

오픈소스SW실습

Docker Compose & Kubernetes

6주차

24.10.10



목차

1 Docker Compose

2 Kubernetes

Minikube 설치 방법

- **Minikube란?**

- 로컬에서 쿠버네티스를 실행하기 위한 경량화된 도구
- 개발 및 테스트 용도로 많이 사용됨

- **설치 단계**

- ① **Minikube 설치**

- <https://minikube.sigs.k8s.io/docs/start/> 방문 후 OS에 맞는 설치 방법을 참고하여 설치

- ② **Kubectl 설치**

- <https://kubernetes.io/docs/tasks/tools/>에서 OS에 맞는 설치 방법을 참고하여 kubectl 설치

- ③ **Minikube 시작**

- 설치가 완료된 후, 터미널에서 `minikube start` 명령어로 Minikube를 실행

도커 컴포즈

Docker Compose

다중 컨테이너 및 도커 네트워크

- MySQL 컨테이너와 Python 컨테이너를 실행
- 네트워크를 생성하여 두 컨테이너 간 통신
- 그럼 여러 개의 컨테이너를 실행할 때, 커맨드를 일일이 입력해야 할까?

목표: 로그인, 회원가입, 방명록 기능을 가진 웹 App

- **컨테이너 1: MySQL**

- 회원의 id, password를 관리하는 테이블
- 방명록이 기록된 txt 파일의 이름을 관리하는 테이블

- **컨테이너 2: Flask(Python)**

- 파이썬으로 작성된 웹 애플리케이션
- 로그인 (메인 페이지), 회원가입, 방명록 페이지 제공

도커 네트워크 생성 명령어

```
docker network create my-network
```

- **my-network**: 생성할 네트워크 이름

MySQL 컨테이너 실행 명령어 예시

```
docker run --name mysql-container --network my-network
-e MYSQL_ROOT_PASSWORD=1234
-e MYSQL_USER=testuser
-e MYSQL_PASSWORD=1234
-v "[실습 폴더 내 init.sql 절대 경로]":/docker-entrypoint-initdb.d/init.sql
-v mysql-data:/var/lib/mysql
-d mysql:latest
```

- **-e** 옵션으로 환경변수 설정 (루트 비밀번호, 사용자 계정 등)
- **-v** 옵션으로 호스트 파일을 컨테이너 내부로 마운트 (SQL 스크립트, 데이터 저장소)
- **--network**로 지정된 도커 네트워크에서 컨테이너 실행
- **-d** 옵션으로 백그라운드 실행

MySQL 컨테이너 실행 명령어 예시

```
> docker network create my-network
0c739443a4be892aaabd6167cd867eb3486b7c57d5aa544e244a56185d337677
> docker run --name mysql-container --network my-network \
-e MYSQL_ROOT_PASSWORD=1234 \
-e MYSQL_USER=testuser \
-e MYSQL_PASSWORD=1234 \
-v "/Users/wd_seo/Desktop/kgu/lecture/2024-2/오픈소스SW실습/lecture_note/week6/code/실습1/mysql/init.sql":/docker-entrypoint-initdb.d/init.sql \
-v mysql-data:/var/lib/mysql \
-d mysql:latest
f5ca74490102509165748c6f0c22d4ad7234b372f7ccb0696d90e72bd73c9419
```

Flask 컨테이너 실행 명령어 예시

이미지 빌드

```
docker build -t flask-app .
```

컨테이너 실행

```
docker run --name flask-container --network my-network
-e FLASK_PORT=5001
-e MYSQL_HOST=mysql-container
-e MYSQL_USER=testuser
-e MYSQL_PASSWORD=1234
-e MYSQL_DATABASE=testdb
-p 5001:5001

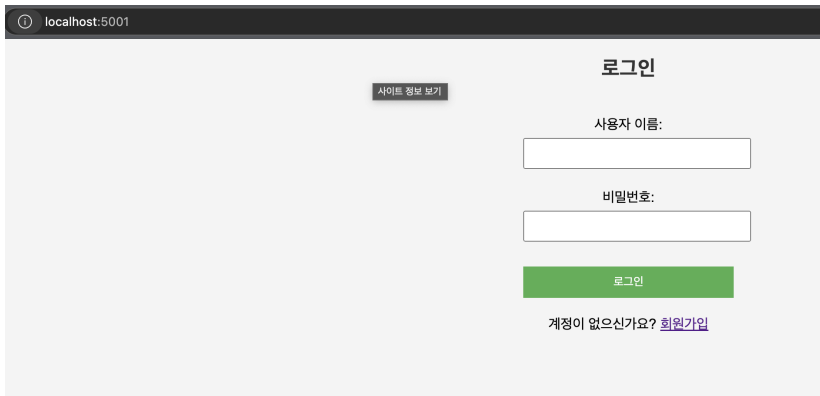
-v guestbook_volume:/app/guestbook_entries flask-app
```

Flask 컨테이너 실행 명령어 예시

```
> docker run --name flask-container --network my-network \
-e FLASK_PORT=5001 \
-e MYSQL_HOST=mysql-container \
-e MYSQL_USER=testuser \
-e MYSQL_PASSWORD=1234 \
-e MYSQL_DATABASE=testdb \
-p 5001:5001 \
-v guestbook_volume:/app/guestbook_entries flask-app
```

웹 브라우저에서 접속

`http://localhost:5001`



The screenshot shows a web browser window with the address bar displaying 'localhost:5001'. The page content includes a '로그인' (Login) title, a '사이트 정보 보기' (View Site Info) link, and two input fields for '사용자 이름:' (Username) and '비밀번호:' (Password). Below the password field is a green '로그인' (Login) button. At the bottom, there is a link for '계정이 없으신가요? [회원가입](#)' (Don't have an account? Sign up).

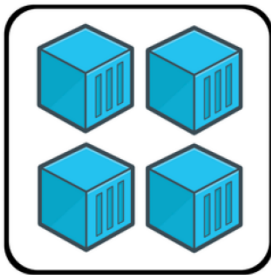
문제점

- 여러 개의 컨테이너를 실행할 때, 각 컨테이너마다 개별적으로 `docker run` 명령어를 실행해야 함
- 네트워크, 볼륨, 환경 변수 등 여러 설정을 수동으로 관리해야 하므로 복잡도가 증가
- 컨테이너 간의 의존성을 고려해 적절한 순서로 실행해야 하며, 이를 일관되게 관리하기 어려움
- 컨테이너를 재시작하거나 전체 환경을 재구성할 때 많은 반복 작업 필요

Docker Compose



Docker



Docker Compose

Docker Compose

- Docker Compose는 여러 컨테이너를 한번에 **build, run, stop**할 수 있도록 도와주는 도구
- 네트워크, 볼륨, 환경 변수 설정을 **한 번에 정의**하여 반복 작업을 줄일 수 있음
- 컨테이너 간의 **의존성**을 관리하고, 적절한 순서로 컨테이너를 실행 가능
- 단일 명령어 `docker-compose up`을 통해 모든 컨테이너를 쉽게 시작, 정지 및 관리 가능

주의할 점

- Docker Compose는 **도커 데몬이 실행 중인 환경에서만 사용 가능**
- Docker Compose는 Dockerfile을 대체하는 것이 아님
- Docker Compose는 이미지와 컨테이너를 대체하는 것이 아님
- Docker Compose는 여러 컨테이너를 하나의 호스트에서 관리하는데 적절함

Docker Compose 파일 구조

Docker Compose 파일의 기본 구조는 각 서비스(컨테이너)를 정의하고 관리하기 위한 상위 레벨 요소들로 구성

- **version** : Docker Compose 파일의 버전을 지정. 버전 3.x가 일반적으로 많이 사용되며, 각 버전은 제공하는 기능이 다름
- **services** : 여러 컨테이너를 정의하는 부분. 각 컨테이너는 하나의 서비스로 취급되며, 서비스의 이름을 지정하고 각 컨테이너의 설정을 세부적으로 작성
- **volumes** : 컨테이너의 데이터를 호스트 머신에 영구적으로 저장하기 위한 볼륨을 정의하는 부분. 각 컨테이너에서 사용할 볼륨을 이 섹션에 정의하고 참조
- **networks** : 컨테이너 간의 통신을 관리하기 위한 네트워크를 정의. Compose 파일 내에서 컨테이너들이 서로 소통할 수 있도록 네트워크를 생성하고, 각 컨테이너에 이를 연결

services의 세부 구조

- **image** : 컨테이너를 실행할 때 사용할 도커 이미지를 지정. 이미 존재하는 이미지를 사용할 때 사용
- **container_name** : 컨테이너의 이름을 지정. 컨테이너의 이름을 지정하지 않으면, Compose가 자동으로 이름을 생성
- **context** : 이미지를 빌드할 때 사용할 Dockerfile의 경로를 지정. 이미지를 빌드할 때 사용 (절대 경로 또는 상대 경로)
- **args** : Dockerfile 내에서 사용할 빌드 인수를 정의. Dockerfile 내에서 ARG 명령어를 사용하여 빌드 인수를 사용할 수 있음
- **build** : 컨테이너 이미지를 직접 빌드할 때 사용. Dockerfile의 경로나 빌드 컨텍스트를 지정하여 이미지를 생성
- **ports** : 컨테이너 내부와 호스트 시스템 간의 포트 매핑을 설정. "호스트포트:컨테이너포트" 형태로 지정하여 외부에서 컨테이너의 서비스를 접근 가능하게 함

services의 세부 구조

- **environment** : 컨테이너 실행 시 필요한 환경변수를 지정.
KEY=VALUE 형식으로 작성하여 컨테이너 내부에서 사용할 수 있는 변수를 설정
- **volumes** : 호스트와 컨테이너 간의 파일 시스템을 공유하는 볼륨을 연결. 데이터를 영구적으로 저장하거나, 컨테이너 재시작 시에도 데이터를 유지하기 위해 사용
- **networks** : 컨테이너가 연결될 네트워크를 지정. 다른 서비스와의 통신을 위해 네트워크에 컨테이너를 연결하여 서비스 간 소통을 가능하게 함
- **depends_on** : 한 서비스가 다른 서비스에 의존하는 경우, 해당 서비스를 먼저 실행할 수 있도록 지정. 예를 들어, Flask 애플리케이션은 MySQL 데이터베이스가 실행된 후에 실행되어야 하므로 `depends_on` 설정을 통해 의존 관계를 관리

YAML 파일 작성 기본 규칙

- **파일 이름:** `docker-compose.yaml`로 파일 이름을 지정해야 Docker Compose가 자동으로 인식
- **들여쓰기:** YAML 파일은 들여쓰기(2칸 스페이스)를 통해 계층 구조를 표현. **탭을 사용하면 안됨**
- **순서 중요:** 각 항목은 순서대로 지정
- **주석:** #로 주석 작성 가능

Docker Compose 주요 명령어

- **docker-compose up**

- docker-compose.yaml 파일을 바탕으로 정의된 모든 서비스를 시작 -
-d 옵션을 추가하면 백그라운드(detached mode)에서 실행 가능
`docker-compose up -d`

- **docker-compose down**

- 현재 실행 중인 모든 컨테이너를 중지하고 네트워크와 함께 삭제
`docker-compose down`

- **docker-compose logs**

- 실행 중인 컨테이너의 로그를 확인 - 특정 서비스의 로그만 보고 싶을 때는 `docker-compose logs <service_name>`으로 지정 가능

env_file 설정

- **env_file**

- Docker Compose에서 환경 변수를 관리하는 방법 중 하나로, .env 파일에 정의된 환경 변수를 불러올 수 있음 - 컨테이너가 실행될 때 필요한 환경 변수를 KEY=VALUE 형식으로 저장한 파일을 사용

- **env_file 사용 예시**

- docker-compose.yaml 파일에서 env_file 항목을 사용하여 환경 변수 파일을 지정

```
services:
```

```
  web:
```

```
    env_file:
```

```
      - .env
```

쿠버네티스

Kubernetes

Kubernetes



kubernetes

Kubernetes란?

Kubernetes는 컨테이너화된 애플리케이션의 배포, 스케일링 및 관리를 자동화하기 위한 **오픈소스 플랫폼**
컨테이너 오케스트레이션 도구로써 복잡한 애플리케이션 운영을 효율적으로 처리할 수 있음

- **컨테이너 관리**: 다수의 컨테이너를 효과적으로 관리
- **애플리케이션의 확장성 제공**: 컨테이너의 자동 확장 및 축소 가능
- **리소스 최적화**: 애플리케이션 리소스를 동적으로 조정하여 비용 절감

Kubernetes가 필요한 이유

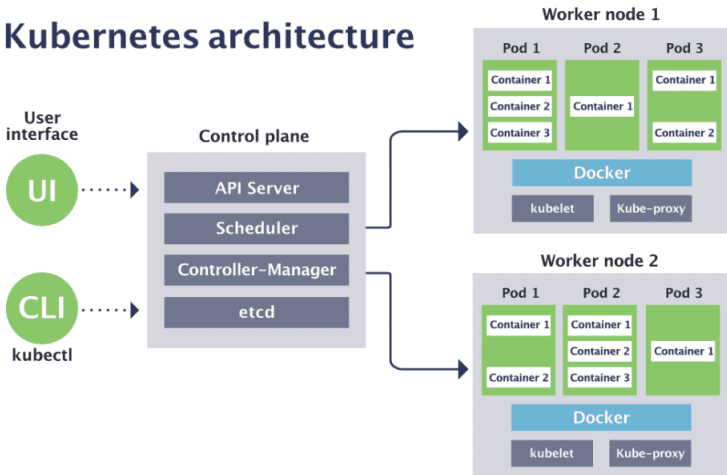
- **복잡한 애플리케이션 배포:** 대규모 애플리케이션에서는 여러 컨테이너를 조정하고 통제해야 하며, 이를 일일이 수동으로 관리하는 것이 어렵고 비효율적
- **자동화된 운영:** 컨테이너의 배포, 스케일링, 복구 작업을 자동으로 수행하여 운영 부담을 경감
- **고가용성:** 애플리케이션이 언제든지 중단 없이 실행될 수 있도록 장애를 감지하고, 자동으로 복구 및 재시작
- **무중단 배포:** 애플리케이션 업데이트 시 다운타임 없이 새로운 버전을 배포하고 이전 버전과 교체

Kubernetes is and is not

Kubernetes is and is not

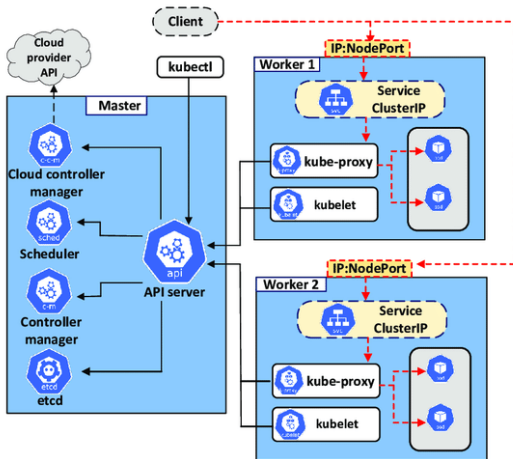
- 쿠버네티스는 배포를 위한 AWS, Azure와 같은 클라우드 서비스가 아님
- 쿠버네티스는 오픈소스 컨테이너 오케스트레이션 도구로, 컨테이너를 관리하고 배포하는 데 사용
- 쿠버네티스는 컨테이너를 관리하는 도구로, 컨테이너를 생성하거나 이미지를 빌드하는 도구가 아님
- 쿠버네티스는 무료 오픈소스 프로젝트로, 사용자가 직접 설치하고 관리해야 함

Kubernetes architecture



Kubernetes 아키텍처

state—and adjusts resources to make the current state close to the designed state.



Kubernetes 아키텍처

Kubernetes는 컨테이너화된 애플리케이션을 자동으로 배포, 관리하는 시스템으로, 크게 **Control Plane**이 포함된 **Master Node**와 **Worker Node**로 구성

- **Control Plane (Master Node):**

- **API Server:** 사용자 요청을 받아들이고, 클러스터의 상태 정보를 관리하는 중심 인터페이스
- **Scheduler:** 노드에 Pod를 배치하는 역할을 함
- **Controller-Manager:** 클러스터 상태를 모니터링하고, 필요한 작업을 자동으로 수행

- **Worker Node:**

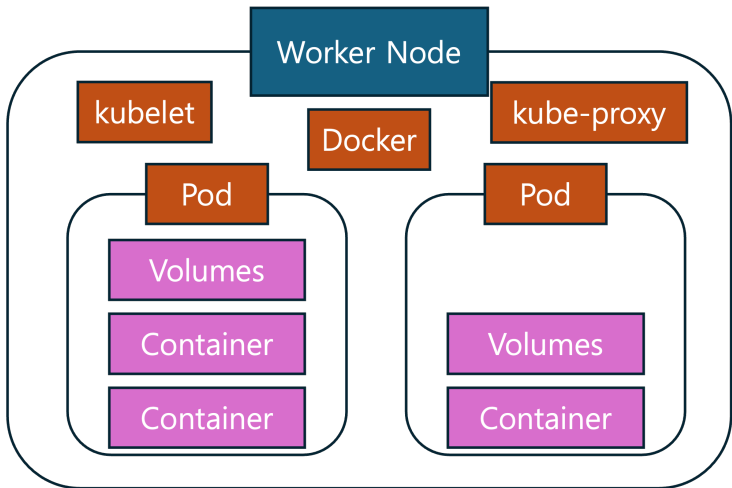
- **Pod:** 컨테이너의 그룹으로, 각 Pod는 애플리케이션 컨테이너를 실행
- **Kubelet:** 각 워커 노드에서 컨테이너가 제대로 실행되도록 관리
- **Kube-proxy:** 각 노드에서 네트워크 규칙을 관리하고, 트래픽이 적절하게 라우팅되도록 설정

kubectl

- **kubectl**은 Kubernetes 클러스터와 상호작용할 수 있는 명령줄 도구로, 사용자가 Kubernetes API를 통해 클러스터의 상태를 조회하고 애플리케이션을 배포하거나 관리할 수 있도록 함.
- 기본적으로 kubectl 명령어는 Control Plane과 통신하여 Pod, 서비스, 노드 등의 리소스를 관리.
- kubectl 명령어는 다양한 서브 명령을 제공하며, 예시로는 다음과 같은 것들이 있음:
 - `kubectl get pods` - 클러스터에서 실행 중인 Pod 목록 조회
 - `kubectl apply -f config.yaml` - YAML 파일에 정의된 리소스를 클러스터에 적용
 - `kubectl logs pod-name` - 특정 Pod의 로그 출력

Worker Node의 구성

Worker Node의 구성



Worker Node의 구성

- Worker Node는 마스터 노드에서 관리되는 노드로, 클러스터에서 컨테이너가 실행되는 노드
- 즉, 하나의 컴퓨터, 또는 가상 머신
- Pod들은 쿠버네티스에 의해 생성 및 관리, 각각의 Pod는 클러스터 내부 IP 주소를 가짐, 교체 및 제거시 내부 데이터도 제거
- 컨테이너 관리를 위해 Docker 또한 설치되어 있어야 함
- **Kubelet**: 각 워커 노드에서 마스터 노드와 통신하며, Pod를 실행하고 관리
- **Kube-proxy**: 각 노드에서 네트워크 규칙을 관리하고, 트래픽이 적절하게 라우팅되도록 설정

Master Node의 구성

- **API Server**: kubelets와 통신하기 위한 API
- **Scheduler**: 새로운 Pod를 생성할 노드를 선택
- **Controller-Manager**: Worker Node의 상태를 모니터링하고, 적당한 수의 포드를 유지

Kubernetes 주요 용어

- **Cluster**: Node들의 집합으로, Kubernetes에서 애플리케이션을 실행하는 환경
- **Node**: 클러스터 내에서 실행되는 물리적 또는 가상의 머신으로 Worker Node와 Master Node로 구성
- **Pod**: Kubernetes에서 실행되는 가장 작은 단위로, 하나 이상의 컨테이너로 구성 (즉, 하나의 사용자 정의 기능 또는 프로세스)
- **Services**: Pod의 집합을 노출하는 방법으로, 클러스터 내의 다른 Pod들이 해당 서비스에 접근할 수 있도록 함 (IP 주소, DNS 이름, 포트 번호를 제공)

Minikube

- Minikube는 로컬 환경에서 Kubernetes 클러스터를 실행할 수 있도록 도와주는 도구
- 로컬 머신에서 Kubernetes를 실행하고 테스트할 수 있으며, 개발 및 테스트 목적으로 사용
- Minikube는 단일 노드 Kubernetes 클러스터를 제공하며, 여러 노드를 사용하는 복잡한 클러스터를 구성할 수 없음

Minikube 설정

- **Minikube 설치:** Minikube를 설치하고 실행
- **minikube start:** 터미널에서 `minikube start` 명령어를 실행하여 Minikube를 시작
- **minikube status:** Minikube 상태 확인
- **minikube stop:** Minikube 중지
- **minikube dashboard:** Minikube 대시보드 (웹 UI) 실행

Minikube 시작

```
minikube start
🐶 Darwin 15.0.1 (arm64) 의 minikube v1.34.0
🌟 자동적으로 docker 드라이버가 선택되었습니다. 다른 드라이버 목록: parallels, ssh
🔧 Using Docker Desktop driver with root privileges
👍 Starting "minikube" primary control-plane node in "minikube" cluster
📦 Pulling base image v0.0.45 ...
🔥 Creating docker container (CPUs=2, Memory=7790MB) ...
🌐 쿠버네티스 v1.31.0 을 Docker 27.2.0 런타임으로 설치하는 중
  ▪ 인증서 및 키를 생성하는 중 ...
  ▪ 컨트롤 플레인을 부팅하는 중 ...
  ▪ RBAC 규칙을 구성하는 중 ...
🔗 bridge CNI (Container Networking Interface) 를 구성하는 중 ...
🔧 Kubernetes 구성 요소를 확인...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 애드온 활성화 : storage-provisioner, default-storageclass
👏 끝났습니다! kubectl이 "minikube" 클러스터와 "default" 네임스페이스를 기본적으로 사용하도록 구성되었습니다.
```

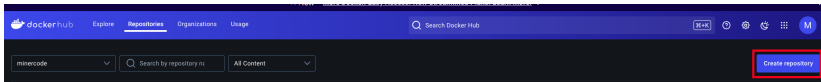
Minikube 제거시 중요 사항

- Minikube를 제거할 때는 `minikube stop` 후 `minikube delete` 명령어를 사용하여 클러스터를 삭제
- Minikube를 제거하면 클러스터에 있는 모든 리소스가 삭제되므로, 주의하여 사용
- Minikube를 도커 데스크톱에서 삭제할 경우, 다시 설치할 때 문제가 발생할 수 있으므로 주의

Minikube를 통한 실습1 예시

flask-app 이미지를 도커 허브에 업로드

- flask-app 이미지 빌드
`docker build -t flask-app .`
- 도커 허브에 로그인
- 도커허브에서 리포지토리 생성



- flask-app 이미지 태그
`docker tag flask-app <username>/flask-app`
- 도커 허브에 이미지 푸시
`docker push <username>/flask-app`

Minikube를 통한 실습1 예시

flask-deployment.yaml 수정

image: <username>/flask-app으로 수정

```
spec:
  containers:
  - name: flask-app
    image: minercode/flask-app:latest
    ports:
```

Minikube를 통한 실습1 예시

yaml 파일 적용

```
kubectl apply -f persistent-volume.yaml
```

```
kubectl apply -f mysql-deployment.yaml
```

```
kubectl apply -f flask-deployment.yaml
```

Pod 확인

```
kubectl get pods
```

대시 보드 실행

```
minikube dashboard
```

포트 포워딩

```
kubect1 port-forward service/flask-service 5001:5001
```

로컬 컴퓨터에서 5001번 포트로 접근한 트래픽을 쿠버네티스 클러스터의 5001번 포트로 포워딩 **웹 브라우저에서 접속**

`http://localhost:5001`

Week 6: Docker Compose, Kubernetes

- Docker Compose
- Kubernetes