

# 3. 컨테이너를 다루는 표준 아키텍처, 쿠버네티스

## ▼ ★ 3.0 Intro

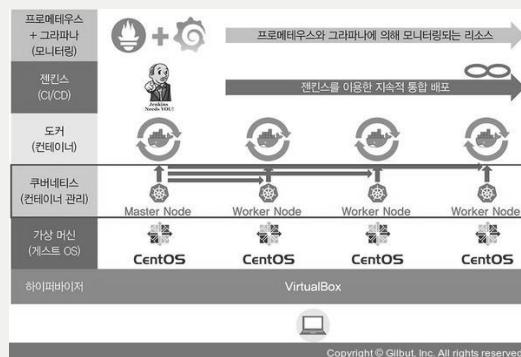


컨테이너 인프라 환경 : 리눅스 운영 체제의 커널 하나에서 여러 개의 컨테이너가 격리된 상태로 실행되는 인프라 환경

- 컨테이너 : 하나 이상의 목적을 위해 독립적으로 작동하는 프로세스
- 기업 환경에서는 다수의 관리자가 수백 또는 수천 대의 서버를 함께 관리하기 때문에 일관성 유지가 중요
- 컨테이너 인프라 환경을 구성하면 눈송이 서버(여러 사람이 만져서 설정의 일관성이 떨어진 서버)를 방지하는 데 효과적임



- 가상화 환경에서는 각각의 가상 머신이 모두 독립적인 운영 체제 커널을 가지고 있어야 하기 때문에 그만큼 자원을 더 소모해야 하고 성능이 떨어질 수밖에 없음.
- 컨테이너 인프라 환경은 운영 체제 커널 하나에 컨테이너 여러 개가 격리된 형태로 실행되기 때문에 자원을 효율적으로 사용할 수 있고 거치는 단계가 적어서 속도도 훨씬 빠름
- 시간이 지나 커널을 공유해 더 많은 애플리케이션을 올릴 수 있는 컨테이너가 도입되기 시작하면서 늘어난 컨테이너를 관리해야 했으나 기존의 컨테이너 관리 솔루션(Docker Swarm, Mesos, Nomad 등)들은 현업의 요구 사항을 충족시키기에는 부족한 점이 있었음
- 구글이 **쿠버네티스(Kubernetes)**를 오픈 소스로 공개하면서 컨테이너 인프라 환경을 좀 더 효율적으로 관리할 수 있게 됨



## ▼ ★ 3.1 쿠버네티스 이해하기

쿠버네티스를 컨테이너 관리 도구라고 설명했지만, 실제로 쿠버네티스는 컨테이너 오케스트레이션을 위한 솔루션입니다.

### • 오케스트레이션(Orchestration)

- 다수의 컨테이너를 다수의 시스템에서 각각의 목적에 따라 배포, 복제, 장애 복구 등 종합적으로 관리하는 것 → 자동화
- 복잡한 단계를 관리하고 요소들의 유기적인 관계를 미리 정의해 손쉽게 사용하도록 서비스를 제공하는 것을 의미합니다.
- 다수의 컨테이너를 유기적으로 연결, 실행, 종료할 뿐만 아니라 상태를 추적하고 보존하는 등 컨테이너를 안정적으로 사용할 수 있게 만들어 주는 것이 컨테이너 오케스트레이션입니다.

## ▼ ♥ 3.1.1 왜 쿠버네티스일까



Copyright © Gilbut, Inc. All rights reserved.

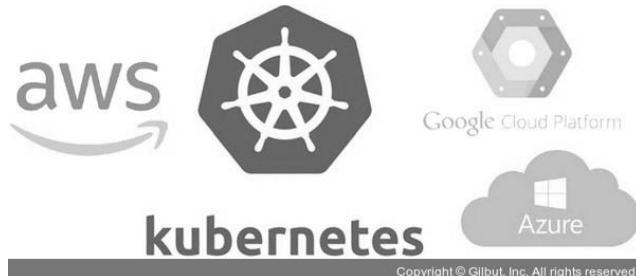
구분	도커 스웜	메소스	노마드	쿠버네티스
설치 난이도	쉬움	매우 어려움	쉬움	어려움
사용 편의성	매우 좋음	좋음	매우 좋음	좋음
세부 설정 지원	거의 없음	있음	거의 없음	다양하게 있음
안정성	매우 안정적임	안정적임	안정적임	매우 안정적임
확장성	어려움	매우 잘 됨	어려움	매우 잘 됨
정보량	많음	적음	적음	매우 많음
에코 파트너	없음	거의 없음	있음	매우 많음
학습 곡선	쉬움	매우 어려움	어려움	어려움

2021년을 기준으로 대부분 IT 기업에서는 쿠버네티스와 관련된 프로젝트를 진행하고 있거나 이미 자사 솔루션으로 흡수 및 통합했습니다. 이에 따라 다양한 종류의 솔루션이 쿠버네티스에 통합되고 있습니다. 그러므로 컨테이너 오케스트레이션을 한다면 쿠버네티스를 우선으로 고려해야 합니다.

### ▼ 3.1.2 쿠버네티스 구성 방법

쿠버네티스를 구성하는 방법은 크게 3가지입니다.

- 퍼블릭 클라우드 업체에서 제공하는 관리형 쿠버네티스인 EKS(Amazon Elastic Kubernetes Service), AKS(Azure Kubernetes Services), GKE(Google Kubernetes Engine) 등을 사용합니다. 구성이 이미 다 갖춰져 있고 마스터 노드를 클라우드 업체에서 관리하기 때문에 학습용으로는 적합하지 않습니다.



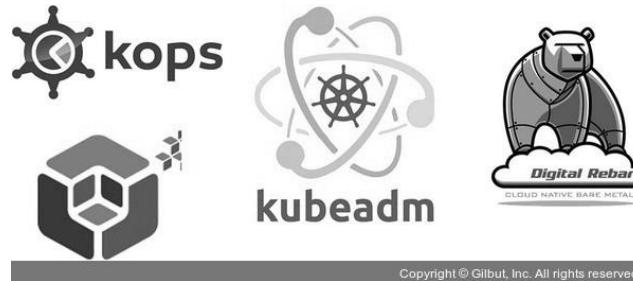
Copyright © Gilbut, Inc. All rights reserved.

- 수세의 Rancher, 레드햇의 OpenShift와 같은 플랫폼에서 제공하는 설치형 쿠버네티스를 사용합니다. 하지만 유료라 쉽게 접근하기 어렵습니다.



Copyright © Gilbut, Inc. All rights reserved.

- 사용하는 시스템에 쿠버네티스 클러스터를 자동으로 구성해주는 솔루션을 사용합니다. 주요 솔루션으로는 kubeadm, kops(Kubernetes Operations), KRIB(Kubernetes Rebar Integrated Bootstrap), kubespray가 있습니다. 4가지의 주요 솔루션 중에 kubeadm이 가장 널리 알려져 있습니다. kubeadm은 사용자가 변경하기도 수월하고, 온프레미스(On-Premises)와 클라우드를 모두 지원하며, 배우기도 쉽습니다. 이러한 솔루션들을 **구성형 쿠버네티스**라고 합니다.

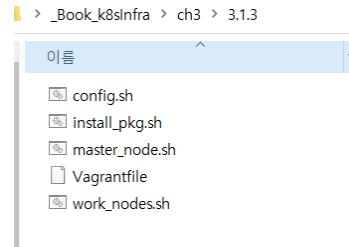


구분	KRIB	kops	<u>kubespray</u>	<u>kubeadm</u>
정보량	적음	많음	많음	매우 많음
세부 설정 변경	가능함	가능함	제한적으로 가능함	다양함
사전 요구 조건	적음	적음	적음	매우 적음
온프레미스 지원	매우 좋음	안 됨	좋음	매우 좋음
클라우드 지원	안 됨	매우 좋음	좋음	좋음
학습 곡선	매우 어려움	어려움	쉬움	어려움
자동화 기능	제공됨	제공됨	쉽게 제공됨	제한적으로 제공됨

### ▼ 🌟 3.1.3 kubeadm으로 쿠버네티스 구성하기

- 베이그런트를 이용해 쿠버네티스 테스트 환경 구축하기
  - 베이그런트 : <https://kgw7401.tistory.com/44#Vagrant란-1>

#### ▼ 🔥 코드 설명



#### • Vagrantfile

- 베이그런트 프로비저닝을 위한 정보를 담고 있는 메인 파일
- 명령 프롬프트를 실행하고 Vagrantfile이 있는 경로에서 vagrant up 명령을 입력하면 현재 호스트 내부에 Vagrantfile에 정의된 가상 머신들을 생성하고 생성한 가상 머신에 쿠버네티스 클러스터를 구성하기 위한 파일들을 호출해 쿠버네티스 클러스터를 자동으로 구성합니다.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  N = 3 # max number of worker nodes
  Ver = '1.18.4' # Kubernetes Version to install

  #####
  # Master Node #
  #####
  config.vm.define "m-k8s" do |cfg|
    cfg.vm.box = "sysnet4admin/CentOS-k8s"
    cfg.vm.provider "virtualbox" do |vb|
      vb.name = "m-k8s(github_SysNet4Admin)"
      vb.cpus = 2
      vb.memory = 3072
      vb.customize ["modifyvm", :id, "--groups", "/k8s-SgMST-1.13.1(github_SysNet4Admin)"]
    end
    cfg.vm.host_name = "m-k8s"
    cfg.vm.network "private_network", ip: "192.168.1.10"
    cfg.vm.network "forwarded_port", guest: 22, host: 60010, auto_correct: true, id: "ssh"
    cfg.vm.synced_folder "../data", "/vagrant", disabled: true
    cfg.vm.provision "shell", path: "config.sh", args: N
  end

  #쿠버네티스 버전 정보(Ver)와 Main이라는 문자를 install_pkg.sh로 넘깁니다.
  #Ver은 각 노드에 해당 버전의 쿠버네티스 버전을 설치하게 합니다.
  #Main은 install_pkg.sh에서 조건문으로 처리해 마스터 노드에만 전체 실행 코드를 내려받게 합니다.
  cfg.vm.provision "shell", path: "install_pkg.sh", args: [ Ver, "Main" ]
  cfg.vm.provision "shell", path: "master_node.sh"
end
```

```

=====
# Worker Nodes #
=====

(1..N).each do |i|
  config.vm.define "w#{i}-k8s" do |cfg|
    cfg.vm.box = "sysnet4admin/CentOS-k8s"
    cfg.vm.provider "virtualbox" do |vb|
      vb.name = "w#{i}-k8s(github_SysNet4Admin)"
      vb.cpus = 1
      vb.memory = 2560
      vb.customize ["modifyvm", :id, "--groups", "/k8s-SgMST-1.13.1(github_SysNet4Admin)"]
    end
    cfg.vm.host_name = "w#{i}-k8s"
    cfg.vm.network "private_network", ip: "192.168.1.10#{i}"
    cfg.vm.network "forwarded_port", guest: 22, host: "6010#{i}", auto_correct: true, id: "ssh"
    cfg.vm.synced_folder "../data", "/vagrant", disabled: true
    cfg.vm.provision "shell", path: "config.sh", args: N
    cfg.vm.provision "shell", path: "install_pkg.sh", args: Ver
    cfg.vm.provision "shell", path: "work_nodes.sh"
  end
end

```

- config.sh

- kubeadm으로 쿠버네티스를 설치하기 위한 사전 조건을 설정하는 스크립트 파일

```

#!/usr/bin/env bash

# vim configuration (vim을 호출하도록 프로파일에 입력)
echo 'alias vi=vim' >> /etc/profile

# swapoff -a to disable swapping
swapoff -a
# sed to comment the swap partition in /etc/fstab (시스템이 다시 시작되더라도 스왑되지 않도록 설정)
sed -i.bak -r 's/(.+ swap .+)/#\1/' /etc/fstab

# kubernetes repo (쿠버네티스를 내려받을 레포지터리 설정)
gg_pkg="packages.cloud.google.com/yum/doc" # Due to shorten addr for key
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://${gg_pkg}/yum-key.gpg https://${gg_pkg}/rpm-package-key.gpg
EOF

# Set SELinux in permissive mode (effectively disabling it) (selinux가 제한적으로 사용되지 않도록 permissive 모드로 변경)
setenforce 0
sed -i '/^SELINUX=enforcing$/ s/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

# RHEL/CentOS 7 have reported traffic issues being routed incorrectly due to iptables bypassed
# 브리지 네트워크를 통과하는 IPv4와 IPv6의 패킷을 iptables가 관리하게 설정
# 파드(Pod, 쿠버네티스에서 실행되는 객체의 최소 단위)의 통신을 iptables로 제어
# br_netfilter 커널 모듈을 사용해 브리지로 네트워크를 구성해 iptables가 활성화 됨
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
modprobe br_netfilter

# local small dns & vagrant cannot parse and delivery shell code.
# 쿠버네티스 안에서 노드 간 통신을 이름으로 할 수 있도록 각 노드의 호스트 이름과 IP를 /etc/hosts에 설정
# 이때 워커 노드는 Vagrantfile에서 넘겨받은 N 번수로 전달된 노드 수에 맞게 동적으로 생성
echo "192.168.1.10 m-k8s" >> /etc/hosts
for (( i=1; i<=$1; i++ )); do echo "192.168.1.10${i} w${i}-k8s" >> /etc/hosts; done

# config DNS (외부와 통신할 수 있게 DNS 서버 지정)
cat <<EOF > /etc/resolv.conf
nameserver 1.1.1.1 #cloudflare DNS
nameserver 8.8.8.8 #Google DNS
EOF

```

- Install\_pkg.sh

- 클러스터를 구성하기 위해서 가상 머신에 설치돼야 하는 의존성 패키지를 명시
- 또한 실습에 필요한 소스 코드를 특정 가상 머신(m-k8s) 내부에 내려받도록 설정

```

#!/usr/bin/env bash

# install packages
yum install epel-release -y
yum install vim-enhanced -y
yum install git -y

```

```

# install docker
yum install docker -y && systemctl enable --now docker

# install kubernetes cluster
yum install kubectl-$1 kubelet-$1 kubeadm-$1 -y
systemctl enable --now kubelet

# git clone _Book_k8sInfra.git
# 실행 코드를 마스터 노드에만 내려받도록 Vagrantfile에서 두번째 변수(Main)을 넘겨 받음
# 내려받은 후 홈 디렉터리로 이동
# 배시 스크립트(.sh)를 find로 찾아서 실행 가능한 상태가 되도록 chmod 700으로 설정함
if [ $2 = 'Main' ]; then
    git clone https://github.com/sysnet4admin/_Book_k8sInfra.git
    mv /home/vagrant/_Book_k8sInfra $HOME
    find $HOME/_Book_k8sInfra/ -regex ".*\.(sh\|)" -exec chmod 700 {} \;
fi

```

- master\_node.sh
  - 1개의 가상 머신(m-k8s)을 쿠버네티스 마스터 노드로 구성하는 스크립트
  - 마스터 노드는 클러스터의 제어와 관리를 담당
  - 여기서 쿠버네티스 클러스터를 구성할 때 꼭 선택해야 하는 컨테이너 네트워크 인터페이스(CNI, Container Network Interface)도 함께 구성합니다.
  - `--token` 옵션 : 클러스터의 워커 노드가 마스터 노드에 참여하기 위한 인증 토큰을 지정하는 데 사용됩니다. 토큰은 클러스터에 대한 신뢰성과 보안을 위해 사용되며, 워커 노드가 클러스터에 참여하고 마스터 노드와 통신할 수 있도록 하용합니다. `--token-ttl` 옵션은 토큰의 유효 기간을 설정하는 데 사용됩니다. `0`을 지정하면 토큰이 만료되지 않도록 설정됩니다. 이렇게 하면 워커 노드가 마스터 노드에 참여하는 동안 토큰이 계속 유효하게 됩니다.

```

#!/usr/bin/env bash

# init kubernetes
# 토큰 지정, ttl(유지시간)을 0으로 설정해서 24시간 후에 토큰이 계속 유지되게 함
# 쿠버네티스가 자동으로 컨테이너에 부여하는 네트워크를 172.16.0.0/16으로 제공
# 워커 노드가 접속하는 API 서버의 IP를 192.168.1.10으로 지정해 워커 노드들이 자동으로 API 서버에 연결
kubeadm init --token 123456.1234567890123456 --token-ttl 0 \
--pod-network-cidr=172.16.0.0/16 --apiserver-advertise-address=192.168.1.10

# config for master node only
# 마스터 노드에서 현재 사용자가 쿠버네티스를 정상적으로 구동할 수 있게
# 설정 파일을 루트의 홈디렉터리에 복사하고 사용자에게 권한 부여
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config

# config for kubernetes's network
kubectl apply -f \
https://raw.githubusercontent.com/sysnet4admin/IaC/master/manifests/172.16_net_calico.yaml

```

- work\_nodes.sh
  - 3대의 가상 머신(w1-k8s, w2-k8s, w3-k8s)에 쿠버네티스 워커 노드를 구성하는 스크립트
  - 워커 노드는 애플리케이션 컨테이너를 실행하는 데 주로 사용 (실제 작업자 노드)
  - 마스터 노드에 구성된 클러스터에 조인이 필요한 정보가 모두 코드화되어 있어 스크립트를 실행하기만 하면 편하게 워커 노드로서 쿠버네티스 클러스터에 조인됩니다.

```

#!/usr/bin/env bash

# config for work_nodes only
# kubeadm을 이용해 쿠버네티스 마스터 노드에 접속합니다.
# 아래 연결에 필요한 토큰은 기존에 마스터 노드에서 생성한 토큰을 사용합니다.
# 인증을 무시하고, API 서버 주소인 192.168.1.10으로 기본 포트 번호인 6443번 포트에 접속하도록 설정합니다.
kubeadm join --token 123456.1234567890123456 \
--discovery-token-unsafe-skip-ca-verification 192.168.1.10:6443

```

1. `vagrant up`

C:\Users\User\Desktop\\_Book\_k8sInfra\ch3\3.1.3>vagrant up

2. superputty 열고 m-k8s 터미널 접속

3. `kubectl get nodes` : 쿠버네티스 클러스터에 마스터 노드와 워커 노드들이 정상적으로 생성되고 연결됐는지 확인

```

[m-k8s] ~ [w1-k8s] ~ [w2-k8s] ~ [w3-k8s]
$ Using username "root".
Last failed login: Fri Dec  6 07:43:55 KST 2019 on pts/0
There was 1 failed login attempt since the last successful login.
[root@m-k8s ~]# kubectl get nodes
NAME     STATUS   ROLES    AGE     VERSION
m-k8s   Ready    master   21m    v1.18.4
w1-k8s  Ready    <none>   15m    v1.18.4
w2-k8s  Ready    <none>   8m59s   v1.18.4
w3-k8s  Ready    <none>   2m36s   v1.18.4
[root@m-k8s ~]#

```

### ▼ 🌟 3.1.4 파드 배포를 중심으로 쿠버네티스 구성 요소 살펴보기

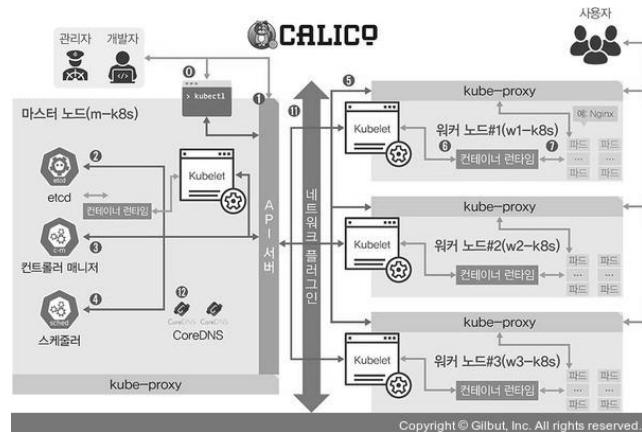
- `kubectl get pods --all-namespaces` : 설치된 쿠버네티스 구성 요소 확인
  - `--all-namespaces` : 기본 네임스페이스인 default 외에 모든 것을 표시하겠다는 의미. 따라서 모든 네임 스페이스에서 파드를 수집해 보여 줌

```

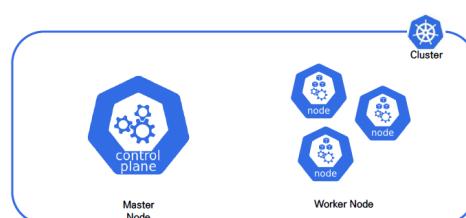
[root@m-k8s ~]# kubectl get pods --all-namespaces
NAMESPACE   NAME                               READY   STATUS    RESTARTS   AGE
kube-system  calico-kube-controllers-99c9b6f64-1787s   1/1    Running   0          22m
kube-system  calico-node-7455j                  1/1    Running   0          22m
kube-system  calico-node-7rkrm                  1/1    Running   0          10m
kube-system  calico-node-9rgc9                  1/1    Running   0          16m
kube-system  calico-node-vc2lq                  1/1    Running   0          4m26s
kube-system  coredns-66bf467f8-775ff           1/1    Running   0          22m
kube-system  coredns-66bf467f8-vqcxg           1/1    Running   0          22m
kube-system  etcd-m-k8s                         1/1    Running   0          23m
kube-system  kube-apiserver-m-k8s              1/1    Running   0          23m
kube-system  kube-controller-manager-m-k8s       1/1    Running   0          23m
kube-system  kube-proxy-5q74b                  1/1    Running   0          4m26s
kube-system  kube-proxy-hrnvn                 1/1    Running   0          16m
kube-system  kube-proxy-k8dv4                  1/1    Running   0          10m
kube-system  kube-proxy-kxvwf                  1/1    Running   0          22m
kube-system  kube-scheduler-m-k8s              1/1    Running   0          23m

```

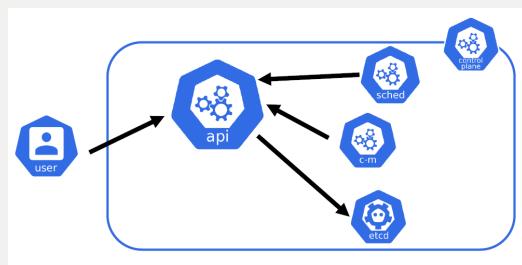
### ▼ 🚫 관리자나 개발자가 파드를 배포할 때



- Cluster = Master Node (Control Plane) + Worker Node (Data Plane)



## 마스터 노드



0. **kubectl**: 쿠버네티스 클러스터에 명령을 내리는 역할을 합니다. 다른 구성 요소들과 다르게 바로 실행되는 명령 형태인 바이너리(binary)로 배포되기 때문에 **마스터 노드에 있을 필요는 없습니다.**

❶ **API 서버**: 쿠버네티스 클러스터의 중심 역할을 하는 풀로입니다. 주로 상태 값을 저장하는 etcd와 통신하지만, 그 밖의 요소들 또한 API 서버를 중심에 두고 통신하므로 API 서버의 역할이 매우 중요합니다. 회사에 비유하면 모든 직원과 상황을 관리하고 목표를 설정하는 관리자에 해당합니다.

→ [명령어를 전달하는 중간 다리](#)

❷ **etcd**: 구성 요소들의 상태 값이 모두 저장되는 곳입니다. 회사의 관리자가 모든 보고 내용을 기록하는 노트라고 생각하면 됩니다. 실제로 etcd 외의 다른 구성 요소는 상태 값을 관리하지 않습니다. 그러므로 etcd의 정보만 백업돼 있다면 긴급한 장애 상황에서도 쿠버네티스 클러스터는 복구할 수 있습니다. 또한 etcd는 분산 저장이 가능한 key-value 저장소이므로, 복제해 여러 곳에 저장해 두면 하나의 etcd에서 장애가 나더라도 시스템의 가용성을 확보할 수 있습니다. (구성 정보를 퍼뜨려 저장하겠다는 의미)

→ [모든 클러스터의 구성 데이터를 저장하는 저장소\(DB\)](#)

❸ **컨트롤러 매니저**: 컨트롤러 매니저는 쿠버네티스 클러스터의 오브젝트 상태를 관리합니다. 예를 들어 워커 노드에서 통신이 되지 않는 경우, 상태 체크와 복구는 컨트롤러 매니저에 속한 노드 컨트롤러에서 이루어집니다. 다른 예로 레플리카셋 컨트롤러는 레플리카셋에 요청받은 파드 개수대로 파드를 생성합니다. 뒤에 나오는 서비스와 파드를 연결하는 역할을 하는 엔드포인트 컨트롤러 또한 컨트롤러 매니저입니다. 이와 같이 다양한 상태 값을 관리하는 주체들이 컨트롤러 매니저에 소속돼 각자의 역할을 수행합니다.

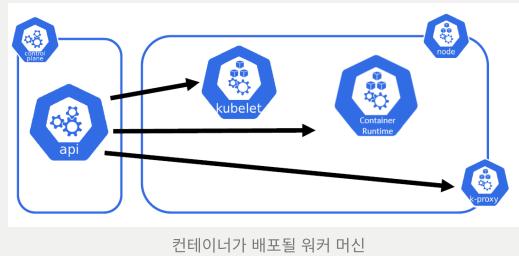
→ [클러스터의 상태를 조절하는 컨트롤러들을 생성, 배포](#)

❹ **스케줄러**: 노드의 상태와 자원, 레이블, 요구 조건 등을 고려해 파드를 어떤 워커 노드에 생성할 것인지를 결정하고 할당합니다. 스케줄러라는 이름에 걸맞게 파드를 조건에 맞는 워커 노드에 지정하고, 파드가 워커 노드에 할당되는 일정을 관리하는 역할을 담당합니다.

→ [Pod의 생성 명령\(ex. 컨테이너 생성\)이 있을 경우 어떤 Node에 배포할지 결정](#)

→ [위치만 결정하고 만드는 건 다른 놈이 만듦](#) (kubelet은 스케줄러가 선택한 워커 노드에서 파드 스펙을 바탕으로 컨테이너를 생성하고 실행하며, 해당 워커 노드에서 파드의 생명 주기와 상태를 모니터링함)

### 워커 노드

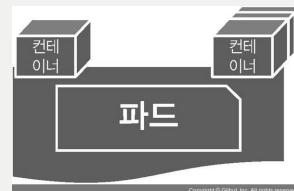


컨테이너가 배포될 워커 머신

❸ **kubelet:** 파드의 구성 내용(PodSpec)을 받아서 컨테이너 런타임으로 전달하고, **파드 안의 컨테이너들이 정상적으로 작동하는지 모니터링**합니다.

❹ **컨테이너 런타임(CRI, Container Runtime Interface):** **파드를 이루는 컨테이너의 실행을 담당합니다.** 파드 안에서 다양한 종류의 컨테이너가 문제 없이 작동하게 만드는 표준 인터페이스입니다.

❺ **파드(Pod):** **한 개 이상의 컨테이너로 단일 목적의 일을 하기 위해서 모인 단위입니다.** 즉, 웹 서버 역할을 할 수도 있고 로그나 데이터를 분석할 수도 있습니다. 여기서 중요한 것은 파드는 언제라도 죽을 수 있는 존재라는 점입니다. 이것이 쿠버네티스를 처음 배울 때 가장 이해하기 어려운 부분입니다. 가상 머신은 언제라도 죽을 수 있다고 가정하고 디자인하지 않지만, 파드는 언제라도 죽을 수 있다고 가정하고 설계됐기 때문에 쿠버네티스는 여러 대안을 디자인했습니다.



### 선택 가능한 구성 요소

❻ 네트워크 플러그인 - 쿠버네티스 클러스터의 통신을 위해서 네트워크 플러그인을 선택하고 구성해야 함

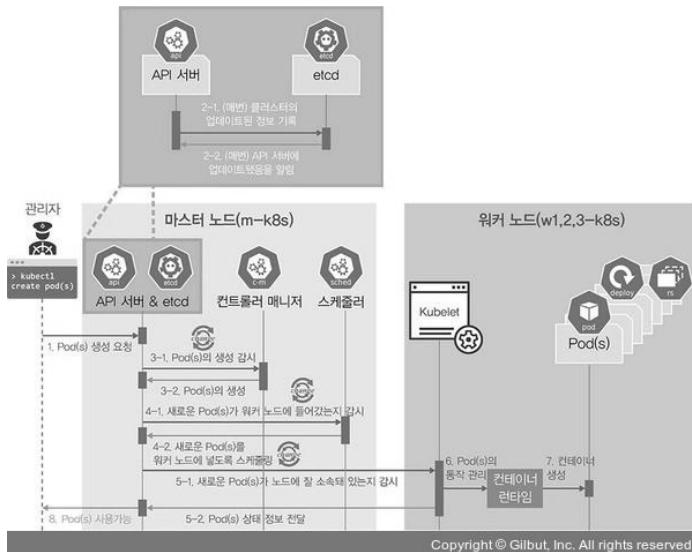
❼ CoreDNS - 클라우드 네이티브 컴퓨팅 재단에서 보증하는 프로젝트로, 빠르고 유연한 DNS 서버

### ▼ 🏁 사용자가 배포된 파드에 접속할 때

- **kube-proxy:** 쿠버네티스 클러스터는 파드가 위치한 노드에 **kube-proxy**를 통해 파드가 통신할 수 있는 네트워크를 설정합니다. 이때 실제 통신은 **br netfilter와 iptables**로 관리합니다.
  - 쿠버네티스 클러스터의 각 노드마다 실행되고 있으면서, **각 노드 간의 통신을 담당**
  - 네트워크 규칙을 만들고, 클러스터 내-외부(인터넷환경)와 통신하도록 pod에게 ip를 할당
- **파드:** 이미 배포된 파드에 접속하고 필요한 내용을 전달 받습니다. 이때 대부분 사용자는 파드가 어느 워커 노드에 위치하는지 신경 쓰지 않아도 됩니다.

### ▼ ❤️ 3.1.5 파드의 생명주기로 쿠버네티스 구성 요소 살펴보기

쿠버네티스의 가장 큰 장점은 쿠버네티스의 구성 요소마다 하는 일이 명확하게 구분돼 각자의 역할만 충실히 수행하면 클러스터 시스템이 안정적으로 운영된다는 점입니다. 또한 역할이 나뉘어 있어서 문제가 발생했을 때 어느 부분에서 문제가 발생했는지 디버깅하기 쉽습니다.

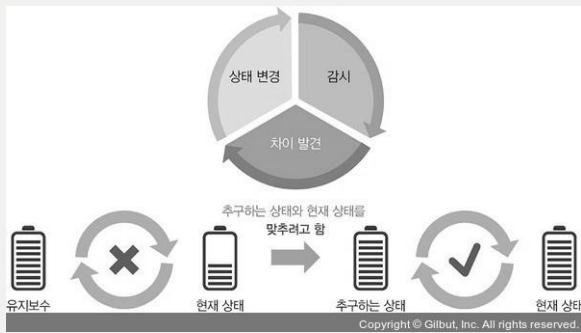


### 생명주기(life cycle) : 파드가 생성, 수정, 삭제되는 과정

1. kubectl을 통해 API 서버에 파드 생성을 요청합니다.
2. (업데이트가 있을 때마다 매번) API 서버에 전달된 내용이 있으면 API 서버는 etcd에 전달된 내용을 모두 기록해 클러스터의 상태 값을 최신으로 유지합니다. 따라서 각 요소가 상태를 업데이트할 때마다 모두 API 서버를 통해 etcd에 기록됩니다.
3. API 서버에 파드 생성이 요청된 것을 컨트롤러 매니저가 인지하면 컨트롤러 매니저는 파드를 생성하고, 이 상태를 API 서버에 전달 합니다. 참고로 아직 어떤 워커 노드에 파드를 적용할지는 결정되지 않은 상태로 파드만 생성됩니다.
4. API 서버에 파드가 생성됐다는 정보를 스케줄러가 인지합니다. 스케줄러는 생성된 파드를 어떤 워커 노드에 적용할지 조건을 고려해 결정하고 해당 워커 노드에 파드를 띄우도록 요청합니다.
5. API 서버에 전달된 정보대로 지정한 워커 노드에 파드가 속해 있는지 스케줄러가 kubelet으로 확인합니다.
6. kubelet에서 컨테이너 런타임으로 파드 생성을 요청합니다.
7. 파드가 생성됩니다.
8. 파드가 사용 가능한 상태가 됩니다.



### 쿠버네티스의 시스템 구조



- **마스터 노드**는 이미 생성된 파드들을 유기적으로 연결하므로 쿠버네티스 클러스터를 안정적으로 유지하려면 선언적인 시스템이 나옵
  - 쿠버네티스는 워크플로(workflow, 작업 절차) 구조가 아니라 선언적인(declarative) 시스템 구조를 가지고 있습니다. 즉, 각 요소가 **추구하는 상태**(desired status)를 선언하면 **현재 상태**(current status)와 맞는지 점검하고 그것에 맞추려고 노력하는 구조로 돼 있다는 뜻입니다.
  - 따라서 추구하는 상태를 API 서버에 선언하면 다른 요소들이 API 서버에 와서 현재 상태와 비교하고 그에 맞게 상태를 변경하려고 합니다. 여기서 API는 현재 상태 값을 가지고 있는데, 이것을 보존해야 해서 etcd가 필요합니다. API 서버와 etcd는 거의 한 몸처럼 움직이도록 설계됐습니다.
- **워커 노드**는 (작업을 순서대로 진행하는) 워크플로 구조에 따라 설계
  - 쿠버네티스가 kubelet과 컨테이너 런타임을 통해 파드를 새로 생성하고 제거해야 하는 구조여서 선언적인 방식으로 구조화하기에는 어려움이 있기 때문입니다.
  - 또한 명령이 절차적으로 전달되는 방식은 시스템의 성능을 높이는 데 효율적입니다.

### ▼ 3.1.6 쿠버네티스 구성 요소의 기능 검증하기

#### ▼ kubectl

앞에서 **kubectl**은 꼭 마스터 노드에 위치할 필요 없다고 했습니다. 실제로 **쿠버네티스 클러스터의 외부**에서 **쿠버네티스 클러스터에 명령을 내릴** 수도 있습니다.

1. 슈퍼푸티 세션 창에서 w3-k8s 접속
2. `kubectl get nodes`

```
[root@w3-k8s ~]# kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

- 쿠버네티스 클러스터의 정보를 kubectl이 알지 못하기 때문에 명령을 실행해도 쿠버네티스의 노드들에 대한 정보가 표시되지 않습니다
- kubectl은 API 서버를 통해 쿠버네티스에 명령을 내립니다. → 따라서 kubectl이 어디에 있더라도 API 서버의 접속 정보만 있다면 어느 곳에서든 쿠버네티스 클러스터에 명령을 내릴 수 있습니다.

3. `scp root@192.168.1.10:/etc/kubernetes/admin.conf .`

```
[root@w3-k8s ~]# scp root@192.168.1.10:/etc/kubernetes/admin.conf .
The authenticity of host '192.168.1.10 (192.168.1.10)' can't be established.
ECDSA key fingerprint is SHA256:16XikZFgOibzSyggZ6+UYHUnEmjFEFhx7PpZw0I3WaM.
ECDSA key fingerprint is MD5:09:74:43:ef:38:3e:36:a1:7e:51:76:1a:ac:2d:7e:0c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.10' (ECDSA) to the list of known hosts.
root@192.168.1.10's password:
admin.conf                                         100% 5452     353.7KB/s   00:00
```

- `admin.conf` : 마스터 노드에서 생성되는 클러스터 관리를 위한 구성 파일입니다. 이 파일은 클러스터 설정 및 인증 정보를 포함하고 있으며, 클러스터에 대한 접근과 제어를 가능하게 합니다.
- **쿠버네티스 클러스터의 정보**(/etc/kubernetes/admin.conf)를 마스터 노드(`192.168.1.10` - `config.sh` 파일 참고)에서 `scp(secure copy)` 명령으로 **w3-k8s의 현재 디렉터리(.)에 받아옵니다.**
- 접속 기록이 없기 때문에 `known_hosts`로 저장하도록 `yes`를 입력합니다.
- 마스터 노드의 접속 암호인 `vagrant`도 입력합니다.

4. `kubectl get nodes --kubeconfig admin.conf`

```
[root@m3-k8s ~]# kubectl get nodes --kubeconfig admin.conf
NAME     STATUS   ROLES    AGE     VERSION
m-k8s   Ready    master   95m    v1.18.4
w1-k8s  Ready    <none>   89m    v1.18.4
w2-k8s  Ready    <none>   83m    v1.18.4
w3-k8s  Ready    <none>   77m    v1.18.4
```

- 워커 노드에서 `kubectl` 명령어를 실행할 때 `--kubeconfig` 옵션을 사용하여 `admin.conf` 파일을 지정하면, 워커 노드도 마스터 노드의 클러스터와 통신할 수 있게 됩니다. `--kubeconfig admin.conf`를 추가하여 실행한 `kubectl` 명령어는 `admin.conf` 파일을 사용하여 클러스터에 접속하고 노드 목록을 조회할 수 있습니다.
- `admin.conf`라는 kubeconfig(클러스터와의 통신에 필요한 클라이언트 설정을 저장하는 파일)을 사용하여 클러스터의 노드 목록을 조회하게 됩니다. 현재 클러스터에 등록된 노드들의 상태와 정보를 확인할 수 있습니다.

▼  **kubelet**

`kubelet`은 쿠버네티스에서 파드의 생성과 상태 관리 및 복구 등을 담당하는 매우 중요한 구성 요소입니다.

1. `kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml`

```
[root@m-m-k8s ~]# kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml
pod/nginx-pod created
```

- 파드 배포 : nginx 웹 서버 파드를 배포합니다. 파드의 구성 내용을 파일로 읽어 들여 1개의 파드를 임의의 워커 노드에 배포하는 것입니다.
- `-f` 옵션 : filename을 의미합니다.

2. `kubectl get pod` 명령으로 배포된 파드가 정상적으로 배포된 상태(Running)인지 확인합니다.

```
[root@m-m-k8s ~]# kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
nginx-pod  1/1     Running   0          99s
```

3. `kubectl get pods -o wide` 명령을 실행해 파드가 배포된 워커 노드를 확인합니다.

```
[root@m-m-k8s ~]# kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE      IP           NODE     NOMINATED NODE   READINESS
GATES
nginx-pod  1/1     Running   0          3m25s   172.16.103.129  w2-k8s  <none>        <none>
```

- `-o` : output의 약어로 출력을 특정 형식으로 해줌
- `wide` : 출력 정보를 더 많이 표시해주는 옵션

4. 배포된 노드에 접속해 `systemctl stop kubelet`으로 kubelet 서비스를 멈춥니다.

5. m-k8s에서 `kubectl get pod`로 상태를 확인하고, `kubectl delete pod nginx-pod`로 파드를 삭제합니다.

6. 기다려도 명령 창에 변화가 없으면 `ctrl+c`를 눌러 명령을 중지하고 `kubectl get pod`로 파드의 상태를 확인합니다. 실행 결과를 보면 Status가 Terminating으로 nginx-pod를 삭제하고 있습니다.

```
[root@m-m-k8s ~]# kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
nginx-pod  1/1     Terminating   0          15m
```

7. 하지만 kubelet이 작동하지 않는 상태라 파드는 삭제되지 않습니다. 워커노드에서 `systemctl start kubelet`으로 kubelet을 복구합니다.

```
[root@w2-k8s ~]# systemctl start kubelet
```

8. 잠시 후 `kubectl get pod` 명령을 실행해 nginx-pod가 삭제됐는지 확인합니다.

▼  **kube-proxy**

kubelet이 파드의 상태를 관리한다면 kube-proxy는 파드의 통신을 담당합니다. 앞서 `config.sh` 파일에서 br\_netfilter 커널 모듈을 적재하고 iptables를 거쳐 통신하도록 설정했습니다.

1. `kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml`

- 테스트하기 위해 마스터 노드인 m-k8s에서 다시 파드를 배포합니다.

2. `kubectl get pod -o wide` 명령으로 파드의 IP와 워커 노드를 확인합니다.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS
nginx-pod	1/1	Running	0	4m20s	172.16.221.129	wl-k8s	<none>	<none>

3. `curl 172.16.221.129` (client URL)로 파드의 전 단계에서 확인한 파드의 IP로 nginx 웹 서버 메인 페이지 내용을 확인합니다.

```
[root@m-k8s ~]# curl 172.16.221.129
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">http://nginx.org/.<br/>
Commercial support is available at
<a href="http://nginx.com/">http://nginx.com/.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

4. `modprobe -r br_netfilter` 명령으로 파드가 위치한 워커 노드에서 br\_netfilter 모듈을 제거합니다.

- `-r` : remove

5. `systemctl restart network` 으로 네트워크를 다시 시작해 변경된 내용을 적용합니다. 이렇게 kube-proxy에 문제가 생기는 상황을 만듭니다.

6. 다시 한 번 `curl 172.16.221.129` 로 nginx 웹 서버 페이지 정보를 받아옵니다. `Connection timed out` 으로 실패합니다.

```
[root@m-k8s ~]# curl 172.16.221.129
curl: (7) Failed to connect to 172.16.221.129:80; Connection timed out
```

7. `kubectl get pod -o wide` 로 파드 상태를 확인합니다.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS
nginx-pod	1/1	Running	0	11m	172.16.221.129	wl-k8s	<none>	<none>

- kubelet을 통해 확인된 파드의 노드 위치와 IP는 그대로고, 상태도 작동 중(Running)으로 문제가 없는 것처럼 보입니다. 하지만 kube-proxy가 이용하는 br\_netfilter에 문제가 있어서 파드의 nginx 웹 서버와의 통신만이 정상적으로 이루어지지 않는(curl로 nginx 서버에 접속했으나 연결이 되지 않음, `Connection timed out`) 상태입니다.

8. 다시 정상적으로 파드의 nginx 웹 서버 페이지 정보를 받아올 수 있는 상태로 만들어 봅시다. 워커 노드에서 `modprobe br_netfilter` 명령을 실행해 br\_netfilter를 커널에 적재하고 `reboot` 로 시스템을 다시 시작해 적용합니다.

```
[root@wl-k8s ~]# modprobe br_netfilter
[root@wl-k8s ~]# reboot
Terminated
```

9. 일정 시간이 지난 후에 `kubectl get pod -o wide` 로 파드의 상태를 확인하면 파드가 1회 다시 시작했다는 의미로 `RESTARTS`가 1로 증가하고 IP가 변경된 것을 확인할 수 있습니다.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE
nginx-pod	1/1	Running	1	13m	172.16.221.130	wl-k8s	<none>

10. `curl 172.16.221.130` 로 정보를 정상적으로 받아오는지 확인합니다.

11. 다음 내용을 진행하기 위해 `kubectl delete -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml` 로 배포한 파드를 삭제(delete)합니다.

```
[root@m-k8s ~]# kubectl delete -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml
pod "nginx-pod" deleted
```

## ▼ ✨ 3.2 쿠버네티스 기본 사용법 배우기

### ▼ ❤️ 3.2.1 파드를 생성하는 방법

## 1. 파드 생성

1) `kubectl run nginx-pod --image=nginx`

- `nginx-pod` : pod 이름
  - `--image=nginx` : 생성할 이미지 이름
- 이렇게 쉽게 파드를 생성할 수 있는데, 왜 그동안 어렵게 `kubectl create`라는 명령을 사용했을까요? `create`로 파드를 생성해서 `run` 방식과 비교해 보겠습니다.

2) `kubectl create deployment dpy-nginx --image=nginx`

- `kubectl create nginx --image=nginx`로 실행시 오류 발생

```
[root@m-k8s ~]# kubectl create nginx --image=nginx
Error: unknown flag: --image
See 'kubectl create --help' for usage.
```

- `create`로 파드를 생성하려면 `kubectl create`와 deployment을 추가해서 실행해야 합니다.

- 이때 기존 파드 이름인 `nginx`와 종복을 피하고자 파드의 이름을 `dpy-nginx`로 지정해 생성합니다.

- `kubectl get pods`로 생성된 파드를 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
dpy-nginx-c8d778df-mtqhr   1/1     Running   0          30s
nginx-pod              1/1     Running   0          4m54s
```

- 이름에서 `dpy-nginx`를 제외한 나머지 부분은 무작위로 생성됩니다.

2. `kubectl get pods -o wide`로 생성된 피드의 IP 확인

```
[root@m-k8s ~]# kubectl get pods -o wide
NAME                  READY   STATUS    RESTARTS   AGE      IP           NODE
DE      READINESS GATES
dpy-nginx-c8d778df-mtqhr   1/1     Running   0          3m10s   172.16.221.131   w1-k8s
<none>
nginx-pod              1/1     Running   0          7m34s   172.16.132.1   w3-k8s
```

3. `curl 172.16.221.131`, `curl 172.16.132.1`로 웹페이지 정보 받아오는지 확인

- run과 create deployment로 파드를 생성한 것의 차이



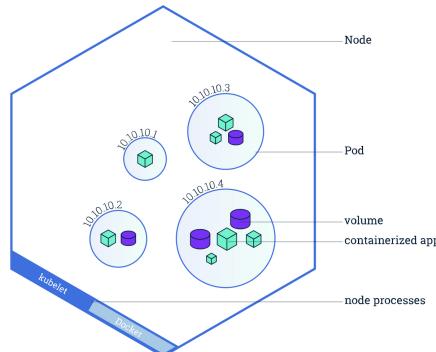
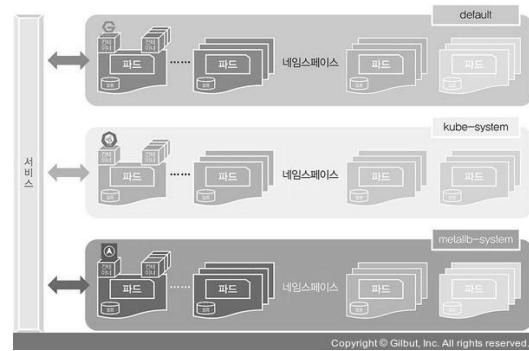
- `run`으로 파드를 생성하면 단일 파드 1개만 생성되고 관리됨 : 초코파이 1개
- `create deployment`로 파드를 생성하면 디플로이먼트(Deployment)라는 관리 그룹 내에서 파드가 생성됨 : 초코파이 상자에 들어있는 초코파이 1개

## ▼ 💛 3.2.2 오브젝트란

파드와 디플로이먼트는 **스펙(spec)**과 **상태(status)** 등의 값을 가지고 있습니다.

이러한 값을 가지고 있는 파드와 디플로이먼트를 개별 속성을 포함해 부르는 단위를 **오브젝트(Object)**라고 합니다. 쿠버네티스는 여러 유형의 오브젝트를 제공합니다.

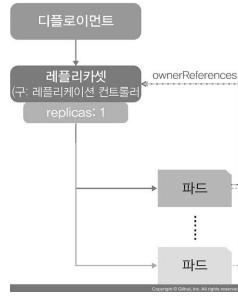
### 1. 기본 오브젝트



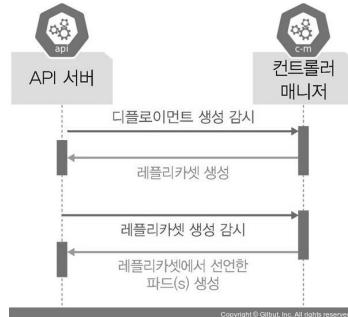
- 클러스터 : 쿠버네티스의 여러 리소스를 관리하기 위한 집합체
- 네임스페이스(Namespace)
  - 클러스터 안의 가상 클러스터. 전체 클러스터에서 특정 이름으로 클러스터의 영역을 구분합니다.
  - 쿠버네티스 클러스터에서 사용되는 리소스들을 구분해 관리하는 그룹입니다.
- 노드(node) : 쿠버네티스 스케줄러에서 파드를 할당받고 처리하는 역할을 합니다.
- 파드(Pod)
  - 쿠버네티스에서 실행되는 최소 단위, 즉 웹 서비스를 구동하는 데 필요한 최소 단위
  - 컨테이너가 모인 집합체의 단위. 이러한 파드는 노드에 배치된다.
  - 독립적인 공간과 사용 가능한 IP를 가지고 있습니다. 하나의 파드는 1개 이상의 컨테이너를 갖고 있기 때문에 여러 기능을 묶어 하나의 목적으로 사용할 수도 있습니다.
  - 쿠버네티스는 거의 모든 부분이 자동 복구되도록 설계되었는데, 파드의 자동 복구 기술을 셀프 힐링(self-healing)이라 함. 이는 제대로 작동하지 않는 컨테이너를 다시 시작하거나 교체해 파드가 정상적으로 작동하게 함
  - 쿠버네티스는 파드 자체에 문제가 발생하면 파드를 자동 복구해서 파드가 항상 동작하도록 보장하는 기능도 있음
- 볼륨(Volume)
  - 파드가 생성될 때 파드에서 사용할 수 있는 디렉터리를 제공합니다.
  - 기본적으로 파드는 영속되는 개념이 아니라 제공되는 디렉터리도 임시로 사용합니다. 하지만 파드가 사라지더라도 저장과 보존이 가능한 디렉터리를 볼륨 오브젝트를 통해 생성하고 사용할 수 있습니다.
- 서비스(Service)
  - 외부에서 쿠버네티스 클러스터에 접속하는 방법
  - 파드는 클러스터 내에서 유동적이기 때문에 접속 정보가 고정일 수 없습니다. 따라서 파드 접속을 안정적으로 유지하도록 서비스를 통해 내/외부로 연결됩니다. 그래서 서비스는 새로 파드가 생성될 때 부여되는 새로운 IP를 기준으로 제공하는 기능과 연결해 줍니다.
  - 쉽게 설명하면 쿠버네티스 외부에서 쿠버네티스 내부로 접속할 때 내부가 어떤 구조로 돼 있는지, 파드가 살았는지 죽었는지 신경 쓰지 않아도 이를 논리적으로 연결하는 것이 서비스입니다.
  - 기존 인프라에서 로드밸런서, 게이트웨이와 비슷한 역할을 합니다.

## 2. 디플로이먼트

- 기본 오브젝트만으로도 쿠버네티스를 사용할 수 있습니다. 하지만 한계가 있어서 이를 좀 더 효율적으로 작동하도록 기능들을 조합하고 추가해 구현한 것이 디플로이먼트(Deployment)입니다.



- 쿠버네티스에서 가장 많이 쓰이는 디플로이먼트 오브젝트는 파드에 기반을 두고 있으며, 레플리카셋 오브젝트를 합쳐 놓은 형태입니다.



- 레플리카셋은 똑같은 팟의 레플리카를 관리 및 제어하는 리소스인데 반해, 디플로이먼트는 레플리카셋을 관리하고 다루기 위한 리소스입니다.
  - 레플리카세트(ReplicaSet)** : 파드를 정의한 매니페스트 파일로는 파드를 하나밖에 생성할 수 없다. 그러나 어느 정도 규모가 되는 애플리케이션을 구축하려면 같은 파드를 여러 개 실행해 가용성을 확보해야 하는 경우가 생긴다. 이런 경우 사용하는 것이 레플리카세트이다. 레플리카세트는 똑같은 정의의 갖는 파드 여러개를 생성하고 관리하기 위한 리소스이다.
  - 실제로 API 서버와 컨트롤러 매니저는 단순히 파드가 생성되는 것을 감시하는 것이 아니라 디플로이먼트처럼 레플리카셋을 포함하는 오브젝트의 생성을 감시합니다.
  - 레플리카셋은 파드 수를 보장하는 기능만 제공하기 때문에 롤링 업데이트 기능 등이 추가된 디플로이먼트를 사용해 파드 수를 관리하기를 권장합니다.**
- 디플로이먼트 생성, 삭제

```

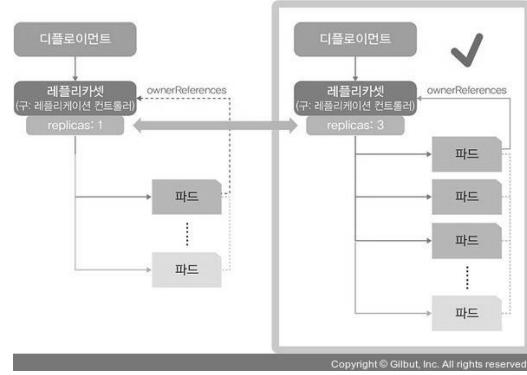
[root@m-k8s ~]# kubectl create deployment dpy-hname --image=sysnet4admin/echo-hname
deployment.apps/dpy-hname created
[root@m-k8s ~]# kubectl get pods
NAME                  READY   STATUS            RESTARTS   AGE
dpy-hname-59778b9bb-t4mvz   0/1    ContainerCreating   0          26s
dpy-nginx-c8d778df-mtqhr   1/1    Running          1          18h
nginx-pod               1/1    Running          1          18h
[root@m-k8s ~]# kubectl delete deployment dpy-hname
deployment.apps "dpy-hname" deleted

```

- `kubectl create deployment dpy-hname --image=sysnet4admin/echo-hname` : 저장소에서 필요한 이미지를 내려받아 디플로이먼트를 생성
- `kubectl get pods` 로 생성된 디플로이먼트 확인
- `kubectl delete deployment dpy-hname` : 디플로이먼트 삭제

### ▼ ❤️ 3.2.3 레플리카셋으로 파드 수 관리하기

- 레플리카셋(ReplicaSet).



- 파드를 정의한 매니페스트 파일로는 파드를 하나밖에 생성할 수 없다. 그러나 어느 정도 규모가 되는 애플리케이션을 구축하려면 같은 파드를 여러 개 실행해 가용성을 확보해야 하는 경우가 생긴다. 이런 경우 사용하는 것이 레플리카셋이다. 레플리카세트는 똑같은 정의를 갖는 파드 여러개를 생성하고 관리하기 위한 리소스이다.
- 예를 들어 파드를 3개 만들겠다고 레플리카셋에 선언하면 컨트롤러 매니저와 스케줄러가 워커 노드에 파드 3개를 만들도록 선언합니다. 그러나 레플리카셋은 파드 수를 보장하는 기능만 제공하기 때문에 롤링 업데이트 기능 등이 추가된 디플로이먼트를 사용해 파드 수를 관리하기를 권장합니다.

#### • code

- `kubectl scale deployment dpy-nginx --replicas=3` : 디플로이먼트로 생성된 dpy-nginx를 scale 명령과 --replicas=3 옵션으로 파드의 수를 3개로 만듭니다.
  - \*오류 : `kubectl scale pod nginx-pod --replicas=3` : nginx-pod는 파드로 생성됐기 때문에 디플로이먼트 오브젝트에 속하지 않습니다. 그래서 리소스를 확인할 수 없다는 에러가 발생한 것입니다.

```
[root@m-k8s ~]# kubectl scale pod nginx-pod --replicas=3
Error from server (NotFound): the server could not find the requested resource
```

- scale 명령으로 추가된 2개의 nginx 파드를 확인

```
[root@m-k8s ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
dpy-nginx-c8d778df-5q6v7	1/1	Running	0	3m39s
dpy-nginx-c8d778df-8zcbl	1/1	Running	0	3m39s
dpy-nginx-c8d778df-mtqhr	1/1	Running	1	18h
nginx-pod	1/1	Running	1	19h

- `kubectl get pods -o wide` → `kubectl delete deployment dpy-nginx` → `kubectl get pods`

```
[root@m-k8s ~]# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE   NOMINATED
DE   READINESS GATES
dpy-nginx-c8d778df-5q6v7  1/1    Running   0          5m47s  172.16.103.132  w2-k8s <none>
dpy-nginx-c8d778df-8zcbl  1/1    Running   0          5m47s  172.16.132.3   w3-k8s <none>
dpy-nginx-c8d778df-mtqhr  1/1    Running   1          19h    172.16.221.132  w1-k8s <none>
nginx-pod        1/1    Running   1          19h    172.16.132.2   w3-k8s <none>
<none>
[root@m-k8s ~]# kubectl delete deployment dpy-nginx
deployment.apps "dpy-nginx" deleted
[root@m-k8s ~]# kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
nginx-pod  1/1    Running   1          19h
```

### ▼ ❤ 3.2.4 스펙을 지정해 오브젝트 생성하기

#### ~ 3.2.5 apply로 오브젝트 생성하고 관리하기

- 디플로이먼트를 생성하면서 한꺼번에 여러 개의 파드를 만들 순 없을까요? `create`에서는 `replicas` 옵션을 사용할 수 없고, `scale`은 이미 만들어진 디플로이먼트에서만 사용할 수 있습니다.
- 이런 설정을 적용하려면 필요한 내용을 파일로 작성해야 합니다. 이때 작성하는 파일을 **오브젝트 스펙(spec)**이라고 합니다. 오브젝트 스펙은 일반적으로  **YAML** 문법으로 작성합니다.

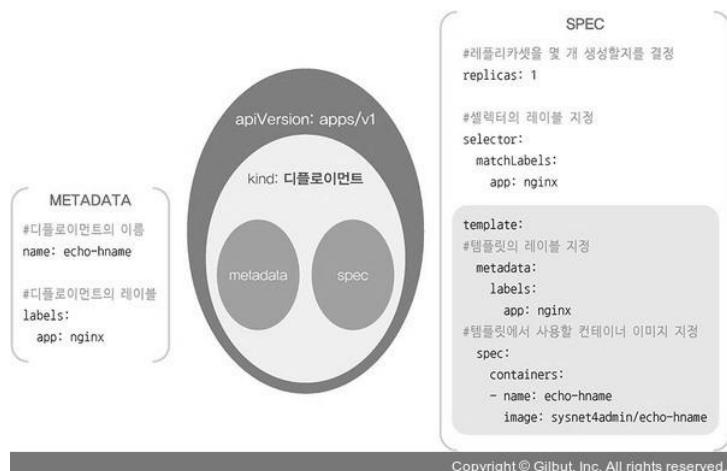
#### ▼ 코드 설명

- echo-hname.yaml

```

apiVersion: apps/v1 # API 버전
kind: Deployment # 오브젝트 종류
metadata:
  name: echo-hname
  labels:
    app: nginx
spec:
  replicas: 3 # 몇 개의 파드를 생성할지 결정
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: echo-hname
        image: sysnet4admin/echo-hname # 사용되는 이미지

```



Copyright © Gilbut, Inc. All rights reserved.

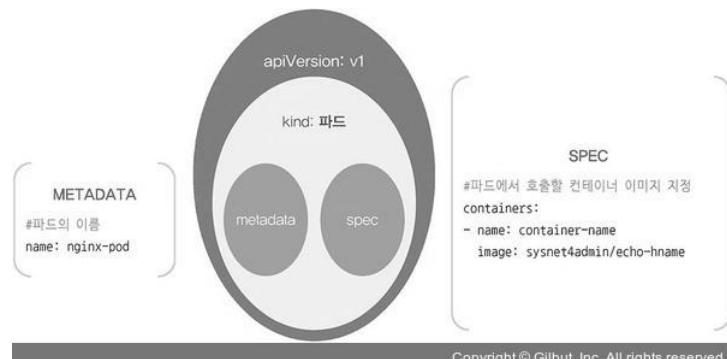
- `kubectl api-versions` : 사용 가능한 API 버전 확인

- nginx-pod.yaml

```

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
    - name: container-name
      image: nginx

```



Copyright © Gilbut, Inc. All rights reserved.

- echo-hname.yaml 파일을 이용해 디플로이먼트를 생성하기

- `kubectl create -f ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml`
- `kubectl get pods` : 파드 확인

```
[root@m-k8s ~]# kubectl create -f ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml
deployment.apps/echo-hname created
[root@m-k8s ~]# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
echo-hname-7894b67f-d87qc  1/1    Running   0          72s
echo-hname-7894b67f-kcdrw  1/1    Running   0          72s
echo-hname-7894b67f-txl28  1/1    Running   0          72s
nginx-pod      1/1    Running   1          19h
```

- `sed -i 's/replicas: 3/replicas: 6/' ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml` : 파일을 6개로 늘리기 (or vim편집기 사용)
  - sed(streamlined editor)
  - -i는 --in-place의 약어로, 변경한 내용을 현재 파일에 바로 적용하겠다는 의미
  - s/는 주어진 패턴을 원하는 패턴으로 변경하겠다는 의미 ( replicas: 3을 replicas: 6으로 변경)
- `cat ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml | grep replicas` : replicas의 값이 3에서 6으로 변경됐는지 확인
  - grep : 특정 문자열 검색
- `kubectl apply -f ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml` : 변경 내용 적용
  - create로 하면 오류 남 : 오브젝트를 처음부터 apply로 생성한 것이 아니어서 경고가 뜹니다. 경고가 떠도 작동에는 문제가 없지만 일관성에서 문제가 생길 수 있습니다. 이처럼 변경 사항이 발생할 가능성이 있는 오브젝트는 처음부터 apply로 생성하는 것이 좋습니다.

```
[root@m-k8s ~]# kubectl apply -f ~/Book_k8sInfra/ch3/3.2.4/echo-hname.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment.apps/echo-hname configured
```

- `kubectl get pods` : 6개로 늘어난 것 확인

- 오브젝트 생성 명령어 비교

구분	Run	Create	Apply
명령 실행	제한적임	가능함	안 됨
파일 실행	안 됨	가능함	가능함
변경 가능	안 됨	안 됨	가능함
실행 편의성	매우 좋음	매우 좋음	좋음
기능 유지	제한적임	지원됨	다양하게 지원됨

### ▼ 3.2.6 파드의 컨테이너 자동 복구 방법

- 셀프 힐링(self-healing) : 파드의 자동 복구 기술. 제대로 작동하지 않는 컨테이너를 다시 시작하거나 교체해 파드가 정상적으로 작동하게 함
- 셀프 힐링 기능

- `kubectl get pods -o wide` : 파드에 접속하기 위해 파드의 IP를 알아낸다.

```
[root@m-k8s ~]# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE   NOMINATED_NODE
echo-hname-7894b67f-d87qc  1/1    Running   0          11m  172.16.103.133  w2-k8s <none>
<none>
echo-hname-7894b67f-gwbh  1/1    Running   0          7m40s 172.16.221.134  w1-k8s <none>
<none>
echo-hname-7894b67f-j7fv4  1/1    Running   0          7m40s 172.16.132.5   w3-k8s <none>
<none>
echo-hname-7894b67f-jv6v}  1/1    Running   0          7m40s 172.16.103.134  w2-k8s <none>
<none>
echo-hname-7894b67f-kcdrw  1/1    Running   0          11m  172.16.132.4   w3-k8s <none>
<none>
echo-hname-7894b67f-txl28  1/1    Running   0          11m  172.16.221.133  w1-k8s <none>
<none>
nginx-pod      1/1    Running   1          19h  172.16.132.2   w3-k8s <none>
```

- `kubectl exec -it nginx-pod -- /bin/bash` : 파드 컨테이너의 셸(shell)에 접속
  - exec는 execute(실행)
  - it는 표준 입력을 명령줄 인터페이스로 작성한다는 의미
    - i 옵션은 stdin(standard input, 표준 입력)
    - t는 tty (teletypewriter)를 뜻합니다.
  - 파드인 nginx-pod에 /bin/bash를 실행해 nginx-pod의 컨테이너에서 배시(bash) 셸에 접속합니다.
- `cat /run/nginx.pid` : nginx의 PID(Process ID, 프로세서 식별자)를 확인
  - nginx의 PID는 언제나 1
  - cat : 파일의 내용을 화면에 출력
- `ls -l /run/nginx.pid` : 프로세스가 생성된 시간 확인

- `ls` 는 "list"를 나타내며, 파일 및 디렉토리의 목록을 보여주는 명령

```
[root@m-k8s ~]# kubectl exec -it nginx-pod -- /bin/bash
root@nginx-pod:/# cat /run/nginx.pid
1
root@nginx-pod:/# ls -l /run/nginx.pid
-rw-r--r--. 1 root root 2 Jul 17 06:09 /run/nginx.pid
```

- `i=1; while true; do sleep 1; echo $((i++)) `curl --silent 172.16.132.2 | grep title` ; done`

- nginx-pod의 IP (172.16.103.132)에서 돌아가는 웹 페이지를 1초마다 한 번씩 요청하는 스크립트를 실행합니다.
- curl에서 요청한 값만 받도록 `--silent` 옵션을 추가합니다.

```
[root@m-k8s ~]# i=1; while true; do sleep 1; echo $((i++)) `curl --silent 172.16.132.2 | grep title` ; done
1 <title>Welcome to nginx!</title>
2 <title>Welcome to nginx!</title>
3 <title>Welcome to nginx!</title>
4 <title>Welcome to nginx!</title>
5 <title>Welcome to nginx!</title>
```

- `kill 1` 로 PID 1번을 kill 명령으로 종료한 후 자동으로 다시 복구되는지 확인

```
root@nginx-pod:/# kill 1
root@nginx-pod:/# command terminated with exit code 137
```

```
62 <title>Welcome to nginx!</title>
>
63 <title>Welcome to nginx!</title>
>
64
65
66
67
68 <title>Welcome to nginx!</title>
```

- `[root@m-k8s ~]# kubectl exec -it nginx-pod -- /bin/bash` : nginx-pod 접속
- `root@nginx-pod:/# ls -l /run/nginx.pid` :nginx.pid가 생성된 시간으로 새로 생성된 프로세스인지 확인
- `exit` : 다시 m-k8s의 배시 셸로 이동

```
[root@m-k8s ~]# kubectl exec -it nginx-pod -- /bin/bash
root@nginx-pod:/# ls -l /run/nginx.pid
-rw-r--r--. 1 root root 2 Jul 17 07:22 /run/nginx.pid
```



#### Tip ☆ kubectl exec에서 '--'의 의미

'--'는 exec에 대한 인자 값을 나누고 싶을 때 사용합니다. 이해하기 쉽게 예제를 봅시다. nginx-pod에서 /run의 내용을 보고 싶다면 파일 이름 뒤에 ls /run을 입력하면 됩니다. '--'를 사용할 때는 보이지 않던 DEPRECATED 메시지가 함께 표시됩니다.

```
[root@m-k8s ~]# kubectl exec -it nginx-pod ls /run
```

```
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
```

이번에는 /run의 권한을 보고 싶다면 어떻게 할까요? -l(long listing format) 옵션을 붙여 확인하면 될 것 같습니다. 하지만 실행하면 다음과 같이 에러가 발생합니다. 이는 -l을 exec의 옵션으로 인식하기 때문에 그렇습니다.

```
[root@m-k8s ~]# kubectl exec -it nginx-pod ls -l /run
```

```
Error: unknown shorthand flag: 'l' in -l
See 'kubectl exec --help' for usage.
```

이런 경우에 명령어를 구분해야 하는데, 이때 '--'를 사용합니다.

```
[root@m-k8s ~]# kubectl exec -it nginx-pod -- ls -l /run
```

이처럼 필요하지 않아도 exec를 사용할 때 명시적으로 '--'를 사용하면 에러를 줄일 수 있습니다. 그래서 '--'을 사용하지 않고 바로 명령을 실행하면 DEPRECATED를 표시해 향후 버전에서는 사용하지 못하게 하면서 명령어에 일관성을 주려는 것입니다.

## ▼ ❤ 3.2.7 파드의 동작 보증 기능

- 파드 자체에 문제가 발생하면 파드를 자동 복구해서 파드가 항상 동작하도록 보장하는 기능

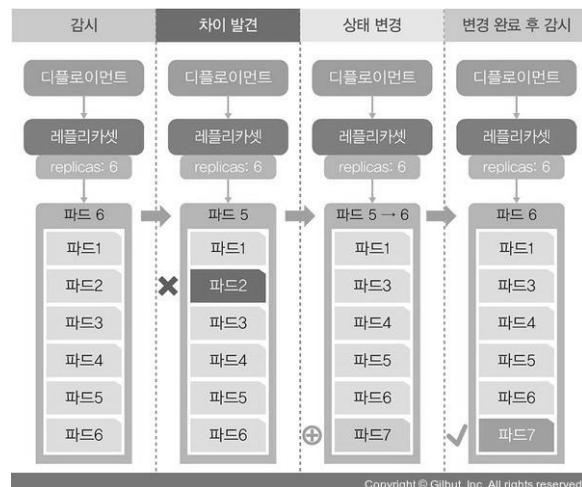
- `kubectl get pods` 로 파드 확인 후 `kubectl delete pods nginx-pod` 로 삭제
- `kubectl delete pods echo-hname-7894b67f-d87qc` 로 삭제
- `nginx-pod`는 삭제한 후 목록에 보이지 않는데 `echo-hname`은 삭제 후에도 6개의 파드가 살아있습니다.

```
[root@em-k8s ~]# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
echo-hname-7894b67f-d87qc   1/1    Running   0          51m
echo-hname-7894b67f-g8wbb   1/1    Running   0          47m
echo-hname-7894b67f-jfv4    1/1    Running   0          47m
echo-hname-7894b67f-jv6j    1/1    Running   0          47m
echo-hname-7894b67f-kcdrw   1/1    Running   0          51m
echo-hname-7894b67f-tx128   1/1    Running   0          51m
nginx-pod                         1/1    Running   2          20h
[root@em-k8s ~]# kubectl delete pods nginx-pod
pod "nginx-pod" deleted
[root@em-k8s ~]# kubectl delete pods echo-hname-7894b67f-d87qc
pod "echo-hname-7894b67f-d87qc" deleted
[root@em-k8s ~]# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
echo-hname-7894b67f-g8wbb   1/1    Running   0          49m
echo-hname-7894b67f-qgfs6   1/1    Running   0          6s
echo-hname-7894b67f-jfv4    1/1    Running   0          49m
echo-hname-7894b67f-jv6j    1/1    Running   0          49m
echo-hname-7894b67f-kcdrw   1/1    Running   0          53m
echo-hname-7894b67f-tx128   1/1    Running   0          53m
```

- nginx-pod는 **디플로이먼트에 속한 파드가 아니며 어떤 컨트롤러도 이 파드를 관리하지 않습니다.** 따라서 nginx-pod는 바로 삭제되고 다시 생성되지도 않습니다.



- echo-hname은 **디플로이먼트에 속한 파드입니다.** 그리고 앞에서 echo-hname에 속한 파드를 replicas에서 6개로 선언했습니다. replicas는 파드를 선언한 수대로 유지하도록 파드의 수를 항상 확인하고 부족하면 새로운 파드를 만들어냅니다. 따라서 임의로 파드를 삭제하면 replicas가 삭제된 파드를 확인하고 파드의 총 개수를 6개로 맞추기 위해서 새로운 파드 1개를 생성합니다.



- 이와 같이 **디플로이먼트로 생성하는 것이 파드의 동작을 보장하기 위한 조건입니다.**

- 파드의 동작 보증 기능을 마무리하면서 이렇게 파드가 자동 복구가 되면 **디플로이먼트에 속한 파드는 어떻게 삭제할까요?** → `kubectl delete deployment echo-hname`

### ▼ ❤️ 3.2.8 노드 자원 보호하기

- 노드(node) : 쿠버네티스 스케줄러에서 파드를 할당받고 처리하는 역할
  - 몇 차례 문제가 생긴 노드에 파드를 할당하면 문제가 생길 가능성이 높으나 어쩔 수 없이 해당 노드를 사용해야 한다면 어떻게 할까요?
  - 이런 경우에는 영향도가 적은 파드를 할당해 일정 기간 사용하면서 모니터링해야 합니다. 즉, 노드에 문제가 생기더라도 파드의 문제를 최소화해야 합니다.
  - 하지만 쿠버네티스는 모든 노드에 균등하게 파드를 할당하려고 합니다. 그렇다면 **어떻게 문제가 생길 가능성이 있는 노드라는 것을 쿠버네티스에 알려줄까요?** 쿠버네티스에서는 이런 경우에 cordon 기능을 사용합니다.
- cordon 노드 관리
  - `kubectl apply -f ~/_Book_k8sInfra/ch3/3.2.8/echo-hname.yaml`
  - `kubectl scale deployment echo-hname --replicas=9`로 파드를 9개로 늘림
  - `kubectl get pods -o wide` 대신에 `kubectl get pods -o=custom-`  
`columns= NAME :.metadata.name, IP :.status.podIP, STATUS :.status.phase, NODE :.spec.nodeName` 사용
    - o는 output을 의미

- custom-columns는 사용자가 임의로 구성할 수 있는 열을 의미
- 명령에서 NAME, IP, STATUS, NODE는 열의 제목
- 콜론(:) 뒤에 내용 값인 .metadata.name, .status.podIP, .status.phase, .spec.nodeName을 넣고 콤마(,)로 구분
- yaml로 파드의 내용 확인하기 : `kubectl get pod echo-hname-5d754d565-69wgg -o yaml > pod.yaml`
- `kubectl scale deployment echo-hname --replicas=3` 로 파드를 3개로 줄이고 `kubectl get pods -o=custom-columns= NAME:.metadata.name, IP:.status.podIP, STATUS:.status.phase, NODE:.spec.nodeName`로 파드 줄어들은 것 확인

```
[root@m-k8s ~]# kubectl get pods \
> -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
NAME          IP           STATUS      NODE
echo-hname-7894b67f-4bx5x 172.16.221.136 Running   w1-k8s
echo-hname-7894b67f-4sgqh2 172.16.221.135 Running   w1-k8s
echo-hname-7894b67f-38vvs8 172.16.103.138 Running   w2-k8s
echo-hname-7894b67f-1n4bt  172.16.103.136 Running   w2-k8s
echo-hname-7894b67f-1xxnw  172.16.103.137 Running   w2-k8s
echo-hname-7894b67f-njz8m  172.16.132.8  Running   w3-k8s
echo-hname-7894b67f-rrlmf  172.16.132.7  Running   w3-k8s
echo-hname-7894b67f-s76s6  172.16.221.137 Running   w1-k8s
echo-hname-7894b67f-z8ksk  172.16.132.6  Running   w3-k8s
[root@m-k8s ~]# kubectl scale deployment echo-hname --replicas=3
deployment.apps/echo-hname scaled
[root@m-k8s ~]# kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
NAME          IP           STATUS      NODE
echo-hname-7894b67f-4sgqh2 172.16.221.135 Running   w1-k8s
echo-hname-7894b67f-1n4bt  172.16.103.136 Running   w2-k8s
echo-hname-7894b67f-z8ksk  172.16.132.6  Running   w3-k8s
```

- w3-k8s 노드에서 문제가 자주 발생해 현재 상태를 보존해야 한다고 가정. `kubectl cordon w3-k8s` 으로 cordon 명령을 실행
- `kubectl get nodes` 으로 w3-k8s가 더 이상 파드가 할당되지 않는 상태로 변경됐는지 확인
- 이처럼 cordon 명령을 실행하면 해당 노드에 파드가 할당되지 않게 스케줄되지 않는 상태(SchedulingDisabled)라는 표시됨

```
[root@m-k8s ~]# kubectl cordon w3-k8s
node/w3-k8s cordoned
[root@m-k8s ~]# kubectl get nodes
NAME        STATUS     ROLES      AGE    VERSION
m-k8s       Ready      master    23h    v1.18.4
w1-k8s      Ready      <none>    23h    v1.18.4
w2-k8s      Ready      <none>    23h    v1.18.4
w3-k8s      Ready,SchedulingDisabled <none>   22h    v1.18.4
```

- `kubectl scale deployment echo-hname --replicas=9` 으로 파드 수 다시 9개로 늘리고 `kubectl get pods -o=custom-columns= NAME:.metadata.name, IP:.status.podIP, STATUS:.status.phase, NODE:.spec.nodeName`으로 w3-k8s에 추가로 배포된 파드가 없음을 확인

```
[root@m-k8s ~]# kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
NAME          IP           STATUS      NODE
echo-hname-7894b67f-22swr 172.16.103.141 Running   w2-k8s
echo-hname-7894b67f-4sgqh2 172.16.221.135 Running   w1-k8s
echo-hname-7894b67f-7ghnfz 172.16.103.140 Running   w2-k8s
echo-hname-7894b67f-99wq2  172.16.221.138 Running   w1-k8s
echo-hname-7894b67f-hvn9c  172.16.103.139 Running   w2-k8s
echo-hname-7894b67f-kz4c7  172.16.221.140 Running   w1-k8s
echo-hname-7894b67f-1n4bt  172.16.103.136 Running   w2-k8s
echo-hname-7894b67f-rqjm4  172.16.221.139 Running   w1-k8s
echo-hname-7894b67f-z8ksk  172.16.132.6  Running   w3-k8s
```

- `kubectl scale deployment echo-hname --replicas=3` → `kubectl get pods -o=custom-columns= NAME:.metadata.name, IP:.status.podIP, STATUS:.status.phase, NODE:.spec.nodeName`로 각 노드에 할당된 파드 수가 공평하게 1개씩인지 확인
- `kubectl uncordon w3-k8s`으로 w3-k8s에 파드가 할당되지 않게 설정했던 것을 해제

- 노드의 커널을 업데이트하거나 노드의 메모리를 증설하는 등의 작업이 필요해서 노드를 꺼야 할 때는 어떻게 하면 좋을까요? → drain 가능

### ▼ ❤️ 3.2.9 노드 유지보수하기

쿠버네티스를 사용하다 보면 정기 또는 비정기적인 유지보수를 위해 노드를 꺼야 하는 상황이 발생합니다. 이런 경우를 대비해 쿠버네티스는 drain 기능을 제공합니다. drain은 지정된 노드의 파드를 전부 다른 곳으로 이동시켜 해당 노드를 유지보수할 수 있게 합니다.

- drain

- `kubectl drain w3-k8s`

```
[root@m-k8s ~]# kubectl drain w3-k8s
node/w3-k8s cordoned
error: unable to drain node "w3-k8s", aborting command...

There are pending nodes to be drained:
w3-k8s
error: cannot delete DaemonSet-managed Pods (use --ignore-daemonsets to ignore): kube-system/calico-node-vc2lq, kube-system/kube-proxy-5q74b
```

- drain은 실제 파드를 옮기는 것이 아니라 노드에서 파드를 삭제하고 다른 곳에서 다시 생성합니다. 파드는 언제라도 삭제할 수 있기 때문에 쿠버네티스에서 대부분 이동은 파드를 지우고 다시 만드는 과정을 의미합니다.
  - 그런데 DaemonSet은 각 노드에 1개만 존재하는 파드라서 drain으로는 삭제할 수 없습니다.
- `kubectl drain w3-k8s --ignore-daemonsets` 으로 DaemonSet을 무시하고 진행합니다. 경고가 발생하지만 모든 파드가 이동됩니다.
  - `kubectl get pods -o=custom-columns= NAME :.metadata.name, IP :.status.podIP, STATUS :.status.phase, NODE :.spec.nodeName` 로 노드 w3-k8s에 파드가 없는지 확인합니다. 그리고 옮긴 노드에 파드가 새로 생성돼 파드 이름과 IP가 부여된 것도 확인합니다.

```
[root@m-k8s ~]# kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
NAME           IP          STATUS    NODE
echo-hname-7894b67f-2t5t2  172.16.103.142  Running   w2-k8s
echo-hname-7894b67f-4sgh2  172.16.221.135  Running   w1-k8s
echo-hname-7894b67f-ln4bt  172.16.103.136  Running   w2-k8s
```

- `kubectl get nodes` 로 w3-k8s의 상태를 확인 (SchedulingDisabled 상태)

```
[root@m-k8s ~]# kubectl get nodes
NAME     STATUS            ROLES      AGE   VERSION
m-k8s   Ready             master    24h   v1.18.4
w1-k8s  Ready             <none>    24h   v1.18.4
w2-k8s  Ready             <none>    23h   v1.18.4
w3-k8s  Ready [SchedulingDisabled]  <none>    23h   v1.18.4
```

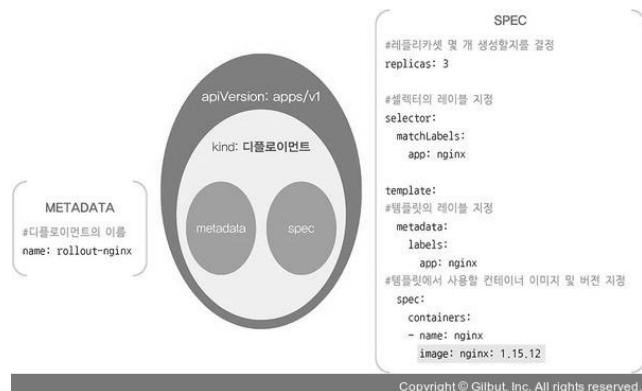
- 유지보수가 끝났다고 가정하고 `kubectl uncordon w3-k8s` 으로 스케줄 받을 수 있는 상태로 복귀
- 다음 진행을 위해 `kubectl delete -f ~/Book_k8sInfra/ch3/3.2.8/echo-hname.yaml` 로 echo-hname 삭제

### ▼ 3.2.10 파드 업데이트하고 복구하기

파드를 운영하다 보면 컨테이너에 새로운 기능을 추가하거나 치명적인 버그가 발생해 버전을 업데이트해야 할 때가 있습니다. 또는 업데이트하는 도중 문제가 발생해 다시 기존 버전으로 복구해야 하는 일도 발생합니다.

- rollout-nginx.yaml의 구조

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rollout-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.15.12
```



- 파드 업데이트하기

- `kubectl apply -f ~/Book_k8sInfra/ch3/3.2.10/rollout-nginx.yaml --record` 으로 컨테이너 버전 업데이트를 테스트하기 위한 파드를 배포합니다.

- --record : 배포한 정보의 히스토리를 기록합니다.

- `kubectl rollout history deployment rollout-nginx` : record 옵션으로 기록된 히스토리는 rollout history 명령을 실행해 확인

```
[root@m-k8s ~]# kubectl rollout history deployment rollout-nginx
deployment.apps/rollout-nginx
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=/root/_Book_k8sInfra/ch3/3.2.10/rollout-nginx.yaml --record=true
```

- `kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName` 으로 파드 정보 확인
- `curl -I --silent 172.16.103.143 | grep Server` 으로 nginx 컨테이너 버전 확인

```
[root@m-k8s ~]# curl -I --silent 172.16.103.143 | grep Server
Server: nginx/1.15.12
```

- `kubectl set image deployment rollout-nginx nginx=nginx:1.16.0 --record` 으로 컨테이너 버전을 1.16.0으로 업데이트
- `kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName` 으로 파드 정보 확인 해보니 파드들의 이름과 IP 가 변경된 것을 확인

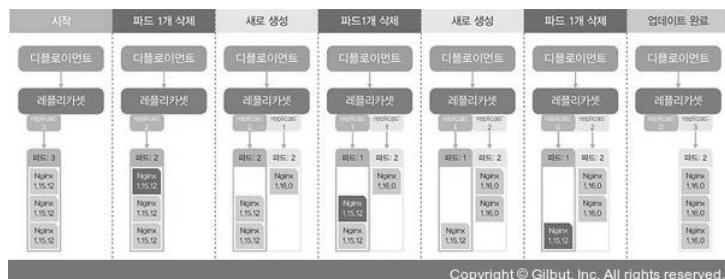
NAME	IP	STATUS	NODE
rollout-nginx-64dd56c7b5-42zjh	172.16.132.9	Running	w3-k8s
rollout-nginx-64dd56c7b5-prtj6	172.16.221.141	Running	w1-k8s
rollout-nginx-64dd56c7b5-vgbcq	172.16.103.143	Running	w2-k8s

이전

NAME	IP	STATUS	NODE
rollout-nginx-8566d57f75-5klw7	172.16.103.144	Running	w2-k8s
rollout-nginx-8566d57f75-m75bx	172.16.132.10	Running	w3-k8s
rollout-nginx-8566d57f75-wfns2	172.16.221.142	Running	w1-k8s

이후

- 파드는 언제라도 지우고 다시 만들 수 있습니다. nginx 컨테이너를 업데이트하는 가장 쉬운 방법은 파드를 관리하는 replicas의 수를 줄이고 늘려 파드를 새로 생성하는 것입니다. 이때 시스템의 영향을 최소화하기 위해 replicas에 속한 파드를 모두 한 번에 지우는 것 이 아니라 **파드를 하나씩 순차적으로 지우고 생성합니다**. 이때 파드 수가 많으면 하나씩이 아니라 다수의 파드가 업데이트됩니다.



- `kubectl rollout status deployment rollout-nginx` 으로 Deployment의 상태 확인
- `kubectl rollout history deployment rollout-nginx` 실행해 rollout-nginx에 적용된 명령들을 확인하고  
`curl -I --silent 172.16.132.10 | grep Server` 으로 업데이트 제대로 이루어졌는지 확인

#### • 업데이트 실패 시 파드 복구

- 의도와 다르게 `kubectl set image deployment rollout-nginx nginx=nginx:1.17.23 --record` 으로 버전을 다르게 입력한 후 `kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName` 를 입력했더니 pending에서 넘어가지 않습니다.

NAME	IP	STATUS	NODE
rollout-nginx-8566d57f75-5klw7	172.16.103.144	Running	w2-k8s
rollout-nginx-8566d57f75-m75bx	172.16.132.10	Running	w3-k8s
rollout-nginx-8566d57f75-wfns2	172.16.221.142	Running	w1-k8s
rollout-nginx-856f4c79c9-cx8jk	172.16.132.12	Pending	w3-k8s

- `kubectl rollout status deployment rollout-nginx` 실행해보니 새로운 replicas는 생성했으나(new replicas have been updated) 디플로이먼트를 배포하는 단계에서 대기 중(Waiting)으로 더 이상 진행되지 않은 것을 확인할 수 있습니다. Deployment를 생성하려고 여러

번 시도했지만, 끝내 생성되지 않았다는 메시지가 출력됩니다.

```
[root@m-k8s ~]# kubectl rollout status deployment rollout-nginx
Waiting for deployment "rollout-nginx" rollout to finish: 1 out of 3 new replicas have been updated...
error: deployment "rollout-nginx" exceeded its progress deadline
```

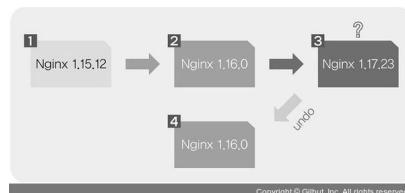
- `kubectl describe deployment rollout-nginx` 로 문제점을 좀 더 자세히 살펴봅니다.

```
Conditions:
  Type        Status  Reason
  ----        ----   -----
  Available   True    MinimumReplicasAvailable
  Progressing False   ProgressDeadlineExceeded
OldReplicaSets: rollout-nginx-8566d57f75 (3/3 replicas created)
NewReplicaSet:  rollout-nginx-856f4c79c9 (1/1 replicas created)
```

- replicas가 새로 생성되는 과정에서 멈춰 있습니다. 그 이유는 1.17.23 버전의 nginx 컨테이너가 없기 때문입니다. 따라서 replicas가 생성을 시도했으나 컨테이너 이미지를 찾을 수 없어서 디플로이먼트가 배포되지 않았습니다.
- 실제로 배포할 때 이런 실수를 할 가능성이 충분히 있습니다. 이를 방지하고자 업데이트할 때 rollout을 사용하고 --record로 기록하는 것입니다.
- 정상적인 상태로 복구하기 위해 `kubectl rollout history deployment rollout-nginx` 로 업데이트 할 때 사용했던 명령들을 확인하고 `kubectl rollout undo deployment rollout-nginx` 로 전 단계로 상태를 되돌립니다.
- `kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName` 로 파드 상태를 다시 확인합니다.

```
[root@m-k8s ~]# kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
NAME                  IP           STATUS      NODE
rollout-nginx-8566d57f75-5klw7  172.16.103.144  Running    w2-k8s
rollout-nginx-8566d57f75-m75bx  172.16.132.10   Running    w3-k8s
rollout-nginx-8566d57f75-wfns2  172.16.221.142  Running    w1-k8s
```

- `kubectl rollout history deployment rollout-nginx` 를 확인해보니 revision 4가 추가되고 revision 2가 삭제됐습니다. 현재 상태를 revision 2로 되돌렸기 때문에 revision 2는 삭제되고 가장 최근 상태는 revision 4가 됩니다.



- `curl -I --silent 172.16.132.10 | grep Server`, `kubectl rollout status deployment rollout-nginx` 으로 상태가 되돌려졌음을 알 수 있습니다.

```
[root@m-k8s ~]# curl -I --silent 172.16.132.10 | grep Server
Server: nginx/1.16.0
[root@m-k8s ~]# kubectl rollout status deployment rollout-nginx
deployment "rollout-nginx" successfully rolled out
```

- 특정 시점으로 파드 복구하기

- `kubectl rollout undo deployment rollout-nginx --to-revision=1` 로 처음 상태로 복구합니다.
- `kubectl get pods -o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName` 로 IP를 확인하고 `curl -I --silent 172.16.103.150 | grep Server` 로 컨테이너 버전을 확인합니다. 1.15.12로 처음 상태로 복구됐습니다.
- 다음 단계 진행을 위해 `kubectl delete -f ~/Book_k8sInfra/ch3/3.2.10/rollout-nginx.yaml` 로 디플로이먼트를 삭제합니다.