특집 02

클라우드 컴퓨팅 효과를 최대화하는 클라우드 네이티브

임철홍 (광주대학교), 전미향 (수원여자대학교), 한성수 (강원대학교)

목 차

1. 서 론

클라우드 컴퓨팅은 내부적으로 연결된 컴퓨팅 자원 또는 가상화된 컴퓨팅 자원의 조합으로 이루어져진 분산, 병렬 시스템의 일종으로서, 사용자들에게 가상화된 자원을 서비스 형태로 공급하는 컴퓨팅 패러다임이다. 이러한 클라우드 컴퓨팅은 다양한 IT 분야에서 적용되고 있으나, 가상머신 등의 인프라 및 플랫폼 영역에 한정되어 있다. 클라우드 컴퓨팅의 장점을 모두 활용하기 위해서는 애플리케이션의 실행 환경부터 구성까지 전반적인 변화가 필요하다. 애플리케이션은 하나의 구조로 구성된 Monolithic에서 작은 기능 단위로 구분되어 구성하는 마이크로 서비스 아키텍처로 구성되며, 애플리케이션 가상화 환경인 Container로 구성되어 실행되고, 개발, 테스트, 운영에 이르는 과정이 자동화되어 지속적으로 통합되고 배포될 수 있는 환경이다. 클라우드 네이티브 애플리케이션으로의 변경은 기존 환경에 많은 변화가 필요하므로 기대되는 장점이 있지만, 적용에는 어려움이 있는 것이 현실이다. 클라우드 네이티브 애플리케이션의 전환을 통해 애플리케이션 환경은 더욱 유연하고 안정적으로 변화되며, 비즈니스 변화에 더욱 빠르게 대응할 수 있도록 개발에서 운영으로의 빠른 통합과 배포가 가능해질 것이다.

2. 클라우드 네이티브 개요

클라우드 네이티브는 클라우드 컴퓨팅 기술의 장점을 모두 활용하는 애플리케이션을 개발하고 실행하는 방법을 말한다[1]. 애플리케이션을 한 가지 업무에 특화된 작은 개별 단위인 마이크로 서비스(Micro-Service)로 개발하고, 이를 컨테이너 가상화 환경에서 생성하고 관리한다. 이러한 구성 방식을 MSA(Micro Service Architecture)

라고 한다. 클라우드 네이티브 애플리케이션은 소규모의 전담팀들이 플랫폼에 빠른 주기 내에 구현하고 배포할 수 있도록 기능을 제공하여 확장하기가 쉽고 변화에 더욱 빠르게 대응할 수 있다.

클라우드 플랫폼 비즈니스가 확대됨에 따라 대형 또는 복잡한 애플리케이션 간의 협업과 통합이 중요하게 되었다. 또한, 비즈니스 변화에 신속한 대응이 필요하게 되었다, 새로운 요구사항

〈표 1〉 클라우드 네이티브 애플리케이션의 장점

장점	설명
경쟁우위 확보	고객의 요구에 부응해 신속하게 애플리케이션을 구현하고 전달할 수 있는 환경이 업계의 경쟁력이 될 것이다. IT 서비스의 목표를 비용 절감에서 비즈니스 성장엔진으로 바꾼다.
유연성	비즈니스 우선 순위를 맞추고 가격을 최적화하기 위해 다양한 클라우드 업체들에 마이그레이션 하거나 배포할 능력을 유지하게 된다.
개발자들에게 최선의 동기 부여	개발자들은 여러 클라우드에서 실행되고 크기를 조정하기 위해 코드를 작성하는 오버헤드를 차단하고 고객가치를 전할 코드 작성에 집중할 수 있다.
IT운영과 비즈니스 정렬	IT운영을 자동화함으로써 비즈니스 우선순위가 높은 곳에 집중할 수 있다. 운영자의 실수에 따른 오류 위험을 제거함에 따라 직원들은 반복적인 작업 대신 생산성과 비즈니스 가치를 높이는 프로세스 개선에 집중할 수 있다.

〈표 2〉 기존의 애플리케이션과 클라우드 네이티브 애플리케이션 비교

클라우드 네이티브 애플리케이션	기존 엔터프라이즈 애플리케이션
(예측가능) 작은 단위의 서비스를 자동화된 체계를 통해 배포하여 오류 가능성을 줄임	(예측불가능) 빌드시간이 오래 걸리고 큰 용량의 실행파일을 배포하여 예상하지 못한 오류발생 가능성이 큼
(다양한 기술) 서비스 단위별로 적합한 언어 및 미들웨어를 활용하여 최적화된 아키텍처를 구성 (Ployglot)	(단일기술기반) 하나의 언어 또는 미들웨어에 의해 아키텍처 구성
(적정용량) 마이크로 서비스 단위로 구성되어 필요한 기능 단위만 용량을 확장할 수 있어 효율적으로 자원을 활용	(용량 비효율) 하나의 거대한 실행파일 형태인 모놀리틱 형태로 구성되어 특정 기능만 용량을 확장할 수 없고 전체를 확장해야 함
(공동작업) 개발자와 운영자간의 긴밀한 협업환경	(사일로 방식) 개발자가 완성한 코드를 운영자에게 이관하여 실행
(지속적인 전달) 지속적으로 통합되고 배포되는 환경 제공	(폭포수형 개발) 분석설계, 개발, 테스트 후에 릴리즈 과정을 거쳐서 배포
(독립성) 애플리케이션이 느슨하게 결합 되어 다른 서비스에 영향을 주지 않고 독립적으로 관리 운영	(종속성) 많은 이질적인 서비스들을 하나의 배포 패키지로 구성
(자동화된 확장성) 자원의 상태와 사용량에 따라 자동으로 동적 자원 할당	(수동 크기 조정) 운영자에 의해서 조정
(빠른 복구) 오류가 발생된 기능 단위만 수정하고 빌드 배포하면 되므로 대응이 빠름	(느린복구) 오류 수정 후에 전체 애플리케이션을 통합하고 빌드·배포 과정을 거쳐야 되므로 대응이 느림

에 대해 빠르게 대응하면서 유연하고 신속한 확장성이 요구되었다. 장애를 사전 예방하고 발생 시에는 신속하게 대응할 수 있는 능력이 중요하게 되었다. 이러한 요구사항을 해결하기 위해 클라우드 네이티브는 매우 중요한 기술이다.

클라우드 네이티브 애플리케이션은 기업간 경쟁우위를 확보할 수 있으며, 기술과 비즈니스 변화에 유연하고 빠르게 대응할 수 있다. 개발자들에게는 최선의 동기 부여를 할 수 있다. 클라우드 네이티브 애플리케이션의 장점은 다음과 같다.

기존 애플리케이션은 모든 기능이 하나의 배포단위로 구성되어 개발과 운영이 매우 복잡하였으나, 클라우드 네이티브 애플리케이션은 작은 기능 단위인 마이크로 서비스로 개발 운영되기 때문에 효율적으로 관리할 수 있으며 독립적으로 운영이 되어 다른 서비스에 대한 영향성을 최소화 할 수 있다. 또한 필요한 기능만 확장하면 되기 때문에 확장의 효율성을 높일 수 있다.

3. 클라우드 네이티브 요소기술

클라우드 컴퓨팅의 장점을 활용하기 위해서 애플리케이션의 실행은 가상머신과 Container를 기반으로 동작하며, 많은 Container를 효율적으로 운영하기 위한 Orchestration 환경을 제공해야 한다. 애플리케이션의 개발, 테스트, 운영에 이르는 과정을 자동화하고 지속적인 통합과 배포를 통하여 유연하고 빠른 애플리케이션 아키텍처를 지원해야 한다. 애플리케이션은 기능 단위의 구성을 통하여 느슨하게 연결되고 독립적인 마이크로 서비스 기반으로 구성되고 운영되는 것이 클라우드 네이티브 환경에서 필요한 기술이다. 다음 그림은 클라우드 네이티브 환경을 구성하는 요소기술로 구성되는 참조 아키텍처이다[2].

(그림 1) 클라우드 네이티브 참조 아키텍처

클라우드 네이티브 환경을 구성하는 Application definition / Development, Orchestration & Management, Runtime, Provisioning, Ingrastructure 기술의 설명은 다음과 같다.

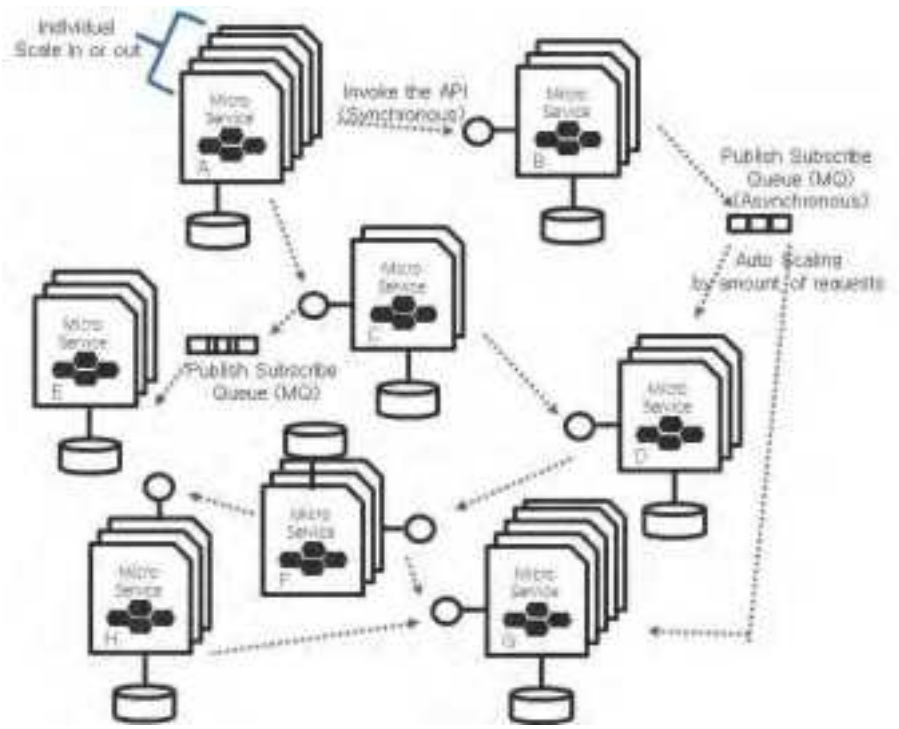
〈표 3〉 클라우드 네이티브 구성 기술

기술요소	설명
Application definition/development	클라우드 네이티브 애플리케이션을 구현하는데 필요한 기술 (MSA 등)
Orchestration & Management	Container 관리 기술 (Kubernetes 등)
Runtime	Container가 동작하고 연계하는 표준 (Docker 등)
Provisioning	Container 환경을 고려하여 개발된 애플리케이션을 배포하고 관리하는 환경 (DevOps 등)
Infrastructure	하드웨어 영역 또는 다른 클라우드와의 연계

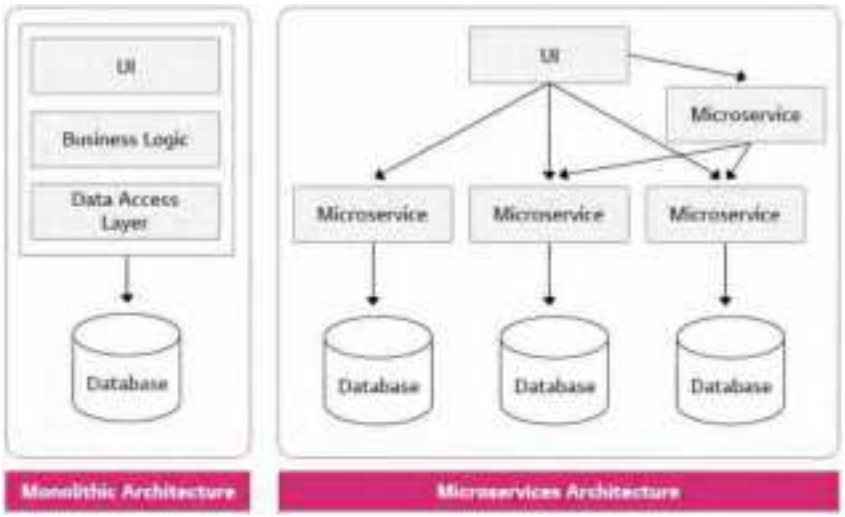
3.1 MSA (Micro Service Architecture)

클라우드 네이티브 애플리케이션은 MSA (Micro Service Architecture)로 실현된다. MSA는 기능 단위의 작은 애플리케이션을 구축하고 이들이 가벼운 통신방식(REST 등)으로 연결되어 동작하는 애플리케이션을 말한다[3]. 기존의 애플리케이션 개발방식은 하나의 배포단위로 구성되어 개발자 전체가 한꺼번에 이를 개발하고 수정하는 방식으로 매우 복잡하고 어려운 작업이다. MSA는 하나의 프로그램 언어나 개발 프레임워크에 국한되어 활용되지 않고, 개별 애플리케이션에 가장 적합한 언어나 개발 프레임워크로 구축되고 이들이 서로 연결하여 전체 애플리케이션을 완성하게 된다.

기능 단위로 분할된 마이크로 서비스들이 서로 연계 동작하여 애플리케이션이 실행된다. 작업의 흐름을 조정하기 위해 메시지큐(MQ)와 같은 미들웨어가 활용되기도 한다. MSA에서는 마이크로 서비스가 동작하고 대기 중인 다른 서비스를 자동으로 실행한다. 이러한 실행방식을 Reactive 또는 Event-Driven 방식이라고 한다. Monolithic 방식은 기존의 전형적인 애플리케이션 구현방식으로 전체 모듈이 하나의 DB와 연동이 되도록 구성되며, 전체 단위로 애플리케이션이 개발되고 배포 운영된다. 이에 비해 MSA는 기능별로 서비스가 나뉘게 되며 분할된 DB와 연동된다. 두 개발방식에 대한 차이는 다음 그림에서 설명하고 있다.



(그림 2) MSA 구성



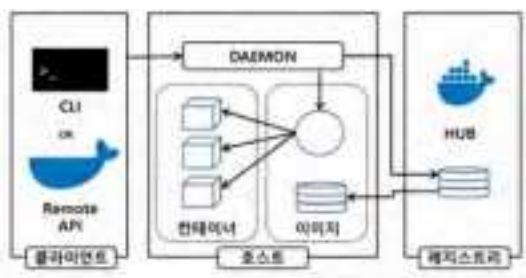
(그림 3) Monolithic과 MSA 비교[4]

3.2 Container 가상화

가상화(Virtualization) 기술은 가상 머신 기반과 리눅스 container 기반의 가상화 기술로 나누어질 수 있다. 가상 머신 기반의 가상화는 호스트(hosted) 방식과 베어메탈 방식을 사용하며, 가상화 방식 모두 게스트 운영체제(OS)의 설치가 필요하다. 이 부분이 리눅스 컨테이너 기반 가상화 기술과 큰 차이를 보여준다.

리눅스 container는 하나의 호스트 운영체제 위에 여러 개의 격리된 시스템 환경을 구성할 수 있는 운영체제 수준의 가상화 기술이다. 리눅스 컨테이너의 컨트롤 그룹은 물리적인 하드웨어의 리소스를 프로세스 그룹 단위로 제어하는 기능이다. 컨트롤 그룹을 통해 사용자는 CPU 시간, 시스템 메모리, 네트워크 대역폭과 같은 자원이나 이러한 자원의 조합을 실행 중인 프로세스 간에 할당하는 것이 가능하다. 게스트 운영체제 유무의 차이로 성능, 배포, 운영 측면의 영향이 달라진다. 게스트 운영체제가 설치되지 않고, 프로세스 레벨의 분리가 가능한 container를 사용하는 가상화 방식은 성능 측면에서 게스트 운영체제로 인한 성능 저하가 발생하지 않기 때문에 경량화 가상화로 분류된다. Container를 생성하는 것은 일반적인 프로세스 실행과 다름없이 빠르게 수행할 수 있다.

Container 기술 기반의 대표적인 가상화기술로 Docker가 있다[5]. Docker는 게스트 운영체제 없이 배포하므로 설치된 이미지 자체의 용량에 따른 네트워크 부하 및 배포 시간을 고려했을 때 큰 장점을 보여준다. 컨테이너 기반의 인프라를 사용하기 때문에 다양한 환경에 대한 종속성을 고민할 필요 없이 운영할 수 있다. Docker는



(그림 4) Docker의 구성과 동작

Docker 클라이언트, Docker 이미지, Docker 데몬, Docker 레지스트리, Docker 네트워크와 Orchestration 및 모니터링이 가능한 에코 시스템으로 구성된다.

3.3 Container Orchestration & Management

클라우드 네이티브 애플리케이션은 실제로 여러 container에 걸쳐있으며 이러한 container는 여러 서버 호스트에 배포되어야 한다. 클라우드 환경은 매우 많은 가상머신으로 구성이 되며, 하나의 가상머신에는 많은 container가 배포되고 운영되어야 한다. 이러한 container의 배포와 협업을 container orchestration이라고 하며, 가장 많이 활용되고 있는 대표적인 플랫폼이 kubernentes이다.

작은 수의 container라면 수동으로 가상머신이나 하드웨어에 직접 배포하면 되지만, 그 수가 많아지고 container의 수가 많아지면, 이 container를 어디에 배포해야 하는지에 대한 결정이 필요하여서 container orchestration이 필요하다. 16 CPU, 32 GB 메모리 머신들에 containr를 배포할 때 사이즈가 2 CPU, 3 CPU, 8 CPU 등 다양할 수 있어서, 자원을 최대한 최적으로 사용하기 위해서 적절한 위치에 배포해야 하고, 애플리케이션 특성들에 따라서, 같은 물리 서버에 배포하거나 가용성을 위해서 일부러 다른 물리서버에 배포되어야 하는 일이 있다. 이렇게 container를 적절한 서버에 배포해주는 역할을 스케쥴링이라고 한다. 또한, container가 정상적으로 작동하고 있는지 체크하고 문제가 있으면 재기동을 해주고, 모니터링, 삭제 관리 등 컨테이너에 대한 종합적인 관리를 해주는 환경이 필요하다.

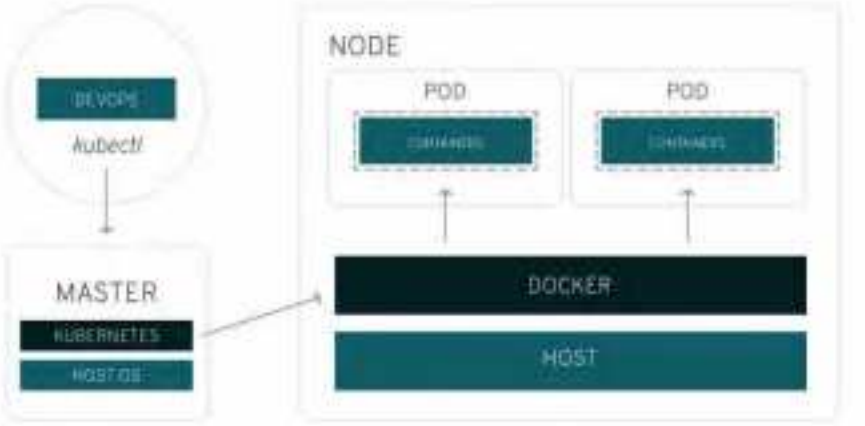
Kubernetes는 규모에 맞는 컨테이너를 배포하는 데 필요한 Orchestration 및 관리 기능을 제공한다. 쿠버네티스 Orchestration을 사용하면 여러 컨테이너에 걸쳐 애플리케이션 서비스를 구축하고 클러스터 전체에서 컨테이너의 일정을 계획하고 이러한 컨테이너를 확장하여 컨테이너의 상태를 지속적으로 관리할 수 있다. Kubernetes는 종합적인 container 인프라를 제공할 수 있도록 네트워킹, 스토리지, 보안, 기타 서비스와 통합해야 한다.

kubernetes가 제공하는 기능은 다음 표와 같다.

Kubernetes는 마스터와 노드로 구성되며, 마스터는 노드를 제어하는 머신으로 여기에서 모든 태스크 할당이 시작된다. 노드는 container가 배포되어 실행되는 시스템이다. 마스터가 할당하는 태스크를 요청대로 수행한다. Container는 포드 단위로 구성되어 실행된다.

〈표 4〉 Kubernetes 제공 기능

기술요소	설명
Container 관리	여러 호스트에 걸쳐 Container를 Orchestration한다. Container 기반 애플리케이션과 해당 리소스를 즉시 확장한다.
애플리케이션 리소스 관리	하드웨어를 최대한 활용하여 엔터프라이즈 애플리케이션을 실행하는 데 필요한 리소스를 극대화한다.
배포 및 제어	애플리케이션 배포 및 업데이트를 제어하고 자동화한다. 자동 배치, 자동 재시작, 자동 복제, 자동 확장을 사용해 애플리케이션 상태 확인과 셀프 복구를 수행한다.



(그림 5) Kubernetes 구성과 동작

3.4 Provisioning

클라우드 환경은 매우 많은 서버, 가상머신, container 들로 구성되기 때문에 애플리케이션의 개발, 빌드, 배포에 이르는 모든 과정이 자동화되지 않으면 활용이 어렵다. 단일 시스템에서 하나의 애플리케이션을 빌드하고 배포하는 기존의 방식에 비해 클라우드 네이티브 애플리케이션은 많은 마이크로 서비스들로 구성이 되며, 수많은 서버 또는 가상머신에 container 형태로 배포되게 된다. PaaS를 활용하는 경우 애플리케이션 코드를 빌드하고 container에 배포하는 과정이 자동화되어 매우 효율적이다. IaaS를 활용하는 경우 하드웨어, 네트워크, 스토리지를 쉽게 활용할 수 있지만, OS, 프레임워크, WAS/DB와 같은 미들웨어, 애플리케이션 빌드 배포 등의 경우는 직접 다루어야 한다. 반면, PaaS를 활용하게 되면 애플리케이션만 개발하면 OS, Java, WAS 등을

(그림 6) PaaS를 활용한 Provisiong 구성[6]

묶어서 container를 생성하고, 생성된 container가 가상머신에 탑재되고 도메인이 할당되어 바로 서비스할 수 있게 된다.

PaaS에 포함되어 있는 Build pack에는 자바환경, 개발프레임워크 및 웹 애플리케이션 구동을 위한 WAS까지 포함하고 있어 애플리케이션과 이들을 모두 묶어서 container로 배포하게 된다. 아래 그림에서 Build pack은 애플리케이션 구성에 필요한 요소를 모두 포함하고 있으며, container들은 가상머신에 배포되어 실행되게 된다. 개발부터 실행 및 운영에 이르는 모든 과정이 자동화되어 실행된다.

4. 클라우드 네이티브 적용 사례

4.1 국내 공공 클라우드 네이티브 인프라 전자정부 클라우드 플랫폼

전자정부 클라우드 플랫폼은 공공정보 서비스 개발에 필요한 전 영역(인프라, 개발·실행환경, 정보자원)을 제공하는 클라우드 환경이다[7]. 전자정부 클라우드 플랫폼이 구축되면 서비스 개발을 위한 인프라, 개발도구, 데이터베이스 등과 함께 서비스 소프트웨어에 대한 배포·관리가 자동으로 가능하다. 빅데이터, 인공지능 등 지능정보기술을 적용하거나 민간에서 개발한 서비스를 전자정부에서 활용할 때도 탄력적으로 활용할 수 있다.

전자정부 클라우드 플랫폼은 전자정부 서비스를 쉽고 편리하게 개발·운영할 수 있는 PaaS 기반의 클라우드 환경을 제공한다. 개발환경에서 구축된 서비스는 쉽게 운영환경으로 이관할 수 있으며, 지속적 서비스 개발을 위한 통합개발지원 서비스를 제공한다. 다중의 사용자가 자신의 전용환경을 활용하는 것처럼 조직간 독립된 서비스 운영을 지원하는 멀티테넌시 환경을 지원한다.

4.2 넷플릭스

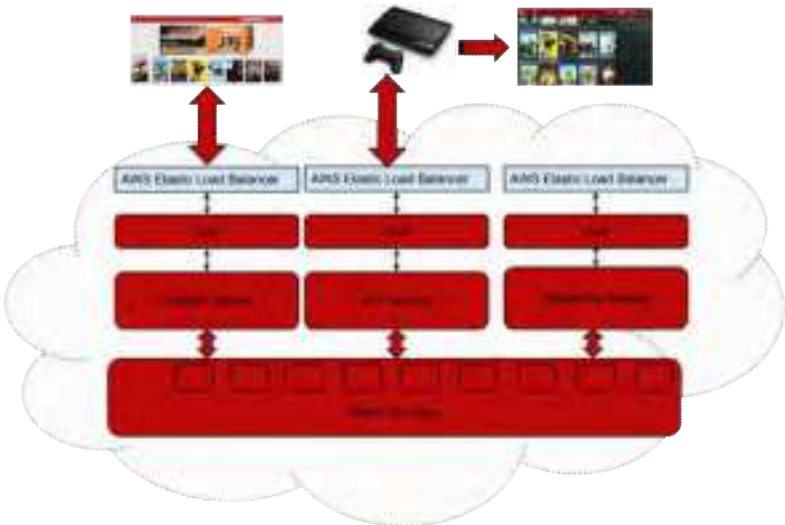
2016년 1월, 세계적인 인터넷 기반 TV 서비스 기업 넷플릭스의 마지막 데이터센터가 문을

(그림 7) 전자정부 클라우드 플랫폼 개념도

닫았다. 넷플릭스의 모든 서비스는 모두 아마존 웹서비스(AWS) 클라우드에서 이루어지고 있다. 넷플릭스는 7년이라는 시간을 들인 끝에 2016년 1월, 스트리밍 서비스를 위한 데이터센터 운영을 중단하면서 사내 모든 컴퓨팅 인프라를 클라우드 환경으로 옮겼다.

넷플릭스는 데이터센터에서 운영한 환경을 고스란히 클라우드 환경으로 옮기기보다는 클라우드 환경에 맞춰 새롭게 컴퓨팅 인프라를 선택하는 법을 택했다. 데이터센터 환경을 클라우드로 그대로 옮기면 마이그레이션 과정은 쉬울지 몰라도 데이터센터 환경에서 겪었던 문제점이 클라우드로까지 고스란히 이어질 수밖에 없다는 이유였다. 그렇기 때문에 넷플릭스는 Cloud Native 방식을 채택하여 클라우드 보안성과 내구성을 오랜 시간을 들여 살피고 회사 운영 방침도 클라우드 환경에 맞게 바꿨다. 그 결과 현재 넷플릭스는 비즈니스 로직, 분산 데이터베이스, 빅데이터 처리 및 분석, 추천, 코드 변환 등 넷플릭스 앱을 구성하는 수백 가지 기능을 처리하는 데 필요한 확장성 있는 컴퓨팅 및 스토리지를 모두 AWS 클라우드로 옮겨 가동 중이다. 2008년 대비 스트리밍 서비스 이용 회원 수가 8배 증가하고, 월간 스트리밍 시간이 무려 1천배 가량 증가했음에도 무리 없이 탄력적으로 시스템을 운영할 수 있게 됐다. 서비스 가용성 역시 대폭 증가했다. 과거의 데이터센터를 이용하던 시절에는 서비스 중단이 발생하는 경우가 빈번했으나, 클라우드 이전 이후 가용성이 꾸준히 증가해 서비스 가용성 목표인 99.99%에도 다가섰다.

넷플릭스의 MSA 구성은 가장 성공적인 사례로 손꼽히고 있고, 많은 회사가 도입 활용하고 있다. 특히 Zuul, Eureka, Config server와 같은 서비스는 Spring Cloud에 포함되어 배포되고 있다. Zuul은 외부에서 접속하는 경로를 내부로 전달해주고 인증 등의 처리를 수행한다. Eureka는 마이크로 서비스를 자동으로 등록하여 다른 서비스에서 접근할 때 필요한 정보를 제공한다. Config server는 분산된 마이크로 서비스들이 서로 설정 정보를 공유할 수 있도록 한다. 설정 정



(그림 8) 넷플릭스 클라우드 MSA 아키텍처[8]

보가 변경되면 각 마이크로 서비스들에 실시간으로 통지가 되고 업데이트된다.

5. 결 론

클라우드의 장점을 모두 활용하여 유연성, 확장성, 안정성을 높일 수 있는 클라우드 네이티브 애플리케이션의 적용을 위해 기존의 애플리케이션 구조의 변화와 Container, PaaS 등 새로운 기술의 적용과 활용이 있어야 한다. MSA의 적용을 통해 애플리케이션은 기능 단위로 분할 하여 연동되어 동작하여 비즈니스 변화에 빠르고 유연하게 대응할 수 있다. Container 기반의 운영을 통하여 애플리케이션은 빠르게 통합, 배포되며 필요한 자원을 실시간으로 확장할 수 있다. PaaS는 이러한 클라우드 네이티브의 중요한 요소기술로 Container의 관리와 함께 애플리케이션의 개발부터 운영에 이르는 과정을 자동화한다. 클라우드 네이티브의 가장 성공한 사례로 넷플릭스가 있다. 넷플릭스의 성공은 기술뿐만 아니라 기업 전체적으로 클라우드 네이티브 환경으로 전환하였기 때문이다. 국내에서도 많은 공공기관과 민간기업에서 클라우드 네이티브의 성공적인 도입과 적용을 위해 노력하고 있다.

참 고 문 헌

[1] CNCF Cloud Native Definition v1.0", https://github.com/cncf/toc/blob/master/DEFINITION.md, 2020-10-13.

[2] Open Container Technologies and OpenStack - Sorting Through Kubernetes, the OCI & the CNCF, Danel Krook.

[3] Newman, Sam. “Microservices” in Building microservices, first ed. Sebastopol: O'Reilly Media, Inc., pp, 1-11, 2015.

[4] 블록을 조립하듯 앱을 조립하는 마이크로서비스, https://blog.lgcns.com/1278, 2020-10-13.

[5] Docker docs, Docker Overview part, Docker architecture, 2019.

[6] 5분만에 보는 PaaS의 기본개념과 특징, SK 주식회사 C&C 블로그

[7] 2020 디지털정부 클라우드 컨퍼런스 발표자료, “전자정부 클라우드 플랫폼을 활용한 개발 및 운영”, 2020.

[8] https://github.com/Netflix/zuul/wiki

저 자 약 력

임 철 홍

이메일 : chim@gwangju.ac.kr

- 1996년 고려대학교 재료공학과 (학사)
- 2003년 고려대학교 전자컴퓨터공학과 (석사)
- 2017년 고려대학교 영상정보처리학과 (박사)
- 2002년~2005년 SK 커뮤니케이션즈 과장
- 2005년~2014년 SK C&C SW공학센터 부장
- 2020년~현재 광주대학교 컴퓨터공학과 교수
- 관심분야 : 에너지 빅데이터, 지능형 영상정보처리, AIaaS, PaaS

전 미 향

이메일 : chic830@naver.com

- 1999년 University of Adelaide Performing art (학사)
- 2011년 한국방송통신대학교 영어영문학과 (학사)
- 2014년 동국대학교 예술경영 (석사)
- 2018년 경기대학교 미디어 커뮤니케이션 (박사)
- 2011년~2018년 안젤라 아트컴퍼니 대표
- 2018년~현재 수원여자대학교 겸임교수
- 관심분야 : 문화 콘텐츠, 글로벌 미디어, 콘텐츠 유통

한 성 수

이메일 : sshan1@kangwon.ac.kr

- 2019년 고려대학교 영상정보처리학과 (박사)
- 2015년~2016년 오리온테크놀리지 이사
- 2018년~2019년 순천향대학교 교수
- 2019년~현재 강원대학교 자유전공학부 교수
- 2020년~현재 한국정보처리학회 상임이사
- 관심분야 : 빅데이터, 분산병렬알고리즘, 영상정보처리, 딥러닝