SaaS OverCloud 기술문서 #16

Multi-site Cloud 상의 동적 재구성을 위한 OverCloud 개념 설계 및 검증

Document No. SaaS OverCloud #16
Version 1.0
Date 2018-12-14
Author(s) GIST#1 Team

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2018. 12. 14.	김종원, 박선, 한정수	
0.2	2018. 11. 05.	한정수	
0.3	2018. 11. 12.	한정수	
0.4	2018. 11. 22.	한정수	
1.0	2018. 11. 24.		

본 문서는 2018년도 정부(미래창조과학부)의 재원으로
정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R7117-16-0218,
이종 다수 클라우드 간의 자동화된 SaaS 호환성 지원 기술 개발)

The research was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R7117-16-0218, Development of Automated SaaS Compatibility Techniques over Hybrid/Multisite Clouds)

Contents

Multi-stie Cloud 상의 동적 재구성을 위한 OverCloud 개념 설계 및 검증

그림 목차

SaaS OverCloud 기술문서 #16.
Multi-site Cloud 상의 동적 재구성을 위한 OverCloud 개념 설계 및 검증

1. OverCloud 개요

1.1. 문서의 목적

o 본 문서에서는 다양한 SaaS 응용 지원을 위해 지리적으로 분산된 클라우드 인프라를 동적으로 활용할 수 있는 새로운 형태의 클라우드 소프트웨어 개념인, 동적 OverCloud에 대한 방식을 제안하고, 이전에 진행했던 버전에서 개선된 부분을 중심으로 개념을 설명하고 이에 대한 지원 소프트웨어를 구현하여 실제 서비스 사례에 적용해서 확인한다. 언더레이의 인프라 자원 집합을 다루는 계층과 오버레이의 SaaS 응용들을 다루는 계층을 분리해 다루기 때문에 이종 하이브리드/멀티사이트 클라우드 인프라/플랫폼 환경에 독립적으로 설치와 운용되도록 보장하는 호환성 지원을 위한 자동화된 실행환경을 통해 동적으로 제공함으로써 SaaS 관리에 대해 하부의 UnderCloud와 호환성에 대해 고민할 필요 없이 추상화된 동적 OverCloud 만을 고려하여 원하는 SaaS 응용을 가능하게 한다.

1.2. OverCloud 연구 배경

o 클라우드 기반의 ICT 기술이 성숙함에 따라, 거대한 클라우드 사업자들에 의해 구축된 글로벌한 클라우드 인프라가 미래 ICT 인프라의 핵심으로 부상하고 있다. 또한 IoT 기술의 성장과 함께, 다양한 IoT 기기들이 급속도로 증가하면서, 집, 건강, 공장, 농장 등 다양한 분야로 확산 중이다. 이러한 IoT와 클라우드 중심의 패러다임 전환에 따라서 IoT 기기와 클라우드를 연동하는 다양한 IoT-Cloud 서비스들이 개발되고 있다. 따라서 다양한 서비스들을 신속하고 효율적으로 개발할 수 있도록, 기존의 획일적인(monolithic) 서비스 합성은 컨테이너 중심의 클라우드-네이티브(cloud-native) 컴퓨팅 기반의 마이크로서비스 구조(MircroServices Architecture: MSA)로 점차 진화하고 있다.

o 마이크로서비스 구조는 독립적으로 구성되는 다수의 자그마한 기능들(functions)의 결합으로 구성된다. 획일적인 서비스 합성을 마이크로서비스 구조로 변환하면, 개발 기간을 단축하고 확장성을 개선되어 서비스 개발의 자동화가 향상된다. 또한 마이크로서비스 구조의 서비스 합성을 위해 복수로 분할된 기능들을 배치하고 연결함에 있어서, stateless (즉 상태보존이 필요치 않은) 대부분의 기능들에 가상머신(Virtual Machine: VM) 보다 가볍고 신속한 컨테이너(container) 기반의 기능 구현이 적합한 것으로 알려져 있다. 따라서 컨테이너 중심의 마이크로서비스 구조에 대응하는 Platform as a Service (PaaS) 차원의 컨테이너 지향 SaaS 서비스 도구들이 클라우드 사업자를 중심으로 출시되어 확산되고 있다.

o 이러한 서비스 도구들은 개발자들이 편리하게 서비스들을 실증하도록 지원하지만, 개발자가 원하는 자유로운 인프라 활용에는 제한적이며 특정 클라우드 사업자에게 개발자를 종속(lock-in)시키는 문제가 있다. 그런데 클라우드 기반 서비스들의 일부는 서비스가 고정적인 형태가 아닌 동적인 서비스 합성을 요구한다. 예를 들면, 지리적으로 분산되어 있는 IoT 기기들은 인접한 위치에 있는 클라우드를 동적으로 활용하면 유리하다. 즉 IoT-Cloud 서비스의 동적 서비스 합성을 위해서는 공용 및 사설 클라우드 자원 중에서 자유롭게 선택할 수 있는 클라우드 인프라와 연동이 요구된다.

o 하지만 대다수의 클라우드 사업자들은 벤더 종속을 유도하면서 고객들을 모으고 있다. 개발자가 특정 클라우드 사업자가 제공하는 도구에 의존하게 되면, 다른 클라우드로 전환하는 것이 어렵다. 즉 Amazon EC2 및 S3와 같은 특정 클라우드 플랫폼에 맞춰 개발된 응용 서비스를 다른 클라우드로 쉽게 이전하기 힘들며, 클라우드 사업자의 서비스 조건 변경에 취약하다. 여러 사업자들이 제공하는 클라우드 구성 및 도구와 API 이질성은 기술적으로 호환되지 않으며, 이를 해결하는 상호운영성 및 이식성에 대한 연구들은 사업자 종속을 피하기 위한 중요한 도전과제이다. 즉 여러 클라우드 인프라들에 대한 상호운영성을 확보하여 동적인 서비스 합성이 가능하도록, 클라우드 서비스 브로커, 클라우드 표준화, SaaS 마이그레이션 등이 활발하게 연구되고 있다.

o 먼저 클라우드 서비스 브로커 (cloud service broker)는 개발자들에게 사용 목적과 요구되는 서비스 품질에 맞춰 클라우드 자원을 선택적으로 활용하게 보조한다. 그러나 대량의 클라우드 자원을 요구하는 서비스에 적합한 형태의 저렴한 자원 선택을 지원하는 도구에 가까우며, 마이크로서비스 기반의 IoT-Cloud 서비스에서 요구하는 동적인 서비스 합성을 지향하지 않는다. 또한 클라우드 표준화 차원에서는 다양한 클라우드 사업자들의 자원들을 활용하는 표준적인 API들을 시도한다. 먼저 CIMI(Cloud Infrastructure Management Interface)는 표준화된 API를 통해 클라우드 자원을 확보할 수 있다. P. Merle 등이 제안한 Open Cloud Computing Interface (OCCI)도 RESTful 형태의 표준화 API로 클라우드 상호운영성을 지원한다. 유사하게 XML 기반으로 클라우드 자원들을 확보하는 명세를 설계하는 연구들도 진행되었다. 하지만 개발자들이 새로운 명세 방식을 익히는 부담이 있으며, 동적으로 서비스 합성을 지원하기 위해 필요한 기능들을 개발자 및 사용자가 추가적으로 구성을 해야 하며 이러한 기능을 자동적으로 구성하는 동적 준비 기능이 부족하다.

o 또한 동적인 서비스 합성을 지원하기 위한 호환성을 위해 오버레이 차원의 가상화 방식을 통해 해결하려는 연구들도 있으며, SaaS 호환성을 해결하도록 컨테이

너 기반의 가상 클러스터를 구성한 것에 주목할 만하다. 그리고 SaaS 지향의 유연한 동적인 서비스 합성에 주목하면서, 이종 다수 클라우드 간의 SaaS 이전(migration) 연구들도 활발하다. 이들은 주로 클라우드 간에 동작중인 서비스를 이전하는 문제에 집중하지만, 서비스를 구성하고 운영하기 위해 필요한 관리 계층을 직접 다루지 않는 추세이다.

o 이와 같이 상기한 연구들은 대부분 클라우드의 상호운영성을 해결하거나 동작중인 서비스의 이전과 같이 한정된 부분에 대한 서비스 운영 이슈들을 개별적으로 해결하는데 집중하고 있다. 일부 상업적인 소프트웨어 중에서는 멀티 클라우드 상의 자원 확보를 코드 형태로 자동적으로 수행할 수 있도록 제공하며, 서비스의 운영 및 동작을 지원하기도 한다. 이러한 소프트웨어는 대규모의 데이터센터를 확보한 상태에서 서비스를 지속적으로 관리 및 운영하기에는 적합하지만, 여전히 벤더 종속의 가능성이 남아있다. 또한 소규모의 자원 중심으로 개발된 서비스를 빠르게 검증 및 운영하는 환경에서는 소프트웨어 비용이 상대적으로 부담이 될 수 있다.

o 다양한 분야에서 개발되고 있는 IoT-Cloud 서비스들을 신속하고 빠르게 검증 및 실증하기 위해서는 클라우드-네이티브 중심의 유연한 환경 구성에 대응할 수 있는 방법론이 필요하다. 또한 클라우드 인프라를 직접 그대로 사용하면서 IoT-Cloud 서비스가 요구하는 동적인 서비스 합성을 유연하고 신속하게 지원하기 힘든 제약을 해결해야 한다. 따라서 자동화 수준으로 특정 사업자에 종속되지 않는 특별한 환경 구성을 통해 클라우드 상호운영성을 확보할 수 있는 오픈소스 기반의 동적인 서비스 합성을 지원하는 기반을 마련하는 포괄적인 접근 방식이 요구된다.

o 이러한 문제점들을 극복하기 위해 본 기술문서에서는 다양한 서비스가 요구하는 동적인 서비스 합성을 지원하는 새로운ㄱ ᅦ층 추가에 기반한 개념을 제안한다. 즉 클라우드 인프라 계층 위에 얇은(razor-thin) 계층을 구성하여, 마이크로서비스 구조에 기반한 동적인 서비스 합성에 대응하는 개념을 목표로 한다.

2. OverCloud 개념 및 주요 구성

2.1. OverCloud 개념

o 다수의 클라우드 인프라에서 제공하는 방식을 그대로 사용해서는 클라우드의 상호운영성 확보와 IoT-Cloud 서비스가 요구하는 동적인 서비스 합성을 지원하기는 어려움이 있다. 첫째로, 클라우드 사업자들마다 서로 다른 인터페이스를 제공하고 있기 때문에, 이를 통한 자원 확보와 서비스 합성 모두를 수행하면 클라우드 상호운영성 확보에 제한이 될 수 있다. 둘째로, 클라우드 사업자들이 컨테이너 지향 SaaS 서비스 도구들을 그대로 사용하면, 특정 클라우드 인프라에 사용자를 종속시키기 때문에 유연하고 동적인 클라우드 인프라 활용에 제약이 된다. 따라서 특정 클라우드 사업자에 종속되지 않는 특별한 환경을 자유롭게 구성하고, 이를 통해 IoT-Cloud 서비스가 요구하는 동적인 서비스 합성에 대응할 수 있어야 한다.

o 본 기술문서에서는 그림 1과 같이 새로운 형태의 클라우드 활용 소프트웨어 개념에 대해 제안한다. 서비스를 위한 자 원 확보 과정과 서비스 합성 과정을 분리하고, 두 과정 사이의 연결을 지원하는 새로운 개념 형태의 계층인 OverCloud를 구성한다. OverCloud 계층은 여러 자원의 형태를 포함하는 UnderCloud 계층 상에서 사용자에게 동적인 서비스 합성을 제공해주기 위해 만들어지는 일관된 형태의 논리적인 계층이다. 제안하는 방법을 활용하면, UnderCloud 계층의 자원 유형이나 속성에 영향을 미치지 않는 상태에서 클라우드 상호운영성을 확보하면서 동적인 서비스 합성이 요구되는 IoT-Cloud 서비스 실증 지원이 가능하다. 제안하고 있는 OverCloud 개념의 특징은 다음과 같다.

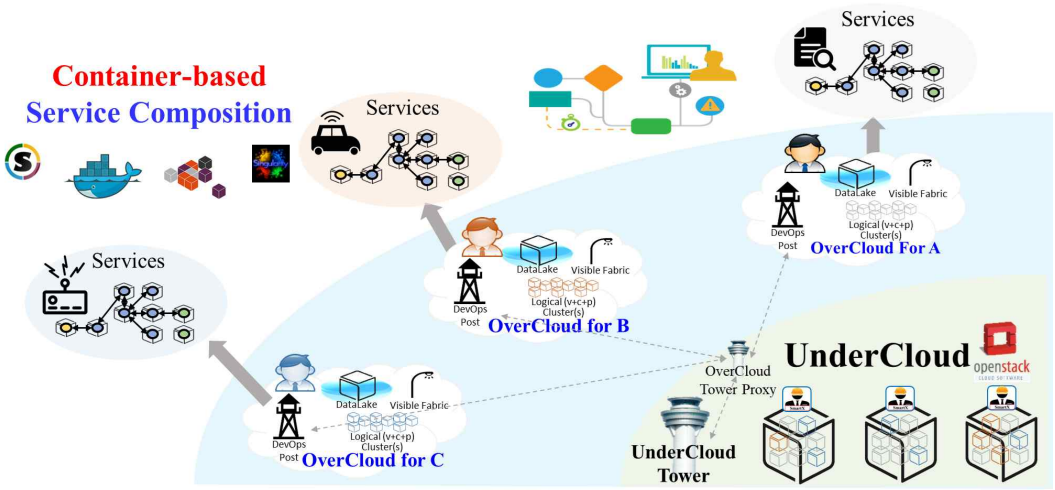


그림 1 제안하는 OverCloud 접근법

o 첫째로, 특별하게 주어진 UnderCloud 계층에서 클라우드 상호운영성을 확보하는 OverCloud 계층의 구성이 가능하다. 특정 클라우드 인프라에 종속적이지 않는 클라우드 자원 활용은 서로 다른 클라우드의 이질적인 특성 때문에 해결이 어려운 문제로 클라우드 간의 이질적인 특성을 최소화하고, 사업자 종속성을 피하기 위한 방법을 고려해야 한다. 하지만 일반적인 다수의 클라우드 환경에서 클라우드 상호운영성을 확보하는 것은 어렵다. 따라서, 미리 준비되어 있는 UnderCloud 환경을 가정하여 클라우드 상호운영성을 확보한 상태에서 OverCloud 계층을 구성하도록 설계하였다. 즉, 미리 준비된 UnderCloud가 주어진다면, UnderCloud가 제공하는 자원을 활용해 OverCloud 계층을 마음대로 손쉽게 구성이 가능하다. 대부분의 클라우드 사업자들은 클라우드 인프라를 활용하기 위한 API를 지원한다. 특정 클라우드 사업자에 종속시키는 컨테이너 지향 SaaS 서비스 API의 사용을 배제하고 자원 확보 중심의 API만을 활용해 리눅스 기반의 환경에서 일관되게 동작할 수 있는 OverCloud를 설계하였다. 따라서 자원을 확보할 수 있는 최소한의 API만을 활용해서, 준비된 UnderCloud 계층에서 클라우드 상호운영성의 확보가 가능하다.

o 둘째로, 동적인 OverCloud 계층의 구성 및 소멸이 가능하다. IoT 기기의 지리적인 위치에 따라 클라우드 인프라를 동적으로 활용하는 IoT-Cloud 서비스 실증을 위해서는 OverCloud 계층을 동적으로 구성 및 소멸할 수 있어야 한다. OverCloud 계층의 동적인 특성을 위해서는 OverCloud 계층을 구성하는 요소들의 오버헤드가 적고 가벼워야 한다. 컨테이너 기술을 활용하면, 리눅스 기반의 클라우드 환경에서 오버헤드가 적은 OverCloud 구성이 가능하다. 또한 복잡한 단계적인 순서를 체계적으로 정리가 가능한 워크플로우 컨셉을 활용하면 효율적인 방식으로 여러 병렬적인 작업들을 자동적으로 진행할 수 있다. 즉, 워크플로우 중심의 소프트웨어를 활용하여 OverCloud 계층의 구성 및 소멸에 관한 전반적인 작업을 효율적인 방식으로 자동적으로 수행한다.

o 셋째로, OverCloud 계층은 서비스 실증을 위해 필요한 기능들을 제공한다. 개발자들은 개발하고자 하는 서비스의 실증을 위해 서비스 합성 및 서비스 운영으로부터 발생되는 데이터들을 관리하고 상황을 파악할 수 있는 도구가 필요하다. OverCloud 계층을 활용하면, 개발자는 마이크로서비스 구조 중심의 서비스 합성을 지원하는 오픈소스 도구들을 이용 가능하다. 따라서 컨테이너 기반의 IoT-Cloud 서비스 실증에 대해서 대응이 가능하다. 또한 서비스를 운영하면서 발생되는 데이터들을 관리하기 위한 필요 요소들을 제공한다. OverCloud 계층은 개발자가 필요한 여러 요소들을 포함하고 있어, 서비스에서 발생되는 데이터들을 관리하거나 자신이 확보한 OverCloud 계층에 대한 지속적인 상황 파악 및 운영

이 가능하다.

2.2. OverCloud 주요 구성

o OverCloud는 미리 준비된 UnderCloud 계층에서 확보한 자원을 바탕으로, UnderCloud 계층과는 독립적으로 만들어지는 새로운 계층이며, 그림 2는 OverCloud 계층의 컨셉을 보여준다. OverCloud 계층은 사용자들에게 마이크로서비스 중심의 서비스 합성과 OverCloud 계층의 전반적인 운영을 지원하기 위해 필요한 여러 요소들을 포함한다. OverCloud 계층의 내부 구성은 DevOps Post, 논리적 클러스터(Logical Clusters) 및 데이터레이크(DataLake), 가시화 패브릭(Visible Fabric)으로 구성되어 있다.

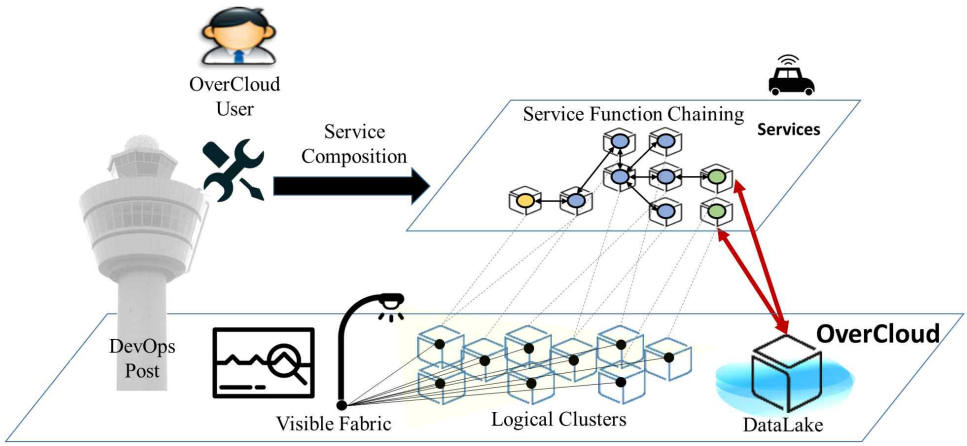


그림 2 OverCloud 계층의 컨셉

o DevOps Post는 OverCloud 계층의 다른 요소들을 유기적으로 연결하며 OverCloud의 전체적인 관리를 담당한다. 또한 사용자가 실증하고자 하는 서비스에 대응하는 미리 설계된 컨테이너 기반의 서비스 패턴을 준비하면, 동적인 서비스 합성을 진행할 수 있도록 도구들을 지원한다. DevOps Post를 활용하면, 마이크로서비스 구조 중심으로 서비스를 기능들로 분리하고, 각 기능들의 배치(Placement), 연결(Stitching), 실행(Execution)과 같은 서비스 합성의 전반적인 단계들을 오픈소스 기반의 도구들로 수행이 가능하다.

o 서비스 실증을 수행하기 위해 논리적 클러스터(Logical Clusters)라는 자원의 집합을 활용한다. 논리적 클러스터는 UnderCloud 계층에서 제공하는 자원과는 다르게 개발자들이 마이크로서비스 구조에 기반한 서비스 합성을 할 수 있도록 미리 준비하는 OverCloud 계층의 논리적인 자원 모음을 말한다. 사용자들은 DevOps Post를 활용하여 서비스 합성에 필요한 논리적 클러스터를 이용해 컨테이너 기반

의 서비스 합성을 수행한다. 서비스 합성이 끝나고 서비스가 정상적으로 수행이 되면, 서비스에서 발생하는 데이터들은 데이터레이크(DataLake)에 저장이 가능하다. 데이터레이크는 논리적인 클러스터에 연결될 수 있는 컨테이너에 대응 가능한 분산 스토리지 형태이다.

o 가시화 패브릭(Visible Fabric)은 OverCloud 계층의 구성요소 간의 다계층 가시성을 지원할 수 있을 정도로 면밀하고 세밀한 연결(Inter-connect)를 뜻한다. 세밀한 연결을 보조하는 시각화을 위해서 OverCloud 계층 내부의 상황을 면밀히 파악할 수 있도록 다수의 가시화 데이터 수집기를 에이전트 형태로 논리적 클러스터에 주입한다. 가시화의 대상으로 OverCloud 계층 내의 논리적 클러스터의 자원 상황이나, 이들의 연결상태를 보여주는 플로우를 대상으로 한다. 이를 통해서 OverCloud 계층의 논리적 클러스터의 사용량 및 네트워킹의 연결 상황을 면밀히 파악 가능하다.

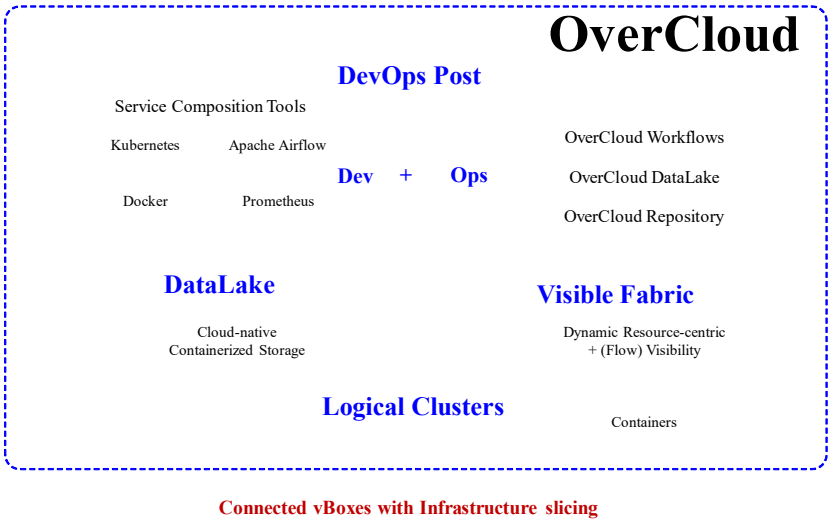


그림 3 OverCloud 구성 요소

o 그림 3은 OverCloud 계층의 내부 구성요소들을 자세히 보여준다. OverCloud 계층을 구성하기 위해서는, 그림 하단에 위치한 슬라이스 형태로 서로 연결된 가상 박스의 자원 모음을 UnderCloud 계층으로 지원 받아야한다. 이 자원 집합을 통해 전반적인 OverCloud 계층의 요소들이 동적으로 구성된다. 전반적인 구성 과정은 다음과 같다.

o 먼저, OverCloud를 운영하기 위한 Ops 중심의 DevOps Post와 OverCloud 계층의 기본 요소들을 병렬적으로 구성한다. UnderCloud 계층으로부터 확보한 자원들은 DevOps Post를 위한 일부 자원을 제외하고는 논리적 클러스터에 할당한다.

논리적 클러스터가 준비가 되고 난 후, 이들의 상황을 파악할 수 있는 가시화 패브릭과 사용자에게 스토리지 기능을 제공해주는 데이터레이크가 병렬적으로 구성이 진행된다. 가시화 패브릭은 동적인 OverCloud 계층의 특성 상 고정적인 토폴로지가 아닌 동적인 토폴로지가 적용 가능한 가시성을 고려한다. 데이터레이크는 서비스 합성에 대응 가능한 스토리지 구성을 고려하여, 간단하게는 LXC 컨테이너에서 제공하는 볼륨 스토리지 API를 활용해 구성하거나, Kubernetes 컨테이너 오케스트레이션 도구와 함께 활용할 수 있는 CNCF의 Rook를 고려해 클라우드-네이티브 컴퓨팅 중심의 컨테이너 기반의 스토리지를 지원하도록 한다.

o 논리적 클러스터, 데이터레이크, 가시화 패브릭이 준비가 되면, 서비스 합성에 필요한 자원의 세 가지 요소(컴퓨팅/네트워킹/스토리지)가 준비가 된 셈이다. 즉, 컴퓨팅은 논리적 클러스터가 담당하며, 네트워킹은 논리적 클러스터가 부분적으로 담당하면서 가시화 패브릭이 보조하는 역할을 한다. 스토리지는 데이터레이크가 담당하는 형태이다.

o 위의 요소들이 동적으로 준비가 되면서, 한편으로는 운영 중심의 기능을 제공하는 DevOps Post는 서비스 합성을 지원하는 기능들을 준비하면서 완전한 DevOps Post로 진화한다. DevOps Post는 마이크로서비스 구조에 대응할 수 있는 오픈소스 기반의 서비스 합성 도구들을 지원한다. 이를 활용하면, IoT-Cloud 서비스의 합성 이외에도 마이크서비스 구조 중심의 서비스 유형에 일부 대응이 가능하다. 즉, Docker 컨테이너를 이용하여 3-티어의 웹 서비스, IoT-Cloud 서비스 및 BigData 서비스 등에 전반적으로 활용이 가능하다. HPC 서비스의 경우 Singularity, Shifter, UGE (Univa Grid Engine) 등 다양한 컨테이너 도구들을 활용해서 대응이 가능하다.

3. OverCloud 개념 검증을 위한 소프트웨어 설계 및 구현

3.1. OverCloud 개념 검증을 위한 설계

o 본 논문에서 제안하고 있는 OverCloud 개념의 실질적인 검증을 위해서 사용자 및 개발자로부터 받은 요구사항을 이해하고 이를 바탕으로 OverCloud 계층을 구성하고 관리할 수 있는 소프트웨어가 필요하다. 이를 구현하기 위해 고려한 요구사항은 다음과 같다.

o 첫째로 준비된 UnderCloud 인프라로부터 동적으로 OverCloud 계층을 구성 및 소멸할 수 있어야 한다. 두 번째로, OverCloud 계층을 최대한 오버헤드가 적도록 구성해야 한다. 마지막으로 OverCloud 계층의 구성 및 소멸에 관한 전반적인 운영 요소들이 체계적이고 효율적인 순서에 의해 진행이 되어야 한다.

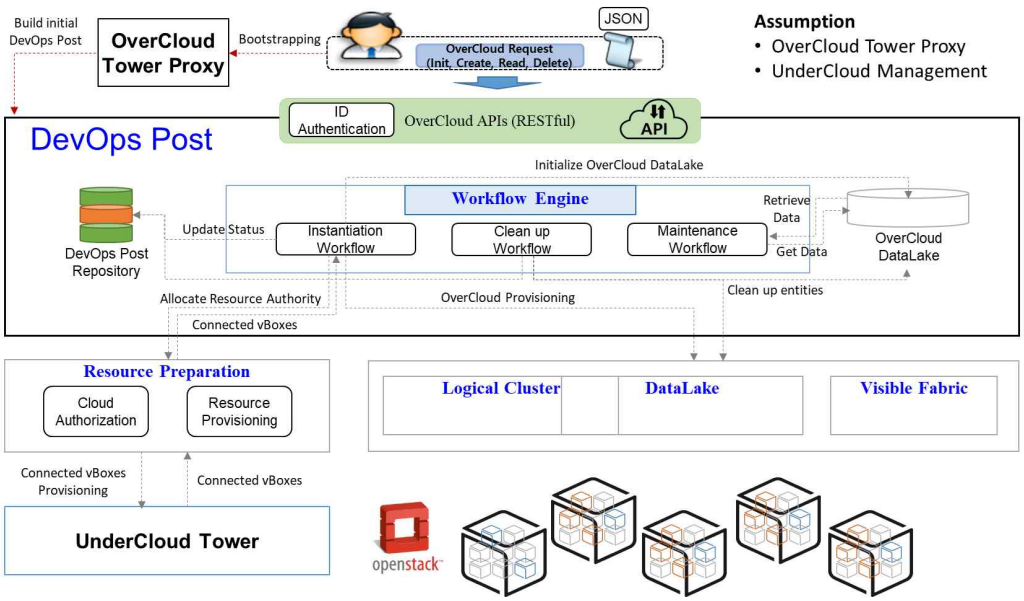


그림 4 OverCloud 개념 검증을 위한 소프트웨어 디자인

o 위의 요구사항을 만족하기 위해서 그림 5와 같이 소프트웨어를 설계하였다. 가정사항으로, 슬라이싱이 적용된 서로 연결되어 있는 가상 박스 모음 제공을 관리하는 UnderCloud Management와, OverCloud Tower Proxy가 준비되어 있다고 가정한다. OverCloud Tower Proxy는 주어진 UnderCloud에서 OverCloud의 전반적인 동작을 담당하는 OverCloud Tower가 없다면, 이를 준비해주는 기능을 한다.

o OverCloud 계층의 구성 및 소멸에 관한 전반적인 동작은 사용자의 API 요청을

통해서 진행이 된다. 그림 5의 왼쪽 하단에 위치한 자원 준비부에서는 사용자의 OverCloud 요청이 들어오면, UnderCloud Tower를 통해 UnderCloud의 자원의 권한을 먼저 확보한다. 즉, 슬라이싱이 적용된 서로 연결된 형태의 가상 박스 집합의 자원 권한을 받는다.

o DevOps Post에서는 자원 준비부로부터 자원의 권한이 확보되면, 사용자가 요청하는 OverCloud 동작에 대한 수행을 진행한다. OverCloud 계층의 구성요소를 최대한 가볍게 구성하기 위해 가상화 방법인 컨테이너를 활용하여, 특정 UnderCloud 인프라에 영향 받지 않는 환경을 가볍게 구성한다. 그리고 내부의 워크플로우 엔진을 활용해 OverCloud 계층의 구성 및 소멸에 대한 전반적인 관리를 자동화 수준으로 진행한다.

o OverCloud 계층의 구성 및 소멸이 완료가 되면, 이를 통해 발생되는 OverCloud 계층의 상태 정보들과 워크플로우의 설계 자료들은 DevOps Post의 레파지토리에 저장하고 관리한다. 또한 OverCloud 데이터레이크는 개발자를 위한 데이터레이크와 별개로, 가시화 패브릭으로부터 수집된 시계열 기반의 데이터들을 저장한다.

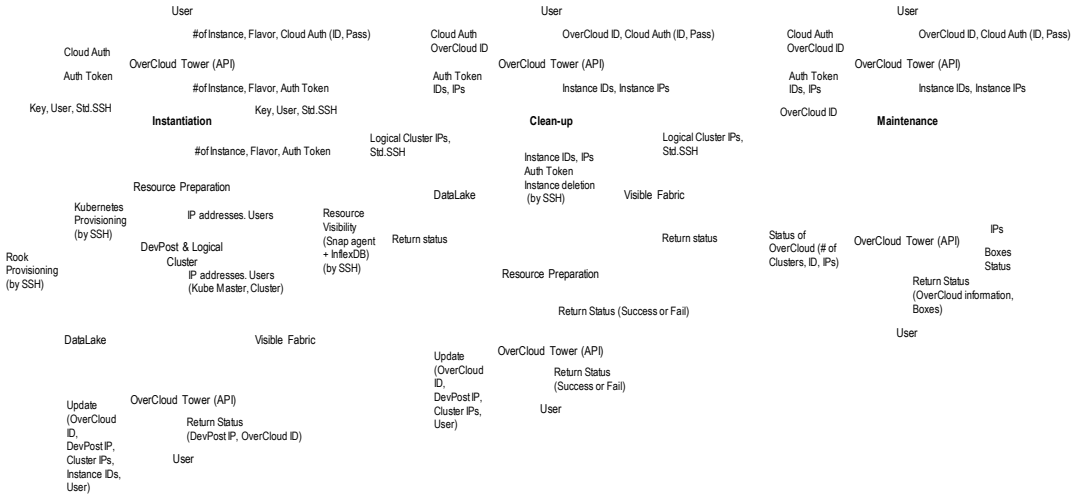
3.2. OverCloud 개념 검증을 위한 구현

o 제안하는 OverCloud 개념을 검증하기 위해 설계한 소프트웨어를 구현하였다. 먼저, UnderCloud 환경은 오픈스택 클라우드와 아마존 AWS를 적용할 수 있도록 구현을 하였다. ID 인증을 위해서, 오픈스택의 사용자 인증은 Keystone의 ID를 통해서 진행하며, 아마존 AWS의 인증은 IAM에서 부여 받을 수 있는 엑세스 키 ID 및 보안 엑세스 키를 통해서 인증을 진행하도록 구현하였다. 클라우드에 제공받는 자원의 형태는 슬라이싱이 적용된 오버레이 네트워킹으로 연결된 가상머신 모음을 지원하도록 가정하였다. 사용자의 요청을 받는 API는 파이썬의 flask를 활용하여, post, get, delete 메소드 기반의 생성, 확인, 소멸에 대한 기능을 RESTful API 형태로 구현하였다.

o UnderCloud에서 제공되는 자원을 통해 로지컬 클러스터를 구성하기 위해서 컨테이너 오케스트레이션 도구인 Kubernetes를 활용하는 형태로 구현하였다. 컨테이너 기반의 로지컬 클러스터에 대응하는 데이터레이크 구현을 위해서 오픈소스인 Rook을 활용해서 Ceph 기반의 분산 스토리지를 동적으로 구성할 수 있도록 구현하였다. 가시성 패브릭을 위해서는 기 개발된 가시성 소프트웨어를 적용하는 수준으로 진행하였으며, 고정된 토폴로지에 대해서 자원에 대한 전반적인 상황을 가시화해서 확인이 가능하다. 또한 서비스 합성에 대한 상황 파악을 지원하기 위해 Weave Scope와 Prometheus 오픈소스 도구를 활용해서 워크로드 가시화를 보

조하도록 구현하였다.

o OverCloud 계층의 구성 소멸에 대한 전반적인 진행을 위해 워크플로우 엔진은 오픈스택의 Mistral을 활용하였다. 오픈스택의 Mistral은 오픈스택 환경에 종속적이지 않고 독립적으로 수행할 수 있어, 특정 클라우드에 제한되지 않고 다양한 클라우드 환경에 적용이 가능한 이점이 있다.



(a) OverCloud Instantiation Workflow (b) OverCloud Clean up Workflow (c) OverCloud Maintenance Workflow

그림 5 OverCloud 운영을 위한 워크플로우

o 워크플로우 구현을 위해 OverCloud 계층의 생성과 소멸, 유지에 대한 워크플로우를 그림 5와 같이 설계하였다. 워크플로우는 오픈스택의 Mistral 서비스에서 제공하는 DSL (Domain Specific Langue)을 이용하였으며, 역 워크플로우 (Reverse Workflow) 특성을 활용해 구현하였다. 역 워크플로우는 각 작업들마다 의존성을 둘 수 있어, 분기를 두면서 병렬적으로 작업을 진행하면서도 작업 간 동기를 맞추면서 작업을 진행 가능하다. DSL은 YAML 형태의 템플릿 형식이며, 명세에 따라 정해진 작업들을 수행한다.

o 워크플로우의 각 작업들은 오픈스택에 의존적일 수 있는 API를 사용하지 않고 SSH를 통한 스크립트 수행으로 구현하였기 때문에 오픈스택이 아닌 다른 클라우드 환경에서도 호환적으로 동작할 수 있도록 설계하였으며, 각 작업에서 생성되는 정보들이 다음 작업들에게 유기적으로 연결되는 형태로 구성하였다. 따라서 OverCloud 계층의 생성과 소멸은 모두 워크플로우 엔진에 의해서 자동화적으로 수행이 된다.

o 그림 5(a)의 OverCloud 계층의 구성 워크플로우의 진행 순서는 다음과 같다. 먼저, 사용자의 API 요청을 확인하고 이에 맞는 워크플로우인 생성 워크플로우를 수행한다. 첫째로 자원의 권한을 UnderCloud Tower로부터 확보한다. 클라우드 인증을 거쳐 자원의 권한이 확보되면, 기본적인 DevOps Post가 구성이 되면서 컨테이너 오케스트레이션 도구를 활용하여 논리적 클러스터를 준비한다. 둘째로 데이터레이크와 가시화 패브릭을 병렬적으로 구성한다. 이 때, 두 요소들은 로지컬 클러스터의 준비가 완료가 되고 난 뒤에 수행이 된다. 마지막으로 OverCloud 계층의 구성요소가 모두 구성이 되었다면, OverCloud의 생성 정보들을 OverCloud 레파지토리에 업데이트 한다. 그리고 사용자 API의 리턴 값으로 OverCloud ID 및 생성 정보들을 반환한다.

o 그림 5(b)의 OverCloud 계층의 소멸 워크플로우의 진행 순서는 위와 비슷하지만 역순이다. 먼저, 사용자의 API 요청이 들어오면, 소멸 워크플로우의 작업을 수행한다. 첫째로, 사용자가 수행하였던 데이터레이크를 보존 작업을 진행하면서 병렬적으로 가시화 패브릭을 제거한다. 둘째로, 데이터의 보존이 끝나면 데이터레이크 회수 작업을 진행한다. 데이터레이크와 가시화 패브릭 회수 작업이 끝났다면, 논리적 클러스터들의 자원을 회수한다. 마지막으로 DevOps Post을 소멸하면서 최종적으로 모든 자원의 권한을 돌려준다. 그리고 OverCloud 레파지토리에 OverCloud 상황 정보를 업데이트하면서 최종 결과를 성공 혹은 실패 형식으로 사용자에게 반환한다.

o 위에서 서술한 모든 과정들은 사용자에 API 요청에 맞춰 적절한 워크플로우를 선택해 수행하게 된다. 워크플로우 내의 모든 작업들은 오픈스택의 Mistral 워크플로우 엔진에 의해서 자동으로 수행이 된다. 위에서 구현한 소프트웨어는 다른 사람들이 사용 및 개선할 수 있도록 오픈 소스로 공개하였으며, http://Github.com/K-OverCloud/Dynamic-OverCloud 주소를 통해 확인이 가능하다.

4. OverCloud 개념 검증을 위한 환경 구성 및 검증 결과

4.1. OverCloud 검증 환경 및 구성/소멸 검증

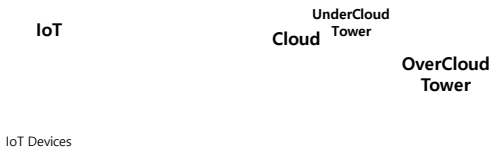


그림 6 OverCloud 검증을 위한 실험 환경

o OverCloud 계층의 구성/소멸에 대한 검증을 진행하기 위해 그림 6과 같이 실험 환경을 구축하였다. 멀티 사이트를 대상으로 OverCloud 검증을 위해서 오픈스택 환경을 직접 구성하였고, 아마존 AWS의 northeast2 리전을 활용하였다. 오픈스택 구성을 위해서 Intel Xeon-D 6 코어, 32G RAM, 512GB SSD로 구성된 SuperMicro E200-8D 모델 4대를 이용하였다. 1대는 오픈스택의 컨트롤러 역할을 수행하고, 3대는 오픈스택의 컴퓨트 역할을 수행하였다. 오픈스택의 컨트롤 역할을 하는 서버는 추가적으로 UnderCloud Tower의 역할 및 OverCloud Tower Proxy 역할을 수행하도록 구성하였다. 동 모델의 E200-8D 1대는 OverCloud Tower 역할을 하도록 구성하였다. 그리고 IoT-Cloud 서비스를 수행하기 위해 5대의 라즈베리 파이 2를 서버실에 배치하였다.

o 먼저, 오픈스택 환경에서 OverCloud 계층의 구성 검증을 위해서 m1.small 사이즈의 3개의 논리적 클러스터와 함께 OverCloud 구성 요청을 진행하였다. 요청을 위한 API 명령은 다음과 같다.

o curl -X POST -H "Content-Type: application/json" -d '{"provider": "OpenStack", "number": "3", "size": "m1.small"}' http://$IP:6125/overclouds.

o 오픈스택의 Mistral은 사용자의 입력 요청에 따라 알맞은 워크플로우를 선택하고 이를 자동적으로 수행한다. UnderCloud Tower로부터 자원을 확보하기 위해, 자원 관리부를 통해 m1.small 3개의 논리적 클러스터를 할당한다. 이때, UnderCloud는 VXLAN 기반의 오버레이 네트워킹으로 연결된 가상머신 인스턴

스 4개를 할당한다. 4개를 할당하는 이유는 DevOps Post 역할을 하는 인스턴스와 논리적 클러스터를 위함이다. 위와 같은 작업이 진행이 되고 난 후, DevOps Post에 워크플로우를 통해 자동적으로 스크립트가 주입이 되면서, DevOps Post에서 필요한 요소들이 채워지게 된다. 즉, OverCloud 운영을 위한 레파지토리 및 데이터레이크가 준비되면서, 이와 동시에 개발자의 서비스 합성을 위한 오픈 소스 도구들이 준비가 된다.

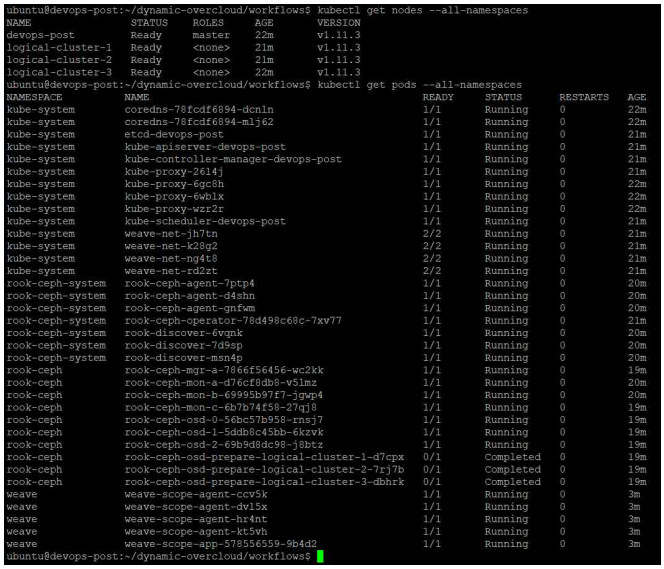


그림 7 DevOps Post의 Kubernetes 현황

o 그림 7은 DevOps Post의 Kubernetes를 활용해서 3개의 논리적 클러스터가 하나의 자원 풀로 묶이고, 데이터레이크 및 가시화 패브릭을 위한 여러가지 컨테이너들이 준비가 되어있는 것을 보여준다. 데이터레이크는 오픈소스인 Rook을 활용해 Ceph 기반의 스토리지 구성을 Kubernetes의 컨테이너 묶음의 단위인 팟(Pod) 형태로 구성된 것을 보여준다. 가시화 패브릭을 위해서는 에이전트 형태의 데이터 수집기가 논리적인 클러스터에 설치하게 되며, 또한 추가적으로 워크로드에 대한 가시화를 지원하기 위해 Weave Scope의 컨테이너가 설치가 되어있는 것을 확인할 수 있다.

o 그림 8은 가시화 패브릭을 활용해 가시화를 보여주고 있다. 상단은 로지컬 클러스터의 자원을 파악할 수 있도록 차트나 시계열 형태의 가시화를 보여주고 있으며, 하단은 컨테이너 기반으로 동작하고 있는 워크로드를 기능별 연결 관계에 따라 그래프 기반으로 보여주고 있다. DevOps Post 내의 존재하는 OverCloud 데이터레이크는 이러한 에이전트로부터 받아지는 각종 데이터들을 저장한다. 데이터레이크는 InfluxDB와 Prometheus와 같이 여러 데이터베이스를 통합적으로 갖고 있다. 위의 과정이 모두 완료가 되면, DevOps Post의 레파지토리에

OverCloud를 생성 정보들을 저장한다. OverCloud ID, DevOps Post IP, 논리적 클러스터 IP 및 개수 등이 저장이 되며, 최종적으로 API을 요청한 사용자에게 위의 정보들을 리턴한다.

그림 8 Visible Fabric을 활용한 가시화

o 마찬가지로 아마존의 AWS를 활용하여 OverCloud 계층을 구성해보았다. C5d.xlarge 사이즈(4 cores, 8GB RAM)를 갖는 논리적 클러스터 3개를 갖는 OverCloud 구성을 요청하였다. 그림 10은 OverCloud를 구성하기 위해 생성된 4개의 아마존 AWS의 인스턴스를 보여주고 있다. DevOps Post의 충분한 컴퓨팅 능력 및 용량 확보를 위해 t2.2xlarge 사이즈(8 cores, 32GB RAM)를 갖고 있으며, 로지컬 클러스터 구성을 위해 C5d.xlarge 사이즈의 인스턴스가 3개가 생성된 것을 볼 수 있다. OverCloud 계층의 구성은, 오픈스택 Mistral과 연동된 Horizon 대시보드를 통해서 워크플로우의 작업 간의 진행 상황을 확인할 수 있다. 이러한 오픈스택의 Horizon은 실험을 위해 구축한 오픈스택 환경과 독립적으로 구성되어 있어서, 아마존 AWS를 활용해 OverCloud를 구성할 때도 마찬가지로 워크플로우 진행 상황을 확인 가능하다.

o 다음은 OverCloud를 구성한 후, 이와 반대로 소멸에 대한 검증을 진행해보았다. 사용자가 OverCloud 소멸을 위한 API를 요청하면, OverCloud ID가 타당한지 확

인한다. 확인하는 절차가 끝나면, 데이터레이크와 가시화 패브릭이 먼저 소멸이 진행이 된다. 이러한 과정이 완료가 되면, 논리적 클러스터들을 제거한다. 제거 과정이 끝나면 최종적으로 DevOps Post도 제거가 되며, UnderCloud Tower는 할당했던 자원들을 다시 회수하게 된다. 위의 모든 과정이 끝나면, OverCloud 소멸에 대한 정보를 OverCloud 레파지토리에 저장하며, 최종적인 결과메시지를 API를 요청한 사용자에게 반환한다.

Instantiation	6 min 33 sec	5 min 47 sec
Clean-up	52 sec	48 sec

그림 9 OverCloud 구성 및 소멸에 대한 평균 시간

o 그림 9는 구축한 오픈스택 클라우드와 아마존 AWS 클라우드를 활용해서 OverCloud를 구성할 때, 준비되는 시간을 평균적으로 계산해 반올림한 값을 보여준다. 총 5번씩 구성 및 소멸을 수행하였으며, 10분 내외의 시간이 주어지면 동적으로 신속하게 OverCloud 계층을 멀티 클라우드 환경에서 자동으로 구성 및 소멸이 가능한 것을 확인하였다. 따라서 자신이 보유한 클라우드 인프라에서 서비스의 특성에 맞게 적절한 자원들을 동적으로 선택해서 서비스를 개발 및 실증할 수 있는 환경을 만들 수 있다. 또한 오픈 소스 중심으로 OverCloud를 구성하기 때문에 특정 벤더에 종속이 되지 않고 클라우드-네이티브 중심의 서비스 지원 환경을 유연하게 준비할 수 있다.

4.2. IoT-Cloud 서비스 지원에 대한 검증

o OverCloud를 활용한 서비스 실증 검증을 위해, 스마트 에너지 IoT-Cloud 서비스 사례를 적용하는 검증을 수행하였다. 스마트 에너지 서비스는 라즈베리 파이2를 활용하여 서버실의 온도와 습도, 그리고 전력 소모량을 수집하며, 서버의 시스템 온도를 수집한다. 수집한 데이터에서 서버의 고온 혹은 전력 과다 소비와 같은 이상 현상을 감지하면 그에 맞는 조치로 서버를 절전모드로 전환시키거나 서버 관리자에게 이를 알린다. 한편으로는 웹 브라우저로 접속할 수 있는 대시보드에 상황을 시각화하여 모니터링 할 수 있게 한다. 위의 서비스는 마이크로서비스 구조 중심으로 스마트 에너지 서비스를 기능 별로 분리하고, 각 기능들을 컨테이너에 담아서 연결할 수 있도록 개발하였다.

o 스마트 에너지 IoT-Cloud 서비스 적용을 위해 그림 7에서 구성한 실험 환경을 재이용하였다. 서비스를 적용하기 위해 구축한 오픈스택 클라우드에서 3개의 m1.small 크기의 논리적 클러스터를 이용할 수 있는 OverCloud 계층 구성을 요

청하였다. OverCloud 구성이 완료가 되면, 사용자는 DevOps Post를 활용해서 서비스 합성에 대한 도구들을 지원받을 수 있는 준비가 된다.

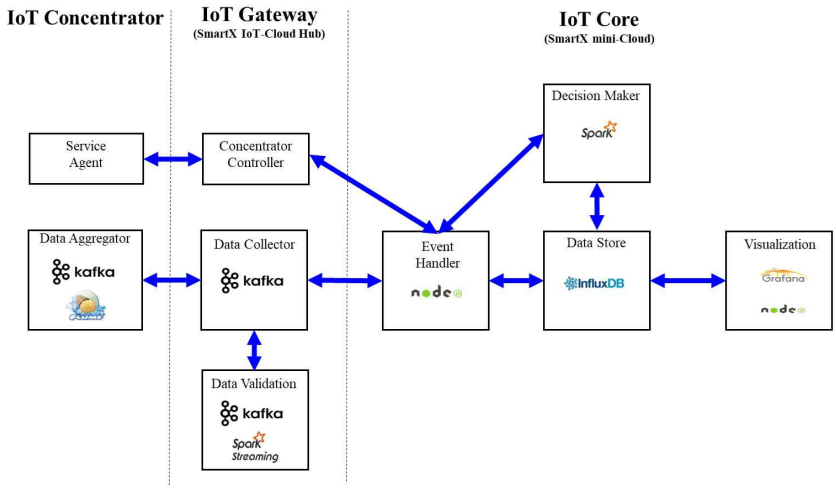


그림 10 스마트 에너지 서비스를 위한 서비스 합성

o 그림 10은 스마트 에너지 IoT-Cloud 서비스 합성을 보여준다. 마이크로서비스 구조에 기반한 컨테이너 합성이 가능하도록 스마트 에너지 서비스를 기능 단위로 분리해서 구현하였다. 그림 내부의 사각형들은 서비스를 이루기 위한 기능(Functions)들을 말한다. IoT-Core에 위치한 기능들은 OverCloud 계층 상에서 동작하는 기능들이며, IoT Concentrator와 IoT Gateway는 IoT 기기 및 IoT 허브에서 동작하는 기능들이다. 이러한 기능들을 연결하기 위한 과정 (Service Function Chaining)을 진행하기 위해서 DevOps Post에서 지원하는 컨테이너 오케스트레이션을 활용해 구성이 가능하도록 명세하였다.

o 명세된 파일을 통해 컨테이너 오케스트레이션 도구가 서비스 합성을 수행하게 되고, 서비스 합성 수행이 완료가 되면 스마트 에너지 서비스를 구성하고 있는 기능들이 정상적으로 배포되고 있는 것을 개발한 스마트에너지 서비스의 대시보드에서 확인할 수 있다. 그림 11과 같이 스마트 에너지 서비스의 대시보드를 접속해 정상적으로 서버실의 온도나 습도와 같은 상황정보를 파악할 수 있다.

o 즉 멀티 클라우드가 준비된 상황에서, OverCloud를 활용해 서비스를 실증하기 위한 환경 구성을 동적으로 구성하고, 개발하고 있는 IoT-Cloud 서비스의 동작 및 운영에 대해서 검증하였다.

o 제안하는 OverCloud를 활용하지 않았다면, 클라우드 환경에서 마이크로서비스 구조 중심으로 서비스를 실증하기 위한 컨테이너 오케스트레이션 환경을 직접 구축해야한다. 아마존 AWS 클라우드를 활용하는 경우를 예를 들자면, EC2 서비스

를 통해 가상 머신들을 준비하고 아마존 Elastic Container 서비스를 통해 컨테이너를 사용하도록 준비해야 한다. 또한 사용자의 데이터를 지속적으로 보관하기 위해 아마존 S3 서비스를 추가적으로 활용해야 하며, 이때 컨테이너에 직접 대응하는 저장소가 아닌 EC2 서비스의 가상머신에 대응하는 스토리지를 제공한다. 이러한 과정에서 상용 소프트웨어를 포함한 다양한 서비스들을 사용하게 되므로 벤더 종속의 위험이 있고, 원하는 환경을 직접 구축하는 많은 노력이 필요하다. 반면에 제안한 OverCloud를 활용하면, 상기한 과정들을 대부분 자동화하여 많은 직접 작업들이 생략된다. 그 결과 사용자 서비스의 개발 및 검증 주기를 더욱 빠르게 만들면서 다양한 클라우드 인프라 상에 서비스를 적용하는 기회를 제공해 양질의 서비스 개발을 하도록 지원한다.

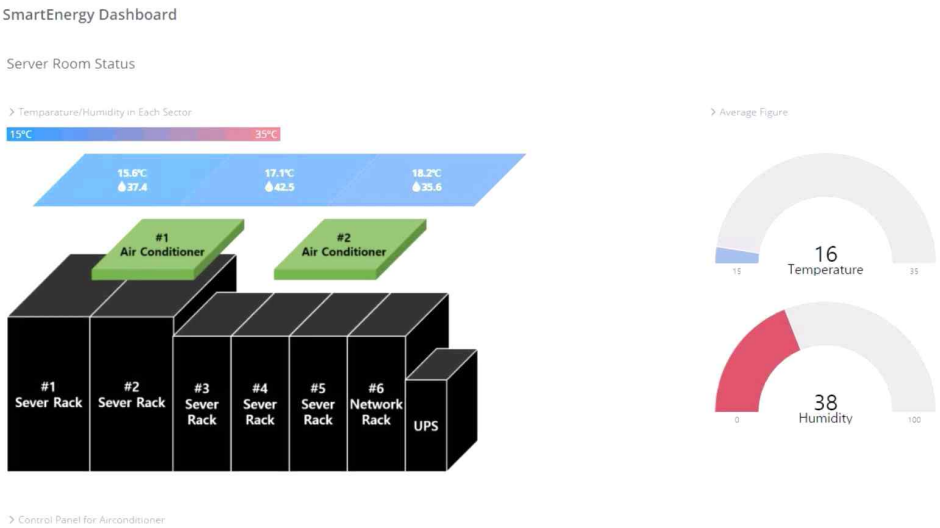


그림 11 OverCLoud를 활용한 스마트 에너지 서비스 수행장면

References

SaaS OverCloud 기술 문서

- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경 될 수 있습니다.
- 본 문서와 관련된 대한 문의 사항은 아래의 정보를 참조하시길 바랍니다. (Homepage: https://smartx.kr, E-mail: contact@smartx.kr)

작성기관: 광주과학기술원
작성년월: 2018/12