논문 19-45-01-23 The Journal of Korean Institute of Communications and Information Sciences '20-01 Vol.45 No.01
https://doi.org/10.7840/kics.2020.45.1.201

오픈소스를 활용한 컨테이너 기반 Private PaaS 구현 방법

정 은 태◆, 박 규 동°, 최 백 규*

Container-Based Private PaaS Implementation Using Open Source

Eun-Tae Jung◆, Gyu-Dong Park°, Baek-kyu Choi*

요 약

최근 국방부에서는 국방정보체계의 클라우드 서비스 도입을 일부 상용 및 오픈소스를 활용하여 구현을 시도하고 있다. 그러나 대부분 IaaS(Infrastructure as a Service) 기반의 인프라 제공과 가상머신을 사용한 OS 가상화에 국한되어 있다. 또한 기존의 모놀리식 개발방식을 사용하고 있기 때문에 요구사항에 대한 민첩한 개발과 서비스 제공에는 한계가 있다. 본 논문에서는 국방정보체계에 빠른 응용소프트웨어 제공을 위해 오픈소스를 활용한 컨테이너 기반 Private PaaS 구현 방법을 연구하였다. 구현을 위해 PaaS(Private as a Service) 기능 요건을 정의 한 후 네트워크 가상화, 컨테이너 오케스트레이션, CI/CD 자동화, PaaS 모니터링, 프로젝트 RBAC 접근제어, 폐쇄망에서 라이브러리 의존성 관리 등의 구현방법을 제안하고 테스트베드 구현을 통해 검증하였다.

Key Words : Cloud Computing, Private PaaS, CI, CD, Container Orchestration

ABSTRACT

Recently, the Ministry of Defense is attempting to implement cloud service of defense information system using some commercial and open source. However, most of them are limited to providing IaaS-based infrastructure and virtualizing OS using virtual machines. In addition, there is a limit to agile provision of requirements and services because the existing monolithic development method is used. in this paper, we researched how to implement container-based Private PaaS using open source to provide fast application software for defense information system. After defining the PaaS service functional requirements for the implementation, we proposed implementation methods such as network virtualization, container orchestration, CI / CD automation, PaaS monitoring, project RBAC access control, and library dependency management in private networks, and verified through test bed implementation.

Ⅰ. 서 론

미군은 효율성, 보안성, 민첩성 향상을 목적으로 클라우드 기술을 기반으로 하는 합동정보환경(Joint Information Environment) 구축을 추진하고 있다. 이를 통해 미군은 네트워크 최적화, 공통 자원, 서비스

※ 이 연구는 국방과학연구소의 국방 지휘통제 통합·연동 기반기술 특화연구실 과제의 지원을 받았습니다(UD180015ED).
※ This research was supported by C2 integrating and interfacing technologies laboratory of Agency for Defense Development(UD180015ED).
◆ First Author : DOSANET Research Institute, infowar@dosanet.co.kr, 정회원
° Corresponding Author : Agency for Defense Development, iobject@add.re.kr, 정회원
* DOSANET Research Institute, undernext@dosanet.co.kr
논문번호 : 201910-244-0-SE, Received October 15, 2019; Revised December 3, 2019; Accepted December 8, 2019

및 응용의 통합화, 모니터링과 통제의 중앙화 또는 일원화를 추진하고 있고, 그 중심에 핵심데이터센터(Core Data Center)[1]가 있다. 국내에서도 이러한 최신 기술 적용 및 서비스 환경 개선을 위해 클라우드를 기반으로 지휘통제체계의 구축 및 운용을 위한 정보체계 환경 개선이 요구되고 있다. 또한 응용소프트웨어의 신속한 개발과 배포 등의 운용 효율성 향상이 요구되고 있기 때문에 국방 지휘통제체계가 정보기술의 발전에 적응하고 민첩한 응용 소프트웨어 개발을 위한 기반 환경이 필요한 상황이다. 따라서 본 연구는 우리 군의 기반환경 개선을 위한 통합데이터센터 구축에 필요한 클라우드 PaaS 서비스의 핵심 구성요소를 식별하고, 구현 방안을 제안하고자 한다. 이를 위해 오픈소스를 활용한 컨테이너 기반 Private PaaS 요구기능과 구현기술을 분류하고, 그 구현방법을 제안하였다. 또한 국방정보체계의 폐쇄망 특성을 고려한 효율적인 개발환경 구현과 시스템 갱신 방안을 제안하였다.

Ⅱ.배 경

최근 우리 군은 빅데이터, 사물인터넷(IoT) 등 최신 정보기술 도입을 추진 중에 있으나 빅 데이터 취합과 대량데이터 처리를 위해 필요한 기반환경으로 요구되는 클라우드 도입은 구체화되지 않고 있다. 응용서비스 구축 및 개발방법도 과거 개발 환경과 동일한 모놀리식(Monolith) 방식을 사용하고 있어 JIE을 위한 기반환경이 미흡한 상태이다. 이 같은 정보기술 환경 변화에 대응 할 수 있는 클라우드 컴퓨팅 기반의 통합데이터센터(Integrated Data Center) 확보와 이 기반환경 운용을 위한 클라우드 컴퓨팅 서비스에 대한 기술 확보가 필요하다. 관련하여 국외에선 현재 PaaS(Platform as a Service)에 대해서도 도입이 적극적으로 검토 및 추진되고 있으나, 국내에서는 그에 대한 이해와 필요성에 대한 인식 부족으로 도입이 지연되고 있는 실정이다. 그리고 KVM(Kernel Virtual Machine) 가상머신과 Docker container 간 성능 비교[2]연구에서 평균서버 부팅시간, 정지 상태 평균 CPU 사용률, 가상화 생성 평균사용 메모리, 리부팅 평균 소요시간, 입출력 임의의 읽기 성능 및 입출력 임의의 읽기/쓰기를 측정한 결과 Docker가 KVM보다 모든 상황에서 성능이 높게 측정되었다. 이는 향후 컨테이너 기반의 클라우드 컴퓨팅으로 전환을 의미하고 있다. PaaS 서비스는 단순한 클라우드 기반 앱부터 엔터프라이즈 응용 프로그램에 이르기까지 응용프로그램이 요구하는 것을 제공할 수 있는 리소스가 포함되어 있으며, 클라우드에서 제공되는 개발 및 배포 환경이다. 즉, 미들웨어, 개발 도구, DBMS 등의 플랫폼 제공과 빌드, 테스트, 배포 관리, 업데이트 등의 응용 프로그램 수명 주기를 관리한다. 또한 다양한 개발언어와 DBMS를 사용 가능하면서, PaaS에서 제공하는 플렛폼을 사용하므로 개발 절차, 지속적 통합, 지속적 배포, 테스트 환경을 표준화 할 수 있는 장점이 있다. Private PaaS 구현을 위한 제품으로 Openshift, Cloud Foundry, AppScale, Tsuru, WSO2, Cloudify 등의 오픈소스 소프트웨어 있다. 오픈소스를 활용한 PaaS를 사용하면 소프트웨어 라이선스, 기본 응용 프로그램 인프라 및 미들웨어 또는 개발 도구와 기타 리소스를 구입하거나 관리하는 비용과 복잡성을 줄일 수 있다. 그리고 개발 및 운영에 필요한 SW 플랫폼 도입 및 개발/실험/운영 환경을 단기간에 구축할 수 있다. 즉, 로컬 개발 PC 환경구성, Source 통합 및 관리 환경구성, 운영서버 환경구성, SW 배포, 장애 시 복구 등의 문제를 단 기간에 확보할 수 있어 개발자는 비즈니스 요구사항에 집중 할 수 있고 민첩하게 SW를 개발 할 수 있는 장점이 있다. 국방정보체계의 보안요소와 폐쇄망 특성을 고려할 때, 특정 벤더나 외산 제품을 사용해야 하는 문제는 오픈소스를 활용한 자체 기술력 확보로 해결할 수 있다.

Ⅲ. 제안시스템

3.1 PaaS 서비스 계층

그림 1은 클라우드 서비스 모델 별 지원 범위를 구분하고, 클라우드 서비스 별 관리자의 역할을 SaaS 운영팀, PaaS 운영팀, IaaS 운영팀, 서비스 운영팀으로 나누고, 각 관리자와 운영자로 그 역할을 식별하였다. IaaS 운영팀은 스토리지, 네트워크, OS, 포털, 앱스토어 등의 클라우드 컴퓨팅 서비스를 구축하고, 서비스를 지속적으로 개발 및 관리하여 PaaS 운영팀과 응용어플리케이션 개발팀에게 IaaS 서비스를 제공한다.

PaaS 운영팀은 IaaS에서 제공받은 인프라를 기반으로 개발언어, 프레임워크, DBMS 등을 응용 개발팀에게 제공 할 플랫폼 서비스를 제공한다. 또한 응용개발팀이 개발한 소스의 지속적 통합, 지속적 배포를 위한 CI/CD (Continuos Integration and Continuous Deploy) 자동화 환경을 응용어플리케이션 개발팀에게 제공한다. 응용 어플리케이션 개발팀은 PaaS에서 제공 받은 개발 언어, 개발 프레임워크, 미들웨어(WAS, DBMS) 등의 플랫폼 통합환경 서비스를 통해 어플리

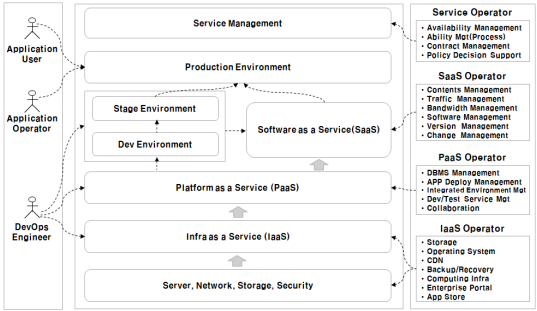

그림 1. 서비스 계층 별 운영자 역할
Fig. 1. Operator role by service tier

케이션을 개발 및 테스트하고, 이후 SaaS 운영팀과 어플리케이션 운영팀은 관련 이해당사자와 협업하여 SaaS 서비스에 등록 및 Production에 서비스를 런칭하도록 한다.

3.2 PaaS 제안 구조

PaaS 서비스는 DevOps 엔지니어에게 Cloud-Native Applications 개발 환경을 제공해야 하며, 그 기능 요건을 충족해야 한다. 첫 번째로 개발자는 Source Build, 응용프로그램 생성, Dev/Stage 테스트 등이 Self Service로 가능해야 한다. 두 번째로 개발자와 운영자는 통합개발환경, Web UI Console, CLI(Command Line Interface)을 통해 협업이 가능하도록 하여 상호 간에 작업 상태를 확인하고, 작동여부와 오류 현황은 함께 공유되도록 한다. 세 번째로 Git, Jenkins, Nexus 등을 통해 Develop과 Stage에 빌드, 테스트, 배포 할 수 있는 자동화된 CI/CD 기능이 제공되어야 한다. 네 번째로 컨테이너 오케스트레이션을 통해 워크로드에 증가 시 서비스가 자동 확장(Auto Scaling) 되도록 한다. 다섯 번째로 개발자/운영자/관리자/서비스 사용자는 RBAC 방식으로 접근통제하고, Develop, Stage, Production 접근은 규칙과 역할에 따라 통제되어야 한다. 여섯 번째로 프레임워크, 미들웨어, 개발도구 등 표준화된 SW의 설치와 구성을 자동화함으로써 신속한 개발과 수분 내로 테스트가 가능하도록 한다. 본 논문에서는 이러한 PaaS 서비스 기능을 구현하기 위해 아래 표1과 같이 개발 환경, 자동화 환경, 사용자 포털 환경으로 나누고, 각 분야별 세부 항목 분류로 PaaS 기능요건을 제안한다.

그리고 표 1에서 기술된 기능요건 구현 방법 제안을 위해 아래와 같이 PaaS 시스템을 도식화하였다.

그림 2에서 PaaS 제안 구조는 IaaS 인프라를 기반으로 PaaS 시스템을 구성하였으며, PaaS 호스트 노드

표 1. PaaS 요구조건
Table 1. PaaS Requirements

Field	Division	Details
Dev Environment	Program Language	Ruby, Java, Python, PHP, C#, .NET, Perl etc
	Dev Framework	Ruby on Rails, Spring, Node.js etc
	Web/WAS	Apache HTTP/Tomcat, Jboss, NGiNX etc
	Database	MySQL, PostgreSQL, MongoDB, MariaDB etc
Automatic Environment	Project Mgt	Project Life Cycle (Create, Update, Delete)
	Build Mgt	Docker Images, Build Pipelnes
	Access Control	RBAC, Storage, CPU/Memory Quota
	Container Orchestration	Kubunetes, Auto Scaling
	CI/CD Tool	Gitlab, Gogs, Jenkins, Sonarqube, Nexus
	Monitoring	Prometheus, Grafana
User Portal	PaaS User	Developer, Administrator, Operator
	User Interface	Web UI Console, Command Line Interface

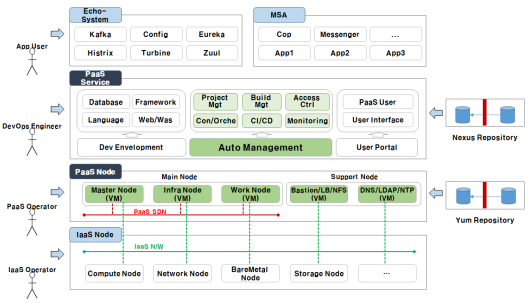

그림 2. PaaS 시스템 구성도
Fig. 2. PaaS System Diagram

를 Master, Infra, Work 주요 노드와 PaaS 서비스 제공을 위해 추가 요구되는 지원 노드로 구분하였다. PaaS 노드는 IaaS가 제공하는 VM 가상머신 위에 각 노드를 구성했다. 그리고 기능 요건 충족을 위해 개발환경, 자동화 관리, 사용자 포털 서비스로 분류하여 구성하였다. 본 연구에서는 PaaS 노드와 NW 구성방법, PaaS 서비스 중 자동화 관리의 6개 서비스 구현 방법, 그리고 Private PaaS의 제약 요소인 호스트 OS 관리 및 소프트웨어 라이브러리 의존성 자동화 구현 방법을 제안한다.

Ⅳ. 시스템 구현 및 결과

4.1 PaaS NW 구현

클라우드 컴퓨팅 환경의 통합데이터센터는 IaaS가 제공하는 가상화 기반 위에 PaaS 운용을 위한 소프트웨어 정의 네트워크가 구성된다. SDN((Software Defined Network)은 사용자가 소프트웨어 프로그래밍을 통해 네트워크 경로 설정, 제어, 운용관리 등의 네트워크 기능을 제어할 수 있는 기술로서, PaaS의 컨테이너 오케스트레이션에서 사용하는 Floating IP를 제공한다. 그림 3은 PaaS 구성 노드를 포함한 네트워크 구성도[3]를 나타낸다. OpenStack 기반의 다수 Hypervisor에 라우터 및 레지스트리 포드(Pod)가 있는 분산된 PaaS 노드를 구성한 제안으로, 서비스 사용자는 로드밸런서를 통하여 각 마스터 노드의 관리자 인터페이스에 접근할 수 있고, 인프라 노드의 레지스트리 관리 기능과 어플리케이션 서비스를 외부에 노출하는 라우터에 접근할 수 있다.

본 연구 검증을 위한 테스트베드에서는 고 가용성의 호스팅 클러스터에서 실행되는 하나의 로드밸런서, 3 개의 마스터, 3 개의 인프라 및 3 개의 애플리케이션을 노드 간 SDN을 통해 PaaS 네트워크를 구성하였다.

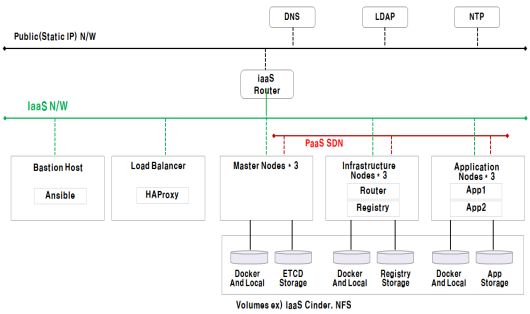

그림 3. PaaS 네트워크 구성도
Fig. 3. PaaS N/W Diagram

4.2 분산 노드 구현

분산 및 복수의 노드 구성을 통한 고 가용성(High Availability) 구성과 도커 컨테이너를 통한 Thin Provisioning 플랫폼 가상화 서비스를 구성하여 Failover 테스트를 포함한 구현 검증을 가능하도록 했다. 노드의 구성은 호스트 자원과 규모를 고려하여 구성한다. 또한 관리 및 유지보수의 편의성을 위해 PaaS 호스트를 IaaS의 VM(Virtual Machine)에 구성하여 PaaS의 노드 추가 등을 자동화할 수 있다.

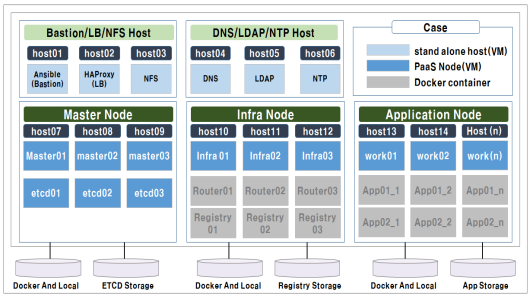

그림 4. PaaS 노드 구성도
Fig. 4. PaaS Node Diagram

그림 4에서 프로덕션 환경에서 PaaS 노드 구성 방법을 제안했으며, 이는 고 가용성 컨테이너 플랫폼 환경 구성이 가능하여 배포 프로세스를 단순화 할 수 있다. 고 가용성 서비스을 고려하여 각 노드를 3개의 호스트로 구성하였고, 노드 자동화 관리를 위해 Ansible을 별도 구성하고, 단일 호스트로 Loadbalance와 NFS, DNS, LDAP, NTP 서버를 별도로 구성하였다. Master 노드는 API 서버, Replication Controller, Scheduler, Data Store 등의 역할을 수행하고, Infra 노드는 Router, Registry, Metrics, Logging 등의 역할을 수행하며, Application 노드는 마이크로서비스가 실행되는 컨테이너 환경을 제공한다. 그리고 마스터와 인프라 Node host의 크기는 예상되는 workload에 따라 다르다. 노드 구성 시 예상되는 workload를 계산하고 10% 가중치를 더하면 된다. PaaS 소프트웨어 제품 별로 다르겠지만, 검증을 위해 테스트베드에 사용한 Openshift v3.11의 경우 클러스터 당 최대 노드는 2,000개, 클러스터 당 최대 Pod는 150,000개, Node 별 최대 Pod는 250개, 파이프라인 빌드 수 10,000개, 최대 서비스 10,000개, 서비스 별 백엔드 5,000개를 한계 할당 수준으로 제시[4]하고 있다.

4.3 컨테이너 오케스트레이션 구현

컨테이너 오케스트레이션(Container Orchestration)은 컨테이너와 서비스들이 대규모로 운영 될 때, 컨테이너의 라이프사이클을 관리한다. 소규모의 컨테이너를 배포하거나 유지하는 일은 어렵지 않지만 수백의 컨테이너와 서비스가 있는 경우 관리가 복잡해지고 어려워진다. 즉, 대규모로 운영하는 경우 컨테이너 배치, 관리, 확장, 컨테이너 가용성을 위한 컨테이너 오케스트레이션 기능은 필수적이다. 컨테이너 오케스트레이션 기술의 대표적 플랫폼으로 Kubernetes, Docker swarm, Mesos가 있다. 사실상의 표준으로 가장 인기 있는 Kubernetes의 경우 다음과 같은 기능[5]

을 제공한다.

- 필요 시 자원을 할당하는 컨테이너 프로비저닝과 배포 관리
- 컨테이너의 가용성과 필요 이상의 미사용 리소스 리던던시 관리
- 애플리케이션 로드 분산을 위한 컨테이너 확장 및 제거
- 호스트에 자원이 부족하거나 Down된 경우 컨테이너 이동 복구
- 컨테이너 간 자원 할당, 컨테이너 그룹에 대한 로드 밸런싱
- 네트워크 포트 제어, 컨테이너와 호스트의 상태 모니터링

쿠버네티스의 구조[6]는 크게 마스터 노드와 일반 노드로 구성 되어있다. 마스터 노드는 쿠버네티스 전체를 관리하는 역할을 한다. 노드에서 발생하는 다양한 이벤트를 감지하고 응답하는 등 의사결정을 수행한다. 마스터 노드는 여러 개의 노드로 구성될 수 있으므로 단일 마스터가 아닌 이중화 또는 삼중화 형태로도 구성할 수 있다. 그림 5에서는 Kubernetes 컨테이너 마스터 노드를 이중화한 경우, 컨테이너 관리자 동화를 위한 구조를 제안하였다. 마스터노드는 API server, Controller Manager, Scheduler, etcd 등으로 구성된다. 컨테이너 오케스트레이션 관리도구가 클러스터되는 경우 서비스 정보를 마스터 노드 간 공유해야 하고, 이러한 역할을 etcd 환경서버가 담당한다. etcd는 Kubernetes 클러스터의 Replicate 데이터 저장, 클러스터의 상태 관리, pod 상태, DNS 데이터 등의 환경정보를 저장한다.

API server는 Controller Manager, Scheduler, Proxy, pod 등이 만들어내는 모든 REST요청을 수신

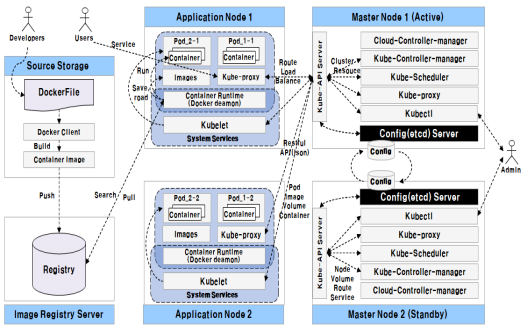

그림 5. 컨테이너 오케스트레이션 구성도
Fig. 5. Container Orchestration Diagram

하여 요청 작업을 실행하거나 전달한다. 쿠버네티스의 동작 상태에 대한 정보를 ectd에 저장하고, 배포된 pod 서비스 세부 사항과 일치하는지 확인하고 관리한다. Controller Manager는 Cloud Controller Manager와 Kubernetes Controller Manager 프로세스로 구동된다. CCM의 주요 기능은 클라우드 종속적인 KCM 내 컨트롤러들을 분리되도록 하고, IP 주소와 지역/영역 레이블 그리고 인스턴스 타입 정보와 같은 클라우드 환경으로 노드를 초기화 한다. KCM은 클라우드 컨트롤러로 Node, Route, Service Controller를 가진다. Node Controller는 클러스터에서 동작중인 노드에 대한 인스턴스 타입 그리고 크기 등 정보를 얻음으로써 노드를 초기화한다. 노드가 무응답일 경우, 클라우드로부터 해당 노드가 삭제된 것인지 확인하고, 삭제된 것이라면 Kubernetes의 노드 오브젝트를 삭제한다. 노드 컨트롤러는 해당 오브젝트를 get, list, create, update, patch, watch, 그리고 delete 하기 위한 모든 접근권한을 필요로 한다. Route Controller는 컨테이너들이 상호 소통할 수 있도록 클라우드에 적합하게 경로를 구성한다. Service Controller는 서비스 생성, 업데이트, 삭제를 담당하고, Kubernetes 내 서비스의 현재 상태를 근거로, 그 서비스의 상태를 나타내기 위해 클라우드 로드밸런서를 구성해준다. Kube Scheduler는 스케줄링의 결정을 위해 자원 요구, 하드웨어와 소프트웨어 정책 명시, 데이터의 위치, 워크로드 간 간섭과 데드라인 요소를 수집한다. 스케줄링은 최저 요구 우선, 균형자원 할당, 서비스 전개 우선권 등의 가중치 우선 순위(Priority)를 지정하여 최적인 노드에 컨테이너가 실행 할 노드를 결정하며, CPU/메모리/동작중인 컨테이너 수를 고려하여 배치하게 된다. Kube Proxy는 Pod 간 그리고 Node 간 통신을 지원한다. 모든 노드에 Daemon이 구동하고, 서비스와 End point의 변화를 감지하고 iptables 규칙을 처리하여 서비스 로드밸런싱 기능을 제공한다.

4.4 이미지 빌드 및 CI/CD 구현

Private PaaS의 CI/CD 구현은 조직, 워크로드 규모, 서비스 중요도, 보유 자원에 따라 다를 수 있다. CI/CD 의 방법과 범위도 DevOps팀의 구성과 개발/관리 조직과 역할, 서비스 환경에 따라 다르게 구성된다.

그림 6에서는 빌드와 배포를 중심으로 CI/CD 절차를 제안하였다. CI/CD 도구의 전체적인 절차를 보면 개발자가 소스를 직접 gitlab에 올리면(push), Gitlab에 소스가 통합되고, 통합된 최종 소스를 체크아웃하고 다운로드 후 Jenkins가 컴파일[7] 한다. 빌드가 성공

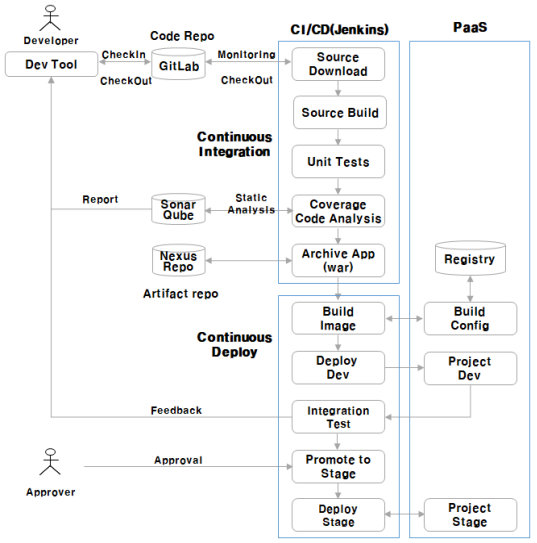

그림 6. 지속통합/지속배포 흐름도
Fig. 6. CI/CD Flow Diagram

하면 단위테스트, 커버리지 분석을 수행하고 리포트를 생성한다. 그리고 응용소프트웨어 종속성 라이브러리를 추가하여 배포용 아카이브 파일을 만들고 도커 이미지를 빌드한다. 이러한 과정이 정상적으로 수행되었을 때 안정적인 버전으로 간주하고 Tagging, 테스트 릴리즈, 통합테스트, 결과 분석 등의 과정을 수행한다. 통합테스트가 완료되면 Stage에 배포한다. Production 배포는 승인 프로세스를 제공하여 승인을 득하여 배포한다. 이러한 CI/CD 절차를 포함한 Pipeline 구성은 Jenkins에서 아래 그림 7과 같이 groovy 스크립트를 사용하여 구현한다.

스크립트의 실행과정은 그림 8과 같이 CI/CD 도구나 PaaS 시스템에서 Pipeline 시각화를 통해 단계별 소요 시간, 로그, 진행상태 등을 실시간으로 볼 수 있다. 또한 이러한 Pipeline은 PaaS의 Build config script에 등록하여 반복 재실행되도록 한다.

중요한 배포는 PaaS의 Deployment Config 스크립트를 개발하여 Blue-Green, Rolling, Canary 방식을 선택 적용하되, 비용, 시간, SW 특성 등 장/단점을 고

```
def mvnCmd = "mvn -s configuration/cicd-settings-nexus3.xml"
pipeline {
  stages {
    stage('Build App')       {  steps { sh "${mvnCmd} install  } }
    stage('Test')            {  steps { sh "${mvnCmd} test"   } }
    stage('Code Analysis') {  steps { script { //.....  }  }  }
    stage('Archive App')   {  steps { sh "${mvnCmd} deploy -DskipTests=true -P nexus3"   } }
    stage('Build Image')   {  steps { script { //...    } } }
    stage('Deploy DEV')    {  steps  { script { //....  } } }
  }
}
```

그림 7. 지속통합/지속배포 그로비 스크립트
Fig. 7. CI/CD groovy Script

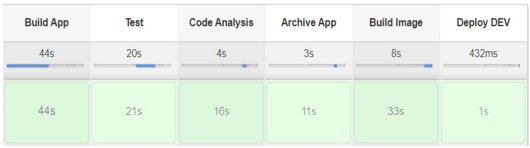

그림 8. 파이프라인 시각화
Fig. 8. Pipeline visualization

려하여 적합한 배포방식을 선택한다.

4.5 프로젝트 생성

위 절에서는 개발된 소스를 통합하고 소스빌드 후 도커 이미지 생성 및 이미지 배포 까지 과정을 설명했다. 이제는 CI/CD 과정을 적용할 프로젝트가 필요하다. PaaS는 단순한 명령어나 UI에서 프로젝트를 생성할 수 있다. 먼저 CI/CD 과정에서 사용하는 Gitlab, Jenkins, Gogs, Nexus, SonarQube 등이 도커 컨테이너에서 동작할 수 있는 CI/CD 프로젝트를 생성한다. 그리고 개발된 응용 소프트웨어와 마이크로서비스가 원활하게 동작할 수 있도록 지원하는 Echo-System을 구성하여 통합 테스트 할 수 있는 Develop 프로젝트와 배포를 위한 Stage와 Product 프로젝트를 생성한다. 다음 표 2는 PaaS에서 프로젝트를 만들고, 생성된 프로젝트에 배포되는 Docker Container를 나타내고 있다.

Echo-System은 Zookeeper, Kafka, Config, Eureka, Zuul, Turbine, Hystrix로 구성 할 수 있다, ZooKeeper는 Queuing 시스템으로 대규모 분산 시스템을 위한 분산 구성 및 동기화 서비스를 제공하고, Kafka는 메시지 발간/구독 관리 기능을 제공한다. Config와 Eureka는 Microservice 프로파일 정보관리와 서비스 등록 및 감지 역할을 한다. Zuul은 API Gateway로 로드밸런싱과 서비스라우팅 기능을 제공하고, Turbine과 Hystrix는 응용소프트웨어 마이크로서비스의 stream 수집과 대시보드 기능을 제공하도록 구성한다.

표 2. 프로젝트 생성
Table 2. Project Creation

Project	Docker Container
CI/CD	Gitlab, Jenkins, Gogs, Nexus, SonarQube
Develop	Echo-System and Microservice(App)
Stage	Echo-System and Microservice(App)
Product	Echo-System and Microservice(App)

4.6 접근제어

RBAC(Role-Based Access Control) 개체는 사용자가 프로젝트 내에서 지정된 작업을 수행 할 수 있는지 여부를 결정한다. 이를 통해 플랫폼 관리자는 클러스터 역할 및 바인딩을 사용하여 PaaS 및 생성된 프로젝트에 대한 다양한 액세스 수준을 사용자별로 제어 할 수 있다. 개발자는 로컬 역할(Role) 및 바인딩(Binding)을 사용하여 프로젝트에 액세스 할 수 있는 사람을 제어한다. 역할은 규칙(Rule)의 모음으로 사용자와 그룹은 동시에 여러 역할과 연결되거나 바인딩 될 수 있다. 규칙은 일련의 객체에 허용되는 create, get, list 등의 동사 세트이고, 바인딩은 역할을 가진 사용자 및 그룹 간의 연관 관계를 말한다. ID 및 프로젝트 범위는 사용자 또는 해당 그룹에 적용되는 모든 바인딩을 찾는 데 사용되고, 바인딩은 적용되는 모든 역할을 찾는 데 사용된다. 역할은 적용되는 모든 규칙을 찾는 데 사용되고, 일치 여부를 찾기 위해 각 규칙에 대해 작업을 확인한다. 일치하는 규칙이 없으면 기본적으로 작업이 거부된다.

아래 그림 9에서는 프로젝트 별 접근제어를 위한 역할, 규칙, 바인딩을 도식화하여 제안하였다. 역할[8]은 프로젝트 관리자, 클러스터 관리자, 편집자(edit), 검토자(view)로 분류한다. 프로젝트 관리자는 프로젝트의 모든 자원을 확인하고 할당량을 제외하고는 프로젝트의 모든 자원을 수정할 수 있는 권한을 가진다. 클러스터 관리자는 모든 프로젝트에서 모든 작업을 수행 할 수 있는 수퍼유저로 자원에 대한 할당량 및 모든 사항을 완전히 제어 한다. 예를 들어 CLI로 로그인 한 후 규칙, 역할 및 바인딩을 제어하여 역할에 대한 규칙 세트를 제공 할 수 있다.

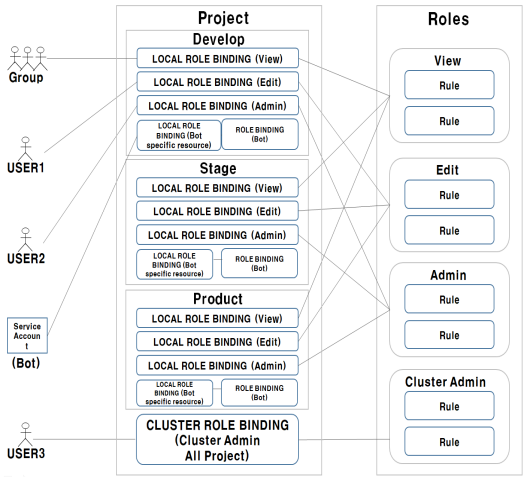

그림 9. 접근 제어
Fig. 9. Access Control

이 외에 기본 사용자, 편집자, 검토자, self-provisioner 역할을 설정한다. 기본 사용자는 프로젝트 및 사용자에 대한 기본 정보를 얻을 수 있는 사용자를 말한다. 편집자는 프로젝트에서 대부분의 객체를 수정할 수 있지만 역할이나 바인딩을 보거나 수정할 권한이 없는 사용자를 말한다. 검토자는 역할이나 바인딩을 보거나 수정할 수는 없지만 프로젝트에서 대부분의 객체를 볼 수 있는 사용자를 말한다. self-provisioner는 자신의 프로젝트를 만들 수 있는 사용자를 말한다. 다음 표3는 Registry와 Develop, Stage, Production 별 개발 관련자의 접근권한을 부여한 예시[9]를 보여준다.

표 3. 프로젝트 접근권한
Table 3. Project access

Division	Registry	Dev Cluster	Stage Cluster	Prod Cluster
Develop leads	pull only	Project administrator	project editor	project viewer, run debug pods
Developers	pull only	project editor	project editor	project viewer
Operators	pull only	project viewer	project editor	project editor

4.7 모니터링

오픈소스 모니터링 도구의 선택 시 소프트웨어의 완전성, 간편한 설치 및 구성, 웹 UI 세부 정보, 외부 서비스에 경고 전달, 커뮤니티 지원 및 참여 수준, Kubernetes 지원, 확장성, 배포 모델 등을 고려하여 선택해야한다. APM(Application Performance Monitoring) 도구에서는 Memory, CPU, Storage 사용률과 Concurrent User, Active User, TPS, Throughput 등의 수치가 기존의 모니터 대상이었다면, Cluster, NameSpace, ReplicaSet, Pod, Container, API Server, Quota, 서비스 간 의존성, 트래픽 흐름 모니터, 분산 트랜잭션 등의 상태 및 그에 할당된 리소스의 사용률이 모니터링 대상으로 추가되어야 한다. 그리고 대부분의 모니터링 솔루션은 Host Agent, Data gethering, Data store, Aggregation, Filtering & Analysis, Visualization, Alerting&Notification의 7계층 모델[10]로 구성된다. 모니터링 7레벨 지원 오픈소스 소프트웨어 제품은 Prometheus, Sysdig, Heapster, Sunsu가 있다.

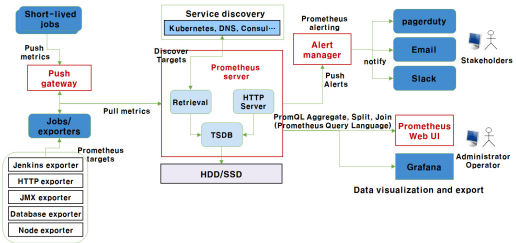

그림 10. 프로메테우스 모니터링 구성도
Fig. 10. Prometheus Monitoring Diagram

컨테이너 모니터링 소프트웨어 중 최근 주목 받는 Prometheus는 오픈소스로 시스템 및 서비스 모니터링을 제공한다. Prometheus 서버는 다양한 소스에서 시계열 데이터를 검색하고 내부 데이터 저장소에 데이터를 저장한다. 그리고 특정 유형의 Metric을 위한 별도의 푸시 게이트웨이 인 서비스 검색과 같은 기능을 제공하며 다차원 데이터 쿼리에 탁월한 내장 쿼리 언어(PromQL)를 제공한다.

Prometheus는 그림10과 같이 Exporter, Prometheus Server, Push gateway, Alert manager로 구성[11]되어 있다. 서버가 주기적으로 클라이언트에 접속해서 데이터를 가져오는 Pull 방식으로 Expoter가 열어놓은 HTTP 엔드포인트에 접속해서 Metric을 수집한다. Exporter는 모니터링 대상의 Metric 데이터를 수집하고 Prometheus가 접속했을 때 정보를 알려주는 호스트 에이전트 역할을 한다. 호스트 서버의 CPU, Memory 등을 수집하는 node exporter, nginx의 데이터를 수집하는 nginx exporter 등의 다양한 Exporter를 지원한다. Exporter를 사용하기 어려운 batch job은 Push gateway를 사용하여 사용자 지정 Metric을 Push하여 노출된 Metric을 스크래핑 하도록 구성할 수 있다. 그리고 웹 애플리케이션 서버 같은 경우 Metric은 클라이언트 라이브러리를 이용해서 Metric을 만든 후 커스텀 Exporter를 사용할 수 있다.

그리고 모니터링을 위한 사용자 인터페이스로 PromQL과 내장된 API, 웹 UI로 모니터링을 구현할 수 있고, 위 그림11처럼 Cluster, pod 등의 기본적인 모니터링 정보를 제공하는 Grafana를 사용할 수도 있다. Grafana를 연동해서 대시보드 형태로 시각화 하고, 알림 규칙을 생성해서 Alert Manager로 보내면 Alert Manager가 규칙에 따라 알림을 보내도록 구현한다.

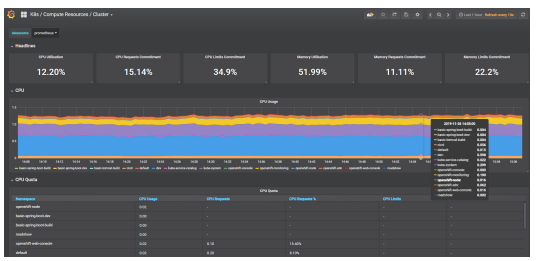

그림 11. 그라파나 시각화
Fig. 11. Grafana Visualization

4.8 라이브러리 의존성 관리 자동화

국방 지휘통제체계 Private Zone에서의 클라우드의 효과적인 활용을 위해서 사전에 해결할 사항이 있다. 첫 번째로 응용 소프트웨어 갱신과 개발 시 요구되는 라이브러리 의존성은 온라인을 통해 자동 추가 되어야 한다. 그러나 지휘통제체계는 폐쇄망에서 운용되어 클라우드 환경에서 중요한 부분인 지속통합(CI) 구현에 제한이 따른다. 개발 시 문제점은 eclipse 같은 IDE 도구에서 의존성 라이브러리를 이용하지 못하기 때문에 플러그인 설치도 어렵고, 패키지 관리 툴도 사용할 수 없다. Maven의 경우 라이브러리를 참조할 때는 Local repository, repository system(Nexus), Maven central repository 순서로 라이브러리를 찾기 때문이다.

해결방안으로 그림 12처럼 Nexus Repository에 원하는 라이브러리가 없을 때 외부에 Nexus를 설치하고, 폐쇄망 환경에서 외부 접속 가능한 서버 한대에 Nexus를 설치하고 Maven 로컬 레포지토리에서 Nexus 레포지토리를 바라보도록 한다. 즉, 외부의 Maven central repository 의존성 라이브러리 서버와 외부 망에 접속 가능한 위치에 Nexus repository를 배치하여 동기화한다. 수신된 Nexus는 망 분리 CDS(Cross Domain Solution)서버의 스트리밍(TCP) 또는 파일 Sync 방식을 사용하여 내부의 Maven Repository에서 수신 동기화 한다. 이렇게 수신된 라이브러리는 개발자의 개발 도구, 내부 개발서버, 운영서버에서 http 프로토콜을 통하여 대상 서버의 의존성 라이브러리를 갱신하도록 한다.

두 번째 문제는 RPM Package를 설치하는 방법은 온라인 환경에서 CentOS의 경우 yum, Ubuntu 계열

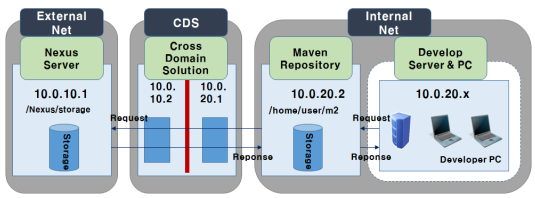

그림 12. 종속성 라이브러리 설치 자동화
Fig. 12. Automating dependency library installation

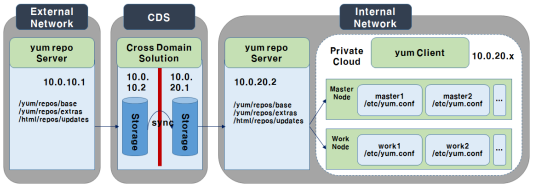

그림 13. RPM 패키지 설치 자동화
Fig. 13. Automating RPM Package Installation

의 경우 apt-get으로 손쉽게 설치 할 수 있다. 그러나 내부 폐쇄망 Private Zone에서 Docker 등 RPM을 설치해야 하는 경우에는 Dependency 이슈 해결을 위한 폐쇄망에서 설치, 갱신, 반복되는 수작업은 한계가 있다. 개별로 하나씩 모든 패키지를 업데이트 하는 것도 불가능하고 업데이트 정보를 주기적으로 받아야 하는 문제가 발생한다.

이 문제 해결을 위해 그림13처럼 외부 인터넷에 접속이 가능한 yum(rpm) Repository 서버를 구축 한다. yum은 RPM 기반 시스템을 위한 자동 패키지 설치/업데이트 도구다. 중앙서버에서 패키지를 관리해 주므로 일관된 버전 관리가 가능하다. Red Hat Linux, Fedora, CentOS 등 RPM 기반의 배포판들은 모두 yum을 사용하고 있다. yum repository는 cloud 자동화 구성을 위해서 yum repository를 만들고 자동화 프레임워크로 ansible 또는 chef를 사용하여 자동화 과정에서 설치할 패키지 등을 저장하기 위한 용도다. rsync를 지원하는 kaist와 neowiz 등의 mirror server를 선택하고, 업데이트 서버 rsync를 이용해서 외부의 yum repository를 정기적으로 sync 시킨다. 망 분리 솔루션을 통과 후 내부 망 yum repository와 동기화되고 내부 개발/운영환경에서 설치할 수 있도록 구성한다.

4.9 구현결과

위 절에서 제안한 구현 방법의 검증을 위해 테스트베드를 구축하였다. Openstack KVM 하이퍼바이저 위에 CentOS Linux 7.6.1을 설치하여 다수의 독립 Host 서버를 구축하고, 그 위에 PaaS 오픈소스 소프트웨어인 Openshift Origin v3.11.0을 설치 후 CI/CD Pipeline과 Echo-System, 샘플 응용 마이크로서비스를 구현하였다. 단, 라이브러리 의존성 관리 자동화는 CDS 환경적 제약으로 구현은 생략하였다. 아래 표4는 테스트베드에서 구현한 호스트 서버 목록 및 CPU, Disk, Memory, Storage 구성정보를 기술하였다.

다음 표 5는 구현에 사용된 소프트웨어 및 버전정보 목록이다.

표 4. 호스트(가상머신) 서버 목록
Table 4. Host(VM) server list

Hostname	SW	CPU/DISK /MEM	Storage
ansible	Ansible	8/300G/16G	-
lb	HAProxy	8/300G/8G	-
nfs	NFS	8/300G/16G	3TB
dns	DNS	4/300G/4G	-
ldap	LDAP	4/300G/4G	-
ntp	NTP	4/300G/4G	-
master01	Openshift, Docker, Etcd, K8S	8/400G/32G	etcd storage (1TB)
master02	Openshift, Docker, Etcd, K8S	8/400G/32G	
master03	Openshift, Docker, Etcd, K8S	8/400G/32G	
infra01	Gateway/Regstry	8/400G/32G	Registry storage (1TB)
infra02	Gateway/Regstry	8/400G/32G	
infra03	Gateway/Regstry	8/400G/32G	
work01	Kube-proxy, Kube-let, Docker	16/600G/64G	App storage (1TB)
work02	Kube-proxy Kube-let, Docker	16/600G/64G	
work03	Kube-proxy Kube-let, Docker	16/600G/64G	

표 5. 소프트웨어와 버전
Table 5. Software & Version

Division	Software & Version
PaaS	Openshift Origin v3.11.0, kubernetes v1.11.0, Docker v1.13.1, ansible v2.6.3, prometheus v2.3.2, grafana v5.2.1, HAProxy v1.5.18
CI/CD	GitLab Community Edition v12.1.6, Jenkins v2.176.3, Sonatype Nexus Repository OSS v3.13.0-0, gogs v0.11.34, gogsdb-postgresql v9.6, sonarqube community v8.0, sonardb-postgresql v9.6
Echo-System	Zookeeper v3.4.8, Kafka v0.10.1.1, spring-cloud-config server v1.4.2, Eureka v1.7.0, ribbon v2.2.4, Zuul v1.3.0, Turbine v1.0.0, Hystrix v1.5.12
Microservice	Messenger Service (Openfire v4.4.1) Map Service (Geo Server v2.15.1)

V. 결 론

본 논문에서는 국방 지휘통제정보체계의 신속한 응용SW 제공을 위해 오픈소스 소프트웨어를 활용한 컨테이너 기반 Private PaaS의 설계 및 구현방법을 제안하였다. 클라우드 IaaS, PaaS, SaaS 서비스 모델 별 운용자의 주요업무와 PaaS 서비스에서 제공해야하는 기능 요건을 식별하였다. 그리고 PaaS 환경구성을 위한 네트워크 가상화 및 분산 노드 구성 방법을 제안했다. 또한 컨테이너 오케스트레이션, 이미지 빌드와 배포를 포함한 CI/CD, 프로젝트 생성, RBAC 방식 프로젝트 접근제어 및 권한부여, Prometheus와 Grafana를 활용한 모니터링 등 PaaS 관리 자동화 6개 주요기능의 구현방법을 제안하였다. 이러한 제안과 Echo-System 및 샘플응용 마이크로서비스를 테스트베드에서 구현 및 시험운용을 통해 검증하였다. 추가적으로 군의 특성을 고려하여 폐쇄망에서 라이브러리 의존성 관리 자동화 구현 방법을 제안하였다. 이로서 향후 우리군의 지휘통제체계 통합데이터센터 구축 시 IaaS 위에 컨테이너 기반 Private PaaS를 구축 할 수 있는 검증된 기술을 확보하게 되었다. 또한 PaaS 플랫폼을 사용하여 개발절차, 지속통합, 지속배포, 테스트 환경을 표준화할 수 있게 되었고, 응용 소프트웨어를 보다 민첩하게 공급하여 지휘통제에 필요한 임무응용 서비스를 적시 제공할 수 있게 되었다. 그리고 오픈소스를 활용한 Private PaaS 서비스 구현으로 특정 벤더 의존성을 낮추고 초기 도입 비용을 대폭 절감 할 수 있는 방안을 제시하였다.

References

[1] *Core Data Center Reference Architecture Version 1.0 Final*, DISA Sep. 2012.

[2] J. Y. Hwang and H. Y. Rho, "Performance comparison and forecast analysis between KVM and docker," *JKIIT*, vol. 13, no. 11, pp. 127-136, Nov. 2015.

[3] M. Curry, *OpenShift Container Platform Reference Architecture Implementation Guides*, Jan. 2017.

[4] *OpenShift Container Platform Cluster Limits*, from https://docs.openshift.com/container-platform/3.11/scaling_performance/cluster_limits.html

[5] *Concepts Underlying the Cloud Controller Manager*, from https://kubernetes.io/docs/concepts/architecture

[6] K. Kim, et al., "Kubernetes architecture for cloud services," *J. KICS Inf. & Commun. Mag.*, vol. 35, no. 11, pp. 11-19, Oct. 2018.

[7] siamaksade, *OpenShift Container Platform-CI/CD Demo*, redhat, Jul. 2019.

[8] *Openshift container-platform cluster Authorization*, from https://docs.openshift.com/container-platform

[9] A. Weitekamp, J. Callen, and P. T. Mauri, *Application CI/CD on Openshift container platform with Jenkins*, 2017.

[10] G. Sissons, *Comparing 10 Docker Container Monitoring Solutions for Rancher*, Oct. 2017 from https://rancher.com/comparing-10-container - monitoring-solutions-rancher

[11] *Prometheus Components & Architecture*, from http://prometheus.io

정 은 태 (Eun-Tae Jung)

2006년 2월 : 서울디지털대학교 경영(e-biz)학과 졸업

2016년 2월 : 고려대학교 소프트웨어공학과 석사

현재 : (주)도전하는사람들 부설 연구소 수석연구원

<관심분야> 클라우드 컴퓨팅, 서비스브로커, 카오스 엔지니어링, 마이크로서비스

[ORCID:0000-0001-5688-0877]

박 규 동 (Gyu-Dong Park)

1994년 2월 : 홍익대학교 컴퓨터공학과 졸업

1994년 2월 : 홍익대학교 컴퓨터공학과 석사

2014년 2월 : 홍익대학교 컴퓨터공학과 박사

현재 : 국방과학연구소 책임연구원

<관심분야> 지휘통제체계(C4I), 가상화, 클라우드 컴퓨팅

[ORCID:0000-0001-7484-5426]

최 백 규 (Baek-kyu Choi)

2005년 2월 : 세명대학교 컴퓨터과학과 졸업

현재 : (주)도전하는사람들 부설 연구소 선임연구원

<관심분야> 클라우드 컴퓨팅, 상호운용성, 컴퓨터공학, 통신공학

[ORCID: 0000-0001-5211-0787]