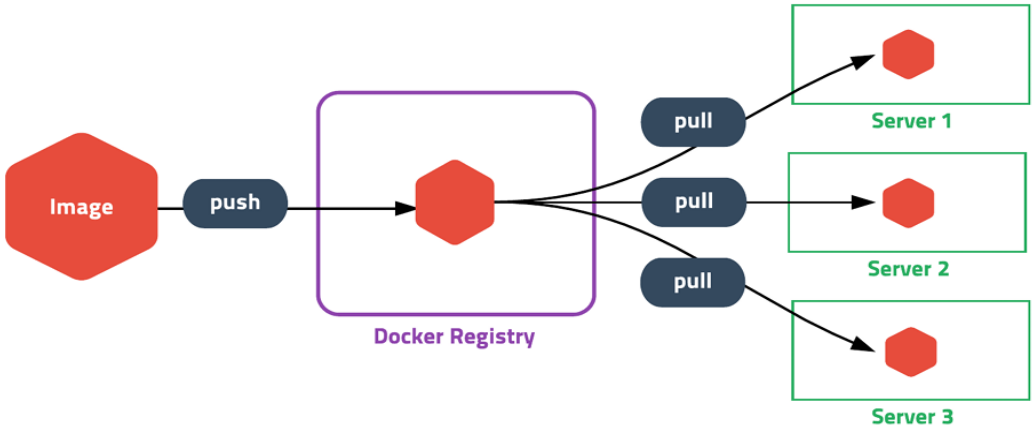
DevOps for SpringBoot

단계	주제	내용
[1]	리눅스 기초 I (명령어와 파일 시스템)	- 리눅스 디렉토리 구조 이해- 기본 명령어 (ls, cd, mkdir, rm, cp, mv, cat, chmod 등)- 사용자 및 퍼미션 이해- 실습: 파일 조작, 사용자 추가, 권한 변경
[2]	리눅스 기초 II (WSL2 네트워크 실습)	- 네트워크 명령어: ping, curl, ss, netstat, ip- 패키지 설치: apt update, apt install- 프로세스/포트 확인: ps aux, top, ss -tuln- 실습: 네트워크 점검, 패키지 관리, 실행 중인 프로세스 추적
[3]	Docker & Spring Boot 컨테이너화	- Docker 기본 구조 및 명령어 실습- Dockerfile 작성- Spring Boot 컨테이너화- Docker Compose 로 DB 연동 실습
[4]	Jenkins 설치 및 CI 구성	- Docker 로 Jenkins 설치- Jenkins 초기 설정 및 Git 연동- Jenkins Job 생성 및 빌드 자동화- Jenkinsfile 작성: Build + Test
[5]	Kubernetes 설치 및 수동 배포	- Minikube 설치 및 kubectl 설정- Kubernetes 개념 (Pod, Deployment, Service)- Spring Boot 앱 수동 배포 (YAML 작성)- 실습: 서비스 접속 및 로깅 확인
[6]	Jenkins → Kubernetes 자동 배포 (CD)	- Jenkins 에 kubectl 연동 (kubeconfig)- Jenkinsfile 수정: Build → Deploy 자동화- 실습: Git Push → Jenkins → K8s 자동배포 구성
[7]	Ingress-Nginx + 경로 기반 라우팅	- Ingress Controller 설치 (Minikube Addon)- 도메인/경로 기반 서비스 분리- 실습: /api, /admin, /user 등으로 분기 라우팅 구성
[8]	Prometheus + Grafana 모니터링	- Prometheus 설치 및 Spring Boot 와 연동- actuator, micrometer 설정- Grafana 설치 및 대시보드 구성- 실습: JVM 메모리, 요청 수, 응답 속도 시각화
[9]	전체 통합 배포 흐름 구성	- Git → Jenkins → Docker → K8s → Ingress → Grafana- 장애 복구 및 배포 실패 대응 실습- 실습: 실제 시나리오 기반 전체 배포 테스트

도커 서비스 환경과 서비스 빌드 및 배포



단계	제목	주요 내용	명령 예시/ 구성

1	초기 인프라 구축	- 애플리케이션이 배포될 인프라를 먼저 준비- 주로 클라우드나 VM 기반 환경 구성	`docker-machine create --driver virtualbox myvm1`
			
			`docker swarm init`

2	Stack 구성 및 실행	- docker-compose.yml 파일 기반으로 서비스 정의 - 여러 컨테이너를 통합해 하나의 Stack 으로 실행	`docker stack deploy -c docker-compose.yml mystack`
3	스케일 조정	- 부하 분산 처리 방식 2 가지 ① 워커 노드를 추가 ② 서비스 복제(replica) 수 증가	`docker service scale mystack_web=5` `docker node ls` (노드 확인)
4	코드 수정 및 이미지 빌드	- 코드 수정 후 새로운 Docker 이미지 빌드	`docker build -t finish07sds/lab02:v2 .`
5	이미지 Push	- 새 이미지를 Docker Hub 에 업로드	`docker push finish07sds/lab02:v2`
6	Stack 업데이트 및 배포	- 변경된 이미지나 파일을 기반으로 Stack 재배포	`docker stack deploy -c docker-compose.yml mystack`
부가 과정	파일 업데이트 및 전송	- bind-mount 방식의 파일이 수정되었다면원격 머신으로 전송 필요	`docker-machine scp ./docker-compose.yml myvm1:~`

Docker Swarm

1. Docker Swarm 이란?

- Docker 에서 제공하는 클러스터링 및 컨테이너 오케스트레이션 도구
- 다중 Docker 서버를 하나의 클러스터로 묶어 관리하는 기능
- 기본 배포 단위는 서비스(Service)

2. Swarm 의 핵심 요소

요소	설명
Node	Swarm 에 참여하는 Docker 머신. Manager Node 와 Worker Node 로 구분
Manager Node	클러스터를 관리하고 작업 분배 및 상태 모니터링 역할 수행
Worker Node	실제 컨테이너를 실행하는 노드 (명령 수신만 수행)
Service	실행하려는 컨테이너 애플리케이션 단위 (예: web, db)
Task	Service 가 생성한 개별 컨테이너 인스턴스
Overlay Network	Swarm 내 노드 간 통신을 위한 가상 네트워크

3. Swarm 의 주요 기능

- **클러스터 관리**

여러 노드를 묶어 클러스터로 구성하고 하나의 엔트리 포인트에서 제어

- **서비스 배포**

컨테이너 수, 배포 정책, 업데이트 전략 등을 정의하고 자동 분산

- **Self-healing (자가 복구)**

컨테이너가 실패하면 자동으로 재시작하거나 다른 노드에 재배포

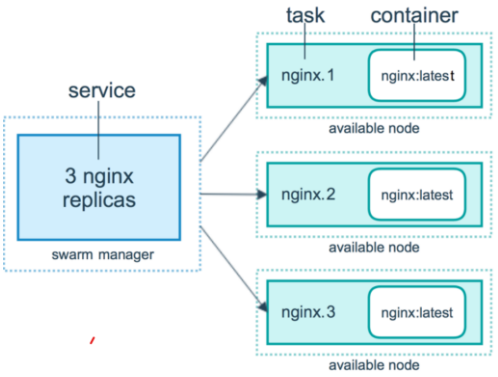
- **롤링 업데이트**

중단 없이 버전 업데이트 가능 (컨테이너를 순차적으로 교체)

- **보안**

TLS 기반의 노드 간 통신 암호화 및 노드 인증

4. Swarm 사용 흐름 https://docs.docker.com/engine/swarm/ingress/



- **nginx:latest 이미지를 탑재한 3 개의 레플리카로 정의된 Nginx 서비스와 태스크의 구성**
- **정의된 서비스의 명세를 스웜 매니저(Swarm Manager)가 파악한 뒤, 가용한 노드에 지정된 레플리카 수 만큼의 태스크를 만들어 할당**

서비스 생성 : docker service create

```
docker service create --name nginx \
  --replicas=3 \
  --publish=8080:80 \
  nginx:latest
```

서비스 조회하기 :docker service ls

```
ID             NAME      MODE          REPLICAS   IMAGE          PORTS
w1v8rn73q95w   nginx     replicated    3/3        nginx:latest   *:8080->80/tcp
```

태스크 목록 및 상태 조회하기 : docker service ps <서비스 명>

```
# docker service scale <SERVICE-ID>
docker service ps nginx

ID             NAME          IMAGE          NODE             DESIRED STATE   CURRENT STATE
ERROR     PORTS
nhxtav0xuwit   nginx.1       nginx:latest   finish07-node-01   Running         Running 24 minutes
ago
vzv1rdk9j7vg   nginx.2       nginx:latest   finish07-node-02   Running         Running 24 minutes
ago
idxn4eijmbah   nginx.3       nginx:latest   finish07-node-01   Running         Running about a
minute ago
```

서비스 삭제하기 : docker service rm <서비스 명>

5.Docker Compose 와 Swarm 연동

docker-compose.yml 파일을 Swarm 에서 그대로 사용할 수 있으며 배포
여러 서비스 (예: Spring Boot, MySQL, Redis 등)를 Swarm 클러스터에 한번에 배포 가능

```
docker stack deploy -c docker-compose.yml mystack
```

6. Kubernetes 와의 비교 : 실무에서 Kubernetes 가 Swarm 보다 더 많이 쓰이는 이유

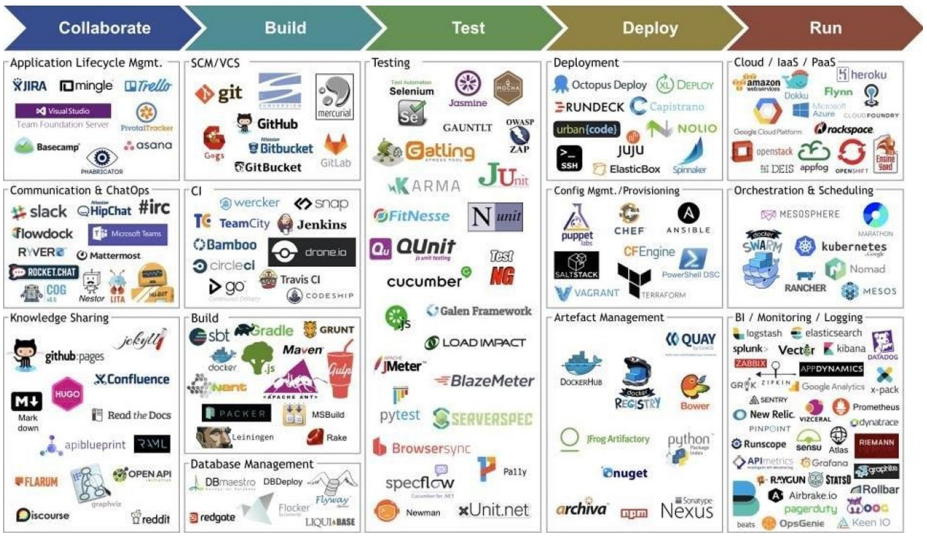
구분	Docker Swarm	Kubernetes
기능 범위	기본적인 오케스트레이션 제공 (서비스, 복제, 롤링 업데이트 등)	서비스 디스커버리, 상태 관리, ConfigMap, Secret, Auto-scaling, Helm 등 강력한 기능
생태계 확장성	제한적 (Swarm 은 Docker 에만 tightly coupled)	Prometheus, Grafana, ArgoCD, Istio, Helm, Kustomize 등과 연동 풍부
커뮤니티/채택률	거의 정체	대부분의 대기업/클라우드/DevOps 팀에서 채택 (사실상 표준)
멀티 클라우드/하이브리드	불편함	AWS, GCP, Azure, 온프레미스 등 모든 환경에 적합
유지보수	Docker 측에서 Swarm 개발 거의 중단	CNCF 에서 지속적이고 활발한 개발과 지원

7. 실무 활용

- WSL = 내 컴퓨터 속 가상의 Ubuntu 리눅스 개발실
- Docker Desktop = Local 클라우드 서버 + 쿠버네티스 클러스터

역할	사용하는 도구
코딩 & CLI 빌드 & git	WSL2 Ubuntu
도커 이미지 빌드	WSL2 or Docker Desktop
쿠버네티스 배포	Docker Desktop 내장 Kubernetes
CI/CD 연동 테스트	WSL + Jenkins or GitHub Actions

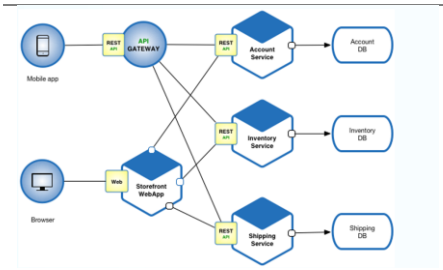
DevOps : 데브옵스도구



출처: https://devopedia.org

마이크로서비스와 데브옵스 :

마이크로서비스 -> 서비스의집합으로서의애플리케이션



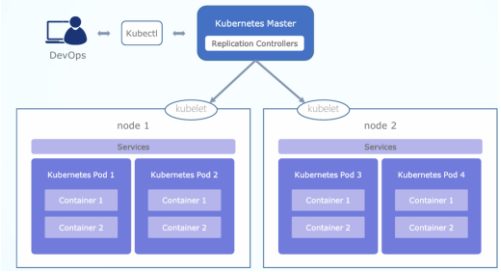
https://microservices.io/

- **마이크로서비스는 작고 독립적인 책임 단위로 구성되며, 각 서비스는 독립적으로 배포 및 운영**
- **HTTP+JSON 같은 기술 중립적인 경량 프로토콜로 통신하여 구현 기술과 무관하게 연결.**
- **각 서비스는 명확한 책임을 가진 소규모 팀이 전체 라이프사이클 관리**

컨테이너 오케스트레이션의 주요 기능 요약

- 컨테이너 애플리케이션의 자동 배포 및 스케줄링
- 컨테이너 그룹(Pod 등)에 대한 로드 밸런싱 지원
- 수평 스케일링 및 오토스케일링 기능 제공
- 장애 발생 시 자동 복구(Self-healing)
- 컨테이너 간 네트워크 연결 및 격리 설정 가능
- 외부 클라이언트에 서비스를 노출 (Ingress, LoadBalancer 등)
- 내부 서비스 간 자동 탐색 (Service Discovery)
- 로그 수집 및 중앙 집중화, 자동화된 로깅 처리
- 모니터링 도구와 연동한 상태 관리 및 시각화 자동화

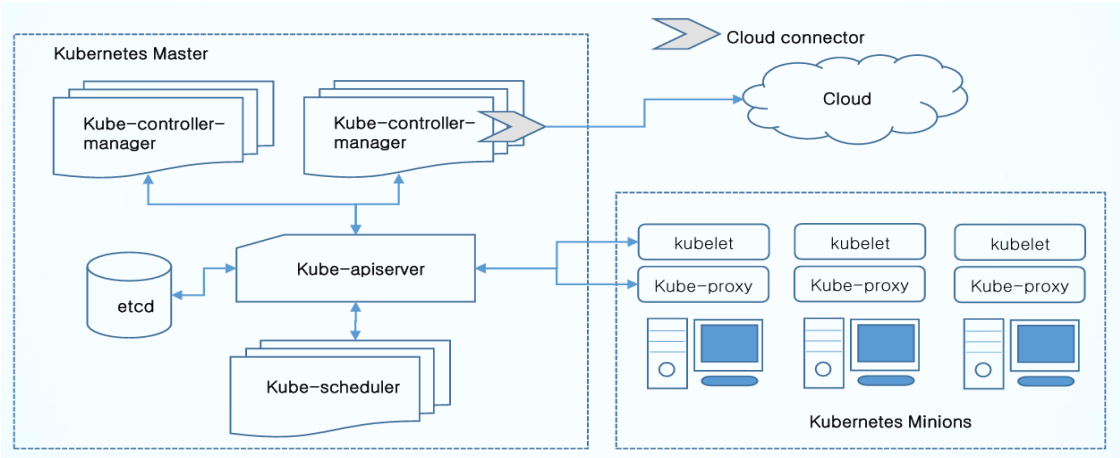
쿠버네티스와 데브옵스운영 방식



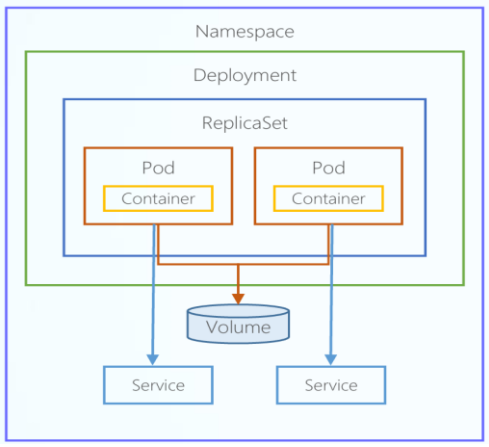
www.bigdatatraining.in

- 컨테이너 기반 개발환경은 CI/CD 도구와 결합되어 코딩, 빌드, 테스트, 배포의 자동화
- CI(지속적 통합)는 빠른 코드 수정과 품질 확보를, CD(지속적 배포)는 변경사항을 효율적으로 서비스에 반영하는 역할.
- 컨테이너와 자동화 도구의 결합은 개발과 운영의 민첩성을 높이며, 이를 Kubernetes 와 함께 사용하면 배치, 확장, 운영 자동화

쿠버네티스아키텍쳐 : Cluster Architecture | Kubernetes



쿠버네티스개념 Concepts | Kubernetes

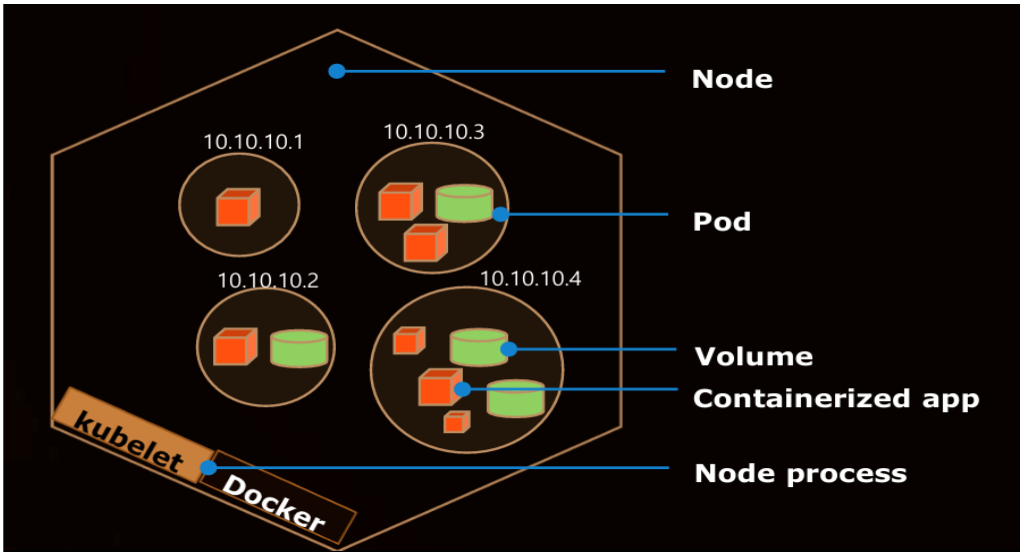


1. Deployment 가 ReplicaSet 을 생성하고 관리
2. ReplicaSet 은 복제된 Pod 를 유지
3. 각 Pod 는 Container 를 포함
4. Volume 을 통해 데이터 공유
5. Service 는 외부 또는 내부 트래픽을 각 Pod 로 연결

Pod, Service, Volume, Namespace

노드컴포넌트

- 노드컴포넌트는 동작 중 인파드를유지시키고 쿠버네티스런타임환경을 제공하며, 모든 노드상에서 동작
- 노드는 쿠버네티스에 있어서 워커머신이며 클러스터에따라 VM 또는물리 머신이 될수 있고 여러 개의 파드는 하나의 노드 위에서 동작한다.



[Practical Kubernetes Setup and Deployment]

구분	실습 01: k3s 설치 및 점검	실습 02: YAML 기반 nginx 배포	실습 03: 배포 자동화 및 커스터마이징
목적	K3s 클러스터 설치 및 상태 점검	Deployment + Service 구성	ConfigMap 기반 nginx 설정 자동 배포
핵심 작업	- K3s 설치 - kubectl get nodes 확인 - export KUBECONFIG 설정	- nginx-deployment.yaml 작성/적용 - nginx-service.yaml 작성/적용	- 기존 리소스 삭제 - nginx-configmap.yaml 작성 - nginx-custom.yaml, nginx -custom-svc.yaml 적용
확인 방법	- kubectl get nodes, get pods 정상 출력 확인	- kubectl get pods, get svc 로 배포 상태 확인 - 브라우저로 접속 (http://<IP>:<NodePort>)	- kubectl describe 로 ConfigMap 적용 확인 - 브라우저에서 HTML 반영 여부 확인

구분	명령어	설명
K3s 설정	export KUBECONFIG=/etc/rancher/k3s/k3s.yaml	kubectl 이 K3s 클러스터와 통신하도록 환경변수 설정
클러스터 상태 확인	kubectl get nodes	노드 상태 확인 (정상 동작 시 Ready)
	kubectl get pods	현재 Pod 상태 확인 (Running, Pending 등)
	kubectl get svc	서비스 목록과 NodePort 등 확인
	kubectl describe pod <pod 명>	Pod 상세 상태 및 이벤트 확인
Deployment 적용	kubectl apply -f nginx-deployment.yaml	nginx Deployment 구성 적용
	kubectl delete -f nginx-deployment.yaml	해당 Deployment 리소스 삭제
Service 적용	kubectl apply -f nginx-service.yaml	nginx Service(NodePort) 구성 적용
	kubectl delete svc <서비스명>	서비스만 삭제 (Deployment 는 유지됨)
ConfigMap 구성	kubectl apply -f nginx-configmap.yaml	HTML 파일 설정을 ConfigMap 으로 생성
Custom Deployment	kubectl apply -f nginx-custom.yaml	ConfigMap 을 마운트한 nginx 배포 구성
Custom Service	kubectl apply -f nginx-custom-svc.yaml	nginx-custom 용 NodePort 서비스 생성
파일 관련	nano <파일명>.yaml	YAML 파일 직접 작성/수정
디스크 용량 확인	df -h /var/lib/kubelet	노드 디스크 용량 확인 (Pending 원인 분석용)

[실습] Kubernetes 설치

1 단계: WSL Ubuntu 터미널에서 아래 명령 실행

```
curl -sfL https://get.k3s.io | sh -
```

-s: 진행률 표시줄을 숨김, -f 는 오류 발생 시 curl 이 자동으로 종료, -L 리다이렉션

```
root@Dominica:~# curl -sfL https://get.k3s.io | sh -
[INFO]  Finding release for channel stable
[INFO]  Using v1.32.5+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.32.5+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.32.5+k3s1/k3s
```

2 단계: Kubernetes 설치확인

```
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml  #클러스터 접속 정보
echo 'export KUBECONFIG=/etc/rancher/k3s/k3s.yaml' >> ~/.bashrc
source ~/.bashrc
kubectl get nodes
```

```
root@Dominica:~# export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
root@Dominica:~# kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
dominica   Ready    control-plane,master   16s   v1.32.5+k3s1
```

항목	설명
kubectl	쿠버네티스 CLI
kubeconfig	클러스터 접속 정보가 담긴 설정 파일
KUBECONFIG	kubectl 이 참고할 설정 파일 경로를 지정
/etc/rancher/k3s/k3s.yaml	k3s 설치 시 자동 생성되는 kubeconfig 경로

- 중지: sudo systemctl stop k3s
- 재시작: sudo systemctl start k3s
- 제거: sudo /usr/local/bin/k3s-uninstall.sh

3 단계: 기본 쿠버네티스 CLI 명령어 설정

deployment 는 Pod 의 자동 생성, 업데이트, 복제, 롤백 등**을 관리하는 추상화 리소스**

```
kubectl get pods --all-namespaces
kubectl get svc           #네임스페이스의 서비스확인
kubectl get deployments # Deployment 리소스를 조회
```

4 단계: 테스트용 Deployment 실행

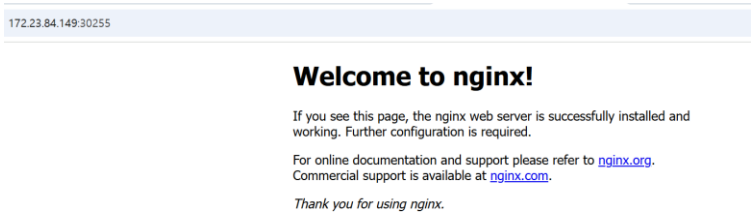
nginx 웹서버를 클러스터 안에 배포하고, 외부에서 접근해 확인해보자.

```
kubectl create deployment hello-kube --image=nginx
kubectl expose deployment hello-kube --type=NodePort --port=80
kubectl get pods
kubectl get svc
```

→ 브라우저에서 http://localhost:<노출된 NodePort>로 접속해 nginx 확인 가능

```
root@Dominica:~# kubectl create deployment hello-kube --image=nginx
deployment.apps/hello-kube created
root@Dominica:~#
root@Dominica:~# kubectl expose deployment hello-kube --type=NodePort --port=80
service/hello-kube exposed
root@Dominica:~#
root@Dominica:~# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-kube-5bd9565b9c-lsbmc   1/1     Running   0          30s
root@Dominica:~#
root@Dominica:~# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
hello-kube   NodePort    10.43.15.132   <none>        80:30255/TCP   21s
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP        3m40s
root@Dominica:~#
```

```
root@Dominica:~# hostname -I
```



[실습 02] YAML 기반 배포_ nginx Deployment 예제 실행

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

의 코드로 nano nginx-deployment.yaml 를 작성 후 실행해보자

1 단계: 파일 작성

```
nano nginx-deployment.yaml
```

2 단계: Deployment 생성

```
kubectl apply -f nginx-deployment.yaml
kubectl get all
```

3 단계: 리소스 확인

```
kubectl get deployments

kubectl get pods

kubectl get svc
```

명령어	설명
kubectl get deployments	생성된 Deployment 목록 확인 (nginx-deployment 포함 여부)
kubectl get pods	파드가 제대로 생성되었는지 확인 (예: Running 상태)
kubectl get svc	서비스(Service)가 생성되었는지 확인 (NodePort 등)

4 단계 브라우저 확인

172.23.84.149:30255

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

[실습 03] YAML 을 이용한 배포 자동화 실습

```
root@Dominica:~# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
hello-kube   NodePort    10.43.15.132   <none>        80:30255/TCP   38m
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP        41m
```

1 단계: 기존 리소스 삭제

```
kubectl delete deployment hello-kube

kubectl delete svc hello-kube
```

---

2 단계 : YAML 파일 작성 _ nginx-deployment.yaml -> 기존 파일 사용

---

3 단계: 배포 실행

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

---

4 단계: 서비스 노출용 YAML 작성 _파일 이름: nginx-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30036  # 선택한 포트 (30000~32767 사이 가능)
```

5 단계: 서비스 배포

```
kubectl apply -f nginx-service.yaml

kubectl get svc
```

6 단계: 브라우저 확인

이전과 같이 http://172.23.84.149: 30036 으로 접속하면 됩니다.

7 단계: 추가 예제 - nginx 설정 커스터마이징

nginx 에 HTML 넣기 (configMap 활용) _nano nginx-configmap.yaml

```
  GNU nano 7.2                                    nginx-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-html
data:
  index.html: |
    <html>
    <head><title>Hello from ConfigMap</title></head>
    <body><h1>Welcome to Kubernetes!</h1></body>
    </html>
```

8 단계 : configMap 을 마운트한 Deployment 작성 nano nginx-custom.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-custom
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-custom
  template:
    metadata:
      labels:
        app: nginx-custom
    spec:
      containers:
        - name: nginx
          image: nginx
          volumeMounts:
            - name: html-volume
              mountPath: /usr/share/nginx/html/index.html
              subPath: index.html
      volumes:
        - name: html-volume
          configMap:
```

```
        name: nginx-html
```

```
kubectl apply -f nginx-configmap.yaml

kubectl apply -f nginx-custom.yaml
```

9 단계 : nginx-custom 에 대한 서비스 생성 _nano nginx-custom-svc.yaml

```
  GNU nano 7.2                                    nginx-custom-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-custom-service
spec:
  type: NodePort
  selector:
    app: nginx-custom
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30037
```

```
kubectl apply -f nginx-custom-svc.yaml

kubectl get svc
```

```
root@Dominica:~# kubectl get svc
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes             ClusterIP   10.43.0.1       <none>        443/TCP        65m
nginx-custom-service   NodePort    10.43.73.136    <none>        80:30037/TCP   0s
nginx-service          NodePort    10.43.51.146    <none>        80:30036/TCP   8m42s
```

Welcome to Kubernetes!

단계	내용	실행 명령어	확인 포인트
1 단계	기존 리소스 삭제	kubectl delete deployment/service ...	삭제 여부 확인: kubectl get all 결과가 비어 있는지
2 단계	nginx-deployment.yaml 작성	kubectl apply -f nginx-deployment.yaml	kubectl get deployments, kubectl get pods 에서 nginx pod 3 개 생성 여부 확인
3 단계	배포 확인	kubectl get deployments, kubectl get pods	pod 가 Running 상태이고 READY 가 1/1 인지
4 단계	nginx-service.yaml 작성	kubectl apply -f nginx-service.yaml	kubectl get svc 에서 NodePort 타입이며 PORT(S)에 포트 노출됐는지 확인 (예: 30036)
5 단계	서비스 배포	(4 단계와 동일)	kubectl get svc 로 실제 노출 포트 확인
6 단계	브라우저 접속 확인	브라우저에서 http://<노드 IP>:30036	기본 nginx Welcome 페이지가 보이는지
7 단계	HTML 커스터마이징용 ConfigMap 작성 (nginx-configmap.yaml)	kubectl apply -f nginx-configmap.yaml	kubectl get configmap, kubectl describe configmap nginx-html 로 데이터 존재 확인
8 단계	ConfigMap 을 마운트한 Deployment (nginx-custom.yaml)	kubectl apply -f nginx-custom.yaml	kubectl get pods, kubectl describe pod 로 마운트 상태 확인
9 단계	nginx-custom-svc.yaml 서비스 생성	kubectl apply -f nginx-custom-svc.yaml	kubectl get svc 에서 nginx-custom-svc 확인 후 포트 확인 (예: 30037)
최종	브라우저 접속 확인	브라우저에서 http://<노드 IP>:30037	HTML 커스터마이징된 페이지 (Custom HTML from ConfigMap!) 확인

- 각 단계에서 반드시 kubectl get all 로 전체 리소스 상태를 점검!!

추가 작업: Helm 설치 _ Kubernetes 패키지 매니저

Ingress Controller, Monitoring, GitLab 등 필요함

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

```
root@Dominica:~# helm version
version.BuildInfo{Version:"v3.18.3", GitCommit:"6838ebcf265a3842d1433956e8a622e3290cf324", GitTreeState:"clean", GoV
ersion:"go1.24.4"}
```

단계	실습 항목	설명
[1]	기본 배포 & 서비스 생성	Deployment, NodePort, ConfigMap 등 기본 리소스 배포 실습
[2]	kubectl rollout restart deployment/nginx-deployment	롤링 재시작: 서비스 중단 없이 새로운 파드로 순차 교체
[3]	kubectl scale deployment nginx-deployment --replicas=5	수평 확장: 파드 개수를 수동으로 조정 (HPA 이전 단계)
[4]	ConfigMap + Secret + Volume 마운트	설정파일, 민감 정보(비밀번호 등)를 파드에 주입
[5]	Ingress Controller 설치 + 도메인 라우팅	nginx-ingress 등 설치 후 /app1, /app2 등 경로 기반 라우팅 테스트
[6]	Helm Chart 로 Wordpress + MySQL 배포	Helm 으로 복합 앱 배포 (Wordpress + DB)