학습 노트 - 도커, 쿠버네티스

도커(Docker)

핵심 개념

1. 컨테이너
    - 애플리케이션과 모든 종속성을 포함하는 독립적인 실행 환경
    - 호스트 OS의 커널을 공유하면서도 다른 컨테이너와 격리됨
2. 이미지
    - 컨테이너를 생성하기 위한 템플릿
    - 애플리케이션 코드, 런타임, 시스템 도구, 라이브러리 등을 포함
    - 불변성(Immutable)을 가짐
3. Dockerfile
    - 도커 이미지를 생성하기 위한 스크립트
    - 기본 이미지 선택, 명령어 실행, 파일 복사 등의 단계를 정의

```
FROM node:14
WORKDIR /app
COPY . .
RUN npm install
CMD ["npm", "start"]
```

1. 도커 컴포즈
    - 여러 컨테이너를 정의하고 실행하기 위한 도구
    - YAML 파일로 구성

```
version: '3'
services:
```

```
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: mongo
```

쿠버네티스(Kubernetes)

핵심 개념

1. 클러스터 구조
    - 마스터 노드: 클러스터 관리 및 제어
    - 워커 노드: 실제 애플리케이션이 실행되는 노드
2. Pod
    - 쿠버네티스의 가장 기본적인 배포 단위
    - 하나 이상의 컨테이너를 포함
    - 같은 Pod 내 컨테이너는 네트워크와 저장소 공유

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: web
    image: nginx:latest
```

1. 주요 리소스
    - Deployment: Pod의 복제본 관리 및 업데이트 전략 정의
    - Service: Pod에 대한 네트워크 접근 방식 정의
    - ConfigMap: 설정 정보 관리
    - Secret: 비밀 정보 관리

- Volume: 데이터 저장소 정의

2. 배포 예시

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: nginx:latest
        ports:
        - containerPort: 80
```

주요 기능

1. 자동 복구
    - Pod가 실패하면 자동으로 재시작
    - 노드 실패 시 다른 노드에 Pod 재배치
2. 스케일링
    - 수평적 확장: Pod 복제본 수 조정
    - 수직적 확장: Pod의 리소스 할당량 조정
3. 서비스 디스커버리
    - 내부 DNS를 통한 서비스 검색
    - 로드 밸런싱 자동 구성

4. 롤링 업데이트
    - 무중단 배포 지원
    - 버전 관리 및 롤백 기능

도커와 쿠버네티스는 함께 사용될 때 더욱 강력한 컨테이너 오케스트레이션 환경을 제공합니다:

- 도커: 컨테이너 생성과 관리
- 쿠버네티스: 컨테이너 오케스트레이션과 확장성 관리

이러한 구조를 통해 마이크로서비스 아키텍처를 효과적으로 구현하고 관리할 수 있습니다.

1. Docker 기초

1.1 Docker 개요

- Docker는 애플리케이션을 컨테이너화하여 개발, 배포, 실행을 단순화하는 플랫폼
- 컨테이너는 애플리케이션과 그 종속성을 포함하는 독립적인 실행 환경

1.2 주요 Docker 명령어

```
# 이미지 관리
docker pull [이미지명]      # 이미지 다운로드
docker images               # 이미지 목록 확인
docker rmi [이미지ID]       # 이미지 삭제

# 컨테이너 관리
docker run [옵션] [이미지명]  # 컨테이너 생성 및 실행
docker ps                   # 실행 중인 컨테이너 목록
docker ps -a                # 모든 컨테이너 목록
docker stop [컨테이너ID]    # 컨테이너 중지
docker rm [컨테이너ID]      # 컨테이너 삭제

# Dockerfile 빌드
docker build -t [태그명] .   # 현재 디렉토리의 Dockerfile로 이미지 빌드
```

2. Kubernetes 기초

2.1 Kubernetes 개요

- 컨테이너화된 애플리케이션의 자동 배포, 스케일링, 관리를 위한 오픈소스 플랫폼
- 여러 컨테이너를 관리하고 운영하기 위한 오케스트레이션 도구

2.2 주요 Kubernetes 개념

- Pod: 가장 기본적인 배포 단위, 하나 이상의 컨테이너 그룹
- Service: Pod를 네트워크 서비스로 노출
- Deployment: Pod의 선언적 업데이트와 스케일링 관리
- ConfigMap/Secret: 설정 정보와 민감한 정보 관리

2.3 기본 kubectl 명령어

```
kubectl get pods           # Pod 목록 조회
kubectl get services       # Service 목록 조회
kubectl apply -f [파일명]  # YAML 파일로 리소스 생성/수정
kubectl delete [리소스명]  # 리소스 삭제
```

3. Vue.js 프로젝트 세팅

3.1 프로젝트 초기화

```
# Vue CLI 설치
npm install -g @vue/cli

# 프로젝트 생성
vue create [프로젝트명]

# 의존성 설치
npm install
```

3.2 프로젝트 구조

```
project/
├── public/
├── src/
│   ├── assets/
│   ├── components/
│   ├── views/
│   ├── App.vue
│   └── main.js
├── package.json
└── vue.config.js
```

4. Django 프로젝트 세팅

4.1 환경 설정

```
# 가상환경 생성 및 활성화
python -m venv venv
source venv/bin/activate  # Windows: venv\\Scripts\\activat
e

# Django 설치
pip install django

# 프로젝트 생성
django-admin startproject [프로젝트명]
```

4.2 프로젝트 구조

```
project/
├── manage.py
└── project/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
    └── wsgi.py
```

5. Git 커밋 가이드라인

5.1 기본 Git 명령어

```
git init                    # 저장소 초기화
git add [파일명]            # 스테이징
git commit -m "메시지"      # 커밋
git push origin [브랜치]    # 원격 저장소에 푸시
```

5.2 커밋 메시지 컨벤션

```
feat: 새로운 기능 추가
fix: 버그 수정
docs: 문서 수정
style: 코드 포맷팅
refactor: 코드 리팩토링
test: 테스트 코드
chore: 빌드 업무 수정
```

5.3 브랜치 전략

- main: 제품 출시 브랜치
- develop: 개발 브랜치
- feature: 기능 개발 브랜치
- hotfix: 긴급 버그 수정 브랜치

6. 추가 참고사항

- 각 기술의 공식 문서를 참조하여 최신 정보 확인
- 보안 관련 설정 및 모범 사례 준수
- 테스트 및 문서화 습관화

Linux 환경의 Docker & Kubernetes 실습 가이드

1. Docker 설치 및 환경 설정

1.1 Docker 설치

```
# 이전 버전 제거
sudo apt-get remove docker docker-engine docker.io containe
rd runc

# 필요한 패키지 설치
sudo apt-get update
sudo apt-get install \\
    ca-certificates \\
    curl \\
    gnupg \\
    lsb-release

# Docker의 공식 GPG 키 추가
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-ke
yring.gpg

# Docker 저장소 설정
echo \\
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/sh
are/keyrings/docker-archive-keyring.gpg] <https://download.
docker.com/linux/ubuntu> \\
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.li
st.d/docker.list > /dev/null

# Docker 엔진 설치
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

1.2 Docker 권한 설정

```
# 현재 사용자를 docker 그룹에 추가
sudo usermod -aG docker $USER
```

```
# 변경사항 적용을 위한 재로그인
newgrp docker
```

2. Docker 실습 예제

2.1 기본 컨테이너 실행

```
# Nginx 컨테이너 실행
docker run -d -p 80:80 --name my-nginx nginx

# 컨테이너 로그 확인
docker logs my-nginx

# 컨테이너 내부 접속
docker exec -it my-nginx bash
```

2.2 Dockerfile 작성 및 빌드

```
# Node.js 애플리케이션 Dockerfile 예제
FROM node:14

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

```
# 이미지 빌드
docker build -t my-node-app .
```

```
# 컨테이너 실행
docker run -d -p 3000:3000 my-node-app
```

2.3 Docker Compose 활용

```
# docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: mongo
    volumes:
      - mongodb_data:/data/db

volumes:
  mongodb_data:
```

```
# Docker Compose 실행
docker-compose up -d

# 서비스 상태 확인
docker-compose ps

# 서비스 중지
docker-compose down
```

3. Kubernetes 설치 및 설정

3.1 kubectl 설치

```
# kubectl 다운로드
curl -LO "<https://dl.k8s.io/release/$>(curl -L -s <http
s://dl.k8s.io/release/stable.txt>)/bin/linux/amd64/kubectl"
```

```
# 실행 권한 부여
chmod +x kubectl

# 시스템 경로로 이동
sudo mv kubectl /usr/local/bin/
```

3.2 Minikube 설치 (로컬 테스트용)

```
# Minikube 다운로드 및 설치
curl -Lo minikube <https://storage.googleapis.com/minikube/
releases/latest/minikube-linux-amd64>
chmod +x minikube
sudo mv minikube /usr/local/bin/

# Minikube 시작
minikube start
```

4. Kubernetes 실습 예제

4.1 기본 Pod 배포

```
# pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

```
# Pod 생성
kubectl apply -f pod.yaml
```

```
# Pod 상태 확인
kubectl get pod nginx-pod
kubectl describe pod nginx-pod
```

4.2 Deployment 생성

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
# Deployment 생성
kubectl apply -f deployment.yaml

# Deployment 상태 확인
kubectl get deployments
kubectl get pods
```

4.3 Service 생성

```
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30000
```

```
# Service 생성
kubectl apply -f service.yaml

# Service 상태 확인
kubectl get services
```

5. 유용한 디버깅 명령어

5.1 Docker 디버깅

```
# 컨테이너 상태 확인
docker ps -a
docker stats

# 컨테이너 로그 확인
docker logs -f [컨테이너ID]

# 컨테이너 상세 정보 확인
docker inspect [컨테이너ID]
```

5.2 Kubernetes 디버깅

```
# Pod 로그 확인
kubectl logs [Pod이름]

# Pod 상세 정보 확인
kubectl describe pod [Pod이름]

# 클러스터 상태 확인
kubectl get nodes
kubectl get events
```

6. 주의사항 및 팁

1. 리소스 관리
    - 사용하지 않는 컨테이너와 이미지는 정기적으로 정리
    - 시스템 리소스 모니터링 필요
2. 보안
    - 컨테이너 실행 시 루트 권한 사용 최소화
    - 민감한 정보는 반드시 Secret으로 관리
3. 네트워킹
    - 포트 충돌 주의
    - 네트워크 정책 설정 확인
4. 백업
    - 중요 데이터는 볼륨 마운트하여 관리
    - 정기적인 설정 백업 필요