ETRI

2016
Electronics and
Telecommunications
Trends

OPNFV Promise 프로젝트

OPNFV Promise Project

백동명 (D.M Baek) 네트워크컴퓨팅융합연구실 선임연구원
이범철 (B.C. Lee) 네트워크컴퓨팅융합연구실 책임연구원

* 본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음[B0101-16-0233, 스마트 네트워킹 핵심 기술 개발].

Promise는 OPNFV의 자원 예약 및 할당, 용량 관리 프로젝트이다. 자원을 많이 필요로 하는 월드컵 경기나 쓰나미 경보 같은 재해를 대비한 자원 예약을 통해 끊임없는 서비스를 제공하기 위함이다. 그러나 기능 구현을 위해 OpenStack 내 많은 컴포넌트의 수정이 필요한 어려운 점이 있다. 그래서 Phase2의 OpenStack 과의 통합된 모델을 목표로, 현재의 Phase1은 Shim-layer 구현물 단계에 있다. Shim-layer는 Node.js 위에서 실행되는 YangForge 프레임워크로 기술된다. 이는 Yang 데이터 모델링로서 스키마를 표현하고, 컨트롤 로직은 YAML로, 설정 데이타는 JSON으로, JavaScript를 만드는 CoffeeScript 언어로서 스펙을 기술한다. 인터페이스는 CLI, Web GUI, REST/JSON, WebSockets이다. 이를 OPNFV summit 2015년 11월에 데모한 바 있다. 이 프로젝트를 분석을 통해 용량 관리, 자원 예약 및 자원 할당하는 예약기법들을 살펴보고자 한다.

Ⅰ. 서론

OPNFV 프로젝트의 목적은 ETSI NFV ISG에서 정의한 NFV 프레임워크를 따르면서 공개소프트웨어 기반으로 NFV 기술 개발과 확산이다. NFV란 표준 제정 목적과 달리 커뮤니티 주도로 이루어지며, 기존의 공개 소프트웨어 솔루션을 최대한 활용해서 활성화 시키는 것이 그 특징이다. 초기 개발범위는 NFV 전 범위가 아니라 NFV 인프라와 Virtual Infrastructure Manager (VIM)에 한정되는 하부 인프라부터 출발한다. 현재 버전은 Arno, Brahmaputra(2016. 4.), Colorado (2016. 9.) 등 알파벳 순서로 나가고 있다. 이 문서는 B version 중심으로 분석된 것이다.

하부 프로젝트는 50여 개가 있으며 요구사항, 통합 및 시험(integration & testing), 협력 개발 문서 범주로 나누고 있다. 대표적인 프로젝트로서는 Doctor(fault management), Copper(virtualized infrastructure deployment policies), Promise(resource management), Octopus(continuous integration), Pharos (testbed infrastructure), ONOSFW(ONOS framework) 등이 있다[1].

이때 Promise는 2014년 12월 출발하여, 2015년 6월에 안정된 버전을 내놓은 요구사항 프로젝트이다. 목적은 나중에 사용할 목적으로 자원을 예약하거나 자원 용량(capacity)을 관리하는 것이다. 이때 '자원'이란 컴퓨터, 네트워크와 스토리지에 대한 자원 풀 (pool)을 말하며, '나중에' 란 즉시 사용할 것과 미리 자원을 확보하는 두 가지 유형을 포함한다.

프로젝트의 리더인 ClearPath Network사의 Peter Lee와 NTT DOCOMO의 Kunzmann이 미팅을 호스팅한다. 팀 미팅은 일주일에 한번 정규 모임을 하며, 사전 의제에 대해서 국제 컨퍼런스 콜과 채팅(#opnfv-promise IRC)으로 회의를 한다. 명단으로 보면 committer가 6명, contributor가 10명, 미팅 참여자가 18명 정도이다[2].

문서는 현재(2016. 9. 13) B version으로 Requirement, User Guide, Configuration Guide 등의 3종 있는데 Requirement 문서 중심으로 구조, 메시지, 정보 요소 등의 핵심이 기술되었다[3]. 나머지 두 개는 매우 짧은 분량이다. User Guide문서는 promise 구현물 (shim-layer)의 3가지 기능인 용량 관리(capacity management), 자원 예약(resource reservation)과 자원 할당(resource allocation)에 대한 개념 설명을 담았다[4]. Configuration Guide 문서는 promise를 Github로 통해 설치하는 방법과 33종의 테스트 케이스를 시험하는 방법을 기술하였다[5].

이 프로젝트는 주변 관련 프로젝트에 의존성이 높다. OPNFV의 규정에 대해서는 ETSI의 NFV GS 중 MANO, INF, IFA 과 관련이 있고, Copper, YangForge, OpenStack의 Blazar 프로젝트과 연관이 많다. Copper는 가상인프라 배치 정책을 다루고, YangForge는 소스 개발의 프레임워크를 제공하며, Blazar는 옛날 이름이 Climate로서 OpenStack 내의 예약 서비스를 다룬다. 현재는 Climate 0.1.0에서 보류되었다.

Ⅱ. OPNFV의 Promise

본 장은 OPNFV의 Requirement 문서[3]와 슬라이드 자료[6][7]를 참조하여 분석한 것이다. 분석에서 유의할 점을 언급한다. 첫째, Requirement 문서는 Phase1과 Phase2를 구분하고 있다. Phase2는 OpenStack과 통합된 목표를 제시하며, Phase1은 그 중간 단계로서 OpenStack을 수정하지 않고 Shim-layer라는 계층을 두어서, 메시지를 가로채고 추출하는 형식으로 구현하였다. 둘째, 일반적인 요구사항과 Shim-layer 구현을 따로 구분하고 있다. 표준 규격을 먼저 정하고 코딩하는

방식이 아닌, 코딩과 규격 수정을 같이 진행하는 방식을 택하고 있다. 셋째, Promise 문서는 전체 NFV 구조 관점이 아니어서 슬라이드 자료를 참조해야 한다. 넷째, NFVO-VIM 관계에서는 매우 추상적인 예약 정보를 다루지만, VNFM-VIM에서는 구체적으로 리소스(가상머신 혹은 인스턴스)를 다루므로 자원 예약과 자원 할당을 잘 구분해야 한다.

1. 목표 및 유저 케이스

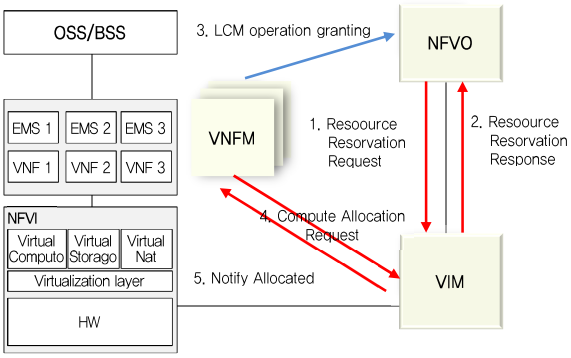
단기 목표로선 예약된 시간에 예약된 자원을 모두 사용되고 할당되는 것이다. 자원이 무한 용량이 아니라면 항상 자원 할당이 되리라는 보장이 없기에 예약이 필요하다. 네트워크 서비스의 경우에는 고가용성이 중요하므로 이 예약 서비스가 매우 중요하다. 예를 들면 유명 가수의 콘서트, 월드컵 경기전 같은 경우는 피크 트래픽이 몰릴 것이므로 미리 예약해두어야 한다. 또한, 쓰나미 경보와 같이 예고없이 닥치는 재난에 대해서도 고가용성을 유지하기 위해 자원 예약이 중요하다.

장기적으로는 효율적인 자원 사용에 있다. 시작 전부터 자원을 미리 할당받아 사용할 수 있다면 '끊임없는' 자원 사용이 가능하다. 마치 선수 교체 허락 전에 후보 선수가 워밍업 하다가, 허락이 나면 바로 실전에 들어가는 것과 비슷하다.

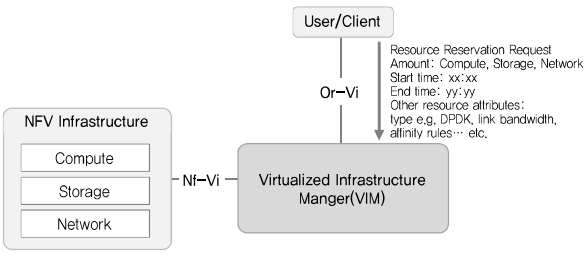
2. 전체 구조

Promise의 이해를 위해 (그림 1)와 같은 VNFV의 전체 구조를 이해해야 한다. NFV 관리 평면의 경우 NFVO, VNFM, VIM의 3단계 구조로 되어있다. 전체적으로는 NFVO가 VIM에게 자원을 요구하고 받고, NFVO가 VNFM에 대해서 라이프사이클관리를 하고, VNFM이 VIM에 대해서 구체적인 자원을 할당하면 VIM은 Notification을 통보하는 순서로 이루어진다[6].

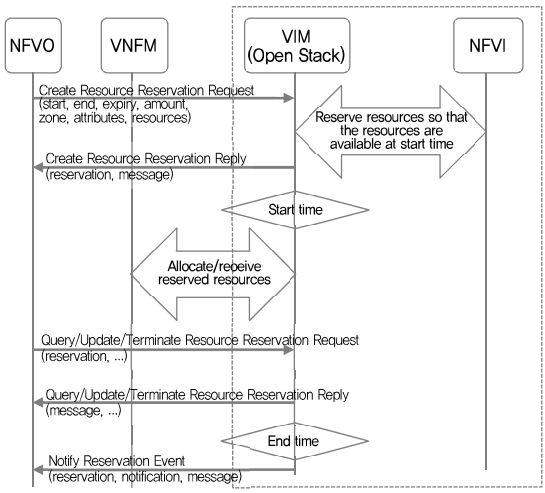
이때 VIM 입장에서 보면 NFVO나 VNFM은 하나의



(그림 1) NFV 전체 구조도[6]



(그림 2) Promise 구조[3]



(그림 3) 자원예약의 흐름도[3]

user/client가 되어 (그림 2)처럼 된다. 즉 NVFO는 VIM에 대해서 예약을 요구(resource reservation Request)하고 응답을 받는다. VNFM도 VIM은에 대해서 구체적인 자원 할당(compute Allocation Request)을 요구하고 응답을 받는다.

자원 예약의 순서도는 (그림 3)과 같다. VIM (OpenStack)

이 하부 Infra를 제어하며, NFVO가 자원 예약을 VNFM이 자원할당을 요구한다. 자원 예약에 필요한 파라메터는 Start time, End time, Expiry time, Amount, Zone, Attributes, Resources 등이 있다.

3. Phase1과 Phase2

Phase1 방법은 OpenStack 수정 없이 Shim-layer를 사용하는 방식으로 현재는 이 방법으로 구현되었다. 정해진 API를 이용하므로 다수 사이트로부터의 접근도 가능하다. 단점은 수신된 API를 받아서 개별 ICT 자원별로 OpenStack의 Blazar API를 사용하는 동기화(Synchronization)문제가 존재하는 것이다. 그래서 좀 더 복잡하고 구조적인 자원 관리가 힘들다는 것이다. 고급 기능인 Affinity, Anti-affinity가 제공되지 않고, VIM 인터페이스가 숨겨진 상태에서 사용된다[(그림 4) 참조].

Shim-layer에서 제공되는 API는 〈표 1〉과 같은데 자세한 설명은 User Guide에 있다. 문제는 ETSI NFV IFA 문서에서 정의하는 방식은 이것과 일치하지 않는다는 것이다. OpenStack 특징상 각 자원별로 컴포넌트 프로젝트가 따로 있기에 컴퓨터, 네트워크, 스토리지를 관할하는 컨포넌트를 다루는 Nova, Neutron, Cinder 프로젝트별로 API로 요구하는 형식이기 때문이다. 주어진 예약 관리, 할당 관리, 용량 관리 API를 개별 ICT 자원

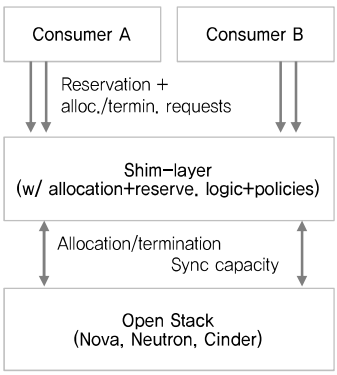
〈표 1〉 API 종류[3]

분류	API 종류
Reservation management	– Create-reservation – Update-reservation – Cancel-reservation – Query-reservation – Subscribe-reservation-event / notify-reservation-events
Allocation management	– Create-instance – Destroy-instance – Query-resource-collection – Subscribe-allocation-events / notify-allocation-events
Capacity management	– Increase/ decrease-capacity – Query-capacity – Subscribe-capacity-events / notify-capacity-events
(Multi-) provider management	– Add-provider – Remove-provider

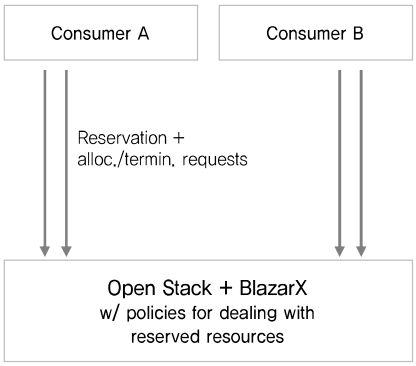
별로 분리하여 해당 컴포넌트 프로젝트에 API 요구하는 것을 동기화라고 한다. Phase 1에서는 이 문제를 잘 풀어야 한다. 자원할당에 대해서는 Requirement 문서에는 없는데 이는 IFA006(Vi-Vnfm)을 다루지 않기 때문이다.

Phase2에서는 (그림 5)와 같이 VIM의 API를 직접 이용하는 형태이므로 동기화가 필요없다. 대신 많은 규격 작업이 이루어지고, OpenStack 내 많은 컴포넌트가 수정되어야 한다. Blazar 프로젝트에서 다룬다.

차후 버전으로 StormForge 프레임워크를 제시하고 있다. 이는 Model-driven Software Architecture로서



(그림 4) Shim-layer 단계[6]

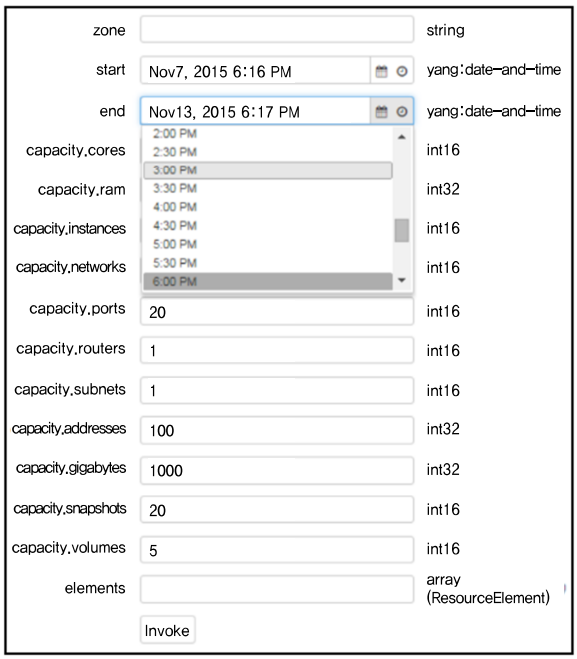


(그림 5) OpenStack과 통합 단계[6]

좀더 유연하고 어떤 플랫폼에서든 통합이 가능한 목적으로 만들고자 한다. 주요 컴포넌트가 Meta-class, Yang-compiler, Storm-compiler, Yang-storm 인데 Yang 스키마 기반으로 설계하려는 방향성과, 비전문가라도 애플리케이션만 잘 작성하면 실행될 수 있을 만큼, 하부 프레임워크를 유연하게 만들려고 한다. Parser가 Yang 스키마 파일을 해석해서 JavaScript meta-class Semantic tree hierarchy를 생성하는 식으로 작동한다. 이런 방식의 접근은 다른 OPNFV 프로젝트(MOVI, Parser, Resource Scheduler, Doctor, Copper)에서도 취하고 있다. API 기반 설계에서 Model 기반의 설계로 옮겨가고 있음을 알 수 있다.

4. Shim-layer 구현물 개요[6]

2015년 11월에 OPNFV Summit 2015에 발표되었다. Start time, End time, Resource 파라메터를 입력하는 방식으로서 (그림 6)와 같다. 자료상으로 보면 소모가 많은 자원은 core, network, addresses여서 여기를 중



(그림 6) Shim-layer 데모 GUI[6]

심으로 예약하였다.

지원 인터페이스는 Command Line Interface(CLI), REST/JSON, WebSocket, Browser이다. CLI는 YangForge의 YFC CLI 명령어를 사용하고, REST/JSON은 RESTful 명령과 JSON 설정 파일로서 외부에서 사용할 때, WebSocket은 Data synchronization에, Browser는 GUI로 사용된다[6].

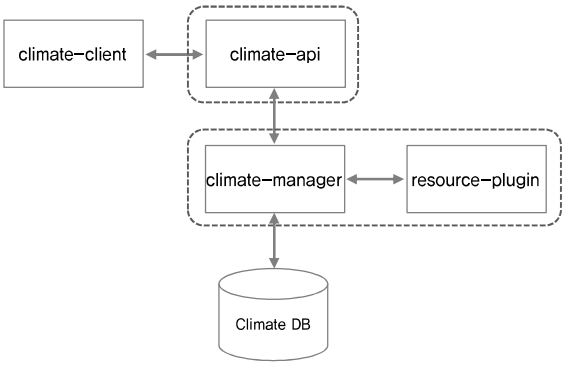
Model-driven 구현 언어는 YANG, YAML, JSON 등이다. YANG은 데이터 모델링 스키마이며, YAML은 컨트롤 로직 정의용으로, JSON은 설정 파일로 사용된다.

Ⅲ. OpenStack Blazar 프로젝트

비록 이 프로젝트는 현재 보류되었으나 promise 이전부터 시작하였기에 promise의 뿌리를 알 수 있다. 본 분석은 2013년 11월 홍콩 OpenStack Summit 자료과 현재 남은 Blazar 문헌을 기반으로 한다. 정식 문서를 남기지 못하고 Wiki 수준의 문서화 작업으로 보류되었다. Climate란 옛날 용어가 구조도에 그대로 남아있고, 소스에서도 신구 용어가 혼재된 상태이다[8][9].

이 프로젝트는 기존 OpenStack 개발 방법론처럼 Python 언어로 전통적인 REST 방식의 API방식으로 개발되었다. 개념도 Reservation와 Lease란 두 개로 분화된 수준이다. 즉 Promise의 YangForge 프레임워크처럼 모델 기반 디자인이 아닌 API 디자인 형태이다. 물리적인 컴퓨터 호스트의 예약을 다루는 것이 매우 특징적이다.

초창기에는 Mirantis 같은 OpenStack에 활발한 활동을 하는 회사가 참여하였다. 목적은 당연히 Promise와 비슷하게 피크 로드나 용량 설계를 준비하는 것으로 비슷하다. Lease란 단어가 특별한데 예약보다는 좀 더 구체적으로 start time, end time, reservations와 events를 파라메터로 정의한다. 그 당시에도 즉각 사용과 미래를 위해 예약하는 개념이 있었다. 특이한 점은 가상 머

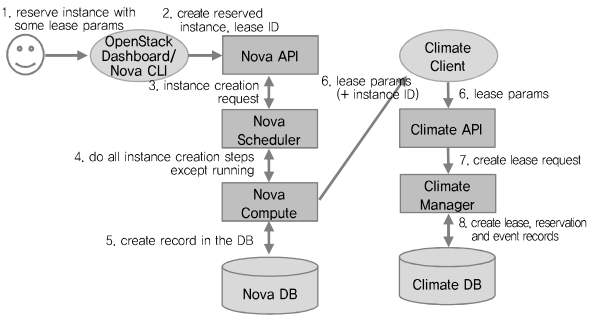


(그림 7) Shim-layer 데모 GUI[8]

신뿐만 아니라 물리적 호스트에 대해서도 예약기능을 다루었다. 문헌[2]에 따르면 Blazar는 현재 'Shelved VM'을 이용하여 오직 컴퓨터 자원 예약만 되고, start time과 end time만 파라메터로 구현된 상태라고 한다.

전체 구조는 climate란 옛 이름으로 남겨진 (그림 7)과 같다. Climate-manager가 중심 주체이다. Climate-client는 REST-API를 통해 시스템에 접근할 수 있다. Climate-api는 Climate-manager와 독립적으로 실행되면서 RPC API로 Climate-manager와 통신한다. Climate-manager는 Client 측의 create, update, get, delete 요구를 처리하며, DB와 연결, 이벤트 관리, 예약 등도 관리한다. Resource-plugin은 외부의 다양한 주체(VMs, volumes 등)를 처리하기 위해 Plugin 메커니즘을 이용한다. OpenStack에서의 일반적인 구조이다.

예약된 VM을 Lease 생성하는 한 제안을 문헌[12]에서 볼 수 있는데 (그림 8)과 같다. OpenStack GUI를 통



(그림 8) VM 생성 단계[9]

해서 Lease 파라메터를 통해 인스턴스를 예약하면, VM 생성을 담당하는 Nova 에 접속된다. 예약 인스턴스와 lease ID가 API를 거쳐서 Nova scheduler에 전달된다. 시간과 환경에 맞게 VM이 만들어져 Lease 파라메터로 climate client로 전달된다. 그 후에는 climate 흐름대로 climate-API, Climate Manager - Climate DB 등을 거치게 된다.

보류된 프로젝트의 옛 제안이어서 큰 의미는 없지만, Nova scheduler, Nova compute과 의존성이 높음을 확인할 수 있다. 아마 여타 다른 프로젝트와의 동시적 진행이 힘들고(gap analysis라는 이름으로 이슈화됨), OPNFV의 Promise 프로젝트가 나타나서 보류된 것으로 보인다.

예측컨데 OPNFV의 모델기반의 SourceForge 프레임워크에 영향을 받아, 지금까지의 API 위주의 설계에서 StormForge 프레임워크 설계로 방향을 선회하는 것으로 판단된다[7].

Ⅳ. 구현 언어 관련 경향

Promise 소스 코드에는 생소한 모델링 언어와 전산언어, 프레임워크들이 많다. 그래서 따로 공간을 여러 IT경향을 기술하고자 한다. 대상은 Yang, JSON, YAML은 모델링 언어이고 Node.js, npm, YangForge, CoffeeScripts, JavaScript, WebSocket 등은 Node.js에 관련된 내용이다.

Yang은 동양의 음양 이론의 '양'을 의미하는 데이터 모델링 언어로서 IETF 소속의 NETMOD 워킹 그룹에서 개발했으며, 2010년 10월에 RFC 6020으로 정의되었다. 처음에는 NETCONF의 전송 내용을 위해 표준화되었으나 점차 발전하면서 NETCONF 프로토콜에 종속되지 않고 다양한 모델링에 적용되고 있다.

이해의 출발은 웹 언어인 HTML의 단점에서 XML의 출현을 이해하는 것이다. Html이 전 세계 인터넷 웹 언

어를 주름잡고 있으나, 태그가 한정적이고, 데이터의 계층 구조를 표현하기 어려웠다. 그래서 XML이 이를 해결한 한 솔루션이었다. W3C에서 제정하였기에 공신력이 있고, 정보를 저장하고 확장하는데 용이하고, 각 언어와 친화력이 좋으며, 작성자가 일련의 규칙을 정할 수 있는 장점을 지닌다. 그래서 10여년 동안 대세를 이루었다.

그 후 JSON이 등장하여 XML을 위협한다. JSON은 태그 대신에 괄호를 사용하므로 가시성이 좋고, 내용이 함축적이고 Markup 언어의 오버헤드가 없는 최소한의 정보를 담고 있다. JSON이란 말 자체도 JavaScript Object Notation이어서 프로그래밍 언어와 친화적임을 알 수 있다. 사람이 읽고 쓰기에 용이하며, 기계가 분석하고 생성함에도 용이하다는 평가이다.

YAML 언어는 좀 더 가독성이 좋다. 실제로 그 코드를 보면 항목을 표현할 때, 태그와 괄호를 이용하지 않고 탭과 줄바꿈을 사용하는 일반 문서와 닮았다. 초창기에는 'YAML Ain't Markup Language'라고 해서 Markup 언어가 아니란 뜻이나, 후에는 'Yet another markup language'로 의미가 바뀌어 지금은 가벼운 마크업 언어로 인식되고 있다. 위키백과에 의하면 'XML, C, 파이썬, 펄, RFC2822에서 정의된 e-mail 양식에서 개념을 얻어 만들어진 '사람이 쉽게 읽을 수 있는' 데이터 직렬화 양식이다. 2001년에 클라크 에반스가 고안했고, Ingy dot Net 및 Oren Ben-Kiki와 함께 디자인했다'고 기술한다.

이런 Markup 언어 흐름과는 달리 Yang은 처음부터 네트워크 환경설정 프로토콜인 NETCONF와 밀접한 연관을 가지고 있었다. 네트워크 환경 설정을 위해서는 가장 원시적인 CLI와 좀 더 나은 Simple Network Management Protocol(SNMP)을 사용하는데 이들은 다양한 벤더들의 장비 타입 정보를 담기에 부족하다. 그래서 IETF에서는 오로지 네트워크 장비의 설정에만 중점을 둔 프로토콜과 효과적인 데이타 모델을 설계하였다. 그것이 바로 프로토콜로서는 NETCONF, 데이터 모델링 언어로서는 Yang이 되었다. 이로서 효율적인 설정관리와 기존 CLI에서 할 수 없던 데이터모델링을 사용하게 되어 이제 NETCONF-YANG의 조합은 네트워크 자동화에 필수가 되고 있다[10].

이런 NETCONF의 특징이 Promise 프로젝트 소스에도 나타나 있다. 수행되는 Action(RPC) 문법이 그것이다. NETCONF의 새로운 오퍼레이션으로 정의되는데 오퍼레이션의 이름, 입력 파라메터, 출력 파라미터는 Yang 데이터 정의 statement로 모델링된다. Notificaiton은 NETCONF에 적합한 사건의 통지를 모델링한다.

이제 Promise의 메인 언어인 Node.js을 알아보자. 'Node.js는 크롬 자바스크립트 엔진인 V8기반으로 만들어진 플랫폼이며, 빠르고 확장 가능한 네트워크 프로그램을 쉽게 작성할 수 있게 한다. 특히, 이벤트 주도(event-driven), 논블록킹(non-blocking) I/O 모델을 사용함으로써, 분산 환경에서 실행되는 가장 가볍고 효과적인 그리고 완벽한 데이터 집중적(data-intensive)인 실시간 애플리케이션을 작성할 수 있게 해줍니다'. 특징은 그동안 웹 브라우저에만 동작하던 저평가된 자바스크립트 언어를 서버 프로그래밍 언어로 사용하는 것이다. Ebay, PayPal에서는 전이중방식의 뛰어난 성능 때문에 아파치 기반에서 Node.js 기반으로 옮겼다. 단 새로운 언어여서 비동기식 프로그래밍이 익숙하지 않으며 라이브러리가 많이 없다는 단점이 있다.

Node.js의 정확한 이해를 위해선 서버 측 프로그래밍의 독보적 자리에 오른 PHP의 선 이해가 필요하다. Professional Hypertext Preprocessor 의 약자인데 1994년에 개발되어 수많은 라이브러리가 존재한다. Apache의 경우 전 세계의 50% 이상의 웹서버를 점유하고 있다. Linux-Apache-MySQL-PHP(LAMP)의 강한 결합을 통해 웹서버가 존재하는데 PHP는 html과 매우

친화적이고, MySQL 등의 DB와 잘 동작한다. 그러나 PC 단독으로 프로그래밍할 때는 서버-클라이언트 개념, 서버 프로그래밍, html 언어에 친숙할 필요가 없으므로 php 언어에 대한 일반인 인식은 약한 상태이다. 이에 비해 Node.js는 독립적인 완벽한 언어이며 클라이언트 및 서버 사이드 무관하게 동작하고, http 서버 라이브러리를 가지고 있기에 Apache 서버 없이도 몇 줄이면 웹서버로 동작하므로 크게 주목 받고 있다. 무엇보다 아파치 서버의 그 무거움에서 벗어나 간결한 언어로서 웹서버 기능을 충실히 수행한다. 자원이 약한 오픈소스 HW에도 이런 Node.js를 이용해서 웹서버로 충분히 동작시킬 수 있다.

Node.js가 활성화된 또 다른 이유는 npm(Node Package Manager)이다. 리눅스 우분투의 apt-get, 안드로이드 스마트폰의 Play Store 같은 패키지 관리 SW이다.. '$ npm install 〈패키지명〉'처럼 명령어를 타이핑하면 그대로 패키지를 다운 및 설치할 수 있다. 이런 방식을 통해 Node.js는 타인이 제작한 모듈을 쉽게 설치할 수 있다. Github처럼 패키지 관리와 버전 관리가 된다.

Promise에 사용되는 YangForge는 바로 Node.js의 패키지의 일종이다. 이것은 Promise 프로젝트를 수행하는 ClearPath Network사의 스폰스를 받아 제작되고 있다. 설치방법은 '$ npm install -g yangforge'이다. Yang 스키마 모델링 언어에 기반한 yfc란 YangForge Controller를 설치하여 cli 명령(build, config, deploy, run 등)을 수행하여 애플리케이션의 라이프사이클을 관리한다. 실행 명령인 '$ yfc run promise.yaml'로서 promise 프로젝트를 실행한다. 이는 CoffeeScript로 코딩되어 JavaScript로 변환되어 Node.js에서 동작한다[11].

CoffeeScript는 아쉬키나스가 2010년에 릴리즈한 언어로, JavaScript로 변환시킬 수 있는 언어이다. 단 몇 한 두줄의 간결한 코드로 10여 줄에 달하는 JavaScript 순환문을 대체할 수 있다. 이것은 jQuery처럼 라이브러리가 아니고, 컴파일러를 통해 컴파일되는 완벽한 또 다른 언어여서 성능 면에서 손실이 없다. JavaScript(이하 JS로 표현)의 최대 단점인 약한 가독성으로 인해 큰 프로그램 관리가 힘든 것을 해결한다. 또한, 패턴 매칭과 이해 구문(comprehension) 을통해 JS에 부족한 어휘를 보완한다. 따라서 YangForge를 통해서 Node.js, CoffeeScript, Yang Schema 등을 이용하는 프레임워크를 제공한다.

이번에는 JS 언어를 기술하고자 한다. 정적인 HTML에 동적 기능을 추가하기 위해 만들어졌다. 그러나 타 언어 비해 생소한 문법과, 오로지 JS만 지원하는 환경에서 코딩하는 상황에 처하므로 매우 저평가되었다. 그러다 Node.js가 등장하면서 다시 주목받기 시작했다. 언어 계통으로 보자면 이름처럼 Java에 가깝기 보다는 Lisp 언어 그리고 Schme 언어와 공통점이 많아서 'C의 옷을 입은 Lisp'라고 할 수 있다[7]. JS는 무척 약점이 많지만 모든 브라우저에서 사용할 수 있는 유일한 언어이다. 장점은 느슨한 타입 체크, 동적 객체, 경량적, 표현적인 객체 리터럴 표기법이며, 단점은 프로그래밍 모델이 전역변수로 기초로 하므로 프로그램의 유연성이 약하다[12].

마지막으로 WebSocket 을 소개한다. 이것은promise의 인터페이스의 하나로서 'http://...'로 접속하지 않고 'ws://..'로 접속한다. 이것은 기존의 http가 느린 원인인 반이중(half-duplex)의 단점을 해결한 전이중 방식(full-duplex)를 통해 성능을 높였다. Node.js 에서는 'npm install websocket' 명령어로 따로 설치해주어야 한다. 다음과 같이 정의된다. '웹 브라우저와 웹서버 사이에 전송 제어 프로토콜(TCP) 연결(80번 포트) 한 개만 생성하여 전이중(full-duplex) 통신을 제공하는 프로토콜. WebSocket은 웹 서버와 웹 브라우저상에 구현되어, 두 지점 간에 실시간 상호작용하도록 지원한다. 따라서 실시간을 요하는 채팅, 게임, 주식 거래 등과 같이 실시

간이 요구되는 응용 프로그램을 한층 효과적으로 구현할 수 있다. WebSocket 프로토콜은 2011년 인터넷 표준화 기구인 IETF에서(IETF RFC 6455) 표준화되었고, WebSocket API는 W3C에서 표준화를 완료했다.'[13]

V. 결론

Promise는 OPNFV의 자원 예약 및 할당, 용량 관리 프로젝트이다. 자원을 많이 필요로 하는 월드컵 경기나 쓰나미 경보 같은 재해를 대비한 자원 예약을 통해 끊임없는 서비스를 제공하기 위해서이다. 그러나 그 기능을 위해선 OpenStack 내 많은 컴포넌트의 수정이 필요한데 이 기능으로 인해 예전 기능에 변화가 없어야되는 어려운 점이 있다. 따라서 최종적으로는 VIM의 공식 API를 사용하는 OpenStack 과의 통합된 모델을 목표로, 현재로선 Shim-layer로서 중간 단계를 밟고 있다. Shim-layer는 Node.js 위에서 실행되는 YangForge 프레임워크으로, 2015년 11월에 데모하였다.

특히 지금까지 통신에서 C언어를, 웹에서는 주로 Java, OpenStack에서는 python을 사용했는데 이 프로젝트에서는 JS 언어와 프레임워크를 사용하는 데 특징이 있다. Node.js가 그것이다. 이는 웹서버의 강자인 아파치와 PHP를 대치할 만큼의 실시간 웹이 가능한 언어이다. 이 언어는 지금까지 저평가되어온 JavaScript 언어로 서버 프로그래밍을 하는 것인데 몇 줄의 코드로서 기존 무거운 아파치 서버 역할을 대치할 만큼 강력하다. 또한, 가독성이 낮은 JavaScript 로 변환시켜주는 CoffeScript 언어로서 스펙을 기술한다. 모델링 언어로서 Yang을 이용하고, 컨트롤 로직 정의는 YAML, 설정 파일은 JSON 등 다양한 데이터 모델링 언어를 이용하는 것도 큰 특징이다.

Shim-layer 접근을 넘어 OpenStack과 통합을 이루게 되면 현재 접근법의 동기화(Synchronization) 문제를 해결하게 될 것이다. 목표도 용량 관리 측면을 넘어서 미래의 수요를 예측하여 자원을 미리 예약, 할당하는 식으로 해서 '끊임없는' 서비스가 가능하도록 할 것이다. 방법은 미리 예측해서 지시하는 Proactive 방식과 계속 사용하면서 적응하며 반응하는 식의 Reactive 방식이 이용될 것이다. 좀 더 Interactive하고 Dynamic한 가시화(visualization)가 필요할 것이다. 한 걸음 더 나아가 현장 시뮬레이션을 통해 시나리오를 계획하는 방식이 나올 것이다[6].

약어 정리

CLI	Command Line Interface
ETSI	European Telecommunications Standards Institute
GS	Group Specification
GUI	Graphical User Interface
HTML	Hypertext Markup Language
ISG	Industry Specification Group
JS	JavaScript
JSON	JavaScript Object Notation
LAMP	Linux-Apache-MySQL-PHP
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
OPNFV	Open Platform for NFV
REST	Representational State Transfer
SNMP	Simple Network Management Protocol
VIM	Virtual Infrastructure Manager
VNFM	VNF Manager
W3C	World Wide Web Consortium
XML	Extensible Markup Language
YAML	Yet Another Markup Language

참고문헌

[1] https://wiki.opnfv.org/display/PROJ/Approved+Projects

[2] https://wiki.opnfv.org/display/promise/Promise

[3] http://artifacts.opnfv.org/promise/docs/requirements/

[4] http://artifacts.opnfv.org/promise/brahmaputra/docs/userguide/

[5] promise Configuration document: http://artifacts.opnfv.org/promise/brahmaputra/docs/configguide/featureconfig.html

[6] P. Lee., “Promise Resource Reservation,” Nov. 9th, 2015, OPNFV Summit, 2015.

[7] P. Lee, ClearPath Networks, “OPNFV Promise Requirements and Implementation Overview,” May 18th, 2015.

[8] https://wiki.openstack.org/wiki/Blazar

[9] D. Belova, Mirantis, “Climate Project,” Nov. 7th, 2013.

[10] http://www.ciscokrblog.com/694

[11] https://github.com/opnfv/yangforge

[12] 한빛미디어, “더글라스 크락포드의 자바스크립트 핵심가이드,” 2008.

[13] IT용어사전, http://terms.naver.com