쿠버네티스를 활용한 다중 서비스 아키텍쳐 설계

전영민, 조민국, 송동훈, 서상민, 고석주*, 정명훈**
*경북대학교 컴퓨터학부,
**구글 코리아
e-mail : og365@naver.com

Multi-Service Architecture Design with Kubernetes

Young-Min Jeon, Min-Kook Jo, Dong-Hoon Song, Sang-Min Seo*,
Seok-Ju Ko**
*School of Computer Science&Engineering, Kyungpook Natioal University
**Google Korea

요 약

쿠버네티스는 구글에서 설계하고 리눅스 컨테이너 재단에서 관리하는, 세계에서 가장 빠르게 성장하는 오픈소스 프로젝트이다. 우리는 이 쿠버네티스의 컨테이너 오케스트레이션 기능을 십분 활용하여 웹페이지를 관리하고 음성 인식 IOT를 관리하는 두 가지 서비스를 하나의 아키텍쳐 설계 안에 녹여내었다.

1. 서론

쿠버네티스(이하 k8s)는 세계에서 가장 빠른 속도로 성장하고 있는 오픈소스 프로젝트이다. 지난 2014년 첫 출범한 이후로 Red hat, Core Os, Microsoft, IBM 등의 거대 IT기업이 주요 기여자로 참여하면서 자리를 확고히 하고 있다. 크고 빠르게 성장하는 생태계와 기술 지원 등이 용이하다는 매력 때문에 IT 인프라 서비스들은 지금 k8s로의 대이주를 준비 중이라고 해도 과언이 아니다.

1-1. 쿠버네티스란?

쿠버네티스(Kubernetes, K8s)는 자동화, 스케일링, 컨테이너화된 애플리케이션의 관리를 위한 오픈소스 프로젝트로서 Goolgle에 의해 설계되었고 Linux Foundation에 의해 관리되고 있다. 여러 Cluster들의 Host간 Application Container의 배치, Scaling, 운영 자동화 플랫폼 등등을 제공하는 것을 목표로 개발되었으며 도커(Docker)와 같은 일련의 컨테이너 툴과 함께 기능한다.

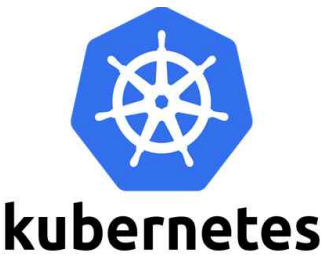

그림 1 쿠버네티스 로고

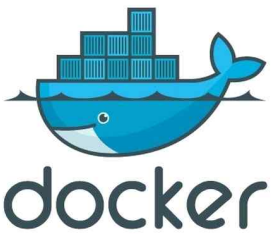

그림 2 도커 로고

언급한 컨테이너 배치나 스케일링과 같은 기능을 하나의 서비스 바운더리 안에서 통합적으로 운영하는 것을 컨테이너 오케스트레이션(Container Orchestration)이라 칭하는데, 쿠버네티스는 이것을 제공하는 통합 솔루션이다.

1-2. 도커와 컨테이너란?

도커 엔진은 리눅스 컨테이너(Linux Container)화된 응용프로그램의 배포를 자동화하여 기술자들이 쉽게 접근할 수 있도록 만든 오픈소스 프로젝트이다.

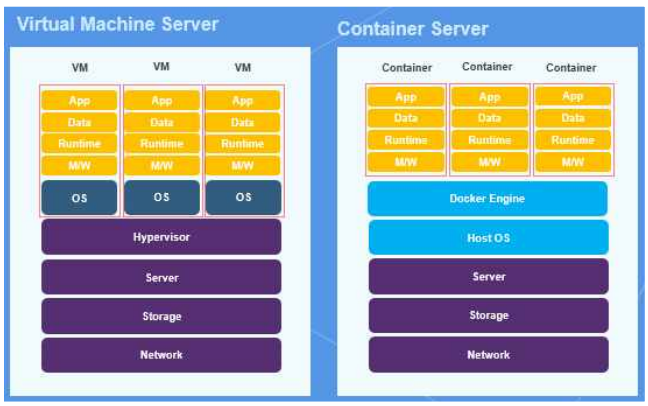

그림 3 Container와 VM의 차이

리눅스 컨테이너는 가상화(Virtualization)과 비슷한 기술이다. 가상화로 만든 VM을 구동하기 위해서는 하이퍼바이저(Hypervizor)가 반드시 필요하지만, 도커는 하이퍼바이저와 달리 게스트 OS를 두지 않는 대신 호스트 OS 커널을 사용한다. 하이퍼바이저 대신에 도커 엔진이 호스트 OS와 여러 애플리케이션을 연결해주는 역할을 한다.

따라서 도커를 사용하면 가상화보다 적은 일을 처리하면서도 애플리케이션을 조금 더 빠르고 효율적으로 실행시킬 수 있다. 업계에서는 가상화와 도커가 서로 부족한 부분을 채우는 보완 기술로 발전하리라고 기대한다.

2. 본론

본 프로젝트는 카페 등 자영업 매장에서 사용할 수 있는

통합형 IOT 솔루션을 제공한다고 가정하고, 그에 필요한 Application들을 컨테이너화해서 클라우드 환경에서 효율적으로 배포, 관리하는 방법을 제안한다.

이에 우리는 2가지의 Application을 구현했다. 첫 번째 Application은 고객이 음성인식을 통해 앉은 자리에서 주문을 하고, 그 주문을 주방에 있는 컴퓨터에 전달한다. 두 번째 Application은 매장 관리자가 매장 내의 조명, 전자기기 등을 음성으로 제어할 수 있게 해준다.

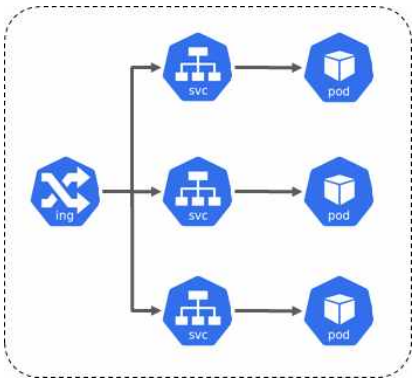

그림 4 초기 프로젝트 구성도

프로젝트의 구조는 하나의 Ingress와 3개의 서비스로 구성하였다. Ingress는 Application Layer(L7) HTTP(S)기반의 Load Balancing을 해주는 컴포넌트이다. Ingress는 URL Path를 기반으로 서로 다른 Service로 라우팅 한다. 그리고 그 Service들은 Pod을 외부에서 접근하게 해준다.

2-1. 다중 컨테이너 단일 Pod 배포

초기에 Pod을 배포할 때, Kubernetes Component의 가장 작은 단위인 Pod 안에 Server Container, Database Container를 함께 배치했다.

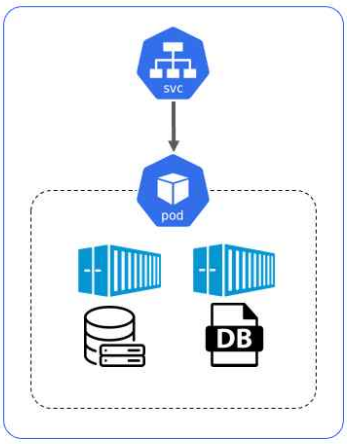

그림 5 초기의 Pod 구성도

하지만 이러한 방식은 몇 개의 문제를 만들어 낼 소지가 있었다. 예를 들어, Server Container에 문제가 생겨서 서버를 잠시 중단시켜야 할 때, Database 컨테이너도 함께 중단되기 때문에 Pod 관리에 있어 상당한 비효율을 감수해야한다. 또한, Scaling시에도 심각한 문제가 발생할 수 있다. Database Container는 변경하지 않고 Server Container의 수를 늘리려고 할 때, 위와 같은 구조 안에서는 Database Container도 같이 복제된다.

또한 Container의 Stateless한 특성상 런타임 중에 발생한 내부 변경사항은 Container가 삭제될 때 함께 삭제되는데, 이 또한 치명적인 오류로 기능할 가능성이 있다.

2-2. 단일 컨테이너 단일 Pod 배포

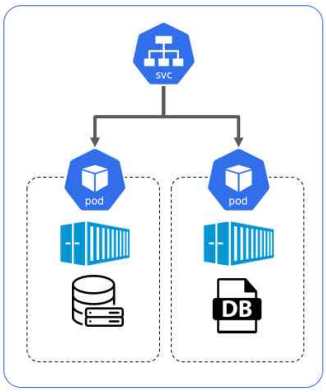

그림 6 수정된 Pod 구조도

따라서 우리는 윗 단락과 같은 문제들을 개선하고자 구조를 변경했다. 초기 설계에서 하나의 Pod 안에 들어있던 Server Container와 Database Container를 각각 하나의 Pod 안에 배치했다. 이런 방식으로 배포하게 되면, 둘 중 한 Pod에 문제가 발생했을 때 그 Pod만 따로 수정하면 되고, 다른 Container에 영향을 끼치지 않는다. 또한 기능별 Pod을 독립적으로 Scaling할 수 있다.

물론 통신 속도의 빠르기라는 기준으로 구조도를 바라볼 때에는 전자의 설계가 조금이나마 더 나을 수 있다. 같은 Pod 내부에서 2개의 Container가 통신하는 것이 아무래도 서비스를 거쳐서 서로 다른 Pod들이 통신하는 것보다는 빠르기 때문이다. 본 프로젝트와 같은 소규모 프로젝트에서 Server로 들어오는 Traffic량이 그렇게 크지 않아 속도 차이가 크게 나지 않지만, 프로젝트 규모가 커져서 Traffic량이 엄청나게 많아진다면, 빠른 Traffic 처리를 위해서 전자와 같은 구조도 고려해볼 수 있다.

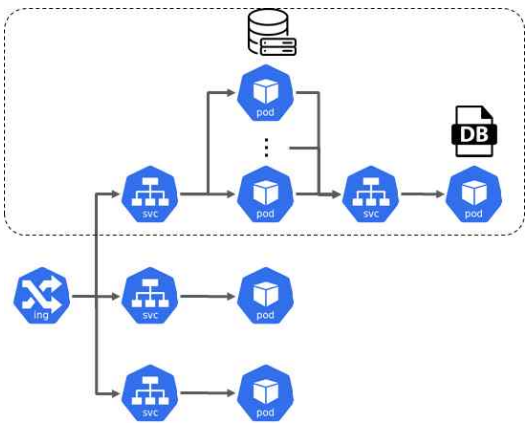

그림 7 수정을 거친 최종 프로젝트 구성도

3. Pod간의 통신

본 프로젝트에서 앞서 제작한 Server Pod는 DataBase Pod로부터 데이터를 읽어 와야 한다. 이러한 통신을 하기 위해서 두 Pod는 서로의 IP를 미리 알고 있어야 한다.

```
thdehdgns@cloudshell:~/capstone (kubernetes-235111)$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
mysql-app-675884897b-7hvjg   1/1       Running   0          3h
node-app-b7f4798d8-4x8wm     1/1       Running   0          2h
thdehdgns@cloudshell:~/capstone (kubernetes-235111)$ kubectl get deployments
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-app   1         1         1            1           3h
node-app    1         1         1            1           2h
thdehdgns@cloudshell:~/capstone (kubernetes-235111)$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)          AGE
kubernetes   ClusterIP      10.15.240.1     <none>         443/TCP          3h
mysql-app    NodePort       10.15.250.188   <none>         3306:30036/TCP   3h
node-app     LoadBalancer   10.15.243.169   34.80.206.42   80:31976/TCP     2h
thdehdgns@cloudshell:~/capstone (kubernetes-235111)$
```

그림 8 Kubernetes Resources

Pod는 생성 시 쿠버네티스로부터 내부 IP를 부여받는다. 그런데 Pod의 IP는 Pod가 Node에 배포될 때마다 바뀌기 때문에, Pod의 IP로 통신을 할 경우 유동적으로 바뀌는 Pod의 IP를 알 수 없다. 이러한 내부 Pod간의 통신을 위해서 Kubernetes에서는 Cluster내부에서 사용할 수 있는 고유 DNS를 제공한다.

이 DNS는 Service와 Pod에 대해 사용할 수 있는데, 특정 Pod에 접근하는 도메인에는 Pod의 IP가 들어가기 때문에 도메인 네임의 장점이 사라진다. 따라서 본 프로젝트에서는 Service에 부여된 DNS를 이용해 Service와 연결된 Pod에 접근하는 방법을 선택했다. 특정 Service에 접근하는 도메인은 아래 그림과 같이 구성된다.

[Name of the Service].[Name of the namespace].svc.cluster.local

그림 9 DNS 구성

아래 사진에서 node-app이라는 이름의 LoadBalancer Service, mysql-app이라는 이름의 NodePort Service가 돌아가고 있다.

Server pod에서 “mysql-app.default.svc.cluster.local” 이라는 host로 연결을 시도하면 mysql-app Service를 거쳐 Database pod에 접근할 수 있다.

```
"database": "iot",
"host": "mysql-app.default.svc.cluster.local",
"dialect": "mysql"
```

그림 10 Server pod의 config파일

4. ReplicaSet

본 프로젝트는 테스트이므로 실사용 유저는 10명 이하이지만, 만약 실제 서비스의 배포를 고려한다면 실사용자는 이보다 훨씬 많을 것이다. 그러므로 쿠버네티스 설계하는 단계에서는 유저수를 대비하여 트래픽 분산처리와 비정상적 서비스 종료를 예방하기 위한 대책을 고려해야 한다.

이러한 점을 고려하여 우리는 본 프로젝트에 ReplicaSet을 사용하였다. ReplicaSet은 Pod을 지정한 수만큼 복제하는 기능을 가진다. 하나의 Pod을 이용하는 사람의 수가 많을 경우 그 ServerPod에 과도한 트래픽이 몰릴 가능성이 있다. ReplicaSet을 통하여 미리 여러 개의 복제 Pod을 만들어두면 이후 Load Balancer를 통해 한 곳에 모여든 트래픽을 분산시켜 속도저하를 방지할 수 있다.

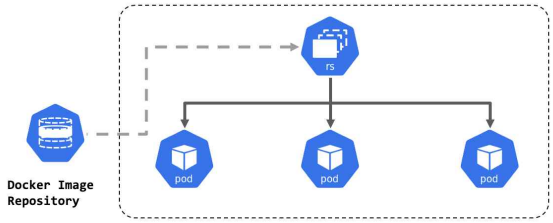

그림 11 ReplicaSet 구조도

또한 ReplicaSet는 복제된 Pod이 비정상적으로 종료될 경우 그것을 다시 복원시켜 항상 일정한 Pod의 수를 유지시키는 기능도 가지고 있다. 본 프로젝트는 설계 단계에서부터 실제적인 서비스 배포의 가능성을 염두에 두고 진행하였기 때문에, ReplicaSet를 사용하여 트래픽 처리와 안정성 이슈에 동시에 대비하려고 하였다.

5. CI/CD

앞서 언급했듯이, Server와 DataBase를 같은 Pod에 배포할 시에는, 오직 Server에만 해당하는 수정사항이 있더라도 Server와 DataBase가 같은 Pod에 배포된 상태이므로 수정하기 위해서는 Pod 전체를 Node에서 내려야 한다.

그런데 이때, 컨테이너의 Stateless한 특성상 수정을 위해 Node에서 Pod을 내릴 경우 DB의 자료가 소실되는 현상이 발생할 수 있다. 이런 경우에 대비하기 위하여 우리는 처음부터 Application을 2개의 Pod에 나누었다. 그리하여 Server에 변경사항이 있을 경우 Server의 이미지를 수정하고, 빌드와 테스트 과정을 거쳐 Dockerfile로 만들어야하며, 이를 다시 Pod으로 배포를 하여야 한다.

하지만 이런 작은 과정 하나하나에 개발자가 개입하기에는 터무니없을 정도로 많은 시간이 요구된다. 이런 애로사항의 해결을 위해 쿠버네티스에서는 CI/CD Application을 지원하며, 본 프로젝트에서는 이를 적극 활용하였다.

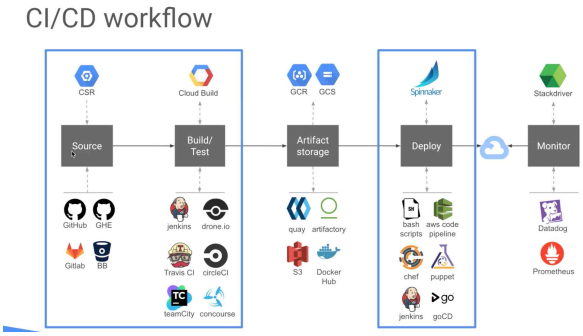

그림 12 CI/CD 과정

본 프로젝트에서는 고객의 요청에 따라 웹 클라이언트의 UI 변경과 WAS의 코드 수정 가능성이 있다. 때문에 효율적인 설계를 위해서라도 CI/CD를 통한 자동화 시스템이 반드시 필요하다. CI/CD는 고객의 피드백을 적극 수용하

고 즉각적으로 반영할 수 있고, 빌드와 테스트 과정을 자동화하여 개발 시간을 줄일 수 있다는 장점이 있다.

5. 결론

본 프로젝트에서 설계한 서비스는 내장된 오토 스케일링 기능을 통해 Pod의 수를 조절해주는 Replicaset이 트래픽 상황에 따라 유동적으로 Pod의 수를 복제하여 조절해주며 지정된 수만큼 Pod의 실행을 보장해준다.

LoadBalancer Service를 이용해 복제된 Pod으로 트래픽분산 처리 또한 자동으로 해주며, 예기치 않게 Pod이 다운되더라도 복구해주기 때문에 쿠버네티스를 사용하지 않은 시스템보다 쿠버네티스를 사용하는 편이 훨씬 안정적이라는 장점이 있다. 이런 이유로 사용자가 많은 대규모 시스템에서 큰 효과를 볼 수 있다. 특히, Pod과 Pod간의 DNS를 활용한 통신은 Pod이 어느 Node에 배치되더라도 통신을 보장해준다는 점에서 매우 효과적이다.

또한 쿠버네티스는 내부 Virtual Machine에 가해지는 부하를 막기 위해 자체적으로 Node의 Pod들의 위치를 바꿔주기 때문에 매우 안정적이다. 또한 설계를 마친 다음부터는 직접 프로젝트를 진행하며, ReplicaSet과 CI/CD 등의 기능을 활용하여 통신은 물론이고 트래픽 처리 속도와 안정성을 동시에 취하고자 하였다.

쿠버네티스를 활용한 컨테이너 관리는 트래픽 분산 처리, 오류 수정, 재배포 등 서비스 관리가 용이하다. 이런 수많은 컨테이너들을 오케스트레이션 하지 않았을 경우, 각각의 컨테이너들을 관리하기가 무척 힘들고 비효율적이다. 본 프로젝트에서는 간소화하여 2개의 서비스만 제공하지만, 실제 Client에게 제공될 IOT 서비스는 거의 수십 가지이며, 계속해서 추가될 것이다. 컨테이너가 늘어나더라도 쿠버네티스는 이를 매우 손쉽고 간편하게 배포 및 관리할 수 있다.

Cluster 내부 Virtual Machine에 부하가 갈 경우 쿠버네티스가 자체적으로 판단하여 내부 Pod들을 다른 Node로 배치하여 부하를 최소화하는 기능 역시 시스템 안정화에 기여한다. 그리고 Pod과 Pod간의 DNS를 활용한 통신은 Pod이 어느 Node에 배치되더라도 통신을 보장해준다는 점에서 매우 효과적이다. 이런 이유로 사용자가 많은 대규모 시스템에서 큰 효과를 볼 수 있다.

현재 많은 국내 기업들이 컨테이너 기반의 설계를 주저하고 있지만, 이미 유수의 해외 기업들은 컨테이너 기반의 인프라를 받아들였다. 국내도 빠른 시일 이내에 컨테이너 기반의 아키텍처로 바뀌게 될 것이나, 쿠버네티스 개발자들을 위한 양질의 지식과 경험은 턱없이 부족한 상태이다.

부족하지만 우리의 짧은 기록이, 쿠버네티스를 익혀보려는 동료 개발자들을 조금이나마 돕게 된다면 더할 나위 없이 기쁘겠다는 생각으로 이 글을 마친다.

Acknowledgment

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학 사업의 연구결과로 수행되었음 (2015-0-00912)

참고문헌

[1] "Kubernetes Documentation," kubernetes.io, 2019년 2월 22일 수정, 2019년 03월 19일 접속, https: //kubernetes.io/docs/home/.

[2] "쿠버네티스#3- 개념이해(2/2) 컨트롤러," 조대협의 블로그, 2018년 05월 30일 수정, 2019년 03월 20일 접속, https://bcho.tistory.com/1257?category=731548.

[3] "3장 네트워킹, 로드밸런서, 인그레스," 조나단 바이에르, 쿠버네티스 기초 다지기 2/e, 1호, pp. 2018.