즉 테넌트 네트워킹 별로 VLAN ID가 태깅이 되면서 Type S의 박스로 나가는 형태로 구성하고 있다.
- o Type O 박스의 경우, 무선과 유선을 관리할 수 있는 가상 스위치가 필요하며, 이 가상 스위치는 VLAN ID를 제어 가능해야 한다. 즉, 테넌트 간의 슬라이싱을 지원하기 위해서는 가상 스위치가 IoT 디바이스의 패킷들을 받아서, 적절하게 VLAN 태깅을 거친 후, 연결된 Type S에게 전달되도록 구성해야 한다. 따라서 무선을 받는 가상스위치 Br-AP와 무선 및 유선을 통합해서 받는 가상 스위치 Br-IoT를 만들고, 그리고 이러한 통합 Br-IoT 가상 스위치에서 나가는 모든 패킷들을 확인하고 VLAN 태깅을 해주는 Br-vlan 가상 스위치를 추가적으로 구성을 하였다.

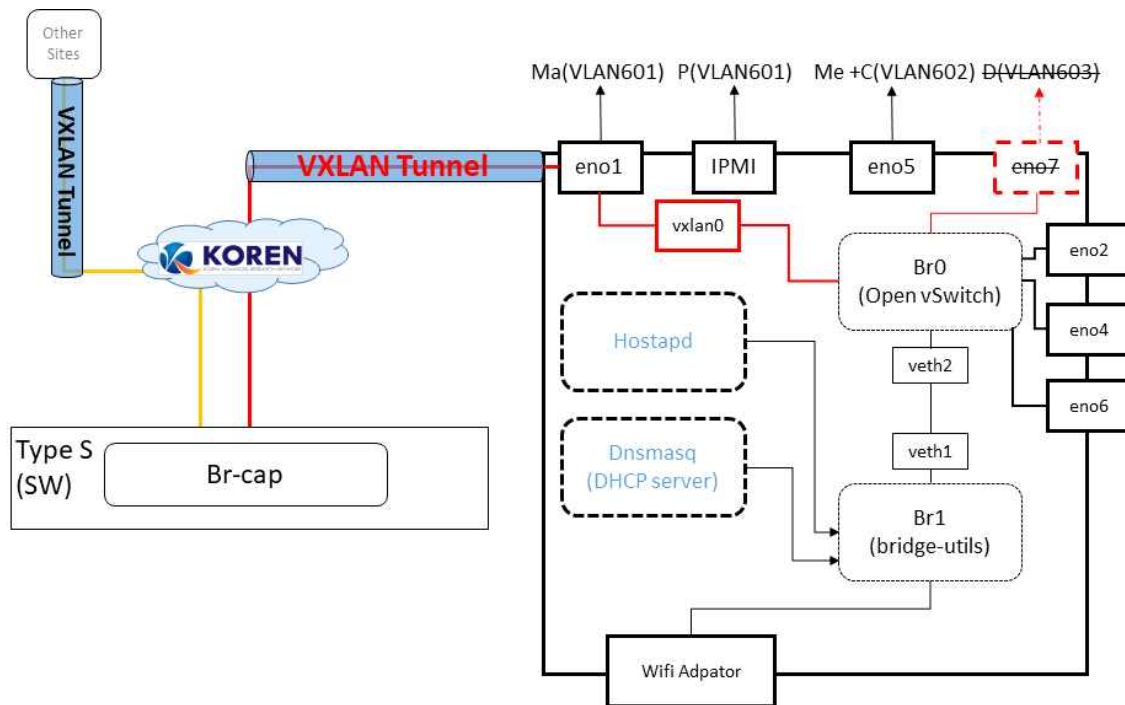


그림 25 Type O의 구조 및 Type S와의 네트워크 연결 구성도

- o 마지막으로 Type S 박스는 Type C와 Type O에서 태깅 되어서 들어오는 포트를 OVS로 묶어서 서로를 연결해주는 역할을 하며 또한 다른 사이트로 패킷을 전달하기 위해서 터널링 기법인 VXLAN 기술을 활용해서 다른 사이트로 패킷을 전달하는 역할을 하게 된다.

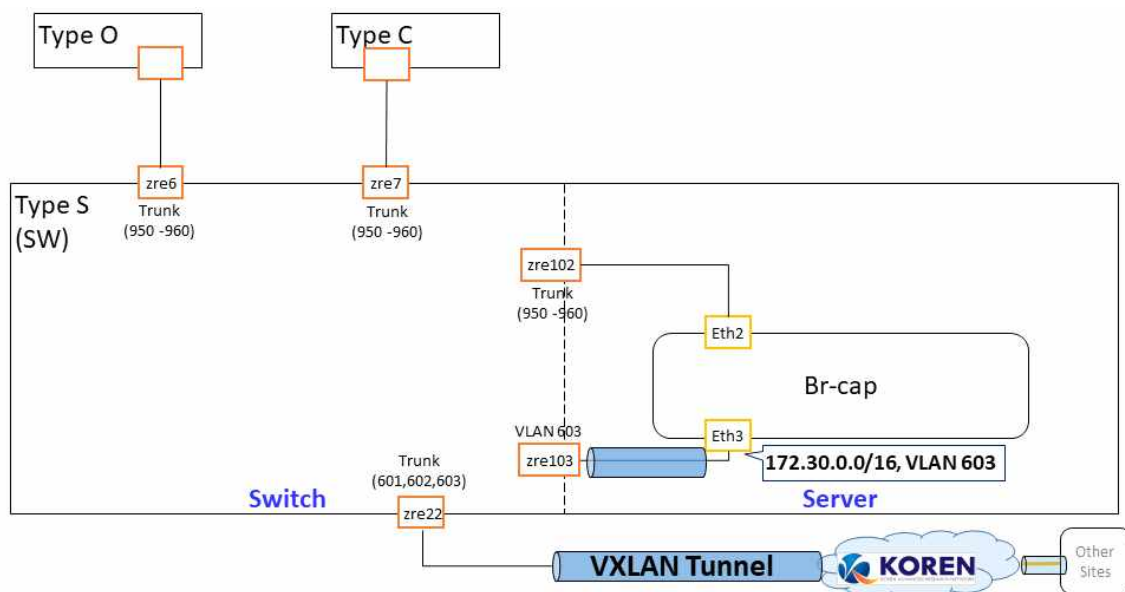


그림 26 Type S 스위치-박스의 내부적인 네트워크 구성도



- o 각 사이트에 위치한 Type S 스위치-박스는 터널링 형태로 서로 연결이 가능하도록 구성하였으며, Type S 스위치-박스의 컴퓨팅 파트에 가상 스위치 Br-cap를 구성하고, 이를 통해 다른 사이트에 위치한 Type S 스위치와 터널링으로 연결이 된다. Type S 스위치-박스의 스위치 파트에서는 Type C, Type O가 직접적으로 연결이 돼 있으며, Type S가 없는 분산된 사이트에 위치한 Type O는 GIST에 위치한 Type S에 직접적인 터널링 연결을 통해 분산된 사이트 간 연동을 완료하였다.
- o 다음 그림 7은 슬라이싱 ID 및 다른 엔티티와의 관계도를 정리한 그림이다. 2절에 슬라이싱 요구사항에서도 언급되었듯이, 각 테넌트별로 슬라이싱을 지원하기 위한 ID들이 정리가 되어야 한다. 즉 슬라이싱 ID 중심으로 다른 요소들이 적절하게 연결이 된 관계를 관리 및 유지해야 한다. 슬라이싱 ID에 연결되는 다른 요소들은 그림과 같이 테넌트 (유저), SDN 제어기, VLAN ID, IoT 기기, VM 인스턴스 등이 있다. 즉 어떤 한 슬라이싱 ID에 연결된 테넌트는 누구이며, 어떤 VM 인스턴스와 어떤 IoT 기기가 연결이 되어 있으며, 이 슬라이싱 ID는 SDN 제어기의 권한은 어떻게 구성되어 있으며, VLAN ID는 어떤 것을 사용하는지를 관리하고 있어야 한다. 여기서 Tenant ID를 위해 오픈스택의 Keystone 프로젝트에서 제공하는 Tenant ID를 그대로 사용하면서, 각 슬라이싱과 관련된 엔티티와의 관계 정보를 관계형 DB로 관리하는 형태로 설계하였다. 즉, 슬라이싱과 관련된 여섯 개의 엔티티와의 관계 DB 테이블을 생성하였다.

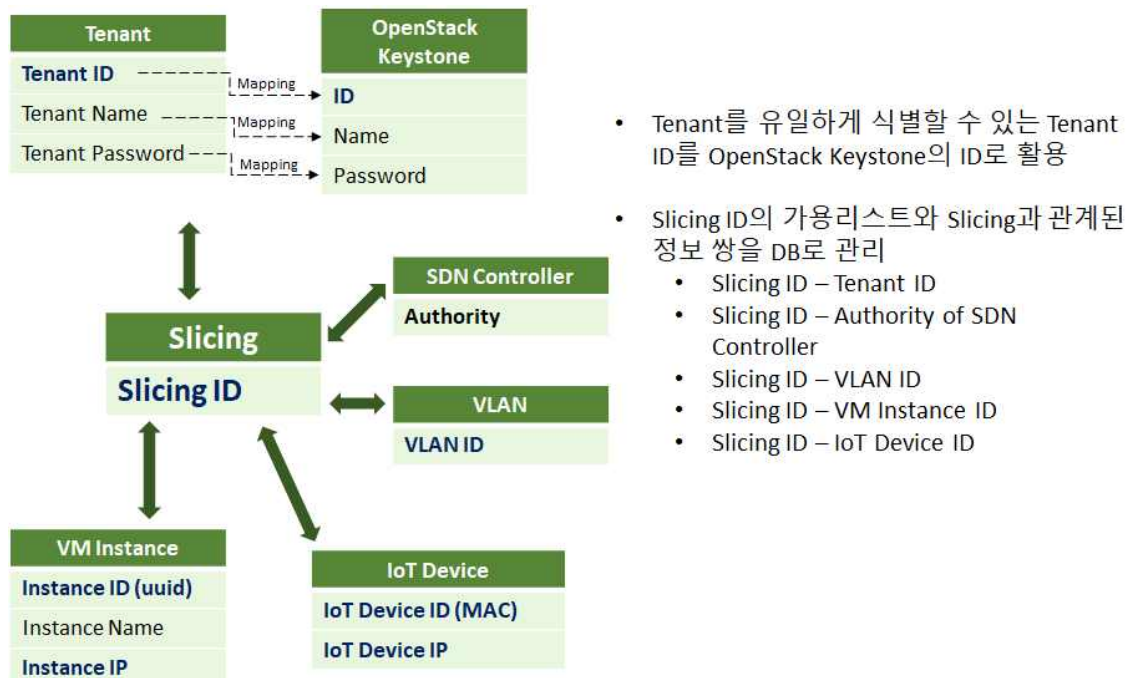


그림 27 슬라이싱 ID 및 다른 엔티티와의 관계도

```

CREATE TABLE Slicing (
  Slicing_ID VARCHAR(100) NOT NULL,
  Tenant_ID VARCHAR(100) NOT NULL,
  Authority VARCHAR(100) NOT NULL,
  VLAN_ID VARCHAR(100) NOT NULL,
  PRIMARY KEY(Slicing_ID)
);

CREATE TABLE Instance (
  Instance_ID VARCHAR(100) NOT NULL,
  Instance_Name VARCHAR(100) NOT NULL,
  IP VARCHAR(100) NOT NULL,
  PRIMARY KEY(Instance_ID)
);

CREATE TABLE IoT (
  MAC VARCHAR(100) NOT NULL,
  IP VARCHAR(100) NOT NULL,
  PRIMARY KEY(MAC)
);

CREATE TABLE Slicing_Instance (
  Slicing_ID VARCHAR(100) NOT NULL,
  Instance_ID VARCHAR(100) NOT NULL,
  PRIMARY KEY (Slicing_ID, Instance_ID),
  FOREIGN KEY (Slicing_ID) references Slicing(Slicing_ID),
  FOREIGN KEY (Instance_ID) references Instance(Instance_ID)
);

CREATE TABLE Slicing_IoT (
  Slicing_ID VARCHAR(100) NOT NULL,
  MAC VARCHAR(100) NOT NULL,
  PRIMARY KEY (Slicing_ID, MAC),
  FOREIGN KEY (Slicing_ID) references Slicing(Slicing_ID),
  FOREIGN KEY (MAC) references IoT(MAC)
);

```

그림 28 슬라이싱 관리 DB 설계

- o 슬라이싱이 VLAN 기술을 기반으로 만들기 때문에 Slicing ID 별 가용 네트워크를 미리 사용자와 약속하는 것이 필요하다. 따라서 다음과 같이 VLAN (951~960)에 대한 가용 네트워크 리스트를 미리 지정하였다.

- 951 : 10.10.10.0/24
- 952 : 10.10.20.0/24
- 953 : 10.10.30.0/24
- 954 : 10.10.40.0/24
- 955 : 10.10.50.0/24
- 956 : 10.10.60.0/24
- 957 : 10.10.70.0/24
- 958 : 10.10.80.0/24
- 959 : 10.10.90.0/24
- 960 : 10.10.100.0/24

그림 29 VLAN 별 가용  
네트워크 리스트

- o 위에서 설계한 슬라이싱 ID와 주변 관계 요소들을 관리하기 위한 부분과 각각의 영역 별 슬라이싱 (SD-WAN/Cloud/SD-Access)를 연결해주는 오케스트레이션 부분이 필요하다. 따라서 SD-WAN 슬라이싱, Cloud 슬라이싱, SD-Access 슬라이싱



과 이들을 엮고 슬라이싱 ID와 주변 요소들의 관계를 정리하는 슬라이싱 통합 이렇게 네 가지로 세분화해서 인프라 슬라이싱을 정리하는 형태로 소프트웨어를 디자인하였으며, 전체적인 인프라 슬라이싱의 워크플로우 순서도는 다음 그림과 같다.

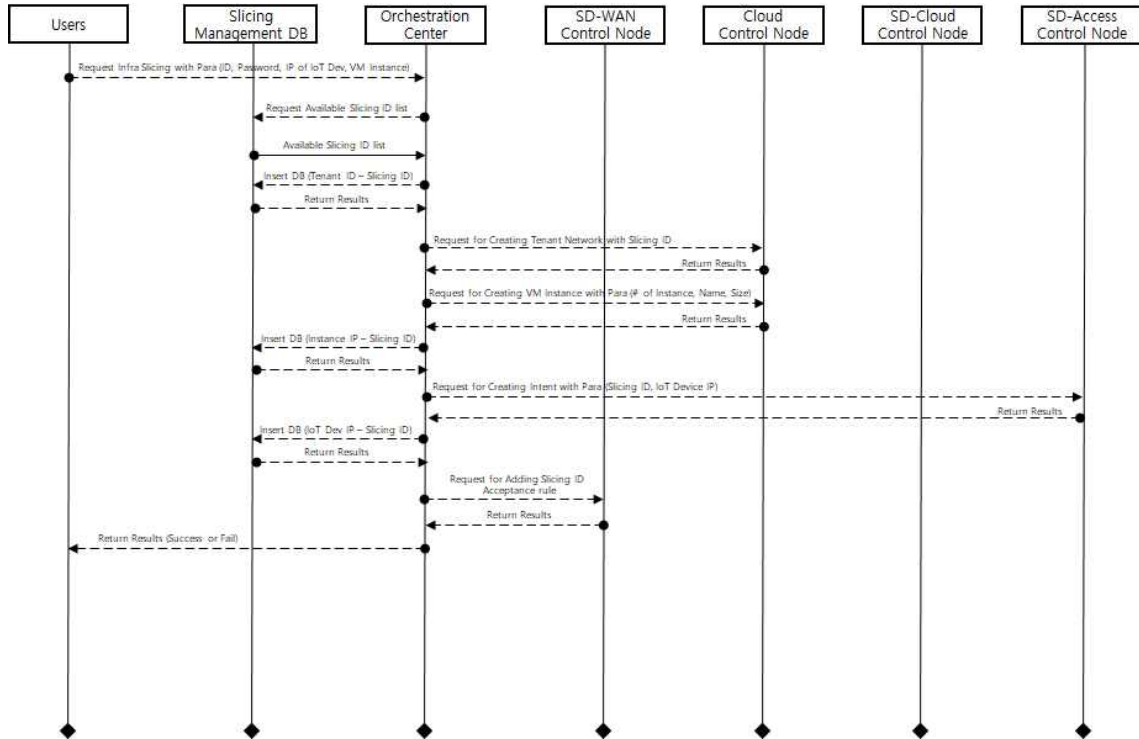


그림 30 인프라 슬라이싱에 대한 전체적인 워크플로우 순서도

- o 초기 설계 시에는 통합적인 인프라 슬라이싱을 제공하는 하나의 API를 사용자가 콜하는 형태로 구축을 진행하였으나, 이렇게 된다면, 사용자별로 슬라이싱을 유연하게 연결하고 관리해주는 부분이 결여되기 때문에, 아래와 같이 API들을 부분적으로 나누어서 정리하였다.

Slicing\_Create (Tenant ID, Password)

Cloud\_Slicing (Tenant ID, Password, Slicing ID, # of VM, Flavor, Image)

SD-Access\_Slicing (Tenant ID, Password, Slicing ID, IoT MAC)

Slicing\_Delete (Tenant ID, Password, Slicing ID)

Slicing\_List (Tenant ID, Password)

Slicing\_Show (Tenant ID, Password, Slicing ID)

## 5.2. KOREN SmartX Platform 상의 인프라 슬라이싱 검증 및 활용

- 본 절에서는 3장에서 설계한 인프라 슬라이싱을 구현하여, 이를 실제 KOREN SmartX Playground 환경에서 검증하는 것을 보인다. KOREN SmartX Playground는 Type S, C, O를 구성하고 있는 특별한 인프라 환경이며, 이러한 환경에서 슬라이싱을 적용하는 것을 가정한다. 또한 모든 슬라이싱의 API 수행은 Slicing Orchestration API가 위치한 p+o 센터에서 수행한다. 먼저 최종 사용자들이 하고자 하는 서비스를 수행하기 위해서 클라우드에서 생성된 가상머신과 Type O에 연결된 IoT 기기가 서로 연결성을 보장받아야 한다. 이를 위해서 구현한 슬라이싱을 최종사용자가 아래 그림과 같은 순서로 API를 활용해야 한다.

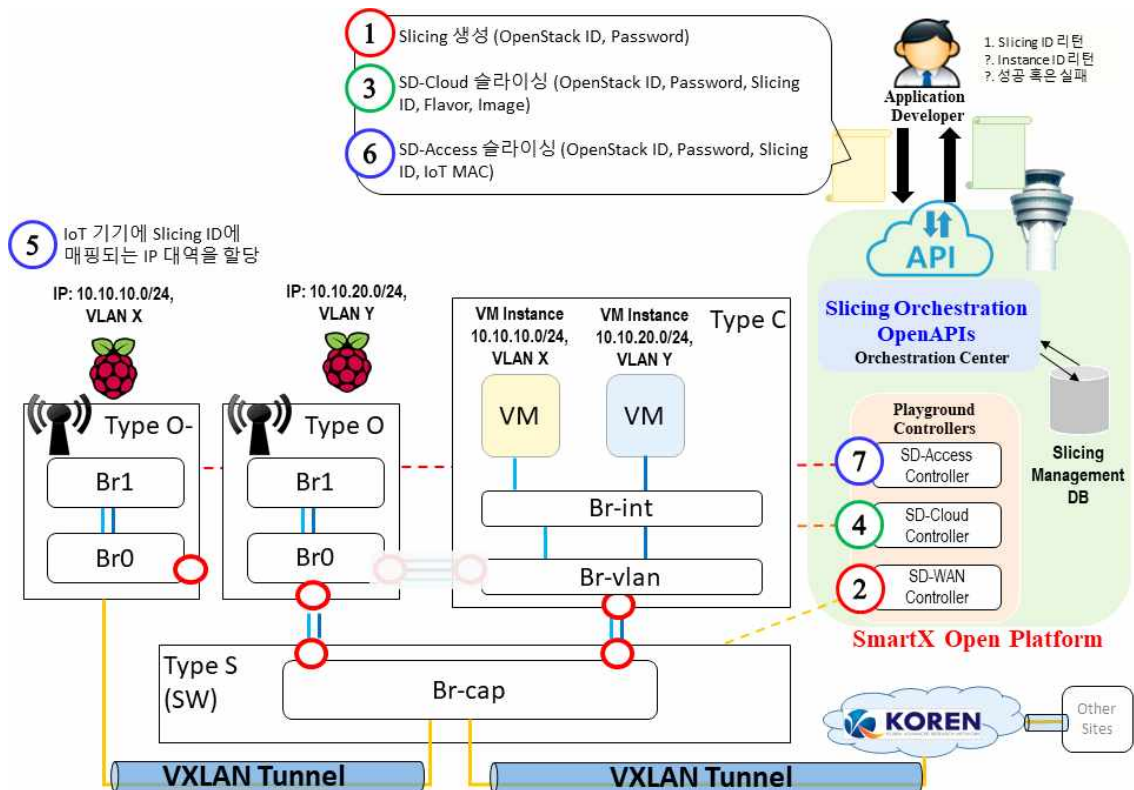


그림 31 인프라 슬라이싱 수행을 위한 시나리오 구성

- 첫 번째로는 사용자가 자신만의 테넌트 네트워킹을 보장받을 수 있는 슬라이싱 ID를 부여받아야 한다. 슬라이싱 ID를 부여받기 위해서 KOREN SmartX Playground의 오픈스택 키스톤의 ID가 필요하며, p+o 센터에서 API 수행을 시도해야 한다. 다음 그림과 같이 p+o 센터에서 Slice\_Create.sh 스크립트를 수행하면, 대화형 인터페이스를 통해 오픈스택 키스톤 ID와 Password를 입력하고, 인증이 정상적으로 완료가 된다면, 다음 그림과 같이 자신의 슬라이싱 ID가 생성이 된다.

```

root@netcs:/home/netcs/slicing# ./Slice_Create.sh
Input your ID: demo
Input your Password:
Slicing has been created

Slicing id: 951

root@netcs:/home/netcs/slicing#

```

그림 32 Slicing\_Create 호출 화면

- 위 그림에서 오픈스택 키스톤 ID가 demo인 사용자가 Slicing ID 951을 할당 받는 장면이다. 사용자의 인증과정이 오픈스택의 Keystone을 통해 이루어지는 것을 볼 수 있다. 그리고 1번 과정이 진행이 되면, 내부적으로 Type S 간의 SD-WAN 영역을 위한 길을 열어주는 플로우가 생성된다. 즉, 각 분산된 사이트의 Type S 서버-스위치 박스에서 해당 VLAN ID에 대해서 터널간의 트래픽 허용을 위한 플로우 룰이 내려지게 된다.

```

root@netcs:/home/netcs/slicing# ./Cloud_Slicing.sh
Input your ID: demo
Input your Password:
Input Slicing ID: 951
Input Flavor: ml.small
Input Image: cirros
Network has been created

```

| Field                       | Value                                           |
|-----------------------------|-------------------------------------------------|
| OS-DCF:diskConfig           | MANUAL                                          |
| OS-EXT-AZ:availability_zone |                                                 |
| OS-EXT-STS:power_state      | NOSTATE                                         |
| OS-EXT-STS:task_state       | scheduling                                      |
| OS-EXT-STS:vm_state         | building                                        |
| OS-SRV-USG:launched_at      | None                                            |
| OS-SRV-USG:terminated_at    | None                                            |
| accessIPv4                  |                                                 |
| accessIPv6                  |                                                 |
| addresses                   |                                                 |
| adminPass                   | trza6NwPAYy1                                    |
| config_drive                |                                                 |
| created                     | 2017-11-09T05:29:17Z                            |
| flavor                      | ml.small (24105f99-67f4-4108-a875-47be4f454cdd) |
| hostId                      |                                                 |
| id                          | 67b22a68-9a77-4bad-921d-4927b9e59ce6            |
| image                       | cirros (583857ff-fb1c-4ba2-8038-ac45daa951c7)   |
| key_name                    | None                                            |
| name                        | Cloud_Instance_951                              |
| progress                    | 0                                               |
| project_id                  | 62acal2392a3459d97053cc533b20357                |
| properties                  |                                                 |
| security_groups             | name='default'                                  |
| status                      | BUILD                                           |
| updated                     | 2017-11-09T05:29:17Z                            |
| user_id                     | 787264914c8a417fb57e1b9e3e8a82fb                |
| volumes_attached            |                                                 |

그림 33 Cloud\_Slicing 호출 화면

- 두 번째 과정으로 부여받은 슬라이싱 ID 중심의 네트워킹을 활용한 가상머신을 생성하는 과정을 진행한다. 그림 14와 같이 p+o 센터에서 Cloud\_Slicing 스크립

트를 수행하게 되면, 오픈스택 키스톤 ID 및 비밀번호를 입력하고, 이전 단계에서 부여받은 Slicing ID와 생성할 가상머신에 대한 정보들 (이미지, 크기)를 입력하게 된다. 이 때, Slicing ID와 동일한 VLAN ID를 갖는 테넌트 네트워크가 오픈스택 Neutron 서비스를 통해 생성이 되며, 가상머신에 이 네트워크를 연결하는 과정을 거치게 된다.

- 입력이 완료가 되면, 그림 14 하단에 결과 창을 볼 수 있으며, Cloud\_Instance\_951이라는 이름으로 가상머신이 생성이 되었고, 오픈스택의 대쉬보드에서 해당 키스톤 ID로 로그인을 하면 생성된 가상머신을 확인이 가능하다.

```

root@netcs:/home/netcs/slicing# ./SD_Access_Slicing.sh
jq has been installed.
Input your ID: demo
Input your Password:
Input Slicing ID: 951
Input Your IoT MAC: B8:27:EB:69:82:DE
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  178  100  178    0     0  68068      0 --:--:-- --:--:-- --:--:-- 89000
MAC Location found!
Making Intents..
Intents have been created
root@netcs:/home/netcs/slicing#

```

그림 34 SD\_Access\_Slicing 호출 화면

- 마지막으로 사용자는 자신의 IoT 기기들의 네트워킹 연결 작업을 진행한다. 먼저, 사용자는 연결하고자 하는 IoT 기기를 Type O에 물리적으로 연결하는 작업을 진행해야 한다. 무선 혹은 유선을 통해 연결이 되었다면, IoT 기기에 접속을 시도하여 IP 대역을 수동으로 설정하여야 한다. 예를 들면, Slicing ID가 951이면 그림 10에서 951에 맞는 대역인 10.10.10.0/24 대역으로 IP를 맞춰야 한다. 내부적인 약속으로 IoT 기기들의 IP대역은 10.10.10.3 ~10.10.10.99 내에서 할당 한다. 이러한 절차가 끝난후, p+o 센터에서 SD-Access Slicing 스크립트를 수행하게 되면, 그림 15와 같이 오픈스택 키스톤 ID 및 패스워드와 슬라이싱 ID를 입력하고, IoT 기기에서 IP를 부여한 인터페이스의 MAC을 입력하게 된다.
- 그림 15는 사용자 Demo가 자신이 할당받은 Slicing ID 951번에 IoT 기기 MAC 주소 B8:27:EB:69:82:DE를 등록하는 과정을 나타내고 있다. 입력받은 MAC 주소가 SmatX KOREN Playground 인프라를 관장하고 있는 범위 내에 있는 지 체크하는 작업을 거친 후, MAC의 위치를 알아내고 이를 기반으로 ONOS의 Intent가 설치가 진행이 된다. Intent는 한 기기 당 양방향의 쌍으로 진행이 된다. IoT 기기에서 나가는 패킷의 경우, Slicing ID에 맞는 VLAN을 태깅하는 플로우 룰을

내려주는 Intent를 설치하며, IoT 기기로 들어가는 패킷에 대해서는 태깅된 VLAN ID를 벗겨내는 플로우 룰을 내려주는 Intent를 설치하게 된다. 이러한 작업이 끝나면, 사설 IP를 가진 IoT 기기들과 가상머신들이 서로간의 통신이 가능하며, 사용자의 IoT 기기가 어떤 위치에 있든 간에, Type O Access 박스를 통해 연결되면, 서로간의 통신이 가능하다.

- 이러한 세 단계의 과정이 모두 끝나면 사용자가 자신의 서비스를 구동하기 위한 인프라 및 네트워크가 준비가 된 것이다. 이후에는 적절한 서비스를 선택해 서비스 합성을 수행시키면 된다. 슬라이싱의 검증을 위해서 Smart Air IoT-Cloud 서비스를 KOREN SmartX Platform 상에 검증해보았다. 이 서비스는 지리적으로 분산되어 있는 몇 개의 사이트에 위치한 IoT 디바이스에 센서들을 붙여서 먼지 값들을 측정하고, 드론이나 핸드폰을 통한 비디오 스트리밍으로 실시간으로 상황을 파악 가능하게 하는 서비스이다. 이렇게 생성된 데이터들은 구축된 SmartX Platform 상의 클라우드를 통해 만들어진 가상머신으로 들어가게 되고 최종적으로 통합 대쉬보드를 통해 그림 16과 같이 전체적인 상황을 파악할 수 있다.

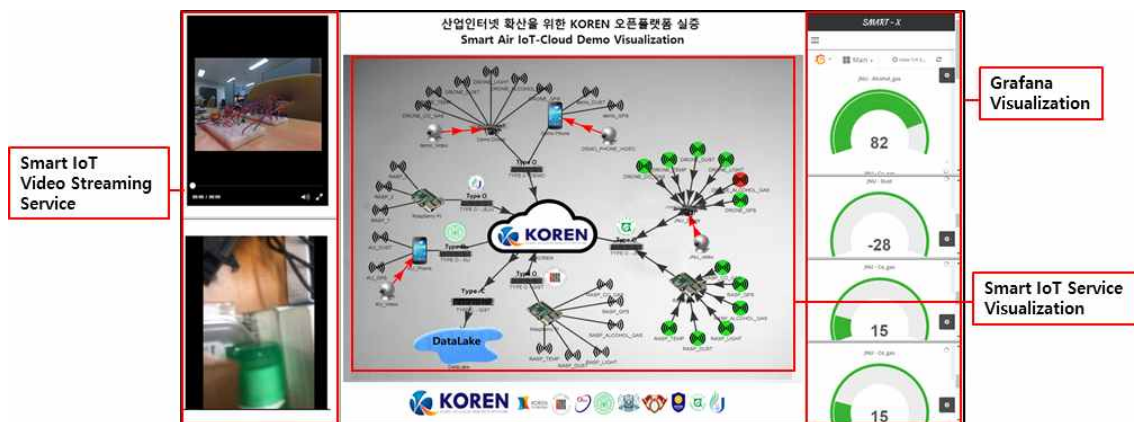


그림 35 Smart Air IoT-Cloud 서비스 통합 시연 데모

- 본 시연을 위해 하나의 슬라이싱 ID를 만들고, 하나의 가상머신 그리고 여러 IoT 디바이스들에 대해 슬라이싱을 진행하였다. 인프라 슬라이싱을 통해, 각 IoT 기기들은 Private IP 범위를 가지고 있지만 지리적으로 떨어져 있어서 정상적으로는 접근이 안되는 GIST에 위치한 가상머신 인스턴스에게 접근하는게 가능한 것을 볼 수 있었고, 이를 통해 위의 서비스가 정상적으로 동작하는 것을 확인하였다.

## References

## *SmartX* 기술 문서

- 광주과학기술원의 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 대한 문의 사항은 아래의 정보를 참조하시길 바랍니다.  
(Homepage: <https://nm.gist.ac.kr>, E-mail: ops@smartx.kr)

작성기관: 광주과학기술원

작성년월: 2018/08