ETRI

2015
Electronics and
Telecommunications
Trends

OpenStack 오케스트레이션 기술 분석 —HOT 템플릿

Analysis of OpenStack Heat Orchestration Technology—HOT Template

Ⅰ. 서론
Ⅱ. Heat 구조
Ⅲ. Heat Orchestration Template
Ⅳ. Nested Stack
Ⅴ. 결론

전홍석 (H.S. Jeon) 네트워크컴퓨팅융합연구실 선임연구원
김병식 (B.S. Kim) 네트워크컴퓨팅융합연구실 책임연구원
이범철 (B.C. Lee) 네트워크컴퓨팅융합연구실 실장

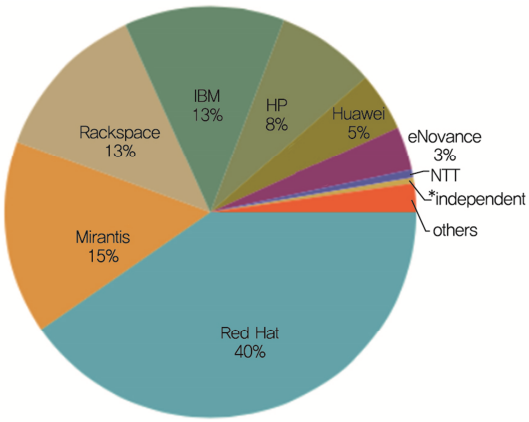
* 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음(14-000-05-001, 스마트 네트워킹 핵심 기술개발).

오픈스택(OpenStack)을 이용한 클라우드 서비스가 점차 확산되어가면서 수많은 가상 자원들을 생성하고 생성된 가상 자원들의 라이프사이클을 관리하는 오케스트레이션 기능의 중요성이 강조되고 있다. 본고에서는 오픈스택 진영에서 템플릿 기반의 오케스트레이션 서비스를 담당하고 있는 Heat의 기본구조와 함께 Heat에서 정의하고 있는 HOT(Heat Orchestration Template)의 구성 요소와 주요 특징을 정리하고 템플릿 간에 참조 관계를 설명하는 Nested Stack 개념을 소개한다.

Ⅰ. 서론

오픈스택(OpenStack)은 클라우드 인프라를 손쉽게 구축할 수 있는 플랫폼으로 2010년에 Rackspace와 NASA의 오픈소스 프로젝트로 시작되어 현재 전 세계 클라우드 업계에서 가장 빠르게 세력을 팽창하고 있는 클라우드 플랫폼 프로젝트이다[1]. 오픈스택은 그동안 컴퓨팅이나 네트워크, 스토리지 등의 기본적인 기능을 중심으로 구현되어 왔지만 최근 들어 과금이나 오케스트레이션, 대시보드 등의 기능들을 추가하면서 기술적인 변곡점을 맞이하고 있다. 특히, 오케스트레이션은 많은 기업들이 오픈스택 기반의 소프트웨어 정의 데이터센터를 구축하기 위한 기반 기술로써 고려되고 있다[2].

오케스트레이션이란 가상 자원들의 구성과 설정을 정해진 방식과 순서에 맞게 자동화하는 로직을 통해 복잡한 인프라의 생성 및 관리를 단순화하고 효율적으로 운영하도록 하는 기술이다. Heat는 템플릿 기반의 오케스트레이션 엔진을 구현하기 위한 오픈스택 프로젝트로 2012년부터 실험적 프로젝트로 진행되어 왔고 오픈스택 Havana 버전부터 공식적인 프로젝트로 배포판에 포함되기 시작했다[3]. 현재 Heat 프로젝트는 Red Hat을 중심으로 Mirantis, Rackspace 등이 적극적으로 참여하고 있다[(그림 1) 참조][4].



(그림 1) 회사별 기여도

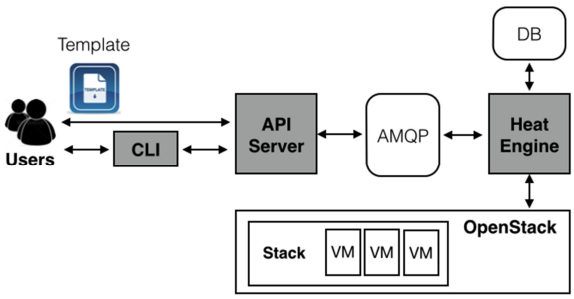
〈출처〉: STACKALYTICS, "Heat Contribution," 2015.

초기의 실험적 프로젝트에서 Heat는 아마존 웹서비스(AWS: Amazon Web Service)의 오케스트레이션 서비스인 클라우드포메이션(CFN: CloudFormation)[5]을 벤치마킹하여 CFN 템플릿을 기반으로 하는 오케스트레이션 엔진을 개발하였다. 하지만, 공식적인 프로젝트로 진행되면서 HOT(Heat Orchestration Template)라는 자체 템플릿을 정의하였고 점진적으로 클라우드포메이션의 색깔을 벗고 독자적인 오케스트레이션 서비스를 제공하는 방식으로 발전해 가고 있다. 본 고에서는 Heat 구조에 대한 간략한 소개와 함께 오케스트레이션 서비스의 중심에 있는 HOT 템플릿의 구조 및 구성 요소들을 정리하고 템플릿 간의 참조 관계를 설명하는 Nested stack 개념을 소개한다.

Ⅱ. Heat 구조

Heat는 여타의 오픈스택 프로젝트와 마찬가지로 Python 언어로 구현되어 있으며 CLI(Command Line Interface) 및 API(Application Programming Interface) 서버와 같은 사용자 인터페이스와 템플릿을 기반으로 가상 자원들을 구성하고 설정하는 로직인 Heat Engine으로 구성된다. (그림 2)는 Heat의 기본 구조를 도시한다.

사용자는 API를 직접 호출하여 오케스트레이션 서비스를 요청하거나 CLI를 통해 동일한 작업을 수행할 수 있다. CLI 역시 사용자 명령을 API로 변환하여 처리하게 된다. 현재 Heat API 서버에서 지원하는 API의 종류는 두 가지로 하나는 아마존 웹서비스의 클라우드포메



(그림 2) Heat 기본 구조

이션 API이고, 또 하나는 Heat에서 독자적으로 정의한 API이다. 두 가지 API 모두 HTTP 프로토콜을 활용하지만 클라우드포메이션 API는 Query API 방식이며 Heat API는 Restful API 방식을 따른다. 향후 Heat에서는 클라우드포메이션 API는 배제하고 Heat API만을 지원할 것이라고 한다. CLI 및 API 호출을 통해 전달된 사용자의 요청 데이터 및 템플릿은 API 서버를 통해 수신되고 API 서버는 AMQP(Advanced Message Queueing Protocol) 규격 기반의 메시지 통신을 통해 Heat Engine으로 이를 전달한다. Heat에서 AMPQ 기반 메시지 통신은 오픈소스인 RabbitMQ[6]를 활용하고 있다. Heat Engine은 수신한 템플릿과 API의 파라미터에 포함된 사용자 요청 데이터를 기반으로 인프라 및 클라우드 애플리케이션을 생성하기 위하여 Nova API, Neutron API 등과 같은 오픈스택 프로젝트들의 API를 호출한다. 이때 Heat Engine은 인프라 및 클라우드 애플리케이션을 구성하는 가상 자원들 간의 상호 의존 관계를 파악하고 어떠한 가상 자원을 먼저 생성해야 할지 의존성 그래프(dependency graph)를 도출한 후 이를 기반으로 오픈스택 프로젝트 API의 호출 순서를 파악한다.

Ⅲ. Heat Orchestartion Template

Heat는 템플릿 기반의 오케스트레이션 서비스를 제공하는 프로젝트로 사용자가 구성하고자 할 가상 자원들을 템플릿에 명시하도록 한다. 템플릿은 JSON(Java-Script Object Notation)이나 YAML(YAML Ain't Mark-up Language)과 같이 사람이 쉽게 읽고 작성할 수 있는 텍스트 기반의 데이터 인코딩 방식을 이용한다. 아마존 웹서비스의 CFN 템플릿이 JSON 기반의 표현 방식을 사용하는 반면 HOT 템플릿은 YAML을 인코딩 방식으로 사용한다. YAML 방식이 JSON 방식에 비해 파싱(parsing)의 부하가 좀 더 크지만 JSON과 달리 멀티라인 스트링을 지원하고 주석 처리가 용이하다는 장점을 갖는다. 본 장에서는 HOT 템플릿에 대한 구조와 간략한 예제 템플릿을 소개한다.

1. HOT 템플릿 구조

HOT 템플릿은 (그림 3)과 같이 두 개의 필수 구성 요소인 heat_template_version 필드와 resources 섹션, 그리고 네 개의 선택적 구성 요소인 description 필드, parameter_groups, parameters, outputs 섹션으로 구성된다[7].

Heat_template_version 필드는 해당 템플릿이 어떤 버전인지 명시한다. 버전은 날짜 형태로 표현되며 현재 '2013-05-23'과 '2014-10-16' 같이 두 가지 버전이 정의되어 있다. '2013-05-23' 버전은 오픈스택의 Ice-house 버전까지 구현된 사항들을 포함하며 템플릿 내에서 Heat 자체 Intrinsic Functions과 함께 일부 CFN intrinsic function을 지원한다(Intrinsic Function은 3절에서 자세히 설명한다). '2014-10-16' 버전은 오픈스택 Juno 버전까지 추가되거나 삭제된 사항들을 포함하고 Fn::Select를 제외한 모든 CFN Intrinsic Function들을 지원하지 않고 Heat 자체 Intrinsic Function만을 지원한다.

Resources 섹션은 가상 자원들을 정의하는 부분으로 HOT 템플릿에는 적어도 하나의 가상 자원이 Re-sources 섹션에 정의되어야 한다. Resources 섹션의 세부 구조는 다음 절에서 소개한다. Description 필드는

```
heat_template_version:
description:
parameter_groups:
parameters:
    set of parameters
resources:
    set of resources
outputs:
    set of outputs
```

(그림 3) HOT 템플릿 구조

템플릿에 대해 소개하는 부분이다. Parameter_groups 섹션은 사용자가 템플릿에 사용되는 일련의 파라미터들의 입력 순서를 정의하고 관련성 있는 파라미터들을 그룹으로 정의하기 위한 용도로 사용된다. Parameter_groups은 해당 그룹을 지칭하는 라벨과 설명 부분 그리고 해당 그룹에 포함된 파라미터들의 목록으로 정의된다. Parameters 섹션은 템플릿에서 사용되는 파라미터들의 유형(e.g. 문자열(string)) 및 의미, 기본값들을 명시한다. Outputs 섹션은 템플릿이 실행된 후, 그 결과값으로 제공할 정보들을 명시하는 부분이다. Out-puts 섹션에서 정의된 결과값은 'show stack' Heat API 호출을 통해 사용자에게 제공된다. Outputs 섹션에서 결과값으로 지정할 수 있는 정보는 생성된 가상 자원들의 속성(attribute)들로 각 가상 자원들의 속성값에 대한 정의는 웹상에 공지된 OpenStack Resource Types[7]을 참조한다.

2. HOT Resources 섹션 구조

가상 자원들은 HOT 템플릿 Resources 섹션에서 두 개의 필수 정보(resource ID, type)와 네 개의 선택적 정보(properties, metadata, depends_on, update_policy, deletion_policy)를 통해 정의된다.

(그림 4)의 Resources 섹션 구조에서 보듯이 가상 자원의 정의는 Resource ID를 명시하는 것으로부터 비롯

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
    metadata:
      <resource specific metadata>
    depends_on: <resource ID or list of ID>
    update_policy: <update policy>
    deletion_policy: <deletion policy>
```

(그림 4) resource 섹션 구조

된다. Resource ID는 가상 자원을 식별하기 위한 용도로 템플릿 내에서 고유한 값을 가져야 한다. Type은 정의하고자 하는 가상 자원의 유형을 지정하는 부분이다. 예를 들어, 컴퓨팅 유형의 가상 자원을 정의하기 위해서는 'OS::Nova::Server'라는 가상 자원 유형을 지정하게 된다. 실제 가상 자원들은 오픈스택의 여러 하부 프로젝트에서 각각 구현된 것들로 가상 자원 유형의 표현은 'OS::프로젝트::가상자원'과 같은 형태로 표기된다(여기서 OS는 OpenStack을 의미함). (그림 5)는 오픈스택 Juno 버전까지 정의된 67개의 가상 자원들의 목록으로 굵게 표기된 가상 자원들은 Juno 버전에서 새롭게 추가된 자원들이다.

Properties는 가상 자원의 특성들을 구체적으로 설정하기 위한 부분이다. 선택된 가상 자원의 유형별로 설정할 수 있는 특성들의 목록은 정의되어 있다. 예를 들어, 'OS::Nova::Server' 가상 자원 유형을 선택한 경우, 서버의 운영체제로 사용할 이미지(image)는 설정해야 할 하나의 특성이 되고 사용할 이미지의 ID가 해당 특성의 값이 된다. 오픈스택 가상 자원별로 정의된 특성(Properties)들의 정보는 웹상에 공지된 OpenStack Resource Types[8]을 참조한다. 메타데이터는 상기 지정된 가상 자원들이 생성된 이후 가상 자원에 의해 사용될 메타데이터를 정의하는 부분이다. Depends_on은 가상 자원들 간에 의존성을 명시적으로 정의하기 위한 정보로 정의하고 있는 가상 자원이 다른 특정 가상 자원에 의존한다면, 해당 특정 가상 자원의 Resource ID를 Depends_on의 값으로 명시하면 된다. Update_policy는 가상 자원의 업데이트 정책을 표현하는 정보로 Heat Engine이 특정 가상 자원의 업데이트를 어떻게 처리해야 할지를 정의하는 부분이다. 하지만, 아직 오프스택의 가상 자원들 중에는 Update_policy를 사용하는 자원은 없다. 다만, 아마존 웹서비스의 AWS::AutoScaling::AutoScalingGroup 가상 자원에 한정되어 사용되고 있다. Deletion_policy는 해당 가상 자원이 삭제되는

OS::Barbican::Order	OS::Heat::StructuredConfig	OS::Neutron::Pool
OS::Barbican::Secret	OS::Heat::StructuredDeployment	OS::Neutron::PoolMember
OS::Ceilometer::Alarm	OS::Heat::StructuredDeployments	OS::Neutron::Port
OS::Ceilometer::CombinationAlarm	OS::Heat::SwiftSignal	OS::Neutron::ProviderNet
OS::Cinder::Volume	OS::Heat::SwiftSignalHandle	OS::Neutron::Router
OS::Cinder::VolumeAttachment	OS::Heat::UpdateWaitConditionHandle	OS::Neutron::RouterGateway
OS::Cinder::VolumeType	OS::Heat::WaitCondition	OS::Neutron::RouterInterface
OS::Glance::Image	OS::Heat::WaitConditionHandle	OS::Neutron::SecurityGroup
OS::Heat::AccessPolicy	OS::Neutron::ExtraRoute	OS::Neutron::Subnet
OS::Heat::AutoScalingGroup	OS::Neutron::Firewall	OS::Neutron::VPNService
OS::Heat::CloudConfig	OS::Neutron::FirewallPolicy	OS::Nova::Flavor
OS::Heat::CWLiteAlarm	OS::Neutron::FirewallRule	OS::Nova::FloatingIP
OS::Heat::HARestarter	OS::Neutron::FloatingIP	OS::Nova::FloatingIPAssociation
OS::Heat::InstanceGroup	OS::Neutron::FloatingIPAssociation	OS::Nova::KeyPair
OS::Heat::MultipartMime	OS::Neutron::HealthMonitor	OS::Nova::Server
OS::Heat::RandomString	OS::Neutron::IKEPolicy	OS::Nova::ServerGroup
OS::Heat::ResourceGroup	OS::Neutron::IPsecPolicy	OS::Sahara::Cluster
OS::Heat::ScalingPolicy	OS::Neutron::IPsecSiteConnection	OS::Sahara::ClusterTemplate
OS::Heat::SoftwareComponent	OS::Neutron::LoadBalancer	OS::Sahara::NodeGroupTemplate
OS::Heat::SoftwareConfig	OS::Neutron::MeteringLabel	OS::Swift::Container
OS::Heat::SoftwareDeployment	OS::Neutron::MeteringRule	OS::Trove::Instance
OS::Heat::SoftwareDeployments	OS::Neutron::Net	
OS::Heat::Stack	OS::Neutron::NetworkGateway	

(그림 5) OpenStack Resource Type(OpenStack Juno 버전 기준)

경우에 적용될 정책을 나타내는 정보로 'Delete', 'Retain', 'Snapshot'의 세가지 정책 중에 하나를 선택하게 된다. 예를 들어, 'Deletion_policy', 'Retain'과 같이 정의된 경우에는 해당 가상 자원을 포함한 스택(stack)이 삭제되는 경우에도 해당 가상 자원의 콘텐츠는 삭제되지 않고 유지하게 된다.

3. Intrinsic Functions

Heat에서 템플릿을 활용한다는 것은 기존에 정의된 템플릿을 재활용할 수 있다는 의미를 갖는다. 따라서 템플릿에는 주로 정적인 정보를 명시하고 사용 목적에 따라 필요한 정보들은 사용자별 입력을 요구한다. 또한, 가상 자원들이 실제 생성되기 이전에 가용하지 못한 값들은 템플릿 내부에서 동적으로 참조되어야 한다. HOT 템플릿은 이렇게 동적으로 요구되는 정보를 처리하기 위한 내부함수를 정의하는데 이러한 내부함수를 'Intrinsic Function'이라고 한다. 오픈스택 Juno 버전까지 정의된 Intrinsic Function은 총 8개로 다음과 같다.

가. Get_Attr

템플릿이 실행된 이후 생성된 가상 자원의 속성값을 얻기 위해 사용되는 내부함수로 해당 가상 자원의 Resource ID와 원하는 속성명을 명시하는 방식으로 사용된다. 주로 HOT 템플릿의 outputs 섹션에서 생성된 가상 자원의 결과값을 도출하기 위해 사용된다.

나. Get_File

템플릿에 외부 파일을 참조하기 위한 내부함수이다. 파일은 URL(Uniform Resources Locator)이나 파일경로를 통해 참조되며 참조된 파일은 Heat API를 호출하는 모듈(i.e., python-heatclient)에 의해 API 'Files' 파라미터에 담겨 전달된다.

다. Get_Param

템플릿의 입력 파라미터를 참조하기 위한 내부함수이다. 본 내부함수를 통해 명시된 파라미터들의 값은 템플릿이 실행될 시에 입력되어야 한다. 예를 들어 템플릿에서 OS::Nova::Server 유형의 가상 자원이 갖는 특성(properties)들 중 flavor값을 get_param로 참조한 경우에는 해당 템플릿이 실행될 때, 사용자가 flavor값을 입력해야 한다.

라. Get_Resource

템플릿 내에서 다른 가상 자원을 참조하기 위해 사용하는 내부함수이다. 참조할 가상 자원은 동일한 템플릿 내에 정의된 자원으로 한정하며 참조 대상의 해당 Resource ID를 이용하여 참조한다.

마. List_Join

오픈스택 Juno 버전에서 처음 소개된 내부함수로 '2014-10-16' 버전의 HOT 템플릿부터 적용된다. 본 내부함수는 지정한 구분문자(delimiter)로 문자열들을 구분하기 위해 사용되는데 예를 들어, list_join: ['-', ['one', 'two', 'and three']]의 경우에는 구분문자 '-'를 사용하여 주어진 문자열 목록을 'one- two-and three' 형태로 표현하게 된다.

바. Resourc_Facade

Provider template에 정의된 가상 자원이 Parent template에 정의된 가상 자원의 데이터를 찾아 가져오기 위한 내부함수이다. Provider template과 paraent template에 대한 자세한 내용은 IV장에서 소개한다.

사. Str_Replace

원하는 문자열을 동적으로 생성하기 위해 사용하는 내부함수이다. 사용 방법은 동적으로 입력 받아야 할 파라미터를 params 변수에 할당하고 상기 파라미터를 포함한 문자열을 template 변수에 입력한다. 예를 들어, (그림 6)과 같이 동적으로 할당되는 서버의 IP 주소를 포함한 URL를 만들기 위해서는 상기 서버의 IP 주소를 'host'라는 이름의 파라미터로 정의하고 이를 params 변수에 할당한 후 상기 'host' 파라미터를 포함한 문자열 'http://host/MyApplication'을 template 변수에 입력하면 된다.

```
outputs:
  Login_URL:
    description: The URL to log into the deployed application
    value:
      str_replace:
        template: http://host/MyApplication
        params:
          host: { get_attr: [ my_instance, first_address ] }
```

(그림 6) Str Replace 내부함수 예제

아. FN::Select

FN::Select 내부함수는 아마존 웹서비스의 CFN 템플릿에서 사용되던 함수로 인덱스값을 이용하여 객체 목록들 중에서 한 객체를 반환하기 위한 함수이다. 예를 들어, {'Fn::Select' : ['1', ['apples', 'grapes', 'oranges', 'mangoes']]} 의 경우, 'apples', 'grapes', 'oranges', 'mangoes'라는 객체 목록들 중에서 인덱스값 1에 해당하는 'grapes'를 반환하게 된다. 참고로 목록에 대한 인덱스값으로 서수를 사용하는 경우에 인덱스값의 범위는 0부터 n-1의 값을 갖게 된다(n은 객체의 개수). 반면 객체 목록이 name: value 형태의 목록으로 표현되는 경우는 인덱스 값으로 서수가 아닌 name을 지정할 수 있다. 예를 들어, OS::Neutron::Pool 가상 자원의 특성(properties) 중 하나인 vip는 name:value 형태의 목록으로 정의되는데 vip의 목록들 중에서 'port_id' name에 해당하는 value를 반환하기 위해서 FN::Select 내부함

수를 사용하면 다음과 같다.

{'Fn::Select': ['port_id', {get_attr: [pool, vip]}]}.

4. HOT 템플릿 예제

(그림 7)은 가상 자원으로 컴퓨팅 자원과 네트워크 포트 자원을 생성하는 HOT 템플릿을 보여준다.

상기 HOT 템플릿은 '2013-05-23' 버전의 템플릿이며 총 5개 파라미터(name, instance_type, private_net_id, private_subent_id, image_id)를 정의하고 있다. 5개의 파라미터는 'General parameters'라는 이름의 파라미터 그룹으로 분류하고 있고 사용자의 파라미터 입력 순서를 name, image_id, instance_type, private_net_id, private_subent_id 순서로 정의하고 있다. Resources 섹션에 생성할 가상 자원으로는 서버와 네트워크 포트 자

```
heat_template_version: 2013-05-23
description: Template (HOT) example
parameter_groups:
    - label: General parameters
      description: General OpenStack parameters
      parameters:
      - name
      - image_id
      - instance_type
      - private_net_id
      - private_subnet_id
parameters:
    name:
        type: string
        description: sever name
    image_id:
        type: string
        description: ID of the image to use for the instance to be created.
        default: centos-6.5-x86_64-cfntools
    instance_type:
        type: string
        label: Instance Type
        description: Type of instance (flavor) to be used
        default: m1.medium
    private_net_id:
        type: string
        description: ID of private network into which servers get deployed
    private_subnet_id:
        type: string
        description: ID of private sub network into which servers get deployed
resources:
    nova_instance:
        type: OS::Nova::Server
        properties:
          name: { get_parm: name }
          image: { get_param: image_id }
          flavor: { get_param: instance_type }
          key_name: { get_param: key_name }
          networks:
            - port: { get_resource: nova_instance_port }
    nova_instance_port:
        type: OS::Neutron::Port
        properties:
          network_id: { get_param: private_net_id }
          fixed_ips:
            - subnet_id: { get_param: private_subnet_id }
outputs:
    nova_instance_private_ip:
        description: IP address of server1 in private network
        value: { get_attr: [ nova_instance, first_address ] }
```

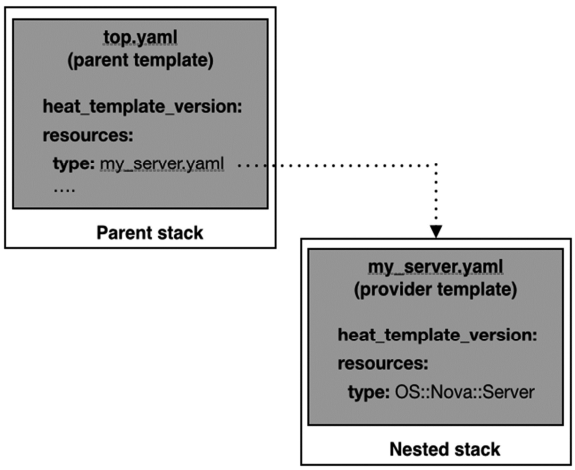
(그림 7) HOT Template 예제

원을 정의하고 있으며 각각의 자원 유형 및 특성값들을 명시하고 있다. 상기 가상 자원들의 특성값들은 Intrinsice Function을 이용하여 템플릿 실행 시에 사용자의 입력을 받거나 동적으로 다른 자원들을 참조하도록 하고 있다. 템플릿 실행 후에 받을 결과값으로는 'get_attr' Intrinsic Function을 이용하여 서버에 할당된 사설 IP 주소를 정의하고 있음을 알 수 있다.

Ⅳ. Nested Stack

Heat에서 템플릿을 통해 생성된 가상 자원들의 집합을 스택(stack)이라고 부르며 스택이 별도의 스택을 참조하는 경우에 참조한 스택을 'Parent stack'이라 하고 참조된 스택을 'Nested stack'이라 한다. 그리고 Parent stack을 정의한 템플릿을 'Parent template', Nested stack을 정의한 템플릿을 'Provider template'이라 한다.

Nested stack을 통해서 사용자가 정의한 가상 자원들의 구성을 사용자 정의형 가상 자원으로 활용할 수 있다. 즉, 가상 자원들의 복잡한 구성으로 이루어진 구조를 하나의 논리적 그룹으로 정의하고 이를 새로운 가상 자원 유형으로 활용할 수 있다. (그림 8)을 참조하면 'top.yaml' 템플릿에서 가상 자원을 정의할 때 가상 자원의 유형으로 'my_server.yaml'을 지정하고 있다. 따라서, top.yaml 템플릿은 Parent template이며 이를 통해 생성된 스택은 Parent stack이 된다. 또한 my_server.yaml 템플릿은 Provider template이고 이를 통해 생성된 스택은 Nested stack이라 부른다.



(그림 8) Nested Stack 구조

Parent template은 Provider template을 URL이나 파일경로 정보를 통해 참조할 수 있고 만일 해당 Provider template을 새로운 가상 자원 유형(e.g. My::Custom::Server)으로 환경변수에 등록하게 되면 resources 섹션의 type에서 상기 등록된 사용자 정의형 가상 자원 유형을 지정하면 된다. Provider template에 명시된 parameters와 outputs 섹션은 각각 Parent template에 해당하는 사용자 정의 가상 자원의 특성(properties)과 속성(attribute)으로 대응된다.

Ⅴ. 결론

가상 인프라 구축을 통해 더 쉬운 관리와 낮은 유지비용을 기대하지만 오히려 관리가 복잡해지는 아이러니한 상황이 발생하곤 한다. 가상 머신의 생성이 너무 간단하고 편리하여 가상 머신의 수가 부지불식간에 폭증하는 경우가 있기 때문이다. 이것이 바로 오케스트레이션 기능이 가상 인프라 구축을 위해 필수적으로 요구되는 이유이다. 이러한 측면에서 Heat는 오픈스택의 일반 사용자보다 기업 사용자들에게 더 많은 가치를 전해주는 기술로 발전해갈 것으로 기대된다.

용어해설

CFN 사용자들이 아마존 웹서비스(AWS) 관련 자원들을 정해진 순서와 방식에 맞게 쉽게 생성할 수 있도록 제공하는 AWS의 오케스트레이션 서비스임.

스택(Stack) 템플릿을 통해 생성된 가상 자원들의 집합

약어 정리

AMQP	Advanced Message Queueing Protocol
API	Application Programming Interface
AWS	Amazon Web Service

CFN	CloudFomration
CLI	Command Line Interface
HOT	Heat Orchestration Template
JSON	JavaScript Object Notation
URL	Uniform Resources Locator
YAML	YAML Ain't Markup Language

참고문헌

[1] OpenStack Project, http://www.openstack.org

[2] 디지털데일리, "대기업 클라우드 표준 노리는 오픈스택," 2013. 11. 11.

[3] Welcome to the Heat Developer Documentation, http://docs.openstack.org/developer/heat

[4] Stackalytics, http://stackalytics.com/?metric=marks&project_type=openstack&release=all&module=heat-group

[5] AWS CloudFormation, http://aws.amazon.com/cloudformation/

[6] RabbitMQ, http://www.rabbitmq.com

[7] Heat Orchestration Template (HOT) Specification, http://docs.openstack.org/developer/heat/template_guide/hot_spec.html

[8] OpenStack Resource Types, http://docs.openstack.org/developer/heat/template_guide/openstack.html