정보기술동향

컨테이너 기반의 클러스터 컴퓨팅 기술 동향

최동훈 (한국과학기술정보연구원)

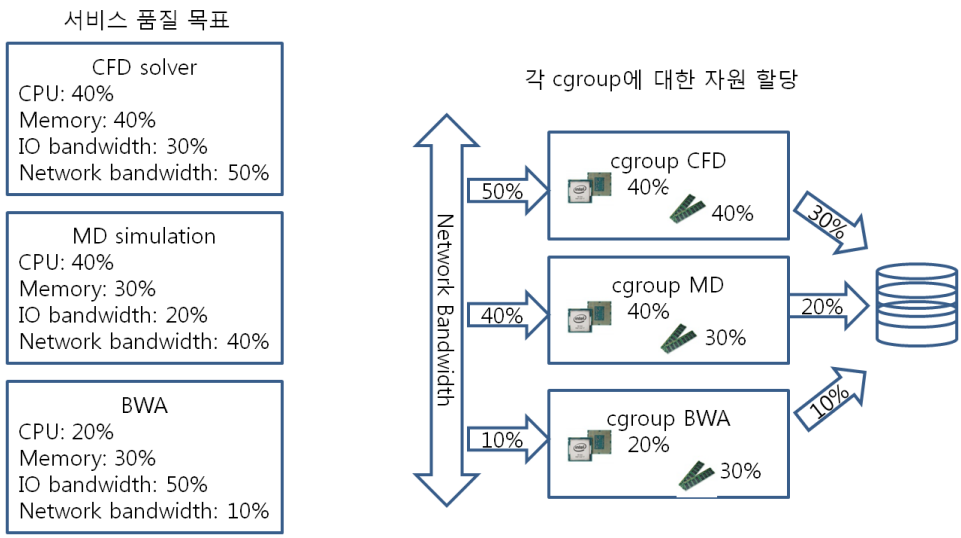
목차

1. 소 개

지난 수년간 정보 시스템은 가상화 기술이 제공하는 자원의 병합과 격리를 통해 어느 정도의 오버헤드를 감수하는 대신에 자원 관리의 효율성과 성능을 추구해 왔다. 최근들어 가상화 기술에 비해 오버헤드가 훨씬 적은 컨테이너(container)에 의한 운영체제 수준의 가상화 기술이 공개되면서 관심을 집중시키고 있다. 마이크로소프트는 윈도우즈용 컨테이너를 개발하여 보급하고 있고, 구글은 컨테이너에 기반한 클러스터 컴퓨팅 기술로 쿠버네테스(Kubernetes)를 개발하여 공개하였다. 불과 10여년 전만 하더라도 운영체제를 비롯해서 시스템 소프트웨어를 돈을 받고 파는 것으로 생각했었으나, 요즘들어 이들 소프트웨어가 공개되고 있다. 이러한 분위기에서, 해외 유능한 개발자들은 신기술에 대한 자신의 능력을 계발하거나 이들 소프트웨어를 개선내지 발전시키려는 기회로 삼고 커뮤니티를 구성하여 적극적으로 활동하고 있다. 반면 이러한 해외 동향에 비해 국내에서는 이들 기술에 대한 관심이 아직은 낮은 수준에 머물러 있다. 이 글에서는 이러한 기술 동향에 대한 국내 개발자의 관심에 부응하고자, 컨테이너 기술과 컨테이너 기반의 클러스터 컴퓨팅 기술에 대해 소개하고자 한다.

2. 가상 머신과 컨테이너

가상 머신은 특정 환경의 실제 하드웨어에서 실행하는 것처럼 프로그램을 실행할 수 있는 에뮬레이션 소프트웨어이다. 이것은 하이퍼바이저를 통해 가상 머신을 만들고 이 가상 머신 위에 운영체제를 설치해야만, 사용자가 자신이 원하는 프로그램을 실행할 수 있는 것을 의미한다. 사용자는 KVM, VMware, Virtualbox, Windows



(그림 1) 서비스 품질관리에 cgroup의 활용

Virtual PC 등 하이퍼바이저에 의해 가상 머신을 생성 및 관리한다.

컨테이너(container)는 운영체제 수준의 가상화 방법으로, 물리적 머신으로부터, CPU, 메모리, 디스크, 네트워크 등을 포함하는 격리된 가상 환경을 제공한다. 컨테이너는 물리적 머신의 운영체제를 사용하기 때문에 프로그램의 실행에 필요한 자족적인 실행 환경을 갖추고 있어서, 가상화와 달리, 운영체제 및 (실행할) 프로그램의 설치를 요구하지 않는다. 컨테이너 내에서 프로그램을 실행할 때, 사용자는 운영체제의 실행에 내포된 오버헤드 없이, 자신이 원하는 프로그램을 실행할 수 있다. 컨테이너에서 실행되는 프로세스는 컨테이너 안에서만 존재한다. 컨테이너는 호스트 운영체제 및 다른 컨테이너와 격리되어 있으며, 가상화와 달리 에뮬레이션을 필요로 하지 않는다.

컨테이너의 기본 아이디어는 공통적인 특성을 갖고 있는 프로세스 집합에 이들이 필요로 하는 자원의 활용을 격리시켜 생성한 하나의 컨테이너를 할당하고, 컨테이너 밖에서는 이들 자원의 활용을 막는 것이다. 컨테이너의 생성은 리눅스 커널의 기능, 즉, cgroup, namespace, apparmor, networking 인터페이스, 방화벽 규칙 등의 활용에 의한다. 즉, 컨테이너에 할당된 CPU, 메모리, 디스크, 네트워크 환경을 생성하기 위해 호스트 운영체제를 이용한다. 궁극적으로 cgroup은 그림 1과 같이 각 프로세스 그룹에 자원을 배분하여 서비스 품질 수준을 유지하는 데 사용된다.

3. 다 커

다커(Docker)는 어플리케이션의 이식성을 위해 개발된 사용하기 쉬운 경량급 가상화 환경으로, 컨테이너를 확장한 어플리케이션이다. 다커를 통해 사용자는 컨테이너의 수명주기를 쉽게 관리할 수 있다. 즉, 어디에서든지 실행할 수 있도록 관리하기 쉬운 단위로 어플리케이션을 전개(deployment)할 수 있다. 전개할 때, 다양한 변종의 리눅스와 다양한 유형의 가상 머신 또는 하

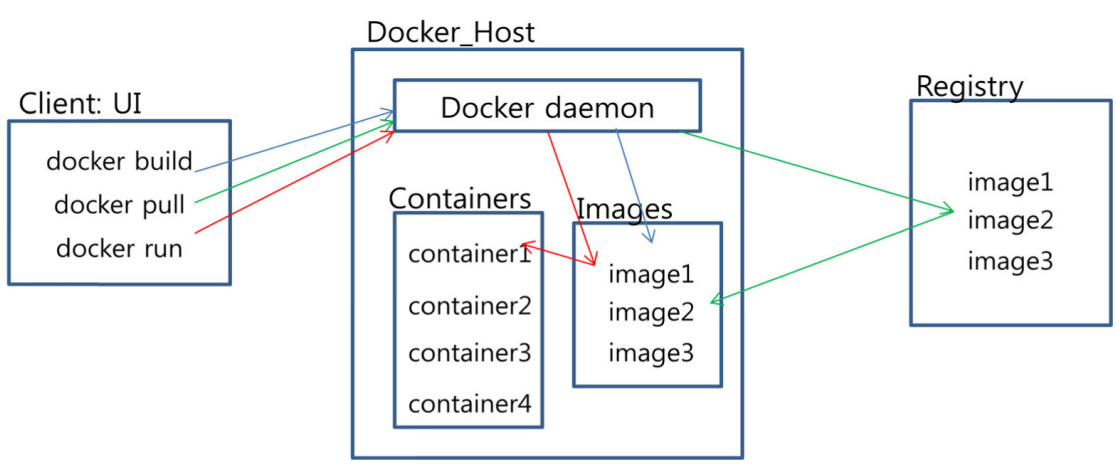
드웨어에 대한 어플리케이션의 종속성으로 인해 발생하는 번잡한 일을 아주 쉽게 다룰 수 있다. 다커를 사용하여 컨테이너 안에 어플리케이션을 빌드(build)하면, 사용자는 실행 환경에 신경쓰지 않고 어플리케이션을 실행할 수 있다.

다커는 그림 2와 같이 RESTful 다커 데몬(daemon), 다커 클라이언트(CLI, Command Line Interface), 공개 및 비공개 다커 이미지 레지스트리로 구성되어 있으며[1], 다커 클라이언트는 다커 데몬을 통해 레지스트리로부터 다커 이미지를 탐색 및 추출할 수 있다. 다커를 사용하려면 사용자는 먼저 자신의 어플리케이션에 적합한 다커 이미지를 생성(build) 하거나 레지스트리에서 어플리케이션에 적합한 기존의 다커 이미지를 탐색하여(search) 추출한다(pull). 다커 이미지에 따라 다커 컨테이너를 생성하여 그 위에서 어플리케이션을 실행한다(run). 다커 이미지를 레지스트리에 등록하여 다른 사용자와 공유할 수 있다.

다커 데몬은 호스트 머신에서 동작한다. 사용자는 데몬과 직접 상호 작용을 하지 않고 다커 클라이언트를 통한다. 다커 클라이언트는 사용자로부터 명령을 받아서 다커 데몬에게 보내거나 다커 데몬으로부터 수행 결과를 사용자에게 보낸다. 다커 컨테이너를 생성하기 위한 템플릿을 다커 이미지라고 한다. 다커는 새로운 이미지를 생성하거나 기존의 이미지를 갱신(변경되어야 할 부분만 편집)할 수 있는 기능을 제공한다. 다커 레지스트리는 사용자가 생성한 이미지를 공개 또는 비공개 저장소에서 관리한다. 다커 컨테이너는 어플리케이션의 실행에 필요한 모든 자원을 포함하고 있으며, 각 컨테이너는 다커 이미지에 따라 생성된다.

항상 더 나은 기술을 사용하려는 사용자 입장에서는 다커와 가상 머신 중에 어떤 것을 쓸 것인가 고민될 것이다. 그것은 사용자가 하려고 하는 것이 무엇인가에 달려 있다. 어떤 어플리케이션 환경에서는 운영체제를 비롯하여 필요한 소프트웨어를 모두 설치하여야만 만족스러운 경우가 있다. 이러한 환경에서는 가상 머신이 필수적이다. 반면에 개발자가 어플리케이션을 하나의 단위로 테스팅 그룹이나 소비자에게 직접 전달하는 경우에는 다커가 더 적절하다.

다커는 가상 머신과 경쟁관계인가? 그렇지 않



(그림 2) 다커의 구조

다. 다커 컨테이너는 공개소스이면서 다양한 리눅스 변종에 어플리케이션을 패키징하고 전개하는 데 따르는 오버헤드가 낮기 때문에, 클라우드와 어플리케이션 개발자들이 선호하고 있다. VMware 조차도 VMworld 2014에서 다커, 구글, 피보탈과 파트너쉽을 통해 가상화와 컨테이너를 통합하고 있다고 강조할 정도이다. 그러나 대기업이나 이미 가상화 기술을 도입한 조직에서는 가상화가 아직 지배적이고 다커를 거의 사용하지 않고 있다. 다커는 가상화에 비해 다커와 리눅스 컨테이너가 CPU, 메모리, 디스크 오버헤드가 훨씬 적기 때문에 성능과 병합(consolidation) 측면에서 더욱 좋다. 이것은 컨테이너화된 어플리케이션을 실행할 때 운영체제의 추가 설치가 필요 없기 때문이다. 이러한 이유로 가상화 기술은 다커를 긍정적으로 받아 들일 필요가 있다. 가상화를 다커로 교체하기보다, 가상화를 보강하기 위해 다커를 사용하는 것이 현실적이다. 가상 머신 안에서 다커 컨테이너를 전개하는 것도 하나의 방법일 것이다.

4. 쿠버네테스

구글은 Gmail, Search, Apps, Maps 등을 컨테이너 안에서 실행하기 때문에, 매주 20억 개의 컨테이너를 관리하고 있다. 이렇게 컨테이너를 가지고 이들 어플리케이션을 서비스하는 이유는 컨테이너가 가상 머신보다 격리 환경을 관리하는 데 효율적이기 때문이다. 구글은 남들이 가상 머신을 개발하고 있었던 지난 십수 년간 리눅스 컨테이너를 비롯하여 대규모 컨테이너 관리 도구를 개발해 왔다. 이러한 노력 덕분에, 구글은 App Engine, Compute Engine 등의 서비스를 통해 클라우드 비즈니스에 진입하게 되었다. 이와 동시에 구글은 다커 기반의 컨테이너 클러스터 엔진인 쿠버네테스를 비롯한 컨테이너 관리 기술을 개발자에게 공개하였다. 쿠버네테스는 그리스어로 조타수라는 의미이며, 배의 키를 로고(logo)로 사용하고 있다.

4.1 컨테이너 클러스터 관리 엔진

다커는 컨테이너의 수명주기를 관리하지만, 쿠버네테스는 컨테이너의 클러스터를 조직하고 관리하는 컨테이너 클러스터 관리엔진이다. 쿠버네테스는 기존의 가상 머신에 컨테이너를 개시하거나 새로운 가상 머신을 생성하여 컨테이너를 개시할 수 있고, 이들 컨테이너를 모니터링한다. 관리자는 쿠버네테스를 이용하여 하나의 어플리케이션의 실행에 필요한 컨테이너의 집합을 하나의 단위(이것을 pod라고 하는데, pod란 하나의 콩깍지 안에 있는 콩들의 집합을 하나의 단위로 상상하면 이해하기 쉽다. 앞으로 이것을 '콩깍지'라고 부르자.)로 생성할 수 있으며, 쿠버네테스는 여러 개의 가상 머신이나 물리적 머신 내에서 이들 컨테이너의 집합을 하나의 단위로 사용자에게 제공한다.

쿠버네테스의 주요 기능은 다음과 같다.

① 어플리케이션 서비스, 네트워크 서비스, 스토리지 서비스:
- DNS(domain name system), 부하 균형 유지, 규모 확장 관리, 어플리케이션 수준의 결함 관리, 서비스 어카운트 관리를 포함하여, 부하 관리 및 분배에 중요한 핵심 기능을 제공한다.
- Google Compute Engine의 persistent disk storage, AWS의 Elastic Block Store, NFS와 같은 로컬 및 네트워크 기반의 디스크

볼륨을 이용하여, 상태 유지 어플리케이션을 지원한다.

- 콩깍지 속에 컨테이너를 전개하여 갱신과 롤백을 쉽게 한다.
- CLI 및 UI를 통해 명령어 실행, 포트 전달, 로그 수집, 자원 모니터링을 수행하여, 어플리케이션을 점검하고(inspect) 디버깅한다.

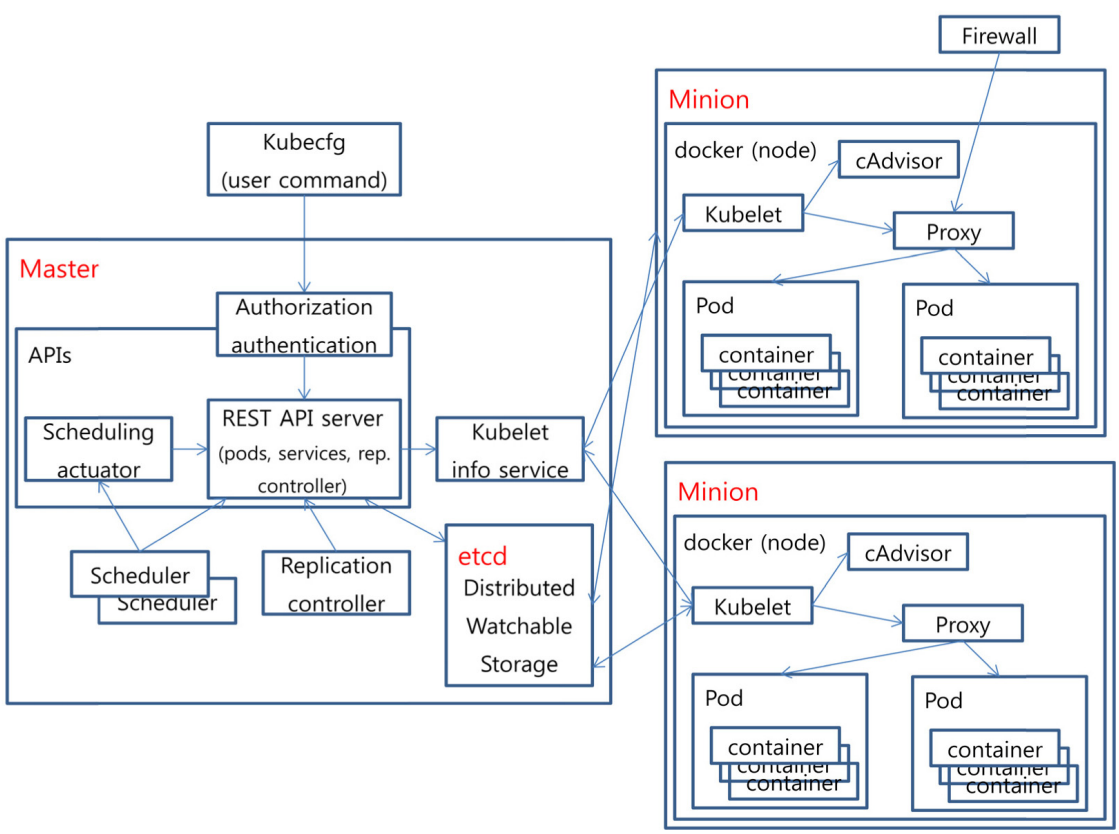
② 동적 클러스터 관리:

- 클러스터를 갱신하고 동적으로 규모(확장)를 관리한다.
- 자원 관리를 강화하기 위해, network namespace에서 전용 서브넷을 정의하여 클러스터를 분할한다. 이렇게 하여, 클러스터를 서로 다른 여러 어플리케이션에 적합하게 분할하거나 테스트용과 서비스용으로 분할할 수 있다.

4.2 쿠버네테스의 주요 구성 요소

쿠버네테스의 구조는 그림 3과 같고, 주요 구성요소를 간략히 서술하면 다음과 같다[2].

- Master: 관리 머신으로 다수의 Minion을 감독한다.
- Minion: 사용자와 Master가 위임한 타스크를 실행시키는 슬레이브이다.
- Pod: 미니언에서 실행되는 어플리케이션 또는 그 일부분에 속하는 컨테이너의 집합으로, 쿠버네테스에서 생성, 스케줄링, 관리할



(그림 3) 쿠버네테스 구조

수 있는 자원의 기본 단위이다.

- Replication Controller: 마스터의 하위 구성요소(자원)로 사용자가 요청한 수만큼의 Pod가 미니언에서 항상 실행되도록 보장한다. (복제 대상은 Pod이다.)
- Kubelet: 각 미니언은 컨테이너를 실행하기 위한 서비스를 실행하고, 마스터는 이러한 미니언을 관리한다. kubelet은 컨테이너 목록(Pod 정의서가 들어 있음)을 읽어서 Pod에 정의된 컨테이너를 개시하는 역할을 수행한다.
- RESTful API: REST API를 통해 마스터는 Pod, Service, Replication Controller에 대한 연산을 검증하고 환경을 설정한다.

4.2.1 Master의 하위 구성요소

Master는 클러스터의 부하와 시스템 간의 통신을 관리한다. Master의 하위 구성요소는 다음과 같다.

- etcd: 쿠버네테스의 가장 기본적인 구성요소로, config 데이터를 저장하는 분산 키-값 저장소이다. config 데이터는 각 구성요소가 스스로 환경을 설정하거나 재설정하기 위해 참조하는 클러스터의 상태를 나타낸다.
- REST API 서버: 비즈니스 로직과 함께 REST 연산을 검증하고 수행한다. 사용자는 이것을 이용하여 쿠버네테스의 부하와 구성 단위를 설정할 수 있기 때문에, API 서버가 전체 클러스터의 주요 관리 포인트라고 할 수 있다. API 서버는 RESTful 인터페이스를 구현하고 있어서, 사용자는 다양한 도구와 라이브러리를 이용하여 API 서버와 통신할 수 있다.
- 인증 서버: Service account는 Pod에서 실행되는 프로세스의 식별자를 제공한다. 사용자가 클러스터를 접근할 때에는 REST API 서버에 의해 특정 user account로 인증받고, Pod 내의 프로세스가 API 서버에 접속할 때에는 특정 service account로 인증받는다.
- Replication Controller 서버: replication controller 서버는 복제 타스크가 정의한 복제 프로세스를 처리하며, 이 연산에 대한 상세 내역을 etcd에 write한다. replication controller는 Minion 서버의 kubelet 서비스에서 변경 사항이 발생하는지 ectd를 주시한다. 변경이 발생하면 replication controller는 새로운 정보를 읽어서 복제 절차를 이행하여, 요청받은 만큼의 사본을 유지한다. Replication controller는 현재 실행중인 컨테이너가 중지될 경우 다른 컨테이너를 찾아서 개시한다. 중지되었던 컨테이너가 복구되면, replication controller는 컨테이너 하나를 중지시킨다.
- Scheduler 서버: scheduler는 부하를 클러스터의 특정 노드의 Pod에 할당 및 회수한다. 스케줄러는 서비스의 운영 요구사항을 읽어 들여서, 현재 인프라 환경을 분석하고, 수용 가능한 노드에 부하를 배치한다. 반면에 각 호스트의 자원 활용을 추적하여 사용 가능한 자원을 초과하는 부하를 스케줄링하지 않는다.

4.2.2 Minion 서버의 하위 구성요소

쿠버네테스에서 실제로 작업을 실행하는 서버를 Minion이라고 한다. 미니언 서버는 마스터와 통신을 주고 받고 컨테이너를 위한 네트워킹 설

정을 하며, 실제 부하를 실행한다.

- 다커: 각 미니언 서버에서는 다커가 실행된다. 다커 서비스를 사용하여, 각 작업 단위를 일련의 컨테이너들로 전개하여 실현한다. 이때 쿠버네테스는 전용 서브넷의 사용으로 네트웍을 격리시킨다.
- Kubelet 서비스: Kubelet 서비스의 역할은 Master와 Minion 간의 커멘드와 작업 정보를 교환하고, etcd 저장소로부터 config 상세 내역을 read/write하는 것이다. 작업 정보는 작업부하와 운영에 관한 파라미터를 포함한다. Kubelet 프로세스는 작업의 상태를 미니언 서버에서 유지한다.
- cAdvisor: cAdvisor는 컨테이너 자원 사용 및 성능 분석 에이전트로, 다커 컨테이너를 지원한다. cAdvisor는 머신 내의 모든 컨테이너에 대한 CPU, 메모리, 파일 시스템, 네트워크 등의 사용 통계치를 수집한다. 컨테이너 클러스터에 대한 사용 정보를 분석하여 전체적인 물리적 머신의 사용 현황을 제시한다.
- 프록시 서비스: 각 호스트의 서브넷을 다루고 외부에 서비스를 사용 가능하게 만들기 위해 프록시 서비스가 각 미니언 서버에서 실행된다. 이 프로세스는 컨테이너에게 요청을 전달하고, 원시적인 부하균형을 유지하며, 네트워킹 환경의 예측 가능성과 접속 가능성을 보장한다.
- Pod: Pod는 쿠버네테스가 컨테이너 자원을 생성, 스케줄링, 관리하는 기본 단위이다. Pod는 단일 어플리케이션에 할당된 컨테이너의 집합을 나타낸다. 쿠버네테스는 '동일한 Pod에 속한 컨테이너들'을 하나의 단위로 스케줄링한다. 볼륨과 IP 공간을 공유할 수 있고, 어플리케이션의 부하 증가에 따라 할당되어야 할 자원의 규모를 확장할 수 있다.

5. 결 론

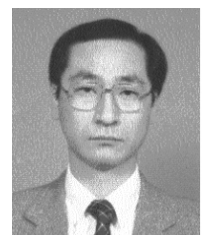
컨테이너는 지난 십수 년간 구글을 중심으로 대규모 데이터 센터의 자원 관리를 위해, 자원 공유에 의한 성능 간섭을 최소화하고 자원 활용을 극대화하기 위한 노력으로 발전을 거듭해 왔다. 이 결과가 집적된 다커와 쿠버네테스가 공개되었다. 이후 개발자의 관심이 집중되면서 향후 급속하게 발전될 것으로 예상된다. 슈퍼 컴퓨팅 컨퍼런스에서도 고성능 클라우드 컴퓨팅을 위해 OpenStack보다 다커가 사용자의 관심을 더 끌고 있다. 더구나 쿠버네테스에 대한 관심은 더욱 뜨겁다. Red Hat, CoreOS, IBM, Intel, Microsoft, VMware 등에 소속된 개발자를 포함하여 400명 이상의 컨트리뷰터로부터 14,000개의 commit을 기록하고[3] 있을 정도로, 쿠버네테스는 아주 짧은 기간에 가장 대중적이고 성공적인 공개 소스 프로젝트 중의 하나가 되었다.

필자의 상식으로 볼 때, 공개소스 소프트웨어에 의해 국가간의 소프트웨어 기술 격차가 좁혀져야 마땅하다. 그럼에도 불구하고, 국내 소프트웨어 역량은 특히 시스템 소프트웨어 분야에서 점점 뒤쳐지고 있다. 공개소스가 대세를 이루면서 앞으로 이 격차가 더욱 벌어질 것이라는 것을 상상하면, 이들 공개소스에 대한 관심과 투자가 절실한 시기라고 생각한다. 이러한 측면에서 이 글이 국내 개발자들에게 조금이나마 도움이 되길 바라면서 끝맺는다.

참 고 문 헌

[1] Understand the architecture: https://docs.docker.com/introduction/understanding-docker/
[2] Kubernetes: https://github.com/googlecloudplatform/kubernetes
[3] Kubernetes V1 Released: http://googlecloudplatform.blogspot.kr/2015/07/Kubernetes-V1-Released.html

저 자 약 력



최 동 훈

이메일: choid@kisti.re.kr

- 1981년 2월 서울대학교 계산통계학과 졸업 (학사)
- 1983년 2월 한국과학기술원 전산학과 졸업 (석사)
- 1989년 6월 Northwestern University 전산학과 졸업 (박사)
- 1983년 2월~1986년 8월 한국증권전산㈜
- 1989년 8월~1992년 2월 한국국방연구원 선임연구원
- 1992년 3월~1999년 2월 동덕여자대학교 부교수
- 2005년 2월 현재 한국과학기술정보연구원 책임연구원
- 관심분야 : 데이터베이스, 고성능 컴퓨팅