쿠버네티스 환경에서 웹 기반의 마이크로 서비스 배포 시스템

신정은O, 권재환*, 김명호**
O숭실대학교 소프트웨어학부,
*숭실대학교 융합소프트웨어학과,
**숭실대학교 소프트웨어학부
e-mail: jungeun9729@gmail.comO, jaehwan@soongsil.ac.kr*, kmh@ssu.ac.kr**

Web-based microservice deployment system in kubernetes environment

Jung-Eun ShinO, Jae-Hwan Kwon*, Myung-Ho Kim**
ODept. of Software, Soongsil University,
*Dept. of Convergence Software, SoongSil University,
**Dept. of Software, Soongsil University

● 요 약 ●

최근 마이크로서비스 아키텍처(Microservice Architecture)를 도입하는 프로젝트가 많아짐에 따라 컨테이너 오케스트레이션 도구인 쿠버네티스의 필요성이 증대되고 있다. 쿠버네티스 환경을 구성하고 서비스를 테스트 및 배포하기 위해서는 클러스터링을 하고 yaml파일을 작성하고 적용하여 자원을 생성하는 등의 작업이 필요하다. 이렇게 반복되는 작업을 자동화하기 위해 기존에는 쉘스크립트를 사용하는데, 이를 사용하기 위해서는 쿠버네티스 및 인프라에 관한 기초지식이 있어야 하는 문제점이 있다. 본 논문에서는 마이크로서비스를 위한 웹 기반의 배포 시스템을 제안한다. 웹을 통해 배포하고 테스트할 수 있어 기초지식이 없어도 사용할 수 있다. 웹에서 프로젝트 개발자로부터 도커 이미지와 사용할 포트번호 및 레플리카 개수를 입력받아 배포를 요청하면 쿠버네티스 환경에 배포하고, 필요한 자원들이 배포가 완료되면 엔드포인트를 제공한다. 그리고 배포된 서버의 엔드포인트에 요청을 전송하고 응답이 오는 것을 확인할 수 있어 활용성이 높다. 이 시스템은 쿠버네티스를 이용하여 복수 개의 파드(Pod)를 유지하여 가용성을 보장하고 빠르게 배포가 가능하게 한다. 또한 이미지가 업데이트되면 웹에서 요청하여 클러스터에 새 버전의 이미지로 배포한다.

키워드: 쿠버네티스(Kubernetes), 마이크로서비스(Microservice), 도커(Docker)

I. 소개

최근에는 마이크로서비스 단위의 테스트 및 배포를 위해 컨테이너를 사용한다[1],[2]. 개발환경과 동일한 환경 제공을 위해 주로 도커(Docker) 이미지를 빌드하고 도커 이미지를 컨테이너 오케스트레이션 도구인 쿠버네티스 환경에 배포한다. 단일 서버로는 가용성을 보장하기가 힘들기 때문에 쿠버네티스 환경에 배포하여 가용성을 보장할 수 있다[3]. 그리고 실제 운영되는 환경에 배포하기 전 여러 번의 테스트를 위해 배포하는 과정에서 쉘스크립트 코드를 작성하여 자동화한다. 그러나 이 방법은 기초지식이 부족한 사용자에게는 적합하지 않다. 따라서 본 논문은 사용성 증대를 위해서 웹을 통해 쿠버네티스 환경에 도커 이미지를 자동 배포하는 시스템을 제안한다. 프로젝트 개발자가 웹에 도커 이미지명과 이미지에서 사용하는 포트 정보 및 레플리카 개수를 입력하고 배포버튼을 누르면 쿠버네티스에 해당 이미지의 deployment, service와 ingress가 생성된다. 배포에서 그치는 것이 아니라 웹을 통해 배포된 서버에 테스트를 요청하고 응답을 확인할 수 있다.

II. 기존 방식

기존의 수동배포 방식으로 쿠버네티스 환경에 이미지를 배포하기 위해서는 우선 리소스 배포를 위한 yaml파일을 구성한다.

디플로이먼트(Deployment), 서비스(Servcie), 인그레스(Ingress) 3개의 리소스를 위한 yaml파일을 구성한다. 그리고 쿠버네티스 환경

에서 yaml파일을 차례로 apply해서 리소스를 생성한다. 배포가 완료되면 테스트를 위해 별도로 구성한 엔드포인트에 요청을 전송하여 확인해야한다.

또 다른 기존 방식으로는 쉘스크립트를 작성하는 방식이 있다. 시간은 처음 작성해 놓으면 얼마 걸리지 않지만, 이 방식 역시 쿠버네티스 기초 지식이 필요하다. 또한 테스트를 위해 별도로 요청을 전송하여 확인해야한다.

제안하는 방식은 개발자가 웹에 사용할 이미지와 포트 번호 및 배포할 레플리카 개수를 입력하여 배포버튼을 누르면 생성해야 하는 자원을 순서대로 쿠버네티스 API Server에 배포를 요청한다. 필요한 배포가 완료되면 웹은 엔드포인트 및 API 테스트 페이지를 제공한다. 테스트도 웹에서 파라미터 및 페이로드를 입력하여 요청하고 응답을 받아볼 수 있다.

Table 1. 기존 방식과 비교

	수동방식	스크립트방식	제안방식
쿠버네티스 기초지식	필요	필요	불필요
시간	10분	1분 미만	1분 미만
요청 방식	별도로 구성	별도로 구성	웹

III. 제안 및 구성

본 논문에서 제안하는 가용성을 보장하는 자동배포 시스템 구축을 위해서 쿠버네티스 클러스터 환경을 구성해야한다. Amazon Web Service에서 제공하는 Elastic Kubernetes Service를 사용하여 쿠버네티스 환경을 구성한다[4].

Fig1의 ①과 같이 클러스터의 네트워크 환경을 위해 VPC를 생성하고 서울 지역의 가용영역 a와 c에 각각 하나씩 서브넷을 생성하고 서브넷을 VPC 영역에 포함시킨다. 그리고 웹에서 쿠버네티스 자원 생성 요청하려면 쿠버네티스 API Server에 요청한다. 따라서 라우트 테이블을 생성하고 각 서브넷을 등록한 후에 인터넷 게이트웨이를 생성하여 외부에 노출한다.

Fig1의 ②와 같이 클러스터를 운영하기 위해서 필요한 정책을 부여한 IAM을 생성한다. 그리고 보안그룹을 생성하고 모든 곳으로 요청이 나갈 수 있도록 이그레스(egress)를 우선 설정한다. 그리고 생성한 IAM과 정책(Policy), 그리고 VPC설정을 가지고 EKS 클러스터를 생성한다. VPC를 설정할 때 엔드포인트에 퍼블릭과 프라이빗 접근을 모두 허용한다. 워커 노드 구성 및 역할에 필요한 정책을 부여한 IAM 생성한다. 그리고 VPC에 속하는 보안그룹을 생성한다. 노드에서도 모든 곳으로 요청이 나갈 수 있도록 이그레스를 설정한다. 그리고 노드끼리의 통신과 EKS와의 통신을 위한 인그레스(ingress)를 설정한다. 설정에는 보안그룹 ID를 명시한다. 그리고 EKS도 워커 노드에서 오는 요청을 허용하기 위해 앞에서 설정한 인그레스를 설정한다. 모두 설정을 하고 앞에서 만든 자원들과 오토스케일링 그룹(Autoscaling Group)을 이용하여 워커 노드로 사용될 인스턴스 3개를 생성한다.

클러스터 구성은 완성되었지만, 외부에서 운영 중인 서비스에 요청을 보낼 수 없는 상태이다. 따라서 Fig1의 ③처럼 AWS의 인그레스 컨트롤러(Ingress Controller)와 네트워크 로드밸런서(Network Loadbalancer)를 생성한다. 인그레스 컨트롤러를 위해 Nginx를 사용하고 Nginx를 위한 서비스와 헬스체크를 위한 서비스를 생성하고 노드포트를 열어둔다. 네트워크 로드밸런서에 두개의 서브넷을 등록하고, 리스너를 생성하여 외부에서 들어오는 요청을 받는다. 그리고 로드밸런싱을 할 대상 그룹을 생성하고 대상으로는 위에서 생성한 3개의 워크노드를 설정한다. 이 때 인스턴스의 ID가 아닌 IP로 해야 보안그룹에 인그레스 설정을 VPC 네트워크 주소로 지정할 수 있다.

쿠버네티스 클러스터 구성이 완료되면 웹을 만들어 웹 기반의 자동배포 시스템을 구축한다[5]. 사용자가 웹에서 배포를 요청하면 클러스터에 요청을 전송하여 디플로이먼트(deployment), 서비스(service)와 인그레스(ingress)를 생성한다. 이 때 자원들은 사용자명의 네임스페이스(namespace)에 배포한다. 그리고 자원들의 이름을 웹에서 사용자가 설정한 레포지토리 명으로 관리한다.

Fig2와 같이 생성할 레포지토리명과 Public 레포지토리에 저장되어 있는 도커 이미지명, 포트번호, 레플리카 개수를 입력한다. 최신의 코드를 반영하므로 태그는 latest인 도커 이미지를 가져온다.

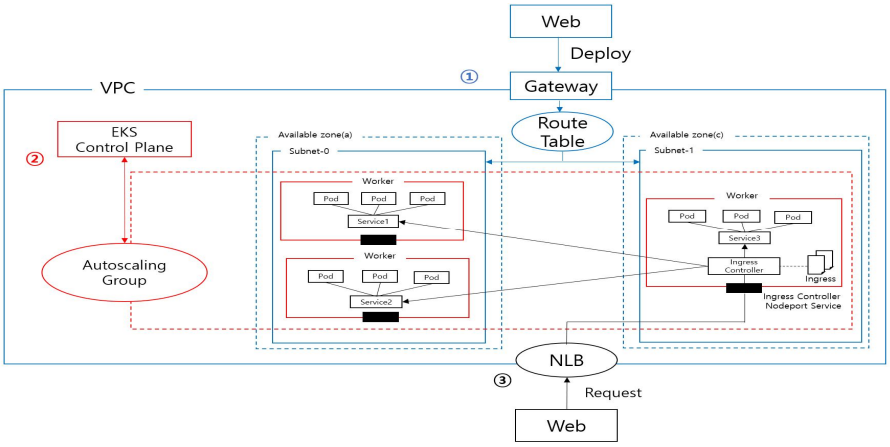

Fig. 1. 쿠버네티스 클러스터 구성

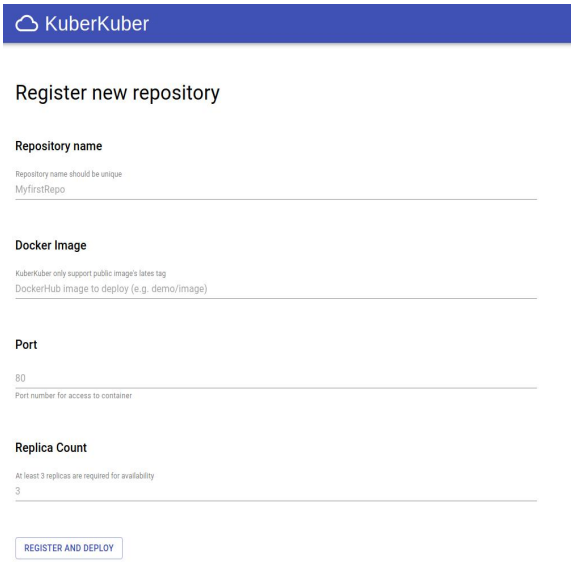

Fig. 2. 레포지토리 생성페이지

deploy버튼을 누르면 우선 사용자의 다른 레포지토리들과 요청하는 레포지토리명의 유효성을 검사하여 네임스페이스내에서 자원의 .metadata.name이 중복되지 않게 설계한다.

해당 도커 이미지가 업데이트되면 웹에서 클러스터에 요청을 전송하여 기존의 파드를 새 이미지의 파드로 중단 없이 배포한다. 마지막으로 Kubernetes Client API를 사용해서 자원 생성을 요청한다. yaml파일로 나타내면 다음과 같다.

Table 2. deployment 생성

```
POST /namespaces/{namespace}/deployments
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {repo-name}
  namespace: {namespace}
  labels:
    app: {repo-name}
spec:
  replicas: {replica-count}
  selector:
    matchLabels:
      app: {repo-name}
  template:
    metadata:
      labels:
        app: {repo-name}
    spec:
      containers:
      - name: {repo-name}
        image: {image-name}
        ports:
        - containerPort: {port-number}
```

Table 3. Service 생성

```
POST /api/v1/namespaces/{namespace}/services
apiVersion: v1
kind: Service
metadata:
  name: {repo-name}
  namespace: {namespace}
spec:
  type: ClusterIP
  selector:
    app: {repo-name}
  ports:
  - protocol: TCP
    port: {port}
    targetPort: {port}
```

Table 4. Ingress 생성

```
POST /namespaces/{namespace}/ingresses
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: {ingress-name}
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - host: {host-url}
    http:
      paths:
      - path: /<:namespace>/<:repo-name>(/|$)(.*)
        backend:
          serviceName: {repo-name}
          servicePort: {port-number}
```

host는 앞에서 생성한 AWS 네트워크 로드밸런서의 DNS로 설정한다. rewrite-target을 설정하여 서비스로 요청이 갈 때는 /namespace/repo-name 이후의 URI로 요청한다.

IV. 실험 및 결과

Fig3과 같이 배포된 레포지토리의 상태를 웹에서 확인할 수 있다. Fig4와 Fig5는 상세 페이지이며 개발자가 등록한 엔드포인트 리스트들을 확인할 수 있고 각 엔드포인트의 파라미터(parameter)나 페이로드(payload)를 수정해서 실제 운영 중인 서버에 요청을 전송하고 오는 응답을 확인할 수 있다.

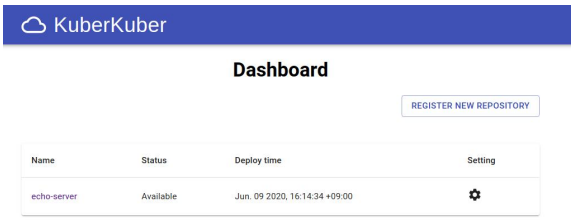

Fig. 3. 자동배포된 레포지토리 리스트

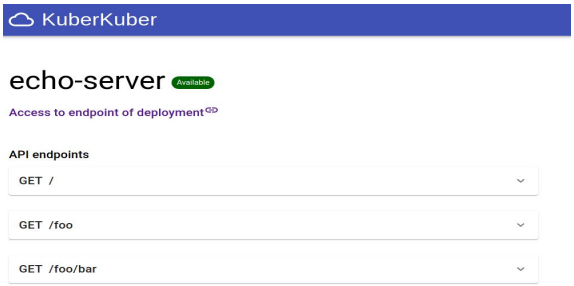

Fig. 4. 레포지토리 상세 페이지

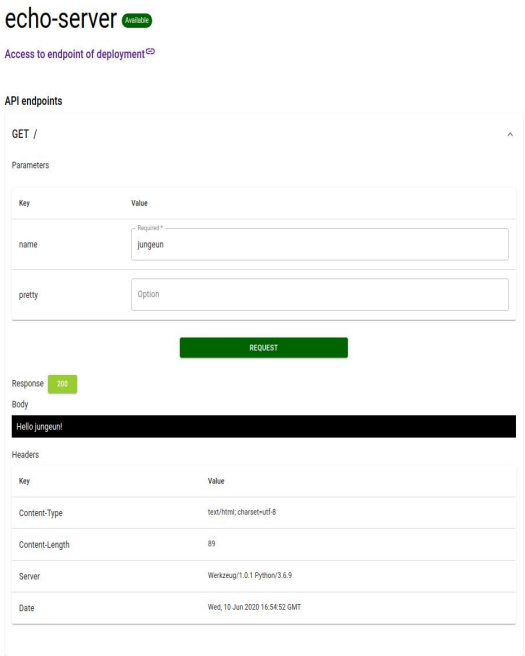

Fig. 5. 상세페이지 요청 및 응답

V. 결론

본 논문에서 제안하는 쿠버네티스를 이용한 웹 기반의 자동배포 시스템은 한번 구축하면 기존의 수동배포에 비해 시간을 단축할 수 있으며 이미지 업데이트 시 자동 배포가 가능하다. 쉘스크립트를 통한 배포에 비해서는 인프라 관련한 기초지식이 없어도 가용성을 보장하는 쿠버네티스 환경에 배포할 수 있다. 그리고 제안하는 시스템을 통해 설정해놓은 테스트 케이스들을 웹에서 바로 요청을 전송하고 응답을 받아서 확인할 수 있다. 향후 연구과제로는 시크릿을 사용하여 프라이빗 이미지를 배포하는 방향과 도커 컴포즈를 통한 배포방식을 추가하는 방향이 있다.

ACKNOWLEDGEMENT

“본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업의 연구결과로 수행되었음(2018-0-00209)”

REFERENCES

[1] H. Kang, M. Le, S. Tao, "Container and microservice driven design for cloud infrastructure devops." 2016 IEEE International Conference on Cloud Engineering (IC2E), pp.202-211, IEEE, 2016.

[2] V. Singh, SK. Peddoju, "Container-based microservice architecture for cloud applications." 2017 International Conference on Computing, Communication and Automation (ICCCA), pp.847-852 IEEE, 2017.

[3] LA. Vayghan,MA Saied, "Deploying microservice based applications with Kubernetes: experiments and lessons learned." 2018 IEEE 11th international conference on cloud computing (CLOUD), pp.970-973, IEEE, 2018.

[4] Amazon Web Service document (website) https://docs.aws.amazon.com

[5] Kubernetes (website) https://kubernetes.io