SaaS OverCloud 기술문서 #10

IoT-Cloud SaaS 응용의 호환성 지원을 위한 워크플로우 기반의 스마트 에너지 서비스 배포 및 운영 방안

Document No. SaaS OverCloud #10
Version 1.0
Date 2017-12-08
Author(s) GIST#1 Team

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2016. 11. 01	김승룡, 한정수	
0.2	2017. 11. 28	김승룡	
0.9	2017. 12. 08	김승룡	

본 문서는 2017년도 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

#10 IoT-Cloud SaaS 응용의 호환성 지원을 위한 워크플로우 기반의 스마트 에너지 서비스 배포 및 운영 방안

그림 목차

SaaS OverCloud 기술문서 #10.
IoT-Cloud SaaS 응용의 호환성 지원을 위한 워크플로우 기반의 스마트 에너지 서비스 배포 및 운영 방안

1. IoT-Cloud SaaS 응용의 호환성 지원을 위한 워크플로우 개요

1.1. 연구의 개요

o 본 문서에서는 워크플로우를 통한 SaaS 응용 호환성 지원 관점에서 컨테이너 기반의 IoT-Cloud SaaS 응용을 대상으로. 서비스 배포와 운영을 위한 워크플로우를 설계하고 다양한 오픈소스 도구들을 활용하여 이를 검증하는 과정을 다룬다. 이를 위한 예제 IoT-Cloud SaaS 응용으로서 소규모 데이터 센터를 위한 스마트 에너지 IoT-Cloud를 활용하며, 해당 SaaS 응용에 대해 설계된 워크플로우를 적용함으로써 서비스의 배포 및 실행 측면에서의 자동화 및 호환성 지원 가능성을 검증한다.

1.2. 연구의 배경

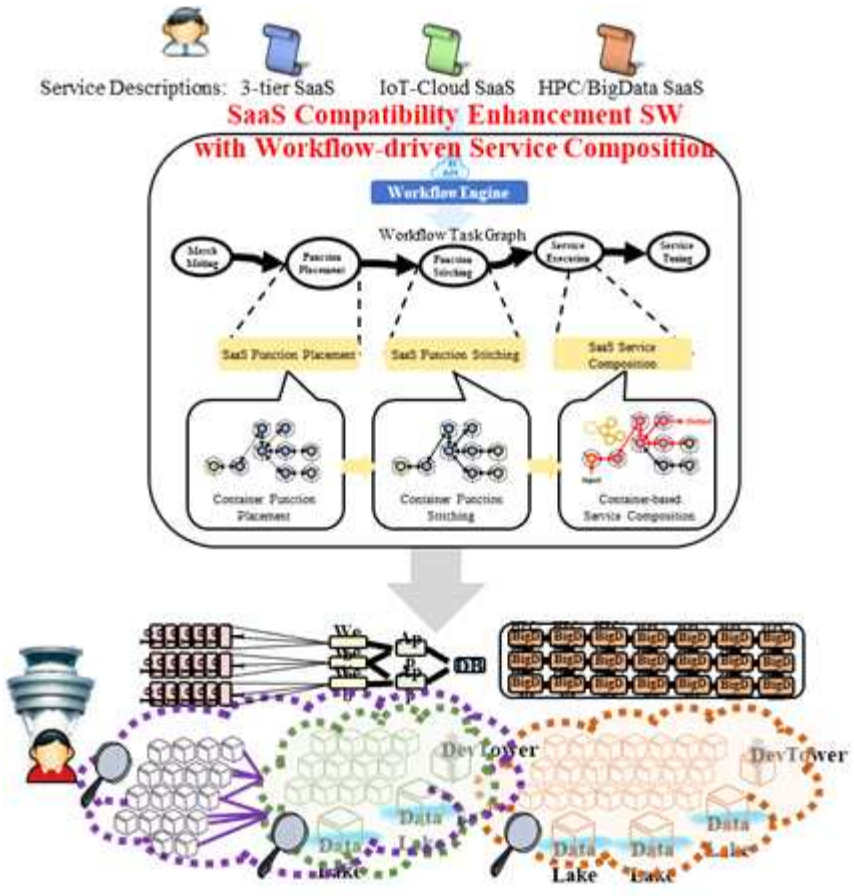

그림 1 워크플로우 기반의 서비스 합성을 바탕으로 하는
다양한 SaaS 응용에 대한 호환성 지원 SW 개념도

o 워크플로우는 일반적으로 원본 데이터로부터 최종 결과를 도출하기 위해 필요한 정형화된 절차를 의미한다. 반복적으로 수행되는 설계, 실행, 모니터링, 재실행 등의 과정들을 최소한의 노력으로 수행할 수 있도록 지원하는 것을 목표로 하며, 이를 위해 Kepler를 비롯한 다양한 워크플로우 엔진 등이 실행 및 런타임 상호작용, 로컬 및 원격 데이터 액세스, 로컬 및 원격 서비스 호출 등의 기능을 바탕으로 사용자가 효과적으로 워크플로우를 다룰 수 있도록 지원한다.

o 본 문서에서 다루고자하는 IoT-Cloud SaaS 응용에 대한 서비스 워크플로우는 서비스와 이와 관련된 자원과의 관계를 고려하여 서비스를 준비하고 수행하며 운영하는 전반적인 과정을 다룬다. 워크플로우가 일련의 작업들의 내용과 그 순서 및 주체에 대한 명세를 바탕으로 하는 흐름을 의미한다는 점에서 바라보면, 서비스와 관련된 각 요소들을 대상으로 관계와 순서를 정의하고 구체적인 요구사항을 바탕으로 그 흐름을 정의한다는 측면에서 다른 의미를 갖는다. 이 때 해당하는 작업들은 기계 뿐 아니라 사람에 의한 내용도 포함한다. 이러한 워크플로우가 사전에 적절히 준비된다면 서비스 운영 측면에서 자동화를 기대할 수 있을 것이다.

o 사물 인터넷(IoT, Internet of Things)이란 전통적인 사물들을 네트워크에 연결함에 따라 모든 사물들이 인터넷을 통해 하나로 연결하고 다루는 기술 및 패러다임을 뜻한다. 이를 통해 다양한 사물로부터 수많은 종류의 데이터를 수집하고 이를 활용하여 여러 산업에서 유용하게 활용할 수 있다. 이러한 사물 인터넷 기기들은 상대적으로 열악한 자원을 보유하고 있는 까닭에 이를 고려한 경량화 된 솔루션이 필요하다. 때문에 컴퓨팅 자원 관점에서는 기존의 가상 머신 위주의 가상화 기술보다는 보다 컨테이너 기술이 각광받고 있는 추세이다. 커널 영역을 공유함으로써 자원을 보다 효율적으로 활용할 수 있는 까닭에 사물 인터넷 기기에 적합하게 활용 가능하다. 이러한 컨테이너 기술 중 도커(Docker)는 도커 허브(Docker Hub) 및 다양한 컨테이너 관리 기능을 포함하는 생태계 구축을 바탕으로 가장 큰 세력을 얻고 있는 컨테이너 기술 중 하나이다.

o 사물 인터넷 기기들로부터 수집한 다양한 정보들은 각 사물들 간의 정보 교환을 통한 처리 뿐 아니라, 전체 데이터를 바탕으로 학습 기반의 예측 및 분석을 통해 유용하게 활용 가능하다. 이를 위해 상대적으로 열악한 사물 인터넷 기기 뿐 아니라 클라우드 기반의 유연하고 동적인 컴퓨팅 자원의 지원이 필수적이다. 이러한 관점에서 IoT-Cloud 라는 개념으로 대표되는 도메인간의 결합이 나타나고 있는 추세이며 Amazon AWS, Microsoft Azure 등 대표적인 클라우드 서비스 제공업체들도 사물 인터넷에 대응하는 부가 서비스를 잇달아 출시하며 이러한 흐름을 반증하고 있다.

1.3. 연구의 필요성

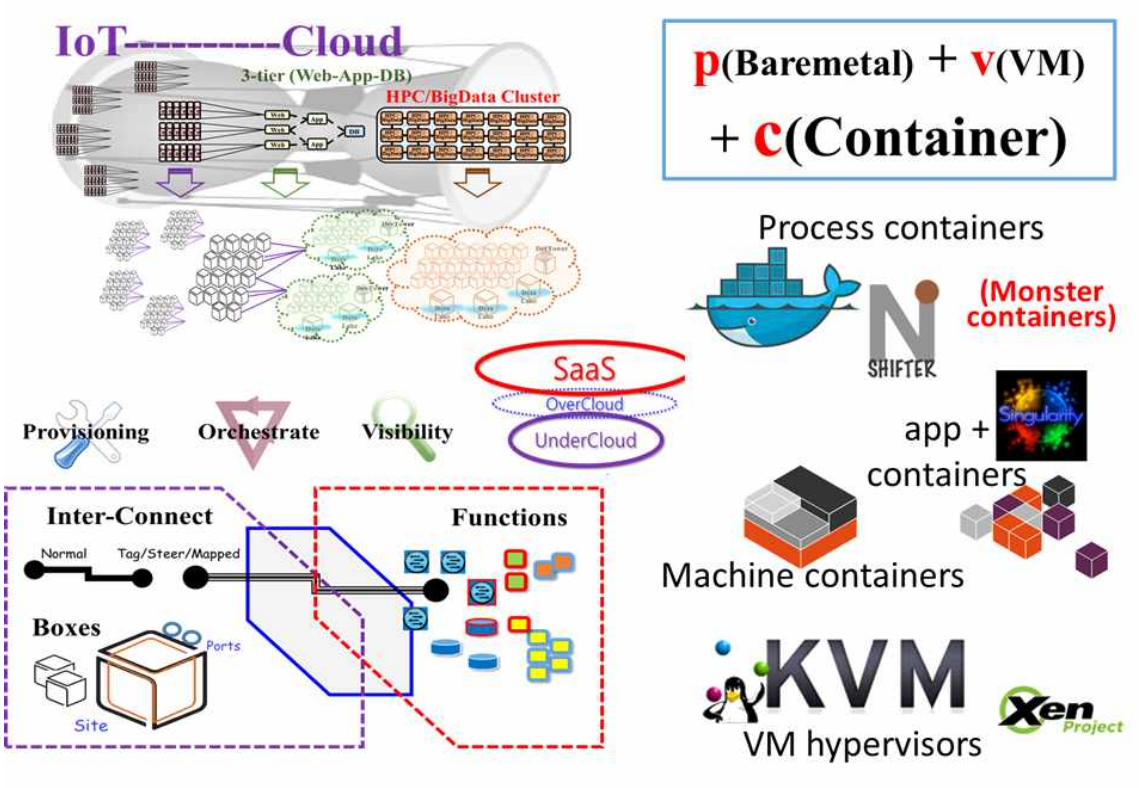

그림 2 IoT-Cloud 패러다임에 따른 다양한 마이크로 서비스 기반 SaaS 응용의 대두

o 현재는 기존과 달리 사물 인터넷과 같은 새로운 패러다임이 대두되는 시기이며, 이에 함께 분산 컴퓨팅을 벗어나 클라우드로 확장된 컴퓨팅 기술 또한 결합되고 있는 추세이다. 이에 따라 IoT-Cloud 기반의 서비스에 대한 이해가 필수적이며, 그 과정에서 컨테이너 기반의 가상화 기술의 활용이 중요한 시점이다. 뿐만 아니라 다양한 환경에 해당 서비스들을 활용하기 위해서는, 마이크로서비스 기반의 SaaS 응용들의 호환성을 향상시킴으로써 다양한 환경에서 반복되는 동일 작업으로 낭비되는 시간과 노력의 절감이 필요하다. 이를 통해 해당 프로세스를 워크플로우를 바탕으로 정의하고 이를 바탕으로 보다 동적이고 유연한 서비스 운영으로의 확장을 모색해야 한다.

2. 컨테이너 기반 IoT-Cloud SaaS 응용을 위한 서비스 워크플로우

2.1. 워크플로우 개요

o 본 문서는 Web-App-DB로 구성되는 3-tier SaaS 응용에 대한 경험을 바탕으로 컨테이너 기반의 IoT-Cloud SaaS 응용에 대한 호환성 확보를 위한 워크플로우를 고려한다. 그 중에서도 서비스의 운영 자동화 측면에서 필수적인 서비스의 배포와 합성과 관련된 내용을 중점적으로 다룬다. 해당 서비스는 동적으로 생성 및 소멸이 가능한 OverCloud 환경 위에서 운용되며 이러한 환경은 OverCloud를 구성하는 Dev Tower, DataLake 등의 개체를 포함하며 UnderCloud와의 관계를 매개하는 유연한 클라우드 환경임을 전제로 한다.

2.2. IoT-Cloud SaaS 응용 지원을 위한 요구사항

o 각 SaaS 응용을 구성하는 요소 기능들은 모두 도커 컨테이너를 통해 구성되어야 한다. 해당 서비스는 이러한 컨테이너들로 구성된 마이크로서비스 형태에 부합해야 하며, 이를 바탕으로 쉽고 효과적인 패키징을 활용하여 배포 및 테스트 진행이 가능하다.

o 해당 요소 기능들에 대한 도커 이미지에 대한 사전 빌드 작업이 필요하다. 사물인터넷 기기의 특성상 각 기기 내에서 빌드 작업을 수행할 경우 많은 시간이 할애되므로 원활한 서비스 운용에 차질이 발생한다. 이 때문에 각 이미지 파일의 사전 확보가 필수적이며, 이들 간의 관계 정리를 위한 명세서 수준의 체이닝이 선행되어야 한다. 이를 바탕으로 워크플로우는 각 기능들을 활용한 서비스 합성을 진행할 수 있다.

o IoT-Cloud SaaS 응용을 지원하기 위하여 자원 및 컨테이너 간의 관계를 고려한 오케스트레이션을 지원할 수 있는 별도의 워크플로우 엔진이 필요하다. 자동화 측면에서의 효율성을 고려하였기에 각 자원에 직접적으로 접근 가능하면서도 원하는 워크플로우를 정의할 수 있으며 이에 대한 시각화 지원이 가능한 도구가 필요하다.

2.3. 컨테이너 기반 IoT-Cloud SaaS 응용을 위한 워크플로우 설계

o 해당 서비스 워크플로우를 활용하는 컨테이너 기반의 자동화된 서비스 합성을 효율적이고 견실하게 수행하려면, 요소 기능들에 대한 매칭(matching), 배포(placement), 접합(stitching) 및 서비스 튜닝(service tuning) 등과 같은 서비스 합성 절차에 대한 세부적 고려가 필요하다.

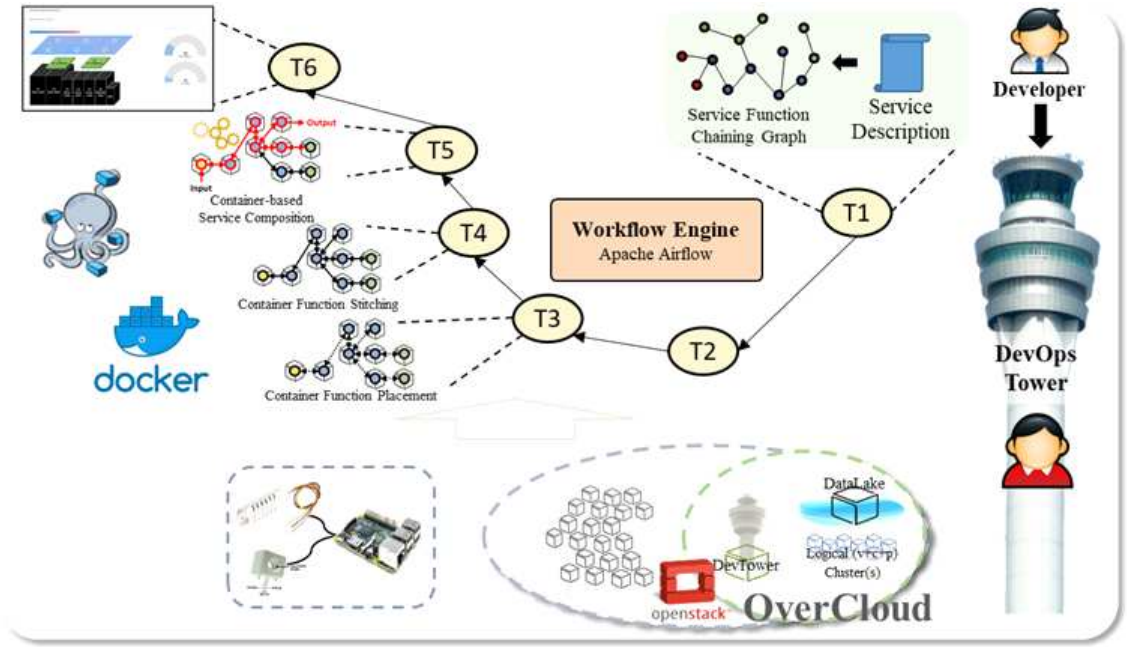

그림 3 IoT-Cloud SaaS 응용에 대한 서비스 워크플로우 개요

o 기능 매칭 절차에서는 목적하는 서비스를 운용하는 대상 환경의 특성을 파악하여 최적으로 부합하는 컨테이너 기반 요소 기능들을 매칭한다. 즉 적절한 컨테이너에 대한 선택을 확보가 가능한 여유 자원의 유형/규모에 맞추고, 컨테이너 설치/설정 유연성이나 엮어질 인근 컨테이너들과의 접합성 등을 확인한다. 이를 통해서 자원 집합의 효율적인 활용을 제고하고 향후 서비스 운용상의 문제 발생을 줄이도록 고려한다.

o 위와 같이 요소 기능 별로 적절한 컨테이너가 선택되면 이를 해당된 자원에 투입하는 기능 배포가 필요하다. 컨테이너 이미지와 이에 부수된 사전 설정들을 함께 배포해야 반복된 컨테이너 배포를 자동화할 수 있다. 더불어 기능 배포 절차에서는 투입을 지원하는 네트워킹의 유연성과 지속성에 따라 기능 배포의 실패나 지연 등의 문제들이 발생하므로 이에 대한 원인 파악을 순서에 따라 진행하도록 지원해야 한다.

o 또한 기능 접합 절차에서는 배포된 기능들을 서비스 명세에 따라 연결함으로써 서비스 합성을 마무리한다. 상호 연결된 기능들이 서로 이어져서 데이터를 주고 받으면서 합성을 통한 서비스 운용이 시작하는 것이다. 즉 SFC(service function chaining)을 통한 유연한 기능 접합이 가능하다면, 컨테이너 기반 서비스 합성을 위한 기능 접합이 변동에 대응하도록 지원할 수 있다.

o 마지막으로 서비스 튜닝 절차에서는 대상 환경에서 수집되는 센서 정보, 사용자 입력 등에 따라 유동적으로 변화하는 IoT-Cloud서비스의 특징과 대상 서비스 합성에 대한 워크로드 수준의 가시성(visibility)를 제공해야 한다. 이를 통해 상황에 따른 동적이면서 자동화가 가능한 서비스의 튜닝을 지원하고, 이를 통해 지속적인 서비스의 품질 개선을 지원하도록 고려한다.

3. 소규모 데이터 센터를 위한 스마트 에너지 IoT-Cloud 서비스

3.1. 서비스 개요

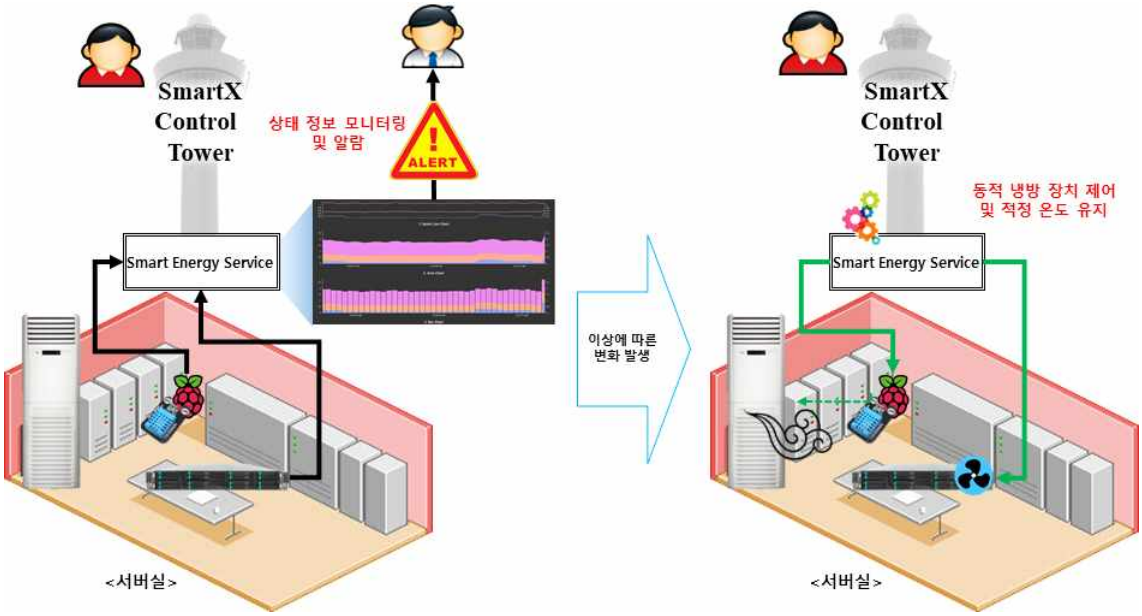

그림 4 스마트 에너지 IoT-Cloud 서비스 시나리오

o 스마트 에너지 IoT-Cloud 서비스는 데이터 센터 내부의 상태 파악을 위하여 오픈 소스 소프트웨어 기반의 데이터 센터 관측 및 제어를 제공하는 것을 목표로 한다. 이를 위한 정보의 전달은 빠르고 신뢰할 수 있어야만 한다. 뿐만 아니라 단순히 수집하고 알려주는 수준에서 그치지 않고 그 결과에 따른 효과적인 온도 제어 및 에너지 관리가 이뤄져야 한다. 실내 공기의 온도가 상승했다면 이를 낮춰주기 위하여 냉각 장치의 온도를 보다 낮게 설정하고, 충분히 온도가 낮은 상태라면 냉각 장비의 상태를 유지시켜주거나 높여줄 필요가 있다. 더불어 서비스의 적용 및 운용에 필요한 비용은 본래 취지에 따라 저렴해야한다.

o 이러한 요구사항을 반영으로 한 스마트 에너지 IoT-Cloud 서비스의 실증 시나리오는 그림 4와 같다. 서버실 내에 센서를 포함한 IoT 장비를 활용하여 각 박스별 상태와 온도 정보를 수집한다. 수집된 정보는 신속하고 안정적으로 내부의 사설 클라우드로 전달된다. 전달된 정보를 바탕으로 사용자가 현재 서버실 내부의 상황을 파악할 수 있게끔 시각화된 인터페이스를 제공하며, 냉방 장치 제어를 위한 명령을 전달하여 적절한 내부 온도를 유지할 수 있도록 한다. 이를 통해 효율적인 에너지 사용 뿐만 아니라 효과적인 냉방 효과를 유지하는 것을 목표로 한다.

3.2. 서비스 상세

o 서비스가 어떠한 환경에서도 잘 동작할 수 있도록 컨테이너 기반으로 서비스의 기능들을 잘 분리를 하고, 종속성이 없는 상태에서 서로간의 연결을 통해 하나의 서비스가 이룰 수 있도록 서비스도 디자인을 하였다. 그림과 같이 많은 종류의 기능들이 구성이 되며, 이러한 각 기능들은 도커 컨테이너 형태로 올라가도록 구성하였다. 서비스를 이루기 위해서 주로 쓰는 도구들은 오픈소스들을 활용하였으며, Apache Kafka, Apache Spark, Node js, Apache Flume 등이 사용되었다.

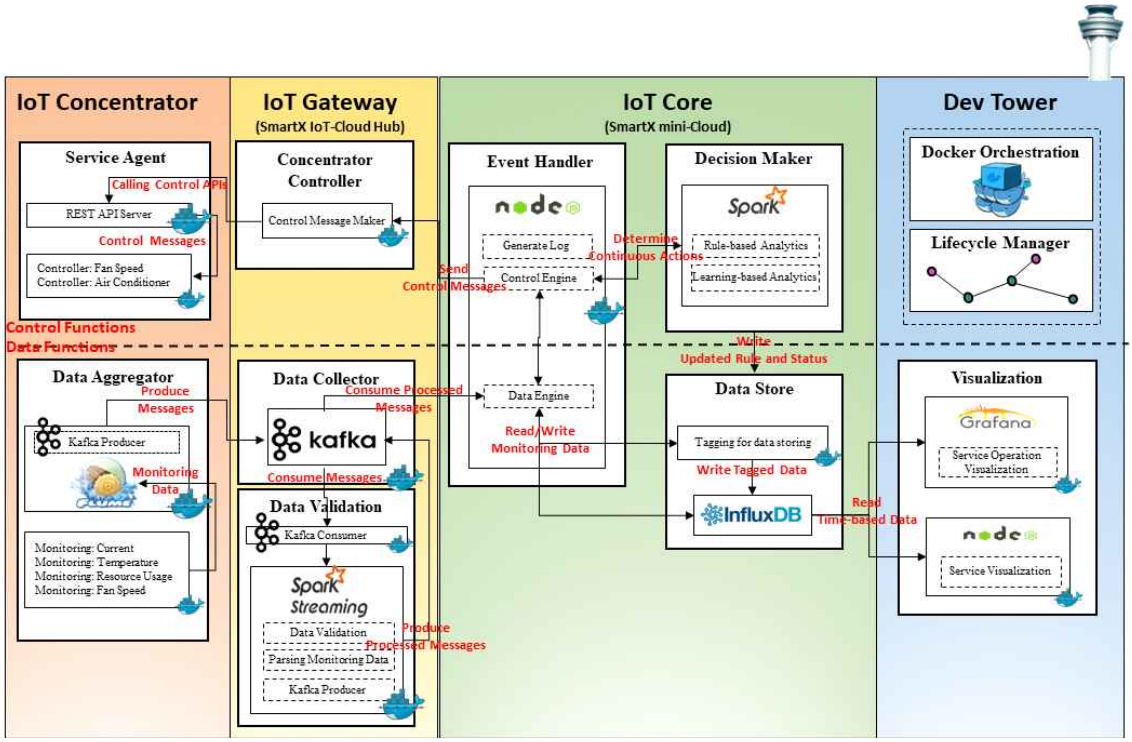

그림 5 스마트 에너지 서비스에 대한 기능 구성도

o 이러한 사물 인터넷과 클라우드를 포함하는 환경을 바탕으로, 그림2와 같이 스마트 에너지 IoT-Cloud 서비스의 요소 기능들을 정의하였다. 서비스 요구사항을 만족하기 위한 수집, 전달, 처리, 제어, 저장, 시각화 기능들을 통해 서비스 시나리오에 따른 원활하고 안정적인 서비스 운용을 고려하였다.

o Raspberry Pi를 활용하여 적외선 신호 송출을 통한 냉방 장치 제어 및 서버실 내부의 온도 및 습도 정보를 수집한다. 각 박스 별 자원 상태는 Net-SNMP를 통해 CPU, 메모리, 디스크 상태를 포함한 14개 항목에 대해 모니터링한다. 각 기능들은 Raspberry Pi의 자원 측면의 제약을 극복하고 서비스의 유연성을 더하기 위하여 Docker 컨테이너 기반의 가상화 기술을 활용한다. 이로 인해 보다 다양한 환경에 대한 적용과 배포 측면의 편의도 기대할 수 있다.

```
FROM resin/rpi-raspbian:wheezy
MAINTAINER Seungryong Kim <srkim@nm.gist.ac.kr>

#Update & Install wget, vim
RUN apt-get update
RUN apt-get -y install wget
RUN apt-get -y install vim

#Timezone
RUN cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime

#Install Oracle JAVA
RUN mkdir -p /opt
RUN wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=http%3
   A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie" "http:
   //download.oracle.com/otn-pub/java/jdk/8u33-b05/jdk-8u33-linux-arm-vfp-hfl
   t.tar.gz" -O - | tar -zxv -C /opt

#Configurate environmental variables
ENV JAVA_HOME /opt/jdk1.8.0_33
ENV PATH $PATH:/opt/jdk1.8.0_33/bin
RUN ln -s /opt/jdk1.8.0_33/bin/java /usr/bin/java

#Install Flume
RUN sudo wget --no-check-certificate http://www.apache.org/dist/flume/1.6.0/
   apache-flume-1.6.0-bin.tar.gz -O - | tar -zxv
RUN sudo mv apache-flume-1.6.0-bin /flume

ADD plugins.d /flume/plugins.d
ADD flume-conf.properties /flume/conf/

#Working directory
WORKDIR /flume
```

그림 6 센서 및 상태 정보 수집을 위한 Apache Flume 컨테이너 Dockerfile

o 수집된 상태 정보는 Apache Kafka를 통해 클라우드로 전달된다. Kafka는 메시지를 파일 시스템에 저장함으로써 빠른 처리가 가능하며, 분산 시스템으로 설계된 까닭에 유연하고 견고하다. 이러한 구조적인 특성 때문에 상대적으로 부족한 처리 능력을 가진 IoT 기기들이 메시지를 보내는 과정에서 발생하는 부하를 최소화할 수 있으며, 동시에 다수의 메시지를 빠르게 처리 가능하다는 점에서 적절한 오픈소스 소프트웨어이다.

```
FROM ubuntu:14.04
MAINTAINER Seungryong Kim <srkim@nm.gist.ac.kr>

#Update & Install wget
RUN sudo apt-get update
RUN sudo apt-get -y install wget

#Install Oracle JAVA
RUN sudo mkdir -p /opt
RUN sudo wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=htt
  p%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie" "
  http://download.oracle.com/otn-pub/java/jdk/8u45-b14/jdk-8u45-linux-x64.tar.
  gz" -O - | tar -zxv -C /opt

#Configurate environmental variables
ENV JAVA_HOME /opt/jdk1.8.0_45
ENV PATH $PATH:/opt/jdk1.8.0_45/bin
RUN ln -s /opt/jdk1.8.0_45/bin/java /usr/bin/java

#Install Kafka
RUN sudo wget --no-check-certificate http://apache.tt.co.kr/kafka/0.8.2.0/kafka_
  2.10-0.8.2.0.tgz -O - | tar -zxv
RUN sudo mv kafka_2.10-0.8.2.0 /kafka

WORKDIR /kafka
```

그림 7 신속하고 안정적인 데이터 전달을 위한 Apache Kafka 컨테이너 Dockerfile

o 클라우드 환경으로 수집된 정보들은 데이터의 유효성을 검증하고, 정형화되어 실시간으로 분석됨과 동시에 향후 배치 분석을 위하여 저장소에 저장된다. 이 과정에서 이상의 발생이 파악될 경우 메일 발송과 사용자 인터페이스를 통해 관리자에게 상황을 전달하고, 냉방 장치의 상태 및 각 박스의 상태를 변경을 위한 명령을 수행한다.

```
const influx = require('influx')
var kafka = require('kafka-node');

// Temp
// InfluxDB - Temp
var tempDB = new influx.InfluxDB({
    // single-host configuration
    host: 'hub1',
    port: 8086, // optional, default 8086
    protocol: 'http', // optional, default 'http'
    username: 'admin',
    password: '#####', // 비공개
    database: 'senics'
});



// Kafka - Temp
var tempKafka = new kafka.Client('hub1:2181');
var tempOffset = new kafka.Offset(tempKafka);

tempOffset.fetch([{
        topic: 'temp',
        partition: 0,
        time: -1,
        maxNum: 1
    },
    {
        topic: 'temp',
        partition: 1,
```

```
        time: -1,
        maxNum: 1
    },
    {
        topic: 'temp',
        partition: 2,
        time: -1,
        maxNum: 1
    }
], function(err, data) {

    var tempConsumer = new kafka.Consumer(tempKafka, [{
            topic: 'temp',
            partition: 0,
            offset: data['temp'][0][0]
        },
        {
            topic: 'temp',
            partition: 1,
            offset: data['temp'][1][0]
        },
        {
            topic: 'temp',
            partition: 2,
            offset: data['temp'][2][0]
        }
    ], {
        autoCommit: false,
        fromOffset: true
    });
        tempConsumer.on('message', function(message) {
        var str = String(message.value).split(',');
        var id = parseInt(str[1]);
        var humidity = parseFloat(str[2]);
        var temperature = parseFloat(str[3]);
```

```
            tempDB.writePoints([{
                measurement: 'temp',
                tags: {
                    id: id
                },
                fields: {
                    humidity: humidity,
                    temperature: temperature
                },
            }])
    });
});
```

그림 8 수집된 데이터를 시계열 데이터베이스에 저장하기 위한 Node.js 스크립트

o 스마트 에너지 서비스를 위한 웹 기반의 유저 인터페이스 Control Room은 Flask 서버 어플리케이션 위에 비동기 처리, 소켓 통신 등을 지원한다. 클라이언트에서 Flask 서버 어플리케이션으로 HTTP 요청을 하면 내부적으로 소켓 통신 및 데이터베이스 쿼리 등을 수행하는 구조이며, 비동기 처리 및 Cron Job을 위해 Celery를 이용했고 캐싱 데이터베이스 및 비동기 처리 큐로서 Redis를 사용했다. 또한 socket.io를 이용해 센서 데이터를 실시간으로 렌더링한다.

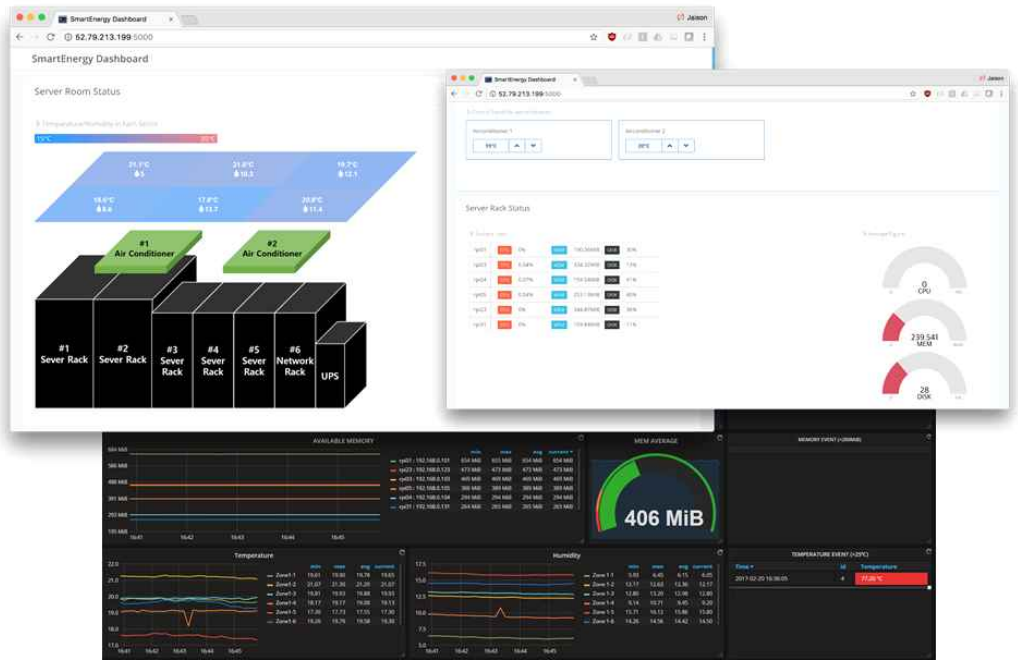

그림 9 스마트 에너지 서비스 유저 인터페이스

```
# SocketIO
def background_thread():
    while True:
        temp = worker.get_keyby_data_last("temp")
        temp = json.dumps(temp)
        resource = worker.get_keyby_data_last("resource")
        resource = json.dumps(resource)
        temp_dump = worker.get_dump_data("temp")
        temp_resource = worker.get_dump_data("resource")
        temp_mean = worker.mean_dump_data(temp_dump, "temperature")
        hum_mean = worker.mean_dump_data(temp_dump, "humidity")
        resource_cpu_mean = worker.mean_dump_data(temp_resource, "cpu")
        resource_mem_mean = worker.mean_dump_data(temp_resource, "memor
  y")
        resource_disk_mean = worker.mean_dump_data(temp_resource, "disk")

        socketio.emit('background_thread', {
                        'temp': temp,
                        'resource': resource,
                        'temp_mean': temp_mean,
                        'hum_mean': hum_mean,
                        'resource_cpu_mean': resource_cpu_mean,
                        'resource_mem_mean': resource_mem_mean,
                        'resource_disk_mean': resource_disk_mean
                        },)
        socketio.sleep(2)
```

그림 10 Control Room의 실시간 렌더링을 위한 socket.io 처리 코드

o Control Room은 IoT-Cloud Hub를 통해 InfluxDB로 수집된 데이터를 가공하고 UI로 실시간 렌더링되도록 전체 시스템이 설계되어있다. Control Room 내부적으로 데이터 흐름을 살펴보자면, 먼저 Celery로 구현된 Cron Job Worker는 InfluxDB에 일정기간 동안 쌓인 데이터들을 조회하고 Redis로 캐싱한다. 그리고 클라이언트에서 HTTP 요청이 발생하면 Flask 서버 어플리케이션과 Socket.io 구현체는 캐싱된 Redis를 쿼리해 데이터를 프런트엔드에 전달하는 방식으로 구성되어있다.

4. 워크플로우에 따른 IoT-Cloud SaaS 응용 배포 및 합성 실증

4.1. 서비스 워크플로우

o 오버클라우드 상에서 IoT-Cloud SaaS 응용의 호환성 지원을 실증하기 위하여, 스마트 에너지 서비스를 위해 구성한 오버클라우드를 바탕으로 워크플로우 기반의 IoT-Cloud SaaS 응용의 배포 및 합성을 실증한다. 오버클라우드는 OpenStack을 활용한 LXD 기반의 컨테이너로 구성된 컨테이너 환경이며, 이를 바탕으로 Apache Airflow 도구를 활용해 IoT-Cloud SaaS 응용을 위한 워크플로우를 구성하고 지원하였다.

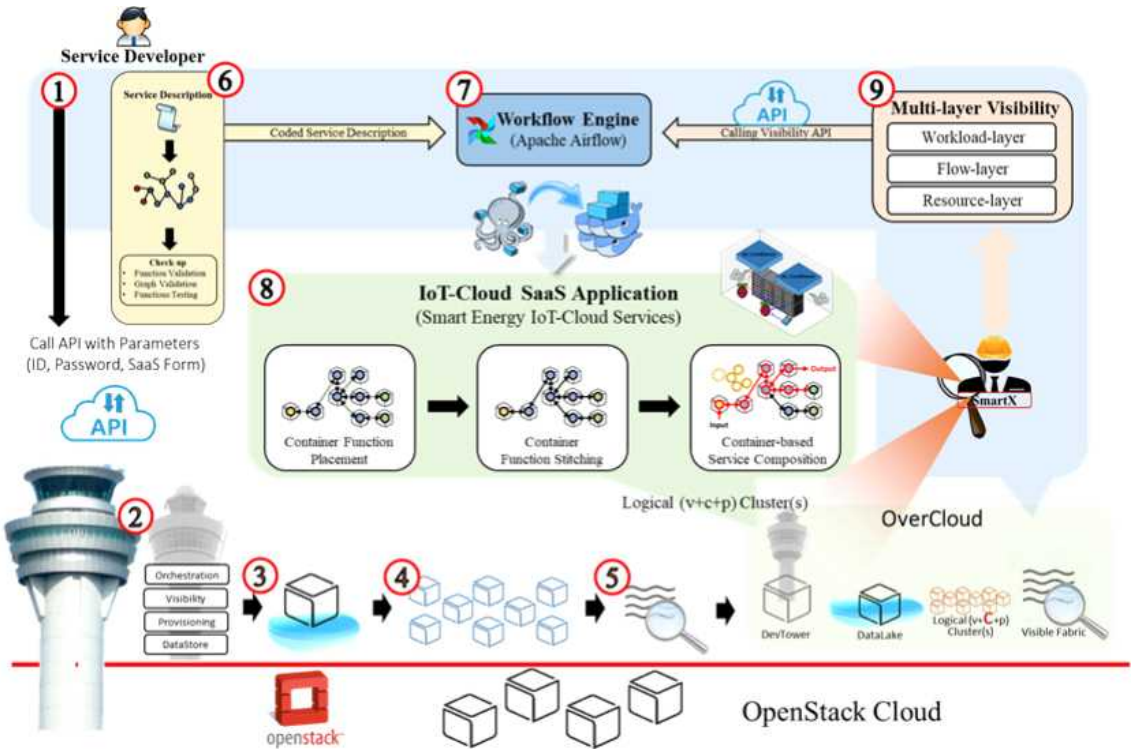

그림 11 오버클라우드 상의 IoT-Cloud SaaS 응용 배포 및 합성 시나리오

o 오버클라우드의 구성이 완료된 후, 확보된 자원을 바탕으로 스마트 에너지 서비스에 대한 워크플로우 기반의 서비스 합성을 수행한다. 워크플로우의 진행에 따라 오버클라우드 상의 Logical Cluster 상에 해당 SaaS 응용을 구성하는 컨테이너 기반의 기능(Function)들의 이미지가 배포 및 실행되는 것을 확인할 수 있다. 해당 작업이 정상적으로 수행 종료되면, 서비스에서 제공하는 웹 기반의 사용자 인터페이스를 통해 서버실 내부의 상황을 파악할 수 있다.

o 기능 배포 워크플로우는 확보한 자원에 각각의 요소 기능들을 적절한 박스에 배포하고 이를 확인하는 과정이다. 이를 위해 각 박스의 주소 정보를 확인하고, 도커 컨테이너 실행을 위한 도커 엔진 설치 및 이미지 풀링을 수행하며 그 결과를 확인하고 검증하는 작업이 필요하다.

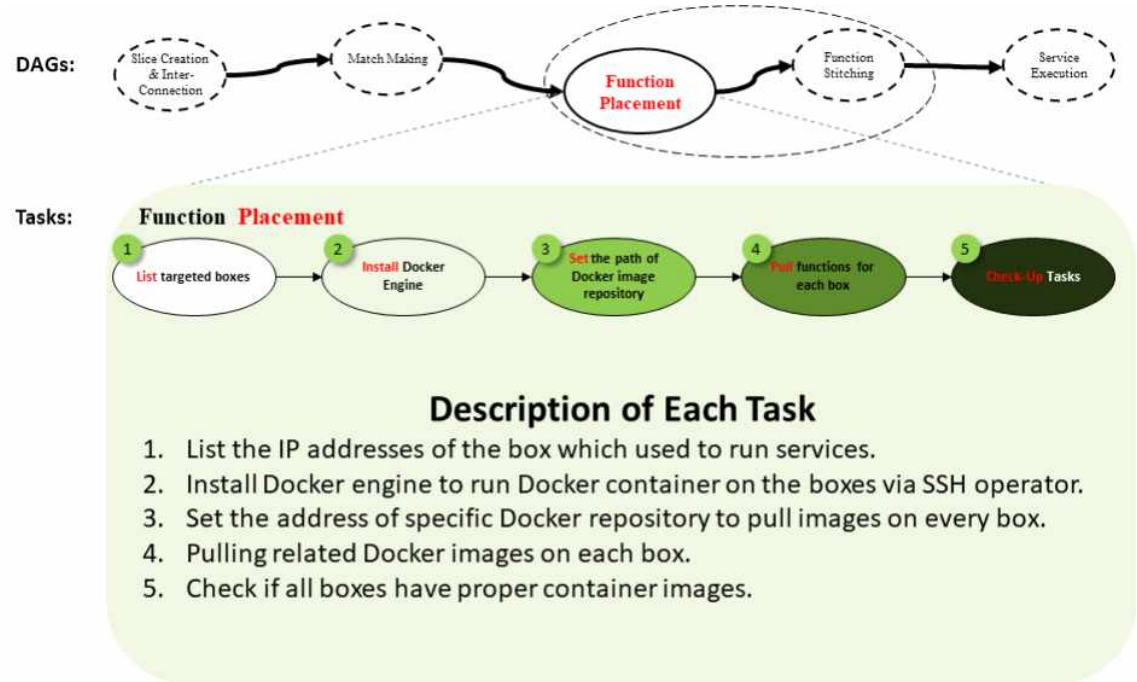

그림 12 기능 배포 워크플로우 상세 흐름

o 각 박스의 주소 정보는 사전에 작성된 서비스 명세를 바탕으로 확인하며, 도커 컨테이너 이미지 저장소는 도커 허브 또는 네크워크 상에서 발생할 수 있는 문제를 감안하여 별도의 이미지 저장소를 구축하여 관리한다. 각 박스에는 최소한의 컨테이너가 실행되게끔 배포하며, 이를 위해 필요한 각 요소 기능 별 스케일 또한 사용자의 요구에 따라 명세하여 확인한다.

o 데이터의 입출력과 관련된 요소 기능의 경우에는 전달 과정에서 발생할 수 있는 문제로 인한 손실 및 변조를 방지하기 위하여 예외적으로 가급적 같은 박스 내에 배포한다. 이에 따라 각 데이터간 전달에서 비롯된 처리 지연의 빈도를 줄이고 서비스의 품질 유지 측면에서 효과를 기대할 수 있다.

o 기능 접합 워크플로우는 배포한 각각의 요소 기능들 간의 관계를 확인하고, 이를 바탕으로 각 요소 기능들 간의 연결성을 확보하는 과정이다. 복수개의 요소 기능들은 각기 병렬적으로 해당 과정을 수행하며 경우에 따라 별도의 도커 컨테이너 오케스트레이션 도구를 활용하는 방안 또한 고려하였다. 이를 활용할 경우 사용자가 구체적인 워크플로우 태스크를 정의할 필요 없이 오케스트레이션 도구의 지원 기능을 활용하여 보다 용이하게 워크플로우를 생성하고 활용할 수 있다.

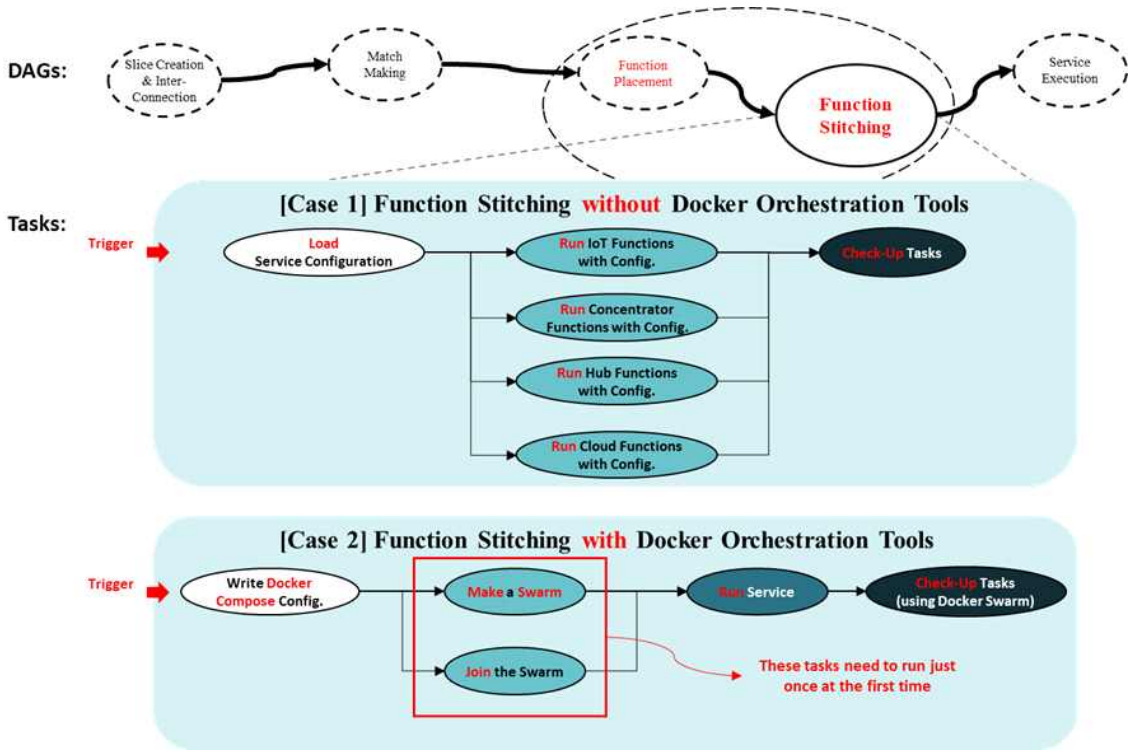

그림 13 기능 접합 워크플로우 상세 흐름

o 각 요소 기능의 실행에 앞서서 필요한 변수 값들은 사전에 정의한 서비스 명세를 바탕으로 확인한다. 해당 값들을 바탕으로 각 기능이 수행됨으로써 사용자의 의도에 적합하게 각 기능들이 수행될 수 있다. 스마트 에너지 서비스의 경우 각 요소 기능들이 보다 범용적인 용도로 활용 가능한 오픈소스 소프트웨어로 구성되어 있으므로, 이를 활용한다면 별도의 프로그램 수정이 없이도 원하는 IoT-Cloud SaaS 응용을 생성할 수 있다.

o 컨테이너 오케스트레이션 도구로서 도커 스웜(Docker Swarm)을 사용할 경우, 최초 수행시 하나의 스웜을 생성하고 각 노드들이 해당 스웜에 참여하는 부가적인 작업이 필요하지만, 이후에는 해당 태스크가 필요하지 않으므로 빈번하게 SaaS 응용의 업데이트가 이뤄지는 경우에는 용이하게 활용할 수 있다.

4.2. 서비스 합성 실증

o 정의된 서비스 워크플로우를 바탕으로 배포된 각 요소 기능들에 대하여 사전에 작성된 명세를 바탕으로 설정을 적용하여 서비스를 수행하고 그 결과를 확인한다. 워크플로우 엔진으로는 Apache Airflow를 활용하여 파이썬 기반으로 서비스 명세를 작성하였으며, 그 결과는 DAG(Directed Acyclic Graph) 형태로써 정리된다.

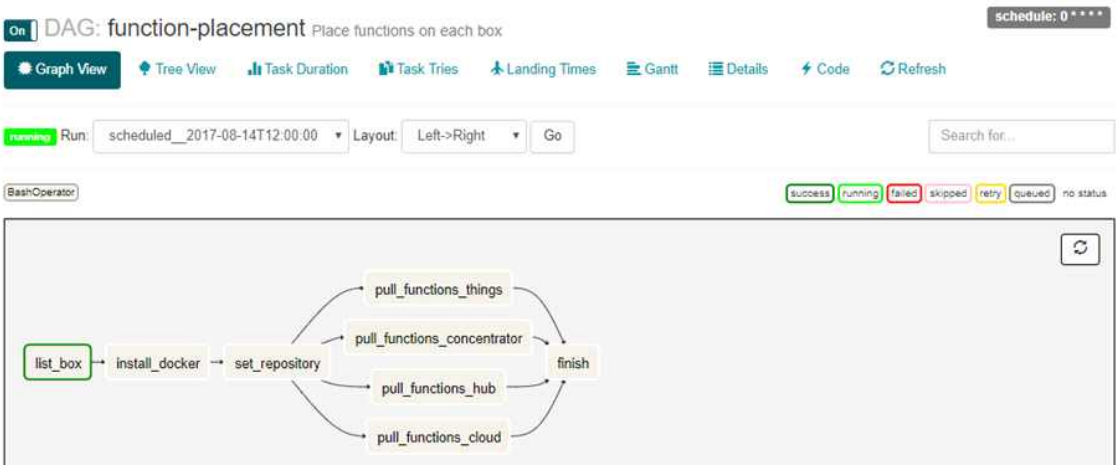

그림 14 서비스 명세를 바탕으로 하는 DAG

o 지정한 스케쥴에 따라서 해당 워크플로우를 수행하며 각 태스크 별 수행에 소요된 시간과 그 결과를 웹 기반의 인터페이스를 통해 확인할 수 있다. 각 유형별 기능 요소들에 대해 병렬적으로 배포 작업을 수행하였으며, 그 결과에 따라 정상적으로 서비스가 수행되는 것을 확인하였다.

그림 15 워크플로우 수행에 소요되는 시간 측정 결과

```
root@devtower:/home/ubuntu# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@devtower:/home/ubuntu# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
smartxenergy/kafka   latest              7284d5844be7        8 months ago        593.1 MB
root@devtower:/home/ubuntu# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
smartxenergy/kafka   latest              7284d5844be7        8 months ago        593.1 MB
root@devtower:/home/ubuntu# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
grafana/grafana      latest              df6d4f43bfa3        5 days ago          285.2 MB
smartxenergy/kafka   latest              7284d5844be7        8 months ago        593.1 MB
root@devtower:/home/ubuntu# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
grafana/grafana      latest              df6d4f43bfa3        5 days ago          285.2 MB
smartxenergy/kafka   latest              7284d5844be7        8 months ago        593.1 MB
root@devtower:/home/ubuntu# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
grafana/grafana      latest              df6d4f43bfa3        5 days ago          285.2 MB
influxdb             1.2                 cfebf3d12055        2 weeks ago         188 MB
smartxenergy/kafka   latest              7284d5844be7        8 months ago        593.1 MB
root@devtower:/home/ubuntu# docker images
```

그림 16 기능 배포 워크플로우 수행에 따른 도커 이미지 배포

o 그림 16과 같이 기능 배포 워크플로우가 수행됨에 따라 도커 엔진이 설치된 Logical Cluster 상에 각각 필요한 도커 이미지를 내려받는 것을 확인할 수 있다. 사용자는 워크플로우 수행 후 별도의 작업 없이도 각 박스에 적절한 이미지가 저장되고 있음을 확인할 수 있다.

```
ubuntu@logical-cluster:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              P
ubuntu@logical-cluster:~$ sudo docker ps
CONTAINER ID        IMAGE                COMMAND                  CREATED             STATUS
4aaa40c7ff85        redis                "docker-entrypoint.sh"   17 seconds ago      Up 13 seconds
2120e2ed8693        jaisonoh/croom       "python3 app/app.py"     32 seconds ago      Up 22 seconds
b70a4c42ab05        smartxenergy/kafka   "/bin/bash"              51 seconds ago      Up 47 seconds
ubuntu@logical-cluster:~$
```

그림 17 기능 접합 워크플로우 수행에 따른 컨테이너 실행 상태

o 그림 17과 같이 기능 접합 워크플로우에 의해 각 박스에서 컨테이너가 실행되는 것을 확인할 수 있다. 해당 워크플로우는 서비스 명세서 기재된 변수값들을 바탕으로 기능 배포 워크플로우의 결과물로서 저장된 컨테이너 이미지들을 실행하여 IoT-Cloud SaaS 응용의 요소 기능들을 실행하고 연결한다. 이에 따른 결과는 각 박스 별로 동작하고 있는 컨테이너 현황을 파악함으로써 확인할 수 있으며, 워크플로우의 수행이 완료되면 해당 IoT-Cloud SaaS 응용이 실행되는 것을 확인할 수 있다.

o 향후에는 디자인 된 워크플로우를 중심으로 HPC/BigData 및 보다 다양한 SaaS 응용을 대상으로 구체적인 서비스 워크플로우 실증과 함께 점진적으로 자동화의 수준을 확장함으로써 궁극적으로는 호환성을 보장하는 다양한 SaaS 응용에 대한 지원을 완성하고자 개선 및 보완할 계획이다.

References

[1] Docker, https://www.docker.com/

[2] Smart Energy IoT-Cloud Service, https://github.com/SmartX-Labs/SmartX-Energy

[3] Apache Airflow, https://airflow.apache.org/

[4] S. Nastic, S. Sehic, D. Le, H. Truong, and S. Dustdar, "Provisioning software-defined iot cloud systems," in Proc. International Conference on Future Internet of Things and Cloud (FiCloud 2014), 2014, pp. 288-295.

[5] F. Khodadadi, R. Calheiros, and R. Buyya, "A data-centric framework for development and deployment of internet of things applications in clouds," in Proc. IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2015), 2015, pp. 1-6.

[6] J. Shuja, K. Bilal, S. Madani, M. Othman, R. Ranjan, P. Balaji, and S. Khan, "Survey of techniques and architectures for designing energy-efficient data centers," 2014.

[7] Z. Sheng, H. Wang, C. Yin, X. Hu, S. Yang, and V. Leung, "Lightweight management of resource-constrained sensor devices in internet of things," IEEE Internet of Things Journal, vol. 2, no. 5, pp. 402-411, 2015.

SaaS OverCloud 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: 2017/12