SaaS OverCloud 기술문서 #3

단일 하이브리드 클라우드 인프라/플랫폼 상의 Web-App-DB 3-tier 대응 SaaS 응용에 대한 SaaS OverCloud 호환성 설계 및 구현

Document No. SaaS OverCloud #3
Version 1.0
Date 2016-10-12
Author(s) GIST#1 Team

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2016. 08. 17	김종원, 박선, 김남곤, 한정수	
0.2	2016. 10. 11	한정수	
0.3	2016. 11. 08	한정수, 김철원	
1.0			

이 논문은 2016년도 정부(미래창조과학부)의 재원으로
정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R7117-16-0218,
이종 다수 클라우드 간의 자동화된 SaaS 호환성 지원 기술 개발)

This work 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

SaaS OverCloud #3. 단일 하이브리드 클라우드 인프라/플랫폼 상의 Web-App-DB 3-tier 대응 SaaS 응용에 대한 SaaS OverCloud 호환성 설계 및 구현

그림 목차

SaaS OverCloud 기술문서 #3.
단일 하이브리드 인프라/플랫폼 상의
Web-App-DB 3-tier 대응 SaaS 응용에 대한
SaaS OverCloud 호환성 설계 및 구현

1. SaaS OverCloud 개요

1.1. 목적 및 개요

o 본 문서에서는 Web-App-DB 3-tier 대응 SaaS 응용을 위한 SaaS OverCloud 호환성 설계 및 구현하는 방법에 대해 상세히 기술한다. 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 SaaS OverCloud 만을 고려하여 원하는 SaaS 응용을 가능하게 한다.

1.2. SaaS OverCloud

o SaaS 응용/서비스들이 기하급수적으로 폭발하면서 SaaS 응용 및 구축에 대한 관심이 증가하고 있다. 하지만 여전히 SaaS 응용/서비스의 범람 속에서도 여전히 SaaS 개발자는 다양한 개발/운영 환경의 이질성과 의존성으로 인해 어려움을 겪고 있다. 또한 기존의 호환성 지원 방식으로는 이질적인 Cloud 인프라/플랫폼 상의 고수준/고부가 SaaS 응용에 대한 체계적인 지원이 불가능하다. 따라서 이러한 고질적인 문제점들을 타파하기 위해 이종 다수의 클라우드 인프라/플랫폼 환경에 독립적으로 원활하게 SaaS 응용/서비스를 개발/운영하도록 지원하는 체계적이고 자동화된 SaaS 호환성 확보 기술이 필요하다.

o 이러한 필요성에 맞물려 공개 클라우드 벤더들이 공용 클라우드와 사설 클라우드를 통합할 수 있는 환경인 하이브리드 클라우드를 구축 및 관리하는데 관심을 가지고 있으며, 여러 클라우드 회사들이 제공하는 인프라들을 중간에서 관리 및 중계해주는 CSB(Cloud Service Broker) 들이 성장하고 있다(공개 SW: OpenStack Cascading, SlapOS, Optimis, Bonfire, Aelous, DeltaCloud 등). 하지만 CSB 기술은 여전히 벤더들이 제공해주는 인프라/플랫폼에 종속되고 있는 성격을 가지고 있으며, 특정한 (AWS, Azure, GCP) 클라우드 인프라/플랫폼에 대한 종속성 (Lock-in) 극복이 계속적으로 필요한 상황이다.

o 따라서 오픈소스에 기반한 클라우드 인프라 서비스의 호환성 개선의 방향으로 특정 클라우드 인프라/플랫폼에 대한 종속성을 제거함과 동시에 다양한 SaaS 응용/서비스를 위한 융합형 Box 자원 인프라 중심으로 혁신적이고 동적으로 다양한 SaaS 응용들을 지원하는 기술이 필요하다.

o 이러한 문제점들을 극복하기 위해 본 기술문서에서 도입하고 있는 핵심 개념 중에 하나인 SaaS OverCloud란 SaaS 응용 개발자의 요구사항에 따라 SaaS UnderCloud가 제공하는 자원집합의 조각 (실제로는 가상/컨테이너/물리 머신)들을 부분적으로 선택하고 엮어 개발자에게 논리적인 클러스터 형태로 제공되는 오버레이 기반의 클라우드를 의미한다. 즉 SaaS OverCloud는 하나 또는 복수의 논리적인 클러스터 형태로 추상화되어 개발자가 필요한 SaaS 환경 호환성 제공을 위한 요소기술들을 기본적으로 내포하면서 제공된다. OverCloud를 이루는 요소는 Logical Cluster, DevTower, DevLake가 있으며, Logical Cluster는 SaaS OverCloud에서 SaaS를 태울 수 있는 자원의 집합조각들 (컨테이너, 가상 머신, 물리 머신)들을 나타내며, DevTower는 OverCloud와 개발자 간의 인터페이스 역할을 하는 동시에, 해당 OverCloud의 전반적인 관리를 담당하는 역할을 한다. 마지막으로 DevLake는 SaaS나 OverCloud 계층에서 모여지는 데이터들의 저장소이며, 이러한 데이터들은 적절한 시각화를 통해 개발자들에게 SaaS에 대한 전반적인 관리 및 상황을 파악하고 개선할 수 있는 기반을 마련해준다.

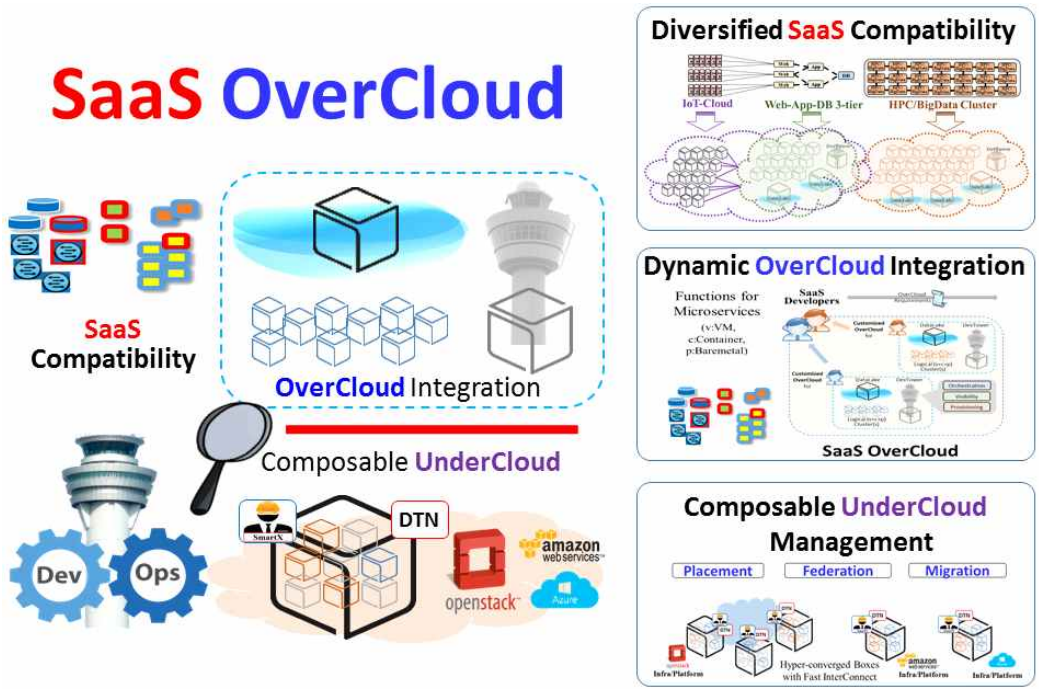

그림 1 SaaS OverCloud의 전체적인 개념도 및 방법론

o 그림 1과 같이 SaaS OverCloud 개념들을 실체화하기 위해서 우측에 보이는 세 가지의 이슈 사항들을 정리해나감과 동시에 DevOps (개발운영병행체제) 패러다임에 근거한 운영자 중심의 자동화된 실행환경을 통해 끊임없이 개발 및 운영하는 형태로 SaaS OverCloud를 구축 및 관리한다.

1.3. Diversified SaaS Compatibility를 위한 Multi-level Visibility

o SaaS OverCloud의 개념을 실체화하기 위해서는 Web-App-DB 3-tier, IoT-Cloud, HPC/BigData Cluster 등에 대응하는 다양한 SaaS 응용들에 대한 호환성 확보가 필요하다.

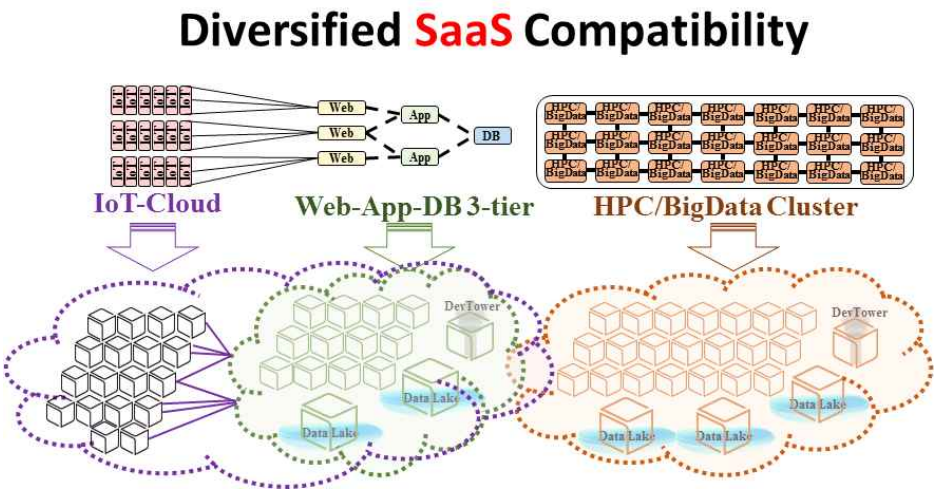

그림 2 세 가지 종류의 SaaS 응용에 대한 관계도

o 그림 2는 Web-App-DB 3-tier, IoT-Cloud, HPC-BigData Cluster와 같은 SaaS 응용들과 언더레이의 인프라와의 관계도를 나타내는 그림이다. 그림에서 왼쪽에서 오른쪽 순으로 End-Edge-Core로 본다면, IoT-Cloud는 각종 엔드 디바이스 장비로부터 데이터들이 모여서 Edge로 향하는 부분에 위치하고 있으며, Web-App-DB 3-tier는 Edge 부분과 Core로 들어가는 부분에 위치하고 있으며, HPC/BigData Cluster는 Core에 위치하고 있다. 이러한 End에서 Core까지 어우르는 인프라를 효율적으로 운영하고, 다양한 SaaS 응용들에 대한 호환성 지원을 위해서는 다계층을 지원할 수 있는 SaaS 응용 모니터링이 필요하다.

o 그림 3은 다양한 SaaS 호환성을 확보하기 위해 필요한 SaaS 응용 모니터링에 대한 개념도를 나타내고 있다. Agent 기반의 다계층 운영/응용 Visibility를 통해서 UnderCloud 계층의 인프라 뿐만 아니라 OverCloud 계층 및 SaaS 응용 까지를 어우를 수 있는 데이터들을 확보하고 이러한 데이터들을 시각화함으로써 다양한 범위에 대한 상황들을 파악하고 개선하는데 활용이 가능하다.

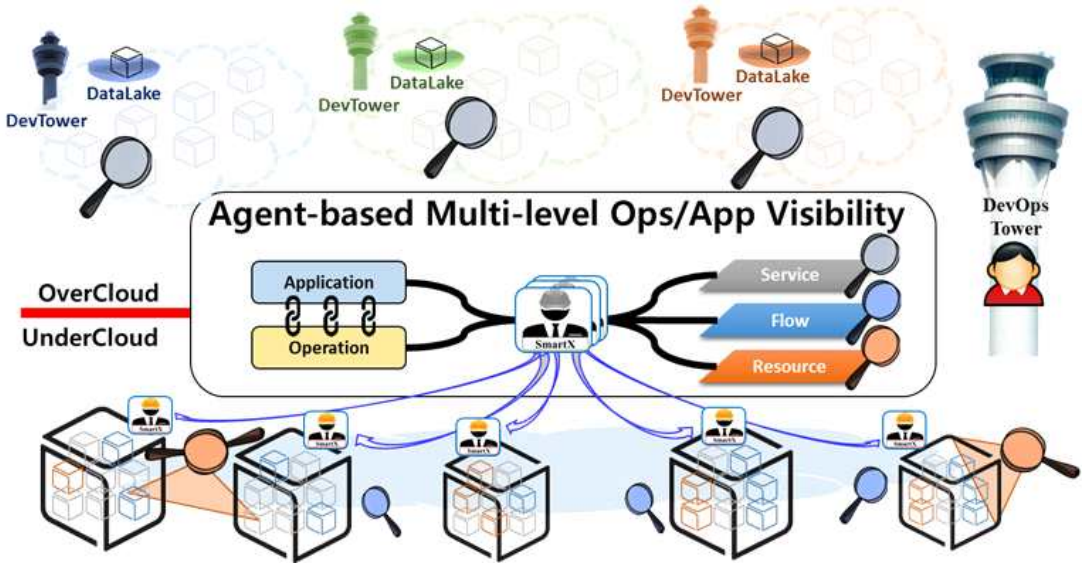

그림 3 Agent 기반의 다계층 운영/응용 Visibility 개념도

o Resource-Flow-Service에 이르는 각 계층별 데이터들과 사용 용도에 따른 Application 데이터, Operation 데이터 등을 분류하여 저장하고, 이렇게 저장된 데이터들은 적절한 처리를 거쳐 새로운 정보의 생성 및 운영자 및 개발자에 적절한 알람 등과 같은 여러 유용한 기능들을 사용가능하게 한다.

1.4. Composable UnderCloud Management

o SaaS OverCloud를 지원하기 위해서는 UnderCloud를 적절하게 관리할 수 있는 기술이 필요하다. Composable UnderCloud Management란 그림 4와 같이 Placement, Federation, Migration과 같은 기술들이 적절하게 지원되는 형태의 관리 기술이다.

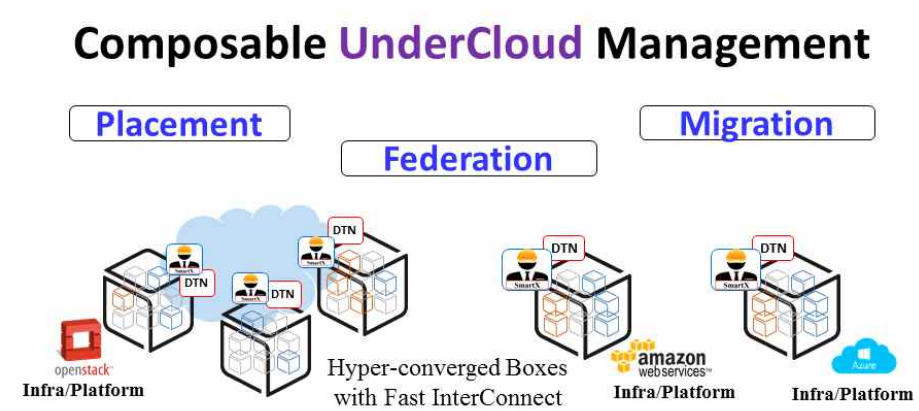

그림 4 Composable UnderCloud Management의 전체적인 구조

o UnderCloud란 SaaS 응용을 위해서 융합형(hyper-converged) 박스 중심의 실제 ICT 자원들로 구축/운영되는 하이브리드/멀티사이트 클라우드 인프라/플랫폼을 넘어서서 사용자 별로 자원집합을 제공하도록 구분된 자원집합의 조각(slice)들의 묶음을 의미한다.

o Placement란 이러한 융합형 박스 중심의 실제 ICT 자원들이 OverCloud를 위해 자원들을 할당할 때, 어떤 방식으로 할당 할지, 전반적인 인프라 상황을 파악하면서 효율적으로 OverCloud를 위한 자원들을 할당하기 위해서 필요한 기술이다. 예를 들면, 네트워킹 중심의 SaaS 응용일 경우, 각 인프라의 네트워킹 상황들을 파악하면서 SaaS 응용을 위한 OverCloud 자원들을 배치한다. 이러한 Placement 기술을 활용하면 각 SaaS 응용의 특징에 맞는 자원 집합들을 효율적으로 배치가 가능하다.

o Federation은 UnderCloud의 하이브리드/멀티사이트 클라우드 인프라/플랫폼을 통합 및 관리하기 위한 기술이다. 따라서 공용 클라우드 (AWS, Azure)나 사설 클라우드 (OpenStack)들이 마치 하나의 거대한 클라우드 인프라를 관리하는 것처럼 연동 및 관리되는 형태를 말한다. 이를 위해서 세부적으로 인증 연동이나, 자원 연동과 같이 연동의 범위를 체계화해서 적절하게 관리될 수 있는 기술들이 지원이 되어야 한다.

o 마지막으로 Migration이란 UnderCloud의 속하는 자원 집합들이 지역적으로 신속하게 이동할 수 있는 기술을 말한다. 동적인 SaaS 응용 호환성 지원을 위해서는 특정한 상황이 발생할 때, 파일 단위가 아니라 OverCloud가 전체적으로 이동할 수 있는 기술이 필요하다. 이 때 OverCloud 단위란 Logical Cluster, DevTower, DevLake라는 구성요소들이 동적으로 이동될 수가 있어야 하며, 이러한 Migration 기술을 소프트웨어 뿐만 아니라 하드웨어 적으로 지원하기 위해 물리 노드인 DTN(Data Transfer Node)을 활용하면서 UnderCloud 자원 집합의 Migration을 지원한다.

2. 단일 하이브리드 클라우드 인프라/플랫폼 상의 SaaS OverCloud 설계

2.1. SaaS OverCloud 구축 요구 사항

o 본 기술문서에서는 UnderCloud의 환경을 단일 하이브리드 클라우드 인프라/플랫폼으로 제한하면서 SaaS OverCloud를 구축하는 방법을 제안한다. 본 기술문서에서 사용되고 있는 UnderCloud의 실제적인 자원 집합은 클라우드 OS의 대표적인 오픈 소스인 OpenStack을 활용하여 UnderCloud 자원집합 (가상머신으로 한정)을 준비하였다.

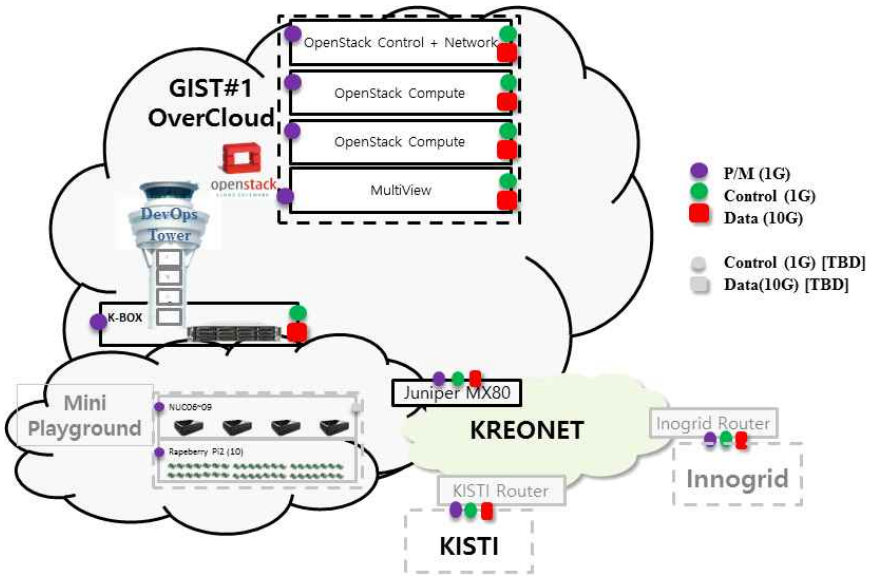

그림 5 실험에 활용된 물리 자원 배치도 (음영은 향후 사용)

o 그림 6은 OverCloud를 구축하기 위한 클라우드 레이어링 컨셉을 나타내고 있다. UnderCloud 층에는 위에서 언급한대로 OpenStack이 구축된 단일 인프라 상에서 가상머신을 제공해주는 자원집합이 있다. 그리고 이러한 UnderCloud를 통합적으로 관리해주는 DevOps Tower가 위치해있어야 하며, 이 Tower에서는 개발자/사용자 별로 자신들의 원하는 SaaS 응용을 수행하기 위한 인터페이스를 제공해주어야 한다. 이러한 개발자/사용자들의 SaaS 요구사항은 적절하게 문서화된 형태로 디스크립션 되어야하며, 또한 이렇게 문서화된 요구사항을 Tower에서 적절하게 해석하고 이를 위한 OverCloud를 생성해줄 수 있어야 한다. 또한 이러한 과정은 DevOps 기반의 동적 실행환경을 제공해줄 수 있어야 한다.

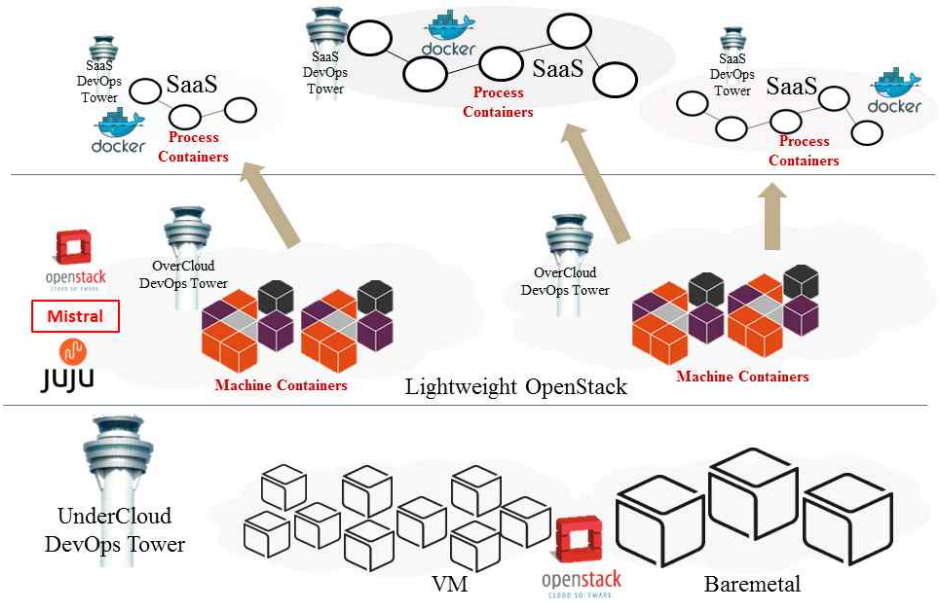

그림 6 Cloud Layering Concept for SaaS OverCloud

o 중간의 위치한 OverCloud 층에서는 각 개발자/사용자별로 요구사항을 만족하는 OverCloud들이 위치해 있으며 OverCloud 내에 존재하는 세 가지 요소 (Logical cluster, DevTower, DevLake)를 포함하여야 한다. 또한 이렇게 구성된 OverCloud는 UnderCloud의 인프라와 독립적이어야 하며, 가볍게 유연한 형태로 제공이 되어야 한다.

o 그리고 OverCloud 위에 SaaS 계층은 개발자/사용자가 처음에 요구했던 SaaS 들을 이루고 있어야하며, SaaS 응용을 이루기 위해 마이크로 서비스 아키텍처 형태를 도입해 SaaS 응용을 여러 개의 펑션들로 나누고 이러한 서비스들이 적절하게 체인(Chain)되는 형태로 SaaS 응용들이 구성되어야 한다.

o 마지막으로 위에서 언급한 과정들이 전체적으로 혹은 단계별로 자동화 형태로 제공되어야하며, 개발자/사용자와 구축되어진 OverCloud와 API 기반의 인터페이스가 제공되어야만, DevOps 패러다임에 근거한 자동화된 OverCloud 실행환경(SW 프레임워크)를 구현할 수 있게 된다.

2.2. Workflow 기반의 OverCloud 설계

o 2.1 절에서 언급했던 요구사항들을 만족하기 위해서는 개발자/사용자로부터 받은 요구사항들을 이해하고 이를 바탕으로 OverCloud를 생성하는 과정이 필요하다. 다음 그림 7은 개발자/사용자로부터 입력을 받고 그것들을 토대로 OverCloud를 만들고, 그 위에 3-tier SaaS 응용을 러닝하는 과정을 시나리오 형태로 보여주고 있다.

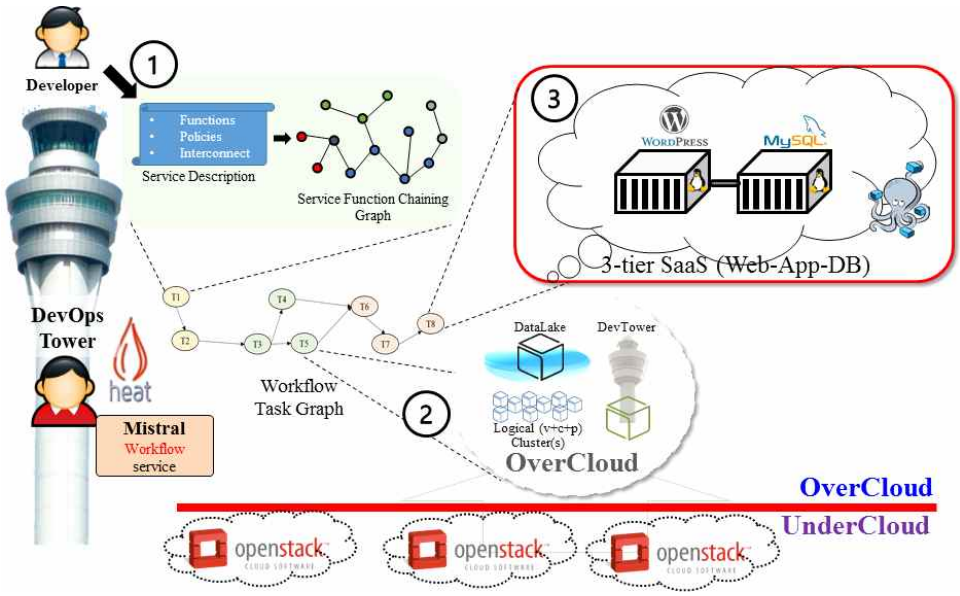

그림 7 Workflow 기반의 OverCloud 구축 및 SaaS 응용을 수행하기 위한 설계도

o 여기서 개발자/사용자의 요구사항부터 OverCloud를 구축 및 3-tier SaaS 응용을 실행하기까지의 일련의 Task들을 Workflow 형태로 관리하는 형태를 보여주고 있다. Workflow란 수행되어야 할 Task 들을 일련의 순서화된 과정으로 정리를 하는 개념을 말하고 있다. Workflow는 보통 그래프 형태로 나타내고 있으며, 그래프에서 노드는 Task를 의미하며, 노드와 노드사이를 연결하는 엣지는 순서에 관련된 Connection을 의미한다.

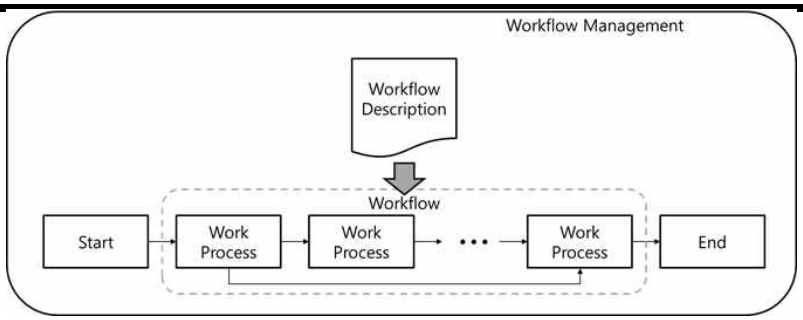

그림 8 Workflow의 관한 정의

o Workflow는 Task들의 수행 단계를 나타내는 State와 도달하려는 목적지인 Result가 정의될 수 있으며, 이를 통해 적당한 Result까지 도달하기 위한 여러 가지 Task들을 지날 수 있다. 이러한 Workflow 개념을 활용하면, 개발자/사용자로부터 입력받은 SaaS 응용의 요구사항들로부터 OverCloud를 생성하고 SaaS 응용을 수행하기까지의 과정들을 그래프 기반의 Workflow를 활용해 나타낸다면 훨씬 더 유연하게 일련의 과정들을 통제 및 관리할 수 있게 된다.

o 오픈스택에서는 이러한 Workflow를 관리할 수 있는 서비스로 Mistral [1]을 제공하고 있다. Mistral을 활용하면 Task 기반의 Workflow를 YAML 형태의 템플릿 파일로 작성 및 관리가 가능하고, 작성된 Workflow를 Workbook 형태로 관리를 하면서 수시로 관련된 Task들을 수행할 수 있도록 도와주는 역할을 한다. 오픈스택에서 제공하는 Heat와 달리 Task들을 병행해서 수행할 수 있는 능력이 있으며, 여러 분기점에서 다음 단계의 Task를 진행하기 위해서, 동기화 기능까지 제공을 해 오픈스택의 Heat 보다 좀 더 유연하게 오케스트레이션 할 수 있다.

o Mistral을 활용하면, UnderCloud 계층과의 독립성을 갖기 위해 가벼운 버전의 오픈스택을 설치하는 과정을 Task로 표현할 수 있게 된다. 여기서 가벼운 버전의 오픈스택을 설치하기 위해 기존의 Nova에서 KVM 하이퍼바이저를 사용하는 대신에, Nova-lxd-compute [2]를 활용해 Machine Container를 생성해주는 LXD 하이퍼바이저를 선택해서 오픈스택 구성을 수행한다.

o Machine Container란 기존의 Container 개념과 동일하게 가상 머신과는 다르게 기존의 커널 층을 호스트와 공유함으로써, 불필요한 오버헤드를 감소시키는 Container의 기존의 특징을 가져감과 동시에, 가상 머신과 같이 일종의 컨테이너도 똑같은 머신과 같은 기능을 수행하도록 만들어진 Container이다. 따라서 기존

의 Container의 철학인 “One process in one container“ 와 달리 Container 내부가 머신과 같이 여러 프로세스들이 동작이 가능하고 가상 머신과 같이 활용이 될 수 있다. 즉, Machine Container를 일종의 물리적 머신으로 생각할 수 있어서, 그 안에 Docker와 같은 Process Container를 담을 수 있고, KVM 하이퍼바이저를 설치해 가상 머신까지 담을 수 있게 된다.

o 따라서 정리를 하자면, OverCloud를 구성하기 위해 오픈스택의 Mistral 서비스를 활용해 Workflow를 구성하되, 이렇게 구성된 Workflow Task에는 오버클라우드를 구축하기 위해 다음과 같은 Task들을 정의를 명세 한다.

o 첫 번째로, LXD 기반의 가벼운 오픈스택 구성을 위한 자동 설치 부분들을 Task를 통해 명세를 한다. 이렇게 명세 된 Task가 수행이 된다면, 개발자/사용자들에게 OverCloud의 구성요소들을 줄 수 있는 일련의 준비 상태가 완료가 된다.

o 두 번째로, 첫 번째 Task에서 구축이 완료된 LXD 기반의 오픈스택에서 OverCloud의 구성요소인 Logical Cluster와 DevTower, DevLake를 구성한다. 이러한 구성요소를 위해 DevTower와 DevLake는 이미 만들어진 이미지를 이용해 구성을 하며 개발자/사용자의 요구사항에 맞게 Logical Cluster를 LXD를 활용해 Machine Container 형태로 제공해준다.

o 마지막으로 개발자/사용자들에게 주어진 OverCloud 상에서 3-tier SaaS 응용을 수행하는 Task를 명세 한다. SaaS 응용을 수행하기 위해서 제공된 Logical Cluster (Machine Container의 집합)에서 Docker Compose를 이용해 마이크로 서비스 아키텍처 방식과 같이 여러 function으로 세분화해서 process container에 담고 이들을 적절하게 체이닝 하는 방식으로 Task를 구성한다. 즉 Machine Conatiner에서 SaaS 응용을 수행하는 단계는 Docker Compose의 서비스 명세 방식을 채택해서 Docker Compose에서 이해할 수 있는 템플릿 형식인 YAML 형태로 명세를 해서 수행하게 된다.

2.3. SaaS OverCloud 구축 과정 검증

o 본 절에서는 2.2 절에서 설계했던 OverCloud의 구축 과정을 실제로 검증하는 부분들을 나타낸다. 2.1 절에서 언급했던 것과 같이 실험에서 쓰이는 물리 환경은 UnderCloud DevOps Tower 머신 1대, 오픈스택을 구성하고 있는 물리 머신 3대를 활용해서 수행하였다.

o UnderCloud에 오픈스택을 구성하고, Mistral을 구성하기 위해서 다음의 Github 레파지토리에서 스크립트를 이용해서 구성하였다.

$ git clone http://github.com/smartx-team/box-openstack-installation

o 오픈스택 Mistral에 대한 설치 공식 매뉴얼이 없으므로, 다음과 같이 소스코드를 내려 받아 설치를 수행한다.

$ git clone https://git.openstack.org/openstack/mistral.git

$ cd mistral

$ pip install -e .

o mistral의 configuration 파일을 생성하기 위해서는 다음과 같은 모듈이 필요하며 설치과정은 아래와 같다.

$ pip install —upgrade oslo.serialization

$ oslo-config-generator --config-file tools/config/config-generator.mistral.conf --output-file etc/mistral.conf

o 이후에 오픈스택의 다른 서비스를 설치하는 것과 마찬가지로 데이터베이스를 MySQL에 생성하고, 오픈스택 endpoint를 등록한다. 여기서 Mistral의 API 주소는 디폴트로 8989 포트를 사용한다.

o 등록이 완료가 되었다면, 데이터베이스에 Mistral의 관련된 DB 테이블을 등록시키는 명령어를 아래와 같이 수행한다.

$ mistral-db-manage --config-file etc/mistral.conf upgrade head

o 수행이 완료가 되었다면 다음과 같은 명령어로 Mistral 서비스를 수행한다.

$ mistral-server --config-file <path-to-config> --server engine,api,executor

그림 9 Mistral 서비스 수행장면

o 이제 Mistral 서비스가 사용 준비가 되었으므로, 준비된 템플릿을 수행한다.

$ mistral workflow-create <file_name>.yaml

$ mistral execution-create <workflow_name> <input_parameter>.json

o 다음은 LXD 기반의 오픈스택을 구성하기 위해서 LXD를 구성 및 세팅을 나타내고 있다. nova-compute-kvm와 nova-compute-lxd는 상호배제 (exclusive) 관계이기 때문에 kvm 하이퍼바이저가 설치가 되어있다면, 패키지가 삭제가 되고 lxd 하이퍼바이저가 설치된다.

o 오픈스택 릴리즈 버전인 Newton 버전부터는 공식적으로 Ubuntu 16.04 OS 매뉴얼을 채택하고 있으므로, LXD 설치 버전인 커널 4.0 이상을 만족한다. 따라서 LXD 기반의 오픈스택을 설치하는 머신은 Ubuntu 16.04를 OS로 선택하고 다음과 같은 명령어를 수행한다.

$ apt-get install nova-compute-lxd

o LXD를 사용하기 위해서는 일반적인 qcow2 이미지로는 불가능하다. 따라서 다음과 같이 raw 버전의 이미지를 받아서 glance의 등록을 시킨다.

$ wget http://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-

amd64-root.tar.gz

$ glance image-create --name lxd --disk-format raw --container-format bare --file xenial-server-cloudimg-amd64-root.tar.gz

$ service nova-compute restart

o 등록을 시킨 후, nova-compute 서비스를 재 수행 시키면, 하이퍼바이저로 LXD를 사용할 준비가 된다. 이후에 대시보드에서 인스턴스를 만들기를 수행한다면 다음과 같이 virsh list (KVM 가상머신 리스트)가 아니라 lxc list (Container 리스트)에 인스턴스들이 등록이 되는 것을 볼 수 있다.

그림 10 인스턴스 생성 후, Container list 화면

3. OverCloud 상의 Web-App-DB 3-tier 대응 SaaS 구축 및 검증

3.1. Web-App-DB 3-tier 적용을 위한 대상 서비스

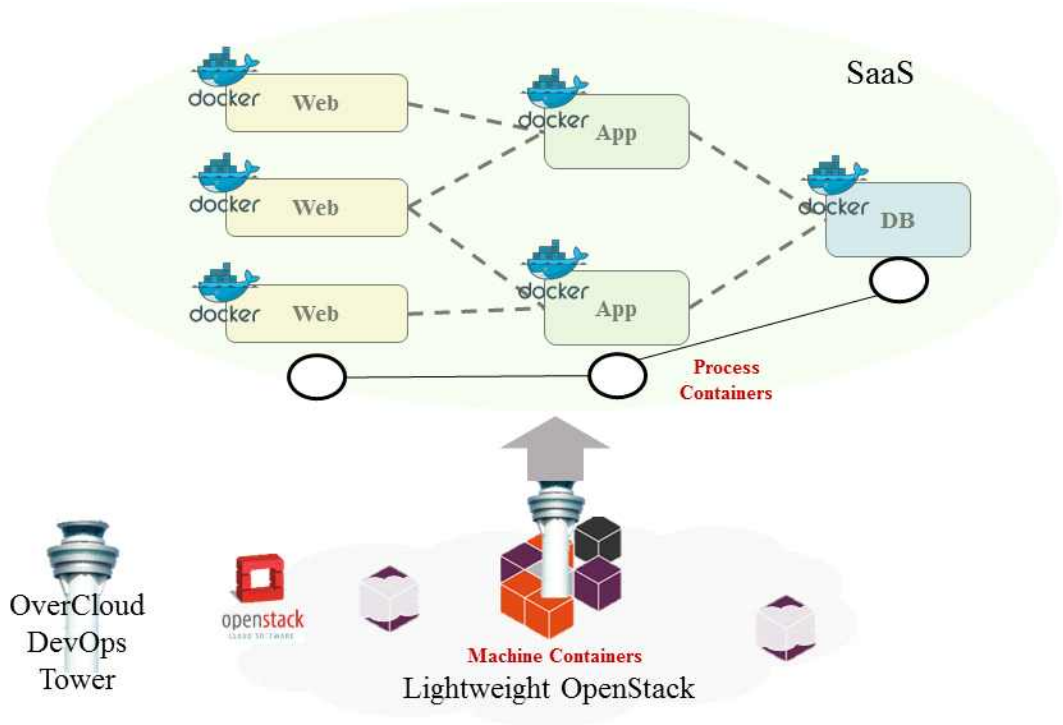

그림 11. Web-App-DB 3-tier 적용을 위한 대상 서비스의 설계도

o 개발자의 요구사항에 따라 생성된 SaaS는 OverCloud 상에서 동작하는데, SaaS의 구축 및 검증도 고민해야 할 문제이다. 따라서 구축 및 검증을 확인하기 위해 Web-App-DB 3-tier 대응의 SaaS를 생각했고, 이에 대한 검증 작업을 수행한다. 언급된 목표를 위한 설계 그림은 위의 그림 11에 나와 있다.

o SaaS 구축 및 검증을 위해 Web-App-DB 3-tier를 적용하는데 이를 위해 Web, App, DB 세 가지의 기능들이 사용될 수 있다. Web-App-DB 3-tier를 위해 오픈 소스를 활용할 수 있는데, Web과 App의 기능으로는 Nginx와 Wordpress를 활용하고, DB로는 MySQL을 활용한다. Nginx [3]는 웹 서버 소프트웨어로, 대표적인 웹 서버인 Apache보다 가볍고 빠른 것으로 유명한 애플리케이션이다. 웹 서버, 리버스 프록시 및 로드 밸런서 등의 기능을 가진다. Wordpress [4]는 웹 페이지 제작 및 관리가 주 목적인 애플리케이션으로, 웹 표준을 따르기에 운영체제나 웹 브라우저의 제약이 없으며 사용이 편리하다. DB의 기능을 수행할 MySQL [5]은 구조화 질의 언어(SQL: Structured Query Language)를 사용하는 관계형 데이터

베이스 관리 시스템(RDBMS)로 리눅스 운영 체제를 지원하면서 사용하기가 쉽다.

o Web-App-DB 3-tier 적용에는 Nginx와 Wordpress, 그리고 MySQL을 체이닝을 통해 엮은 후, 이에 대응하는 SaaS 구축을 수행한다. 해당 기능들은 도커 [6] 컨테이너를 기반으로 생성할 것이고, 컨테이너 생성 시에는 도커 컴포즈라고 하는 컨테이너 자동화 도구를 사용할 것이다. 언급된 도구들을 활용해 생성된 SaaS의 검증은 SaaS 내부의 모듈로 존재하는 웹 서버의 구동을 확인하면 된다.

3.2. Docker compose를 활용한 컨테이너 기반의 3-tier SaaS 설계

o 상기 절에서 제시한 디자인에 따라 언급된 각 역할에 맞는 오픈소스 도구들을 활용해 이를 엮는 체이닝 작업을 수행한 후, SaaS 설계를 수행했다.

o 각 역할을 수행할 오픈소스들은 모두 도커 컨테이너를 통해 기능화된다. 도커 컨테이너는 컨테이너 기술의 하나로 마이크로서비스를 패키징하여 배포하고 테스트할 수 있다. 호스트 운영 체제의 커널을 활용하면서도, cgroup, namespace와 같은 기술을 활용하여 서비스를 위한 격리된 가상 환경을 제공하므로, 가상 머신보다 빠른 속도로 배포 및 테스트 진행이 가능하다. 또한 개발자가 필요한 SaaS 환경 호환성을 제공해야 하므로 Web-App-DB 3-tier 대응 SaaS 구축 시 배포가 용이한 도커 컨테이너를 활용한다.

o Web-App-DB 3-tier 대응 SaaS 구축을 위해 사용할 오픈소스 도구들은 도커 컨테이너를 활용해 패키징하고 배포한다. 도커 컨테이너는 가상 머신과 가상 머신 위에서 작동할 운영 체제의 경우와 비슷하게 이미지를 활용한다. 공개된 이미지 저장 공간인 Docker Hub 또는 개인 이미지 저장 공간인 Docker Private Registry에서 이미지로 패키징된 마이크로서비스들을 다운로드한 후, 다운 받은 이미지를 활용해 구축하면 된다. 일련의 과정에는 배포의 과정도 포함된다. Web-App-DB 3-tier 구축을 위해 언급된 기능들은 이미 Docker Hub에 공개되어 있다.

o Web-App-DB 3-tier 구축을 위해서는 Wordpress, nginx, MySQL도커 이미지가 필요하다. 따라서 최소 세 개의 도커 이미지를 기반으로 기능들을 배포해야 하는데, 이들 각각에 대한 작업은 반복적이다. 또한 각각의 기능에 대한 컨테이너를

생성한 후, 이들의 관계를 정리해 연결해주는 이른바 체이닝의 작업이 필요하다. 왜냐하면 Wordpress는 콘텐츠관리시스템(CMS)으로 웹에서 콘텐츠를 만들 수 있는데, Wordpress를 통해 생성된 콘텐츠를 저장하기 위해서는 데이터베이스를 활용해야 하고, 이는 MySQL이 DB 역할을 하면서 콘텐츠에 대한 로드 밸런싱 및 리버스 프록시 등에 대한 부분은 Nginx가 담당을 하고, 이는 세 가지의 다른 기능이 서로 엮여 유기적으로 동작해야 하기 때문이다. 따라서 각 기능의 생성 후에도 이들을 엮는 체이닝이 필요한데, 이것은 Docker Compose [7]를 통해 구현할 수 있다.

o 도커 컴포즈는 도커의 컨테이너 자동화 도구 중 하나로, YAML과 작동해 애플리케이션 설명을 작성하고 애플리케이션에 사용되는 다양한 컨테이너가 어떻게 서로 연결되는지를 보여준다. 도커 컴포즈를 사용할 경우, docker-compose.yml이란 파일을 활용하는데 이는 애플리케이션을 설명하기 위한 파일이고, 이를 접근할 수 있게 해주는 것이 YAML이다. 따라서 docker-compose.yml 파일에 각 기능 구축에 대한 설명을 작성한다면 단순한 명령어 한 줄만으로도 해당 기능을 실행할 수 있다. Docker-compose.yml의 예로는 그림 12에 제시한다.

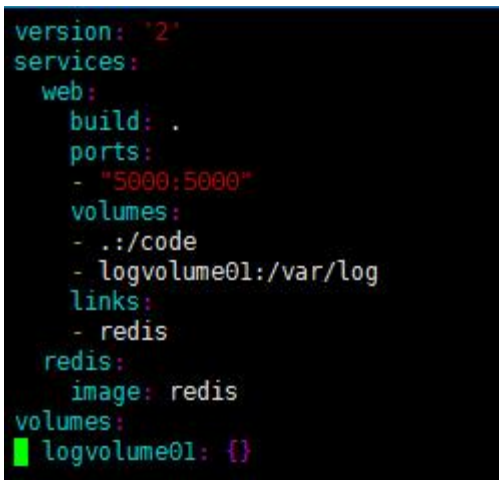

그림 12. docker-compose.yml 예시

3.3. OverCloud 상의 3-tier SaaS 동작 검증

o 생성된 LXD 내부의 OverCloud에는 SaaS Cloud가 생성된다. 이후 생성된 SaaS Cloud에서 해당하는 3-tier SaaS가 동작한다. 이를 나타내는 그림은 아래 그림 13 이다.

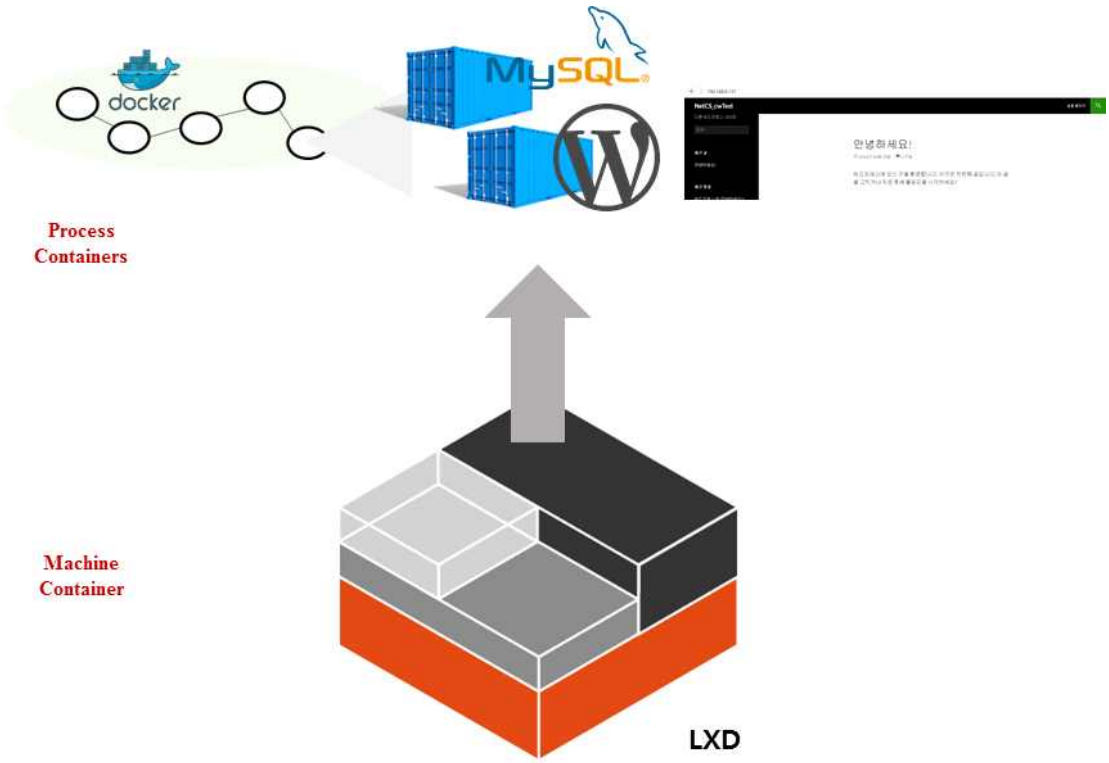

그림 13. OverCloud 상의 3-tier SaaS 동작 검증 전체

o 융합형 Box 내부에 생성된 OverCloud에 LXD 컨테이너가 생성된다. 생성된 LXD Container 위에 개발자가 요구한 SaaS Cloud가 만들어지고 SaaS Cloud 내부에서 서비스가 동작한다. 여기서 지칭할 서비스는 Web-App-DB 3-tier이다. 따라서 생성된 OverCloud 안의 Machine Container 내부에 도커 기반의 Container를 사용해 서비스를 구성하는 내부 모듈들을 생성한다. 이는 그림 9에Process Container 부분이다. Docker 기반 컨테이너는 각각의 모듈을 담고 있는데, 여기서 사용하는 것은 오픈 소스로 App 모듈로 Wordpress와 DB 모듈로 MySQL이다. 웹 서버의 구동만을 확인하기 때문에, 안정적 서비스 제공을 위한 로드 밸런서 및 리버스 프록시의 기능을 가지는 Nginx가 반드시 필요한 것은 아니다.

o OverCloud 내부에 생성된 Machine Container 내부로 3-tier SaaS가 들어가고 이의 동작을 검증해야 한다. 검증을 위해 우선 컨테이너의 구동을 확인하고, 이후

Web-App-DB 3-tier SaaS의 기능을 검증해 동작을 확인해야 한다. 기능에 대한 검증으로는 브라우저 접속 후 직접적인 사용을 통해 검증을 할 수 있다.

o OverCloud 내부의 3-tier SaaS를 생성을 하는데, 여기서는 도커 기반의 컨테이너를 사용하기로 했으므로, LXD 컨테이너 내부에 도커 컨테이너를 생성해야 한다. 이를 위해 우선 도커 컨테이너로 이루어진 서비스의 모듈이 돌아갈 수 있는 환경을 생성해야 한다. 따라서 LXD 컨테이너와 LXD 컨테이너 위에 도커 컨테이너가 있어야 한다. LXD 컨테이너에서 도커 컨테이너를 생성하는 방법은 다음의 명령어를 실행하면 된다 [6].

$ lxc launch ubuntu:14.04 docker -p default -p <container name>

여기서 사용한 ubuntu:14.04는 우분투 14.04에 대한 이미지를 나타내는데, 만일 해당 이미지가 머신 컨테이너 내부에 없는 경우에는 해당 이미지를 다운로드받아 명령어를 수행한다. 따라서 머신 컨테이너인 LXD 컨테이너 위에 도커 컨테이너가 생성되고, 생성된 컨테이너를 확인하기 위해서는 다음의 명령어를 실행한다. 이는 그림 14에 나와 있다.

$ lxc list

```
root@ubuntu:/home/chorwon# lxc list
+--------+---------+--------------------------------+------+------------+-----------+
|  NAME  |  STATE  |              IPV4              | IPV6 |    TYPE    | SNAPSHOTS |
+--------+---------+--------------------------------+------+------------+-----------+
| docker | RUNNING | 192.168.0.140 (eth0)           |      | PERSISTENT | 0         |
|        |         | 172.17.0.1 (docker0)           |      |            |           |
+--------+---------+--------------------------------+------+------------+-----------+
| xen1   | RUNNING | 192.168.0.139 (eth0)           |      | PERSISTENT | 0         |
+--------+---------+--------------------------------+------+------------+-----------+
```

그림 14. 머신 컨테이너(LXD 컨테이너) 위에 생성된 도커 컨테이너

이후, 생성된 도커 컨테이너 내부에는 Web-App-DB 3-tier 구축에 필요한 각 기능들의 컨테이너를 만들어야 한다. 여기서는 Wordpress와 MySQL을 활용하는데, 해당 기능들의 설명 및 기능들을 엮는 체이닝까지의 작업을 수행하는데 도커 컴포즈를 활용한다. Docker-compose.yml 파일을 활용해 사용할 기능들과 이들의 체이닝을 작성한 후, 작성한 docker-compose.yml 파일을 다음의 명령어를 입력해 실행한다.

$ docker-compose up -d

Web-App-DB 3-tier 구축을 위해 작성한 docker-compose.yml 파일의 내용은 그림 15에 제시했다.

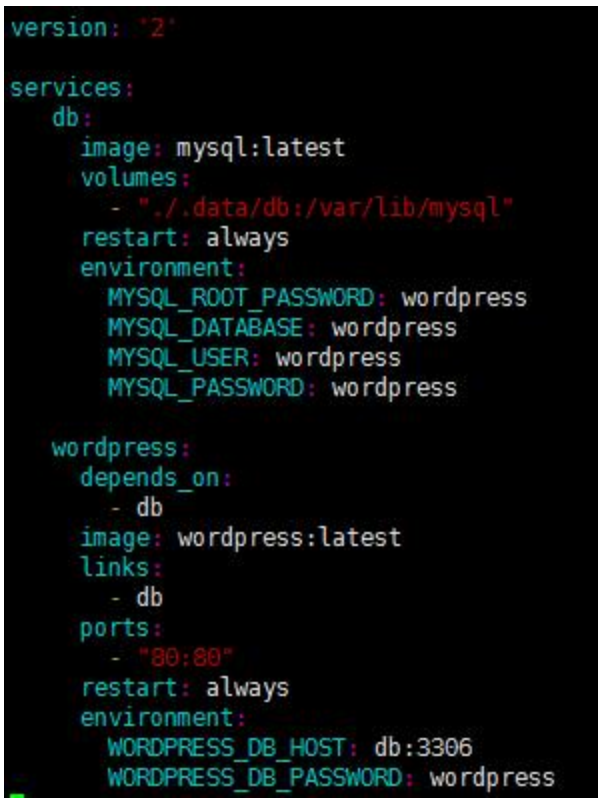

그림 15. Web-App-DB 3-tier 구축을 위한 docker-compose.yml

o docker-compose.yml 파일과 docker-compose up -d 명령어를 사용하면 Web-App-DB 3-tier 구축을 위해 필요한 wordpress와 MySQL 기능들에 대한 도커 컨테이너가 생성됨과 동시에 실행된다. 웹 페이지 제작 및 관리의 기능인 Wordpress와 웹 페이지에서 만들어진 콘텐츠 저장을 위한 MySQL 데이터베이스가 실행되므로, 웹 브라우저를 활용해 3-tier 대응 SaaS 구축 및 동작 검증이 가능해진다. 따라서 웹 브라우저를 통해 다음의 주소를 입력한 후 그림 10과 같은 결과가 나온다면 Web-App-DB 3-tier 대응 SaaS 구축에 성공했음을 알 수 있다. 그림 10은 wordpress 처음 생성시 나오는 wordpress 초기 설정에 대한 내용이고, 이에 맞춰 설정을 하면 웹 페이지를 생성할 수 있다.

http://<container_ip_address>:80

그림 16. 웹 브라우저를 통해 3-tier 대응 SaaS 구축 확인 결과

References

[1] Mistral,
http://docs.openstack.org/developer/mistral/guides/installation_guide.html
[2] Introduction to nova-compute-lxd,
https://insights.ubuntu.com/2015/05/06/introduction-to-nova-compute-lxd/
[3] Nginx, https://www.nginx.com/
[4] Wordpress, https://ko.wordpress.com/
[5] MySQL, https://www.mysql.com/
[6] Docker, https://www.docker.com/
[7] Docker Compose, https://docs.docker.com/compose/
[8] LXD, https://linuxcontainers.org/lxd/

SaaS OverCloud 기술 문서

- 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: 2016/11