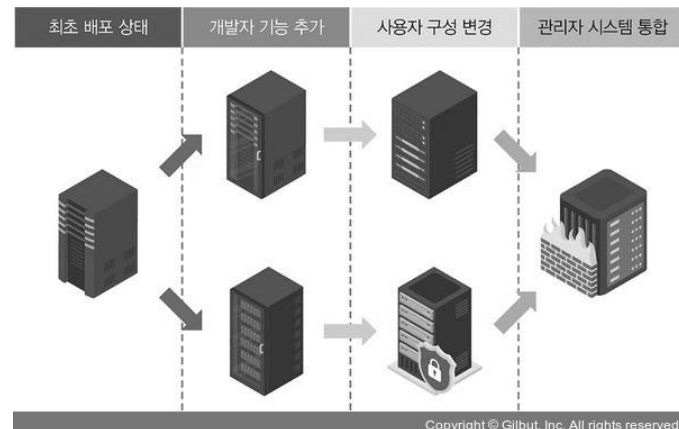


3장 컨테이너를 다루는 표준 아키텍처, 쿠버네티스

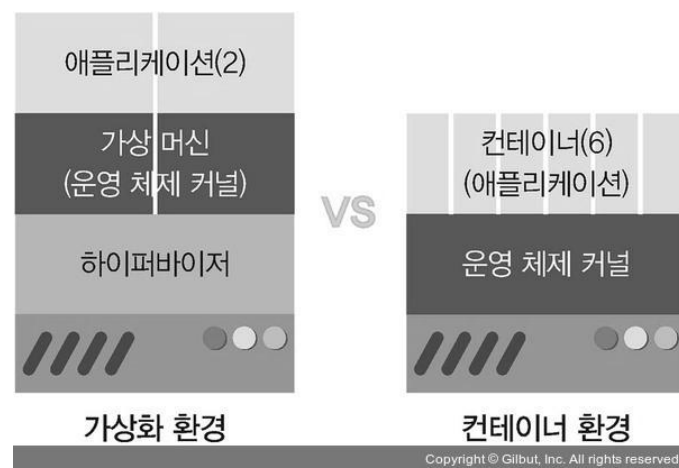
컨테이너 인프라 환경

- 리눅스 OS의 커널 하나에서 여러 개의 컨테이너가 격리된 상태로 실행되는 인프라 환경
- 컨테이너 : 하나 이상의 목적을 위해 독립적으로 작동하는 프로세스
- 기업 환경에서 일관성을 유지하는 것이 중요한데, 컨테이너 인프라 환경을 구성하면 눈송이 서버를 방지하는데 효과적임



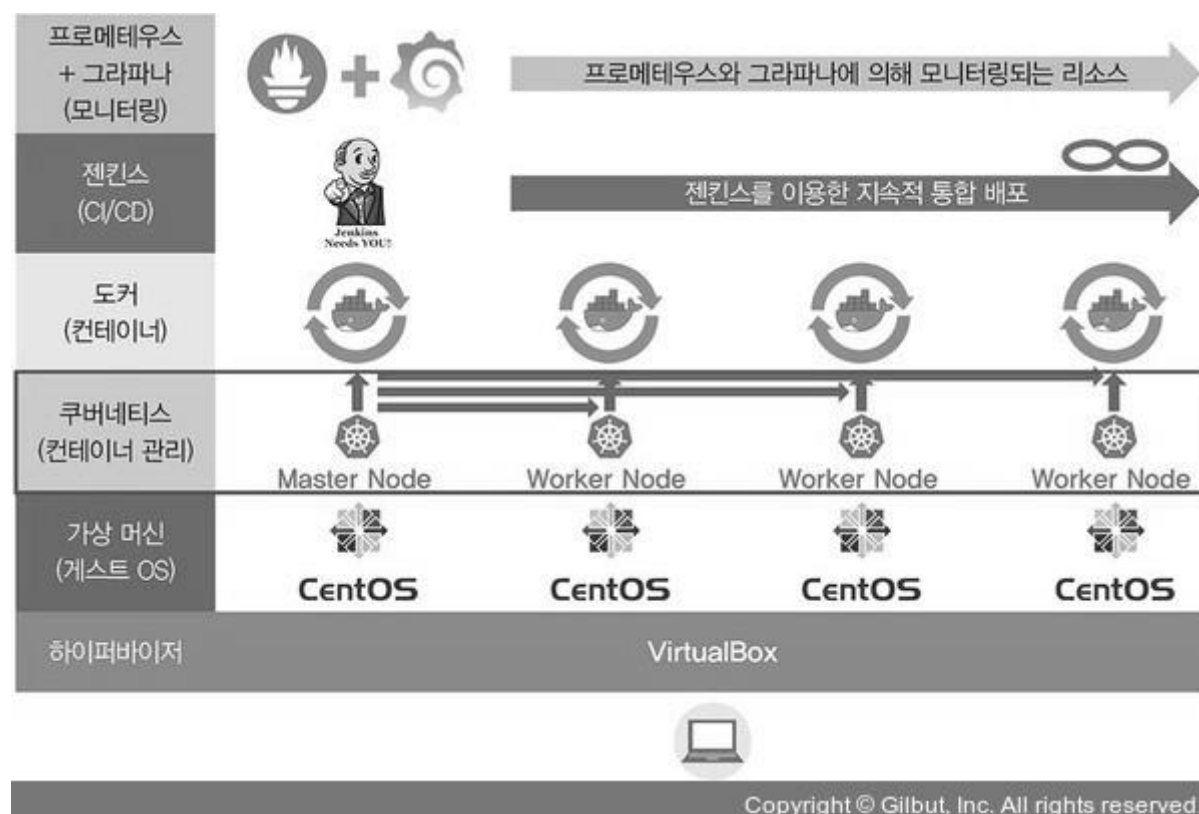
설정의 일관성이 떨어진 눈송이 서버

- 가상화 환경 vs 컨테이너 인프라 환경



- 가상화 환경 - 각각의 가상 머신이 모두 독립적인 os 커널을 가져야하므로 자원 소모 심하고 성능 떨어짐
- 컨테이너 인프라 환경 - os 커널 하나에 컨테이너 여러 개가 격리된 형태로 실행되므로 자원을 효율적으로 사용하고 속도도 빠름

3장에서 다룰 쿠버네티스 부분



3.1 쿠버네티스 이해하기

쿠버네티스 - 컨테이너 오케스트레이션을 위한 솔루션

오케스트레이션 - 복잡한 단계를 관리하고 요소들의 유기적인 관계를 미리 정의해 손쉽게 사용하도록 서비스를 제공하는 것을 의미

컨테이너 오케스트레이션 - 다수의 컨테이너를 유기적으로 연결, 실행, 종료할 뿐만 아니라 상태를 추적하고 보존하는 등 컨테이너를 안정적으로 사용할 수 있게 만들어주는 것이 컨테이너 오케스트레이션

3.1.1 왜 쿠버네티스일까

컨테이너 오케스트레이션을 제공하는 대표적인 솔루션은 다음과 같다.



▼ 표 3-1 컨테이너 오케스트레이션 솔루션 비교

구분	도커 스웜	메소스	노매드	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

3.1.2 쿠버네티스 구성방법

쿠버네티스를 구성하는 방법은 크게 3가지이다.

- 1. **관리형 쿠버네티스 사용**
 - 퍼블릭 클라우드 업체에서 제공하는 관리형 쿠버네티스 EKS, AKS, GKE
 - 구성이 이미 다 갖춰져 있고 마스터 노드를 클라우드 업체에서 관리하기 때문에 학습용으로는 부적합하다.
- 2. **설치형 쿠버네티스 사용**
 - Rancher, OpenShift - 유료라 쉽게 접근 어려움
- 3. **구성형 쿠버네티스 사용**
 - 사용하는 시스템에 쿠버네티스 클러스터를 자동으로 구성해주는 솔루션을 사용
 - 주요 솔루션 - kubeadm, kops, KRIB, kubespray

- kubeadm이 가장 널리 알려져있으며 배우기도 쉬움

구분	KRIB	kops	kubespray	kubeadm
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 쿠버네티스 구성하기






1. 소스 코드 다운로드

다운로드 링크 > https://github.com/sysnet4admin/_Book_k8sInfra

2. 압축을 풀고 폴더 이동

C:\HashiCorp_Book_k8sInfra-main

3. ch3/3.1.3 폴더에서 파일 확인

:크 (C:) > HashiCorp > _Book_k8sInfra-main > ch3 > 3.1.3				
이름	수정한 날짜	유형	크기	
 config.sh	2023-06-10 오후 1:31	Shell Script	2KB	
 install_pkg.sh	2023-06-10 오후 1:31	Shell Script	1KB	
 master_node.sh	2023-06-10 오후 1:31	Shell Script	1KB	
 Vagrantfile	2023-06-10 오후 1:31	파일	2KB	
 work_nodes.sh	2023-06-10 오후 1:31	Shell Script	1KB	

Vagrantfile

- 베이그런트 프로비저닝을 위한 정보를 담고 있는 메인 파일
- cmd를 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 구성한다.

▼ 코드

```
# -*- 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"
  end
end
```

```

        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
        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
end
end

```

config.sh

- kubeadm으로 쿠버네티스를 설치하기 위한 사전 조건을 설정하는 스크립트 파일

▼ 코드

```

#!/usr/bin/env bash

# vi를 호출하면 vim을 호출하도록 프로파일에 입력합니다. 이렇게 하면 코드에 하이라이트를 넣어 코드를 쉽게 구분할 수 있습니다.
echo 'alias vi=vim' >> /etc/profile

# 쿠버네티스의 설치 요구 조건을 맞추기 위해 스왑되지 않도록 설정합니다.
swapoff -a
# 시스템이 다시 시작되더라도 스왑되지 않도록 설정합니다.
sed -i.bak -r 's/(.+ swap .+)/#\1/' /etc/fstab

# 쿠버네티스의 리포지토리를 설정하기 위한 경로가 너무 길어지지 않게 경로를 변수로 처리합니다.
gg_pkg="packages.cloud.google.com/yum/doc" # Due to shorten addr for key

# 쿠버네티스를 내려받을 리포지토리를 설정하는 구문입니다.
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

# selinux가 제한적으로 사용되지 않도록 permissive 모드로 변경합니다.
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

# 브리지 네트워크를 통과하는 IPv4와 IPv6의 패킷을 iptables가 관리하게 설정합니다.
# 파드(Pod, 쿠버네티스에서 실행되는 객체의 최소 단위로, 뒤에서 자세히 설명할 예정)의 통신을 iptables로 제어합니다.
# 필요에 따라 IPVS(IP Virtual Server) 같은 방식으로 구성할 수도 있습니다.
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
# br_netfilter 커널 모듈을 사용해 브리지로 네트워크를 구성합니다.
# 이때 IP 마스커레이드(Masquerade)를 사용해 내부 네트워크와 외부 네트워크를 분리합니다.
# IP 마스커레이드는 쉽게 설명하면 커널에서 제공하는 NAT(Network Address Translation) 기능으로 이해하면 됩니다.
# 실제로는 br_netfilter를 적용함으로써 28~31번째 줄에서 적용한 iptables가 활성화됩니다.
modprobe br_netfilter

# 쿠버네티스 안에서 노드 간 통신을 이름으로 할 수 있도록

```

```
# 각 노드의 호스트 이름과 IP를 /etc/hosts에 설정합니다.
# 이때 워커 노드는 Vagrantfile에서 넘겨받은 N 변수로 전달된 노드 수에 맞게 동적으로 생성합니다.
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

# 외부와 통신할 수 있게 DNS 서버를 지정합니다.
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 위해 가상 머신에 설치돼야 하는 의존성 패키지를 명시
- 실습에 필요한 소스코드를 특정 가상 머신 (m-k8s) 내부에 내려도록 설정되어 있음

▼ 코드

```
#!/usr/bin/env bash

# install packages
yum install epel-release -y
yum install vim-enhanced -y
yum install git -y # 깃허브에서 코드를 내려받을 수 있게 깃(git)을 설치합니다.

# 쿠버네티스를 관리하는 컨테이너를 설치하기 위해 도커를 설치하고 구동합니다.
yum install docker -y && systemctl enable --now docker

# 쿠버네티스를 구성하기 위해 첫 번째 변수($1=Ver='1.18.4')로 넘겨받은
# 1.18.4 버전의 kubectl, kubelet, kubeadm을 설치하고 kubelet을 시작합니다.
yum install kubectl-$1 kubelet-$1 kubeadm-$1 -y
systemctl enable --now kubelet

# 이 책의 전체 실행 코드를 마스터 노드에만 내려받도록 Vagrantfile에서
# 두 번째 변수($2 = 'Main')를 넘겨받습니다.
# 그리고 깃에서 코드를 내려받아 실습을 진행할 루트 홈디렉터리(/root)로 옮깁니다.
# 배시 스크립트(.sh)를 find로 찾아서 바로 실행 가능한 상태가 되도록
# chmod 700으로 설정합니다.
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
- 쿠버네티스 클러스터를 구성할 때 꼭 선택해야 하는 컨테이너 네트워크 인터페이스(CNI)도 함께 구성

▼ 코드

```
#!/usr/bin/env bash

# kubeadm을 통해 쿠버네티스의 워커 노드를 받아들이기 준비를 합니다.
# 먼저 토큰을 123456.1234567890123456으로 지정하고
# ttl(time to live, 유지되는 시간)을 0으로 설정해서 기본값인 24시간 후에 토큰이
# 계속 유지되게 합니다. 그리고 워커 노드가 정해진 토큰으로 들어오게 합니다.
# 쿠버네티스가 자동으로 컨테이너에 부여하는 네트워크를
# 172.16.0.0/16(172.16.0.1~172.16.255.254)으로 제공하고,
# 워커 노드가 접속하는 API 서버의 IP를 192.168.1.10으로 지정해
# 워커 노드들이 자동으로 API 서버에 연결되게 합니다.
kubeadm init --token 123456.1234567890123456 --token-ttl 0 \
--pod-network-cidr=172.16.0.0/16 --apiserver-advertise-address=192.168.1.10

# 마스터 노드에서 현재 사용자가 쿠버네티스를 정상적으로 구동할 수 있게
# 설정 파일을 루트의 홈디렉터리(/root)에 복사하고
# 쿠버네티스를 이용할 사용자에게 권한을 줍니다.
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

```
# 컨테이너 네트워크 인터페이스(CNI)인 캘리코(Calico)의 설정을 적용해
# 쿠버네티스의 네트워크를 구성합니다.
kubectl apply -f \
https://raw.githubusercontent.com/sysnet4admin/IaC/master/manifests/172.16_net_calico.yaml
```

work_nodes.sh

- 3대의 가상머신 (w1-k8s, w2-k8s, w3-k8s)에 쿠버네티스 워커 노드를 구성하는 스크립트
- 마스터 노드에 구성된 클러스터에 조인이 필요한 정보가 모두 코드화되어 있어 스크립트를 실행하기만 하면 편하게 워커노드로서 쿠버네티스 클러스터에 조인된다.

▼ 코드

```
#!/usr/bin/env bash

# kubeadm을 이용해 쿠버네티스 마스터 노드에 접속합니다.
# 이때 연결에 필요한 토큰은 기존에 마스터 노드에서 생성한
# 123456.1234567890123456을 사용합니다.
# 간단하게 구성하기 위해 --discovery-token-unsafe-skip-ca-verification으로
# 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
```

4. vagrant up 명령어

- Vagrantfile에 정의된 가상 머신들을 생성하고 생성한 가상 머신에 쿠버네티스 클러스터를 구성하기 위한 파일들을 호출해 쿠버네티스 클러스터를 자동으로 구성

```
C:\Users\leeoo>cd C:\HashiCorp\_Book_k8sInfra-main\ch3\3.1.3
```

```
C:\HashiCorp\_Book_k8sInfra-main\ch3\3.1.3>vagrant up
```

3.1.4 파드 배포 중심으로 쿠버네티스 구성 요소 살펴보기

m-k8s w1-k8s w2-k8s w3-k8s

🖱 Using username "root".

```
[root@m-k8s ~]# kubectl get nodes
NAME      STATUS    ROLES    AGE   VERSION
m-k8s     Ready     master   119m  v1.18.4
w1-k8s     Ready     <none>   112m  v1.18.4
w2-k8s     NotReady  <none>   105m  v1.18.4
w3-k8s     Ready     <none>   35m   v1.18.4

[root@m-k8s ~]# kubectl get pods --all-namespaces
NAMESPACE   NAME                                     READY   STATUS                    RESTARTS   AGE
kube-system  calico-kube-controllers-99c9b6f64-79fmm  1/1     Running                   1           120m
kube-system  calico-node-65mnr                        1/1     Running                   1           113m
kube-system  calico-node-6k9qp                        0/1     Init:ImageInspectError    0           106m
kube-system  calico-node-fn4zh                        1/1     Running                   0           35m
kube-system  calico-node-g9nnr                        1/1     Running                   1           120m
kube-system  coredns-66bff467f8-dvfjm                 1/1     Running                   1           120m
kube-system  coredns-66bff467f8-rbgzz                 1/1     Running                   1           120m
kube-system  etcd-m-k8s                               1/1     Running                   2           120m
kube-system  kube-apiserver-m-k8s                     1/1     Running                   2           120m
kube-system  kube-controller-manager-m-k8s            1/1     Running                   3           120m
kube-system  kube-proxy-6lwnj                         1/1     Running                   1           120m
kube-system  kube-proxy-9zhgx                         1/1     Running                   0           35m
kube-system  kube-proxy-k8mjd                         1/1     Running                   1           113m
kube-system  kube-proxy-nmttj                         1/1     Running                   1           106m
kube-system  kube-scheduler-m-k8s                    1/1     Running                   3           120m

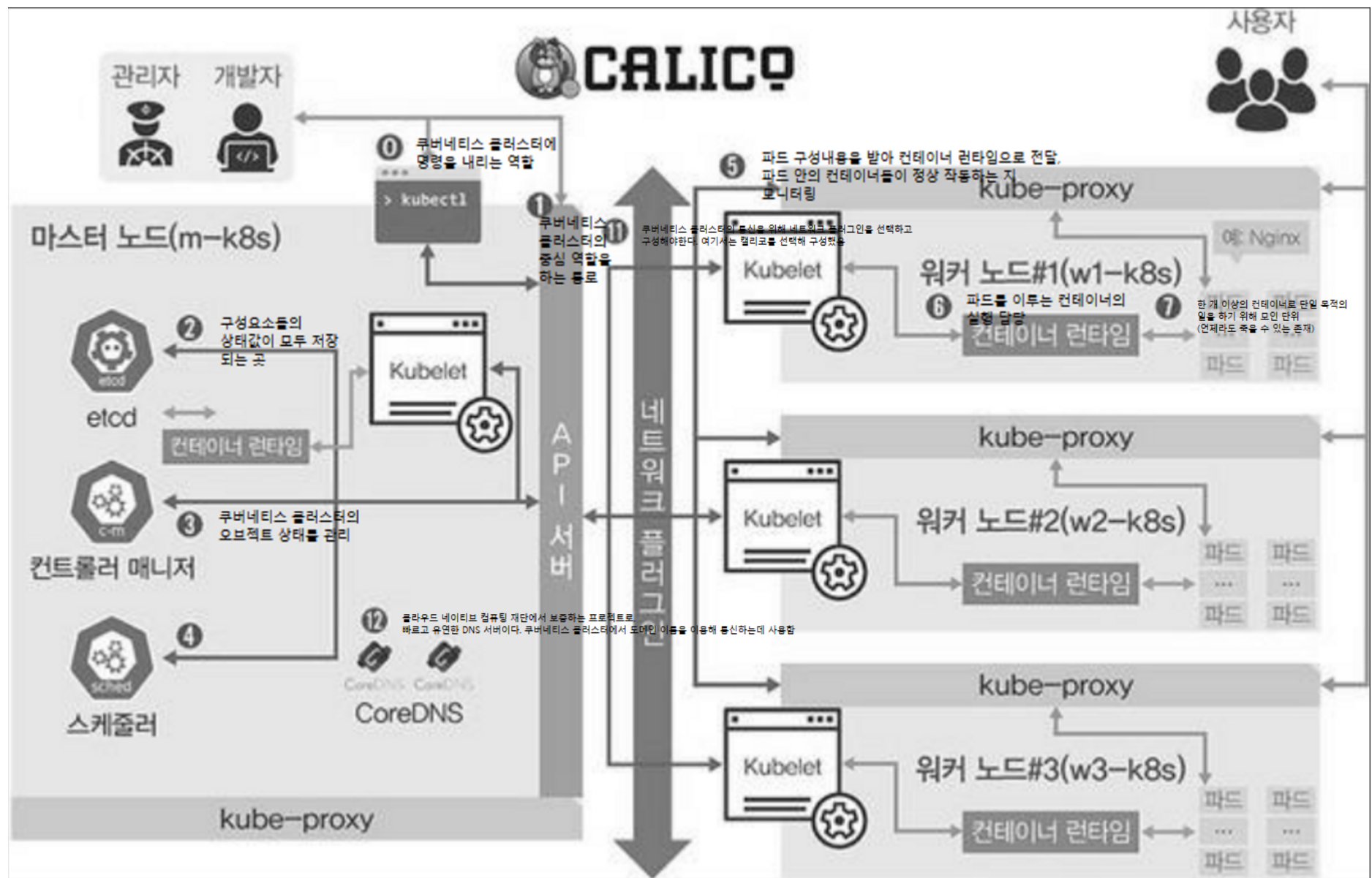
[root@m-k8s ~]# █
```

3장 컨테이너를 다루는 표준 아키텍처, 쿠버네티스

6

관리자나 개발자가 파드를 배포할 때

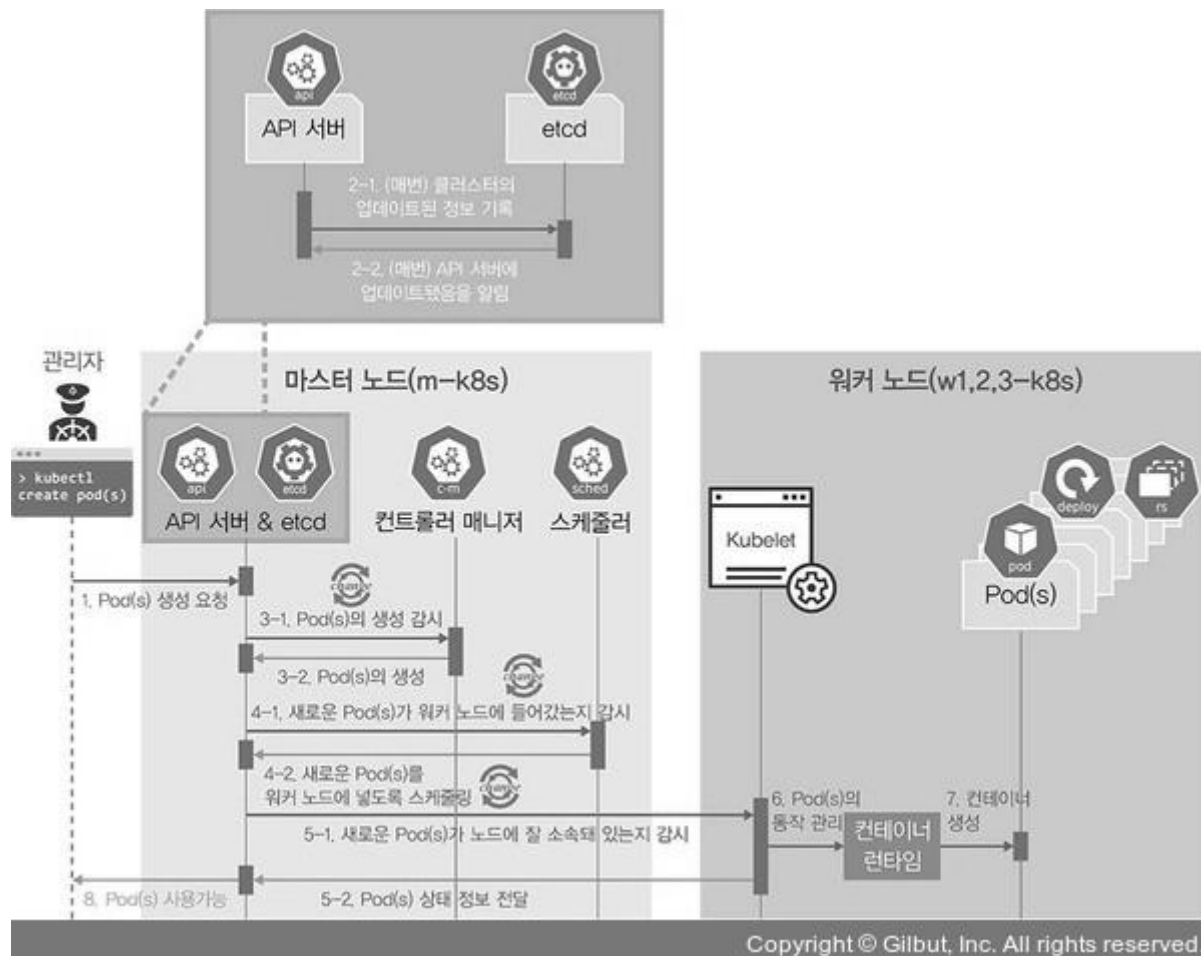
다음은 실제로 관리자나 개발자가 파드 배포 명령을 수행했을 때 실행되는 순서이다.



사용자가 배포된 파드에 접속할 때

- 1. kube-proxy:** 쿠버네티스 클러스터는 파드가 위치한 노드에 kube-proxy를 통해 파드가 통신할 수 있는 네트워크를 설정합니다. 이때 실제 통신은 br_netfilter와 iptables로 관리합니다. 두 기능은 Vagrantfile에서 호출하는 config.sh 코드를 설명할 때 다뤘습니다.
- 2. 파드:** 이미 배포된 파드에 접속하고 필요한 내용을 전달받습니다. 이때 대부분 사용자는 파드가 어느 워커 노드에 위치하는지 신경 쓰지 않아도 됩니다.

3.1.5 파드의 생명주기로 쿠버네티스 구성요소 살펴보기



쿠버네티스는 작업을 순서대로 진행하는 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.

3.1.6 쿠버네티스 구성 요소의 기능 검증하기

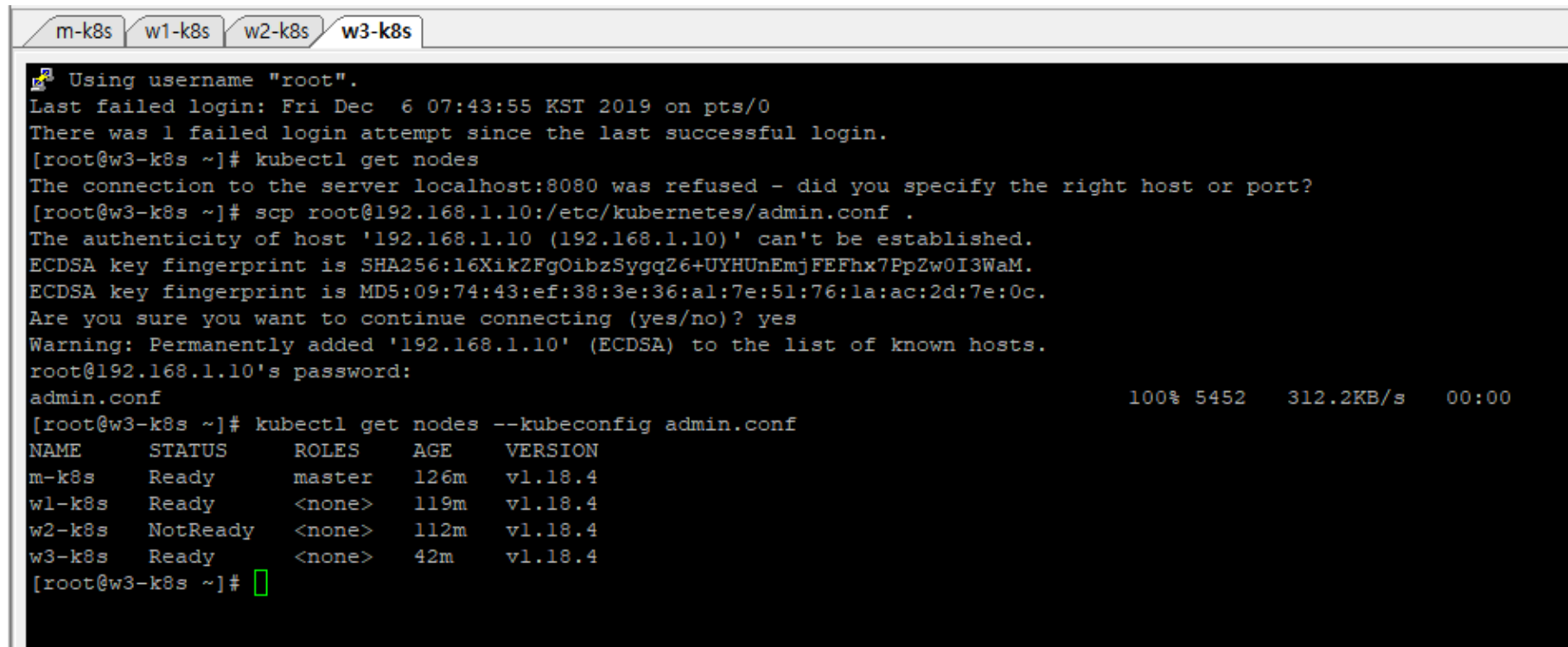
kubectl

- kubectl은 꼭 마스터 노드에 위치할 필요 없다.
- 실제로 쿠버네티스 클러스터의 외부에서 쿠버네티스 클러스터에 명령을 내릴 수도 있다.
- 어느 곳에서든지 kubectl을 실행하기 위한 설정을 알아보자.

▼ 설정 방법 및 결과

1. w3-k8s 접속
2. `kubectl get nodes` - 정보가 표시되지 않음
3. `scp root@192.168.1.10:/etc/kubernetes/admin.conf . yes vagrant`

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



```
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@w3-k8s ~]# kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[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 SHA256:16Xik2FgOibzSyygq26+UYHUnEmjFEFhx7Pp2w0I3WaM.
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 312.2KB/s 00:00
[root@w3-k8s ~]# kubectl get nodes --kubeconfig admin.conf
NAME        STATUS    ROLES    AGE   VERSION
m-k8s       Ready     master   126m  v1.18.4
w1-k8s       Ready     <none>   119m  v1.18.4
w2-k8s       NotReady  <none>   112m  v1.18.4
w3-k8s       Ready     <none>   42m   v1.18.4
[root@w3-k8s ~]#
```

노드 정보가 정상적으로 표시된다.

kubelet

- kubelet은 쿠버네티스에서 파드의 생성과 상태 관리 및 복구 등을 담당하는 매우 중요한 구성 요소이다.
- kubelet에 문제가 생기면 파드가 정상적으로 관리되지 않는다. ⇒ 실습을 통해 확인

▼ 과정 및 결과

1. `kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml`
- 기능을 검증하려면 실제 파드를 배포해야한다.
2. `kubectl get pod` - 배포된 파드가 정상적으로 배포된 상태인지 확인한다.
3. `kubectl get pods -o wide` - 파드가 배포된 워커 노드 확인
4. `systemctl stop kubelet` - kubelet 서비스 멈추기
5. `kubectl get pod`
`kubectl delete pod nginx-pod` - 파드 삭제
6. 명령 창에 변화가 없으니 ^C로 중지
7. `kubectl get pod` - 파드 상태 확인 (Terminating)
8. `systemctl start kubelet` - kubelet 복구
9. `kubectl get pod` - nginx-pod 삭제 여부 확인

```
m-k8s w1-k8s w2-k8s w3-k8s
[root@m-k8s ~]# kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml
pod/nginx-pod created
[root@m-k8s ~]# kubectl get pod
NAME          READY   STATUS             RESTARTS   AGE
nginx-pod     0/1     ContainerCreating   0          5s
[root@m-k8s ~]# kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod     1/1     Running   0          8s
[root@m-k8s ~]# kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE    NOMINATED NODE   READINESS GATES
nginx-pod     1/1     Running   0          15s   172.16.132.2  w3-k8s  <none>           <none>
[root@m-k8s ~]# systemctl stop kubelet
[root@m-k8s ~]# kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod     1/1     Running   0          27s
[root@m-k8s ~]# kubectl delete pod nginx-pod
pod "nginx-pod" deleted
[root@m-k8s ~]# ^C
[root@m-k8s ~]# kubectl get pod
No resources found in default namespace.
[root@m-k8s ~]#
```

▼ 문제

 kubelet 서비스를 중지했는데도 delete가 잘 되는 문제 발생!

GPT 답변

Kubernetes에서 kubelet은 각 노드에서 컨테이너를 관리하는 역할을 합니다. 즉, kubelet이 작동하지 않으면 해당 노드에서는 새로운 파드를 시작하거나, 기존의 파드를 정지시키는 등의 작업을 할 수 없습니다. 하지만 kubelet이 멈춰 있어도, API 서버는 여전히 파드를 삭제하는 요청을 받아들일 수 있습니다.

따라서 여기서의 상황은, kubelet이 멈춘 상태에서 API 서버에 'kubectl delete pod' 명령을 통해 파드 삭제 요청을 보냈고, API 서버가 이를 받아들여 파드를 삭제한 것입니다.

그리고 'kubectl get pod' 명령은 API 서버에 저장된 정보를 바탕으로 출력되기 때문에, kubelet이 멈추어도 해당 파드의 상태 정보는 API 서버에 계속 저장되어 있습니다. 이러한 이유로 kubelet 서비스가 멈춘 상태에서도 파드 삭제가 가능한 것입니다.

그러나 kubelet이 멈춘 상태에서는 물리적으로 파드를 삭제하거나 새로운 파드를 시작하는 등의 작업이 이루어지지 않을 것입니다. 이는 kubelet이 다시 시작될 때까지 반영되지 않습니다.

kube-proxy

- kube-proxy는 파드의 통신을 담당한다.
- 앞서 [config.sh](#) 파일에서 br_netfilter 커널 모듈을 적재하고 iptables를 거쳐 통신하도록 설정했는데, 해당 설정이 정상적으로 작동하지 않는다면 (즉, kube-proxy에 문제가 생긴다면) 어떻게 될지 확인해보자

▼ 과정 및 결과

1. m `kubectl create -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml` - 파드 배포
2. m `kubectl get pod -o wide` - 파드의 IP와 워커 노드 확인
3. m `curl 172.16.103.130` - 파드의 IP로 nginx 웹 서버 메인 페이지 내용을 확인한다.

4. w2 `modprobe -r br_netfilter` `system restart network` - w2-k8s에 접속해 파드가 위치한 워커 노드에서 br_netfilter 모듈을 제거하고 네트워크를 다시 시작해 변경된 내용을 적용한다. ⇒ kube-proxy에 문제가 생기는 상황 만드는 것임
5. m `curl 172.16.103.130` - m-k8s에서 다시 한 번 파드의 nginx 웹 서버 페이지 정보를 받아온다. 오래걸리는데 아직도 파드에서 정보를 받아오지 못했다면 `^C` 로 종료
6. m `kubectl get pod -o wide` - 파드 상태 확인
7. w2 `modprobe br_netfilter` `reboot` - 다시 정상 상태로 복구
8. m `kubectl get pod -o wide` - 일정 시간 지난 후 파드 상태를 확인하면 RESTARTS가 1로 증가하고 IP 변경된 것을 확인 할 수 있음
9. m `curl 72.16.103.131` - 바뀐 IP로 파드로부터 정보를 정상적으로 받아오는 지 확인
10. m `kubectl delete -f ~/Book_k8sInfra/ch3/3.1.6/nginx-pod.yaml` - 다음 내용 진행을 위한 파드 삭제

3.2 쿠버네티스 기본 사용법 배우기

3.2.1 파드를 생성하는 방법

방법1 - run으로 파드 생성

- `kubectl run nginx-pod --image=nginx`
- `kubectl run [파드이름] [생성할 이미지의 이름]`
- `kubectl get pod` - 파드 생성 확인

방법2 - create로 파드 생성

- `kubectl create nginx --image=nginx` - 파드가 생성되지 않음

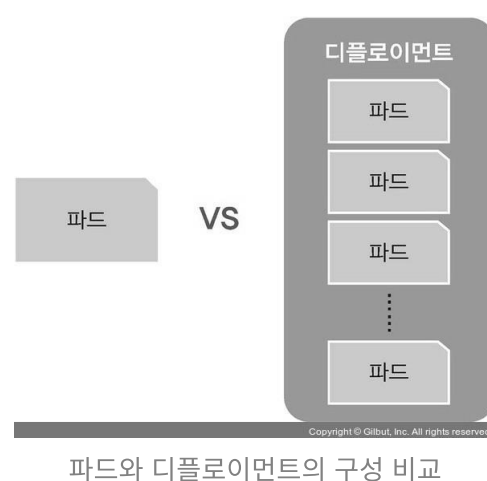
(deployment 추가해서 실행해야함)

- `kubectl create deployment dry-nginx --image=nginx` - 파드 생성됨 (중복을 피하기위해 dry-nginx라고 이름 설정)
- `kubectl get pods` - 생성된 파드 확인

run vs create 비교하기

- `kubectl get pods -o wide` 생성된 파드의 IP를 확인
- `curl [IP1]` - 웹 페이지 정보를 받아오는지 확인
- `curl [IP2]` - 웹페이지 정보를 받아오는지 확인

⇒ 둘 다 정상 작동한다. 어떤 차이가 있을까?



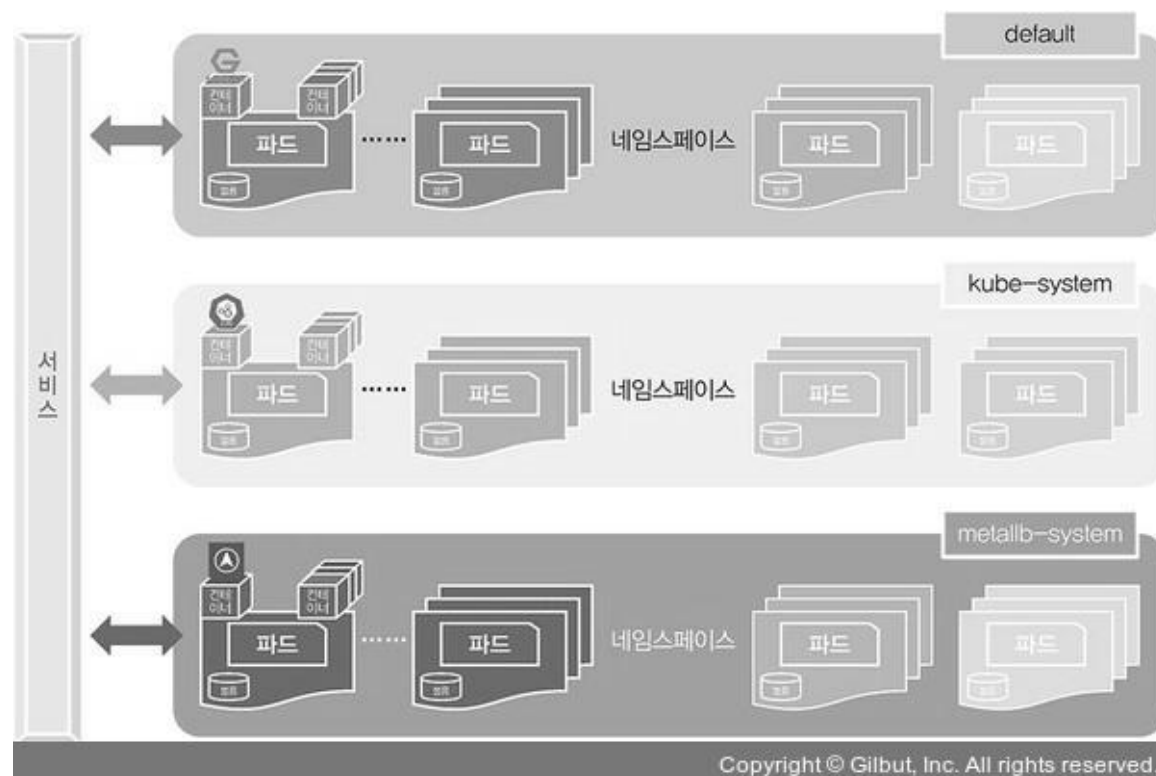
run으로 파드 생성 - 단일 파드 1개만 생성되고 관리 됨

create deployment로 파드 생성 - 디플로이먼트라는 관리 그룹 내에서 파드가 생성됨

3.2.2 오브젝트란

- 파드와 디플로이먼트는 스펙과 상태등의 값을 가지고 있음
- 이러한 값을 가지고 있는 파드와 디플로이먼트를 개별 속성을 포함해 부르는 단위를 오브젝트라고 한다.

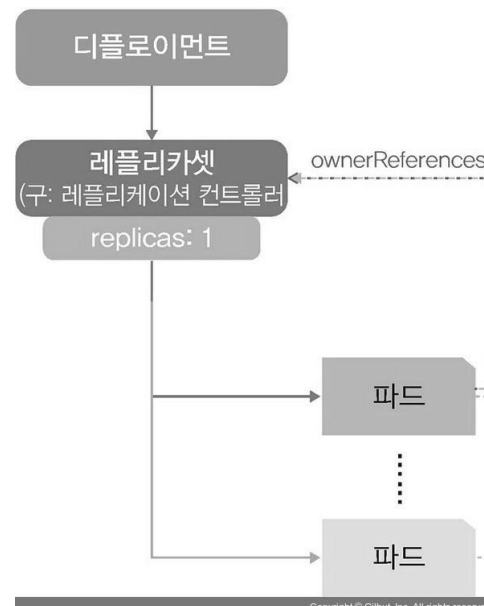
기본 오브젝트



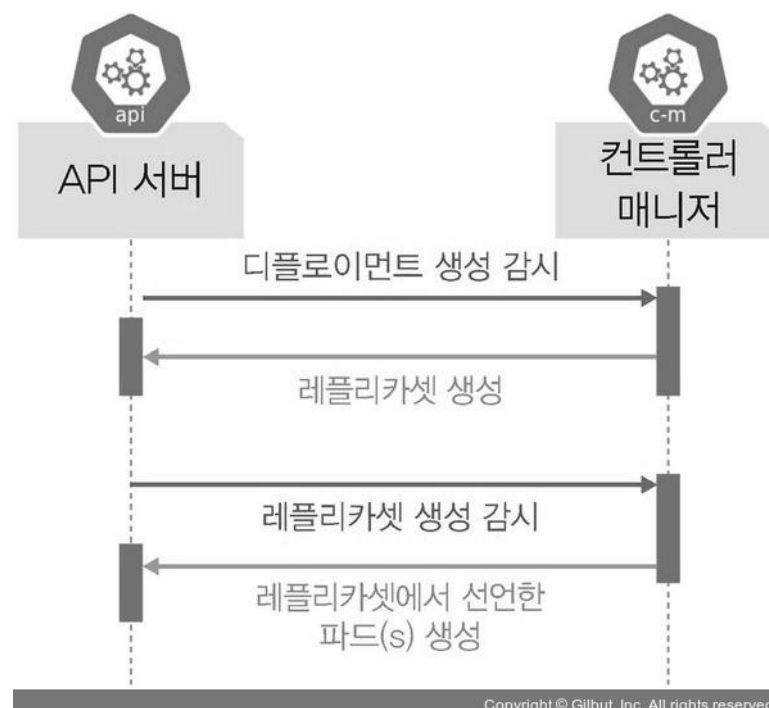
- **파드**
 - 쿠버네티스에서 실행되는 최소단위
 - 웹 서비스를 구동하는 데 필요한 최소 단위
 - 독립적인 공간과 사용가능한 IP를 가지고 있음
 - 하나의 파드는 1개 이상의 컨테이너를 갖고 있으므로 여러 기능을 묶어 하나의 목적으로 사용가능
 - 범용으로 사용할 시 대부분 1개의 파드에 1개의 컨테이너를 적용함
- **네임스페이스**
 - 쿠버네티스 클러스터에서 사용되는 리소스들을 구분해 관리하는 그룹
 - e.g.) 해당 장에서는 3가지 네임스페이스를 사용
 - **default** - 특별히 지정하지 않으면 기본으로 할당됨
 - **kube-system** - 쿠버네티스 시스템에서 사용됨
 - **metallb-system** - 온프레미스에서 쿠버네티스를 사용할 경우 외부에서 쿠버네티스 클러스터 내부로 접속하게 도와주는 컨테이너들이 속함
- **볼륨**
 - 파드가 생성될 때 파드에서 사용할 수 있는 디렉터리 제공
 - 기본적으로 영속되는 개념이 아니므로 제공되는 디렉터리도 임시로 사용
 - But, 파드가 사라지더라도 **저장/보존이 가능한 디렉터리를 볼륨 오브젝트를 통해** 생성하고 사용할 수 있음
- **서비스**
 - 파드는 클러스터 내에서 유동적이므로 접속 정보가 고정일 수 없음. **따라서, 파드 접속을 안정적으로 유지하도록 서비스를 통해 내/외부로 연결됨**
 - 서비스는 새로 파드가 생성될 때 부여되는 새로운 IP를 기존에 제공하던 기능과 연결해줌.
 - 즉, 쿠버네티스 외부에서 쿠버네티스 내부로 접속할 때 내부가 어떤 구조로 되어 있는지, 파드가 살았는지 죽었는지 신경쓰지 않아도 이를 논리적으로 연결하는 것이 서비스임.
 - 기존 인프라에서 로드밸런서(네트워크 트래픽을 여러 서버 사이에서 분산시켜 처리하도록 하는 장치나 프로그램), 게이트웨이(두 개의 다른 네트워크를 연결하는 노드)와 비슷한 역할을 함

디플로이먼트

- 기본 오브젝트만으로도 쿠버네티스를 사용할 수 있지만, 더 효율적으로 작동하도록 기능들을 조합하고 추가해 구현한 것이 디플로이먼트이다.
- 이외에도 데몬셋, 컨피그맵, 레플리카셋, PV, PVC, 스테이트풀셋 등이 있음
- 디플로이먼트 오브젝트는 파드에 기반을 두고 있으며, 레플리카셋 오브젝트를 합쳐 놓은 형태임
- 앞에서 생성한 dpy-nginx의 디플로이먼트 계층구조



- 실제로 API 서버와 컨트롤러 매니저는 단순히 파드가 생성되는 것을 감시하는 것이 아니라 디플로이먼트처럼 레플리카셋을 포함하는 오브젝트의 생성을 감시함



디플로이먼트 생성/삭제

- 디플로이먼트 생성
 - `kubectl create deployment dpy-hname --image=sysnet4admin/echo-hname`
 - `kubectl create deployment dpy-hname --image=[계정 이름]/[이미지 이름]`
- 생성된 디플로이먼트 확인
 - `kubectl get pods`
- 디플로이먼트 삭제
 - `kubectl delete deployment dpy-hname`

- 디플로이먼트 삭제 확인

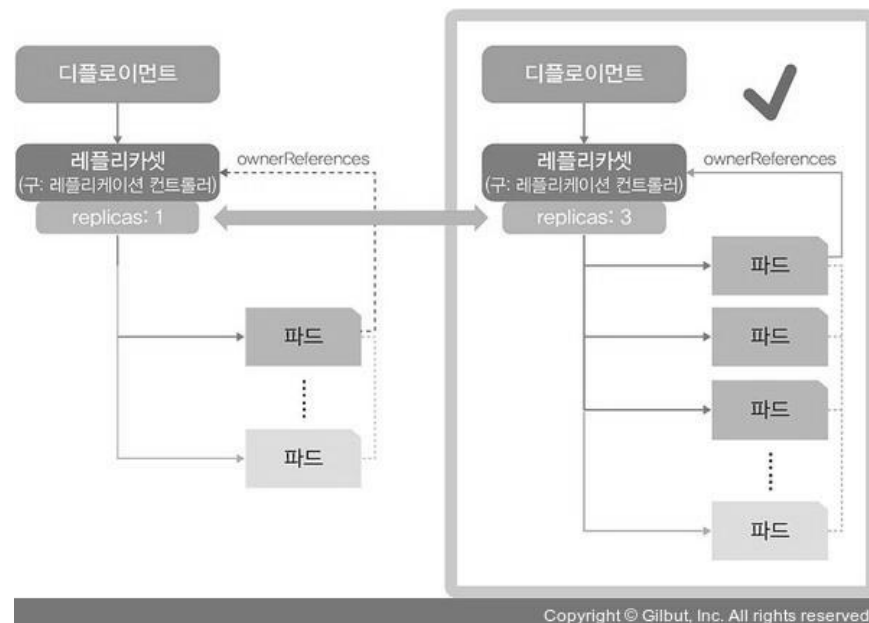
- `kubectl get pods`

아직 왜 디플로이먼트를 생성하는지 이해안되니 왜 필요한지 알아보자

3.2.3 레플리카셋으로 파드 수 관리하기

💡 레플리카셋 오브젝트를 제공하는 이유?

- 많은 사용자를 대상으로 웹 서비스를 하려면 다수의 파드가 필요한데, 이를 하나씩 생성하는 것은 매우 비효율적인 일
- 그래서 쿠버네티스에서는 다수의 파드를 만드는 레플리카셋 오브젝트를 제공함
- 레플리카셋으로 파드 수를 관리하는 과정



위의 과정 실습

- 배포된 파드의 상태 확인
 - `kubectl get pods`
- `nginx-pod`를 `scale` 명령으로 3개로 증가시키기
- ⇒ 에러 발생 : `nginx-pod`는 파드로 생성되었으므로 디플로이먼트 오브젝트에 속하지 않기 때문
 - `kubectl scale pod nginx-pod --replicas=3`
- 디플로이먼트로 생성된 `dpy-nginx`를 `scale` 명령과 `--replicas=3` 옵션으로 파드의 수를 3개로
 - `kubectl scale deployment dpy-nginx --replicas=3`
- 추가된 2개의 `nginx` 파드 확인
 - `kubectl get pods`
- `dpy-nginx`의 모든 파드가 정상적으로 워커 노드에 적용되고 IP가 부여되었는 지 확인
 - `kubectl get pods -o wide`
- 다음 진행을 위해 `dpy-nginx` 삭제
 - `kubectl delete deployment dpy-nginx`
- 삭제된 후 디플로이먼트의 상태 확인
 - `kubectl get pods`

3.2.4 스펙을 지정해 오브젝트 생성하기



문제

위에서 `kubectl create deployment` 명령으로 디플로이먼트를 생성했지만, 1개의 파드만 만들어졌을 뿐이다. 디플로이먼트를 생성하면서 한꺼번에 여러 개의 파드를 만들 수는 없는 것인가?



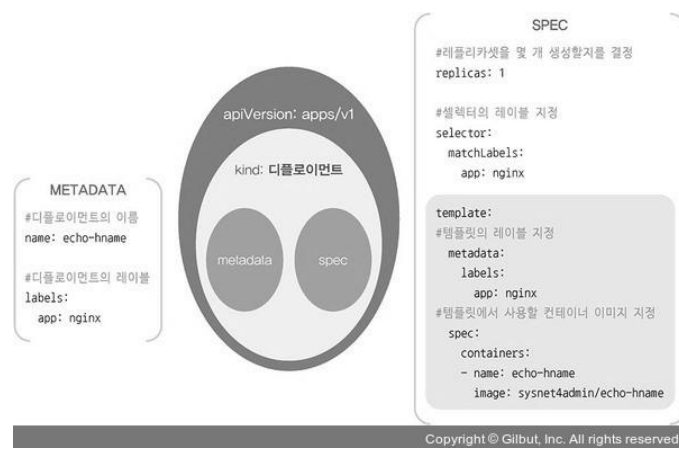
오브젝트 스펙

이런 설정을 적용하기 위해 필요한 내용을 파일로 작성하는데 이 파일을 오브젝트 스펙이라고 하며 보통 야믈 문법으로 작성한다.

3개의 nginx 파드를 디플로이먼트 오브젝트로 만들기

- 명령어로는 3개의 파드를 가진 디플로이먼트를 만들 수 없으므로 오브젝트 스펙을 작성해 디플로이먼트를 만든다.

- `echo-hname.yaml`



echo-hname.yaml 파일의 구조

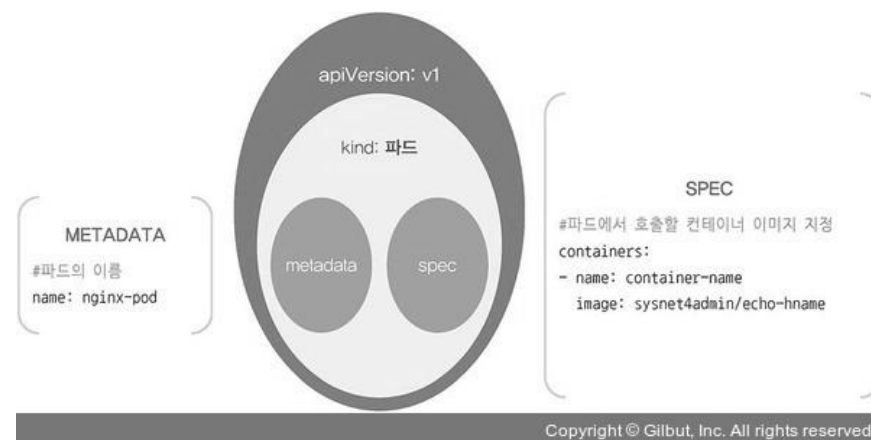
```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-hname
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: echo-hname
          image: sysnet4admin/echo-hname

```

- 앞에서 생성한 파드의 스펙과 비교하자. 위의 template 부분과 같음

`nginx-pod.yaml`



nginx-pod.yaml 파일의 구조

```

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

```

1. echo-hname.yaml 파일을 이용해 디플로이먼트 생성

```
kubectl create -f ~/_Book_k8sInfra/ch3/3.2.4/echo-hname.yaml
```

2. 새로 생성된 echo-hname 파드가 3개인지 확인

```
kubectl get pods
```

3. echo-hname.yaml 파일을 수정해 파드를 6개로 늘리기. sed 명령어 사용

```
sed -i 's/replicas: 3/replicas: 6/' ~/_Book_k8sInfra/sh3/3.2.4/echo-hname.yaml
```

4. replicas의 값이 3에서 6으로 변경됐는지 확인

```
cat ~/_Book_k8sInfra/sh3/3.2.4/echo-hname.yaml | grep replicas
```

5. 변경된 내용 적용 ⇒ 아무런 일 X 다음 챕터 확인하자.

```
kubectl create -f ~/_Book_k8sInfra/sh3/3.2.4/echo-hname.yaml
```

3.2.5 apply로 오브젝트 생성하고 관리하기

변경사항을 바로 적용하기 위해서 쿠버네티스는 apply라는 명령어를 적용한다!

1. replicas를 6으로 수정한 echo-hname.yaml 파일을 apply 명령으로 적용

```
kubectl apply -f ~/_Book_k8sInfra/sh3/3.2.4/echo-hname.yaml
```

2. 명령이 적용된 후에 echo-hname이 6개로 늘어났는지 확인. 특히 AGE 볼 것

```
kubectl get pods
```

▼ 표 3-4 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

- 쿠버네티스는 거의 모든 부분이 자동복구 되도록 설계되었음
- 특히 파드의 자동 복구 기술을 **셀프 힐링**이라고 한다.
- 셀프 힐링 - 제대로 작동하지 않는 컨테이너를 다시 시작하거나 교체해 파드가 정상적으로 작동하게 함

셀프 힐링 기능 체감 테스트

1. 파드의 IP 확인

```
kubectl get pods -o wide
```

2. 파드 컨테이너의 shell에 접속

```
kubectl exec -it nginx-pod -- /bin/bash
```

3. 컨테이너에서 구동하는 nginx의 PID 확인

```
cat /run/nginx.pid
```

4. 프로세스가 생성된 시간 확인

```
ls -l /run/nginx.pid
```

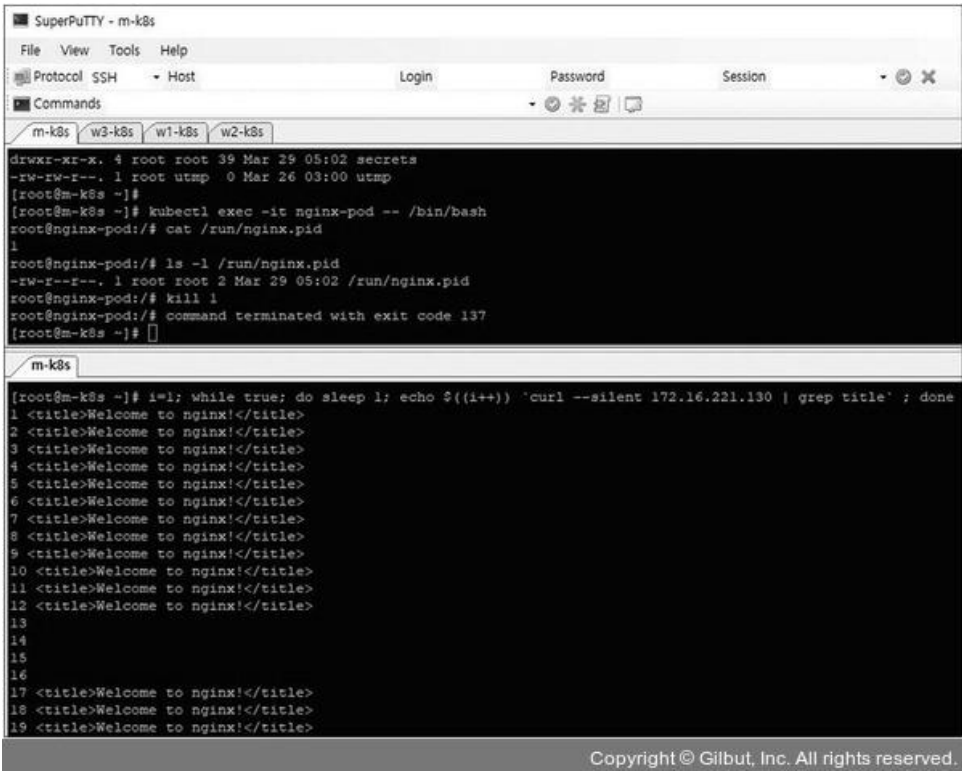
5. 슈퍼푸티에서 m-k8s 터미널을 1개 더 띄워 nginx-pod의 IP에서 돌아가는 웹 페이지를 1초마다 한번씩 요청하는 스크립트 실행.

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

6. 종료

```
kill 1
```

7. 추가한 터미널에서 1초마다 nginx 웹 페이지를 받아오는 스크립트 잘 작동하는지 확인



8. 웹페이지 복구되는 것을 확인 후에 다시 nginx-pod에 접속

```
kubectl exec -it nginx-pod -- /bin/bash
```

3.2.7 파드의 동작 보증 기능

쿠버네티스는 파드 자체에 문제가 발생하면 파드를 자동 복구해서 파드가 항상 동작 하도록 보장하는 기능도 있다.

1. 현재 파드 확인

```
kubectl get pods
```

2. 파드에 문제 발생하는 상황을 만들기 위해 앞서 생성한 파드 삭제

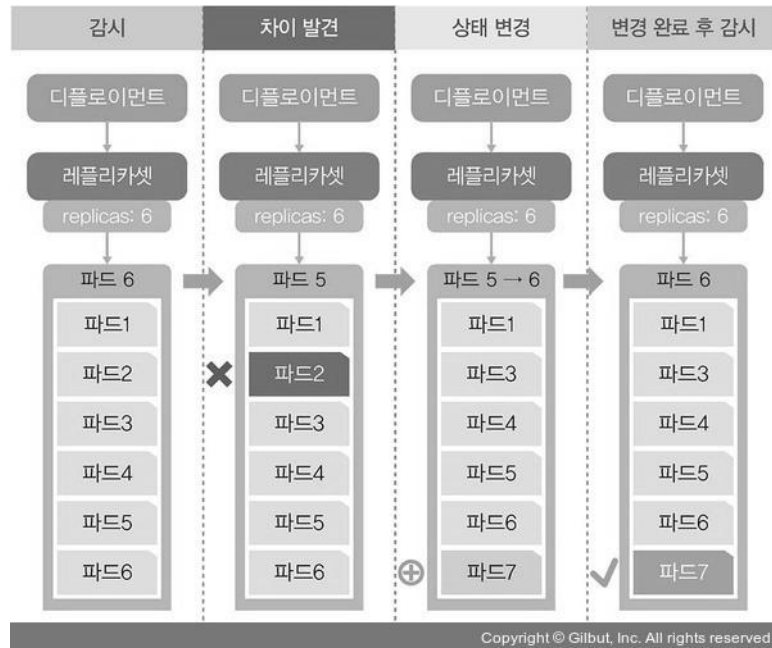
```
kubectl delete pods nginx-pod
```

3. 파드의 동작을 보증하려면 어떤 조건이 필요함. 조건을 확인하기 위해 다른 파드도 삭제해 서로 비교

```
kubectl delete pods echo-hname-5d754d565-7bzfs
```

4. 삭제 확인 ⇒ 6개 파드가 살아있다! (nginx-pod는 디플로이먼트에 속하지 않아서 바로 삭제되고 다시 생성되지 않았지만 echo-hanme 은 디플로먼트에 속한 파드고 ehco-hname에 속한 파드를 replicas에서 6개로 선언했으므로 다시 생성된다!)

```
kubectl get pods
```



디플로이먼트에 속한 파드를 삭제하는 방법

5. 상위 디플로이먼트를 삭제

```
kubectl delete deployment echo-hname
```

6. 배포된 파드가 남아있는지 확인

```
kubectl get pods
```

3.2.8 노드 자원 보호하기

- **노드** - 쿠버네티스 스케줄러에서 파드를 할당받고 처리하는 역할
- 문제가 있는 노드에 파드를 할당하면 문제가 생길 가능성 高. 하지만 어쩔 수 없이 해당 노드를 사용해야한다면? ⇒ 모니터링을 해서 파드 문제 최소화해야함. **어떻게 문제가 생길 가능성이 있는 노드라는 것을 쿠버네티스에 알려줄수있나?**
- 바로 **cordon** 기능을 사용

cordon으로 노드 관리하는 방법

1. echo-hname.yaml을 적용해 파드 생성

```
kubectl apply -f ~/Book_k8sInfra/sh3/3.2.4/echo-hname.yaml
```

2. scale 명령으로 배포한 파드를 9개로 늘리기

```
kubectl scale deployment echo-hname --replicas=9
```

3. 배포된 9개의 파드가 제대로 작동하는지, IP 할당이 잘 됐는지, 각 노드로 공평하게 배분됐는지 확인

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

4. scale로 파드의 수를 3개로 줄인다.

```
kubectl scale deployment echo-hname --replicas=3
```

5. 각 노드에 파드가 1개씩만 남았는지 확인

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

6. w3-k8s 노드에서 문제가 자주 발생해 현재 상태를 보존해야해 cordon 명령 실행

```
kubectl cordon w3-k8s
```

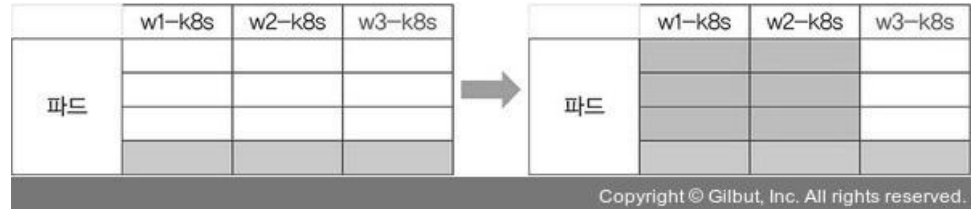
7. cordon 명령이 제대로 적용됐는지 확인 ⇒ w3-k8s가 더이상 파드가 할당되지 않는 상태로 변경되었음

```
kubectl get nodes
```

8. 이 상태에서 파드 수를 9개로 늘린다.

```
kubectl scale deployment echo-hname --replicas=9
```

9. 노드에 배포된 파드 확인 ⇒ w3-k8s에 추가로 배포된 파드가 있다! (w1 4개 w2 4개 w3 1개)



```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

10. 이번에는 파드의 수를 3개로 줄인다.

```
kubectl scale deployment echo-hname --replicas=3
```

11. 각 노드에 할당된 파드 수가 공평하게 1개씩인지 확인 ⇒ 그렇다

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

12. w3-k8s에 파드가 할당되지 않게 설정했던 것을 해제

```
kubectl uncordon w3-k8s
```

13. uncordon 적용 확인

```
kubectl get nodes
```

3.2.9 노드 유지보수하기

- 쿠버네티스를 사용하다 보면 정기/비정기적인 유지보수를 위해 노드를 꺼야하는 상황이 발생한다. 이를 대비해 쿠버네티스는 **drain 기능** 을 제공한다.
- drain은 지정된 노드의 파드를 전부 다른 곳으로 이동시켜 해당 노드를 유지보수할 수 있게 한다.

drain 기능 사용방법

1. 유지보수할 노드(w3-k8s)를 파드가 없는 상태로 만든다 ⇒ 수행 X

```
kubectl drain w3-k8s
```

2. DaemonSet을 무시하고 진행

```
kubectl drain w3-k8s --ignore-daemonsets
```

3. 노드 w3-k8s에 파드가 없는지 확인 ⇒ 없다!

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

4. w3-k8s 노드의 상태 확인 > SchedulingDisabled

```
kubectl get nodes
```

5. (유지보수가 끝났다고 가정) w3-k8s을 스케줄 받을 수 있는 상태로 복귀시킨다.

```
kubectl uncordon w3-k8s
```

6. 다시 노드 상태 확인

```
kubectl get nodes
```

7. 다음 진행을 위해 echo-hanme 삭제

```
kubectl delete -f ~/Book_k8sInfra/sh3/3.2.4/echo-hname.yaml
```

8. 배포된 파드가 없음을 확인

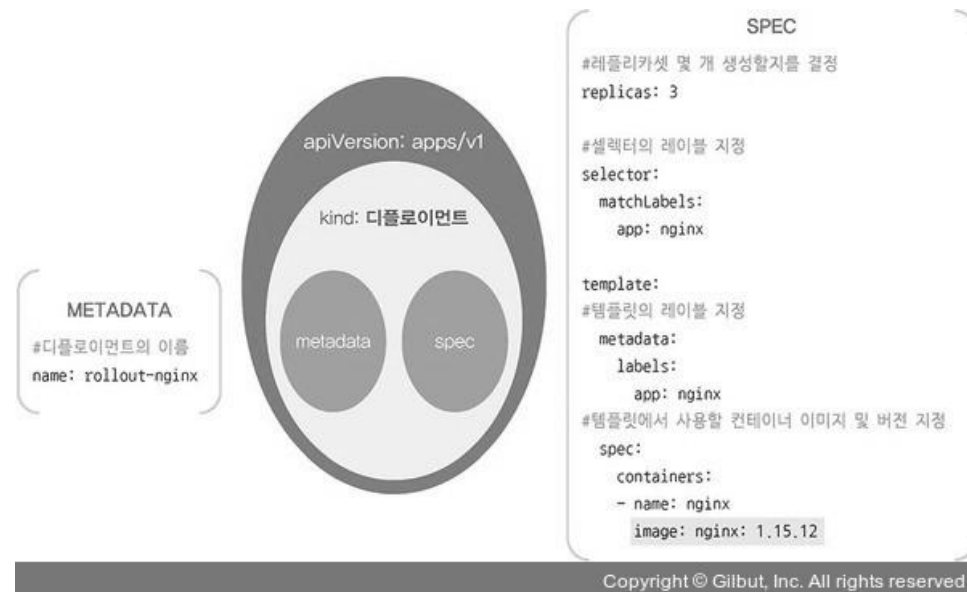
```
kubectl get pods
```

3.2.10 파드 업데이트하고 복구하기

파드 업데이트하기

1. 컨테이너 버전 업데이트를 테스트하기 위한 파드 배포

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.2.10/rollout-nginx.yaml --record
```



rollout-nginx.yaml 파일의 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
```

2. record 옵션으로 기록된 히스토리를 rollout history 명령으로 확인

```
kubectl rollout history deployment rollout-nginx
```

3. 배포한 파드의 정보 확인

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

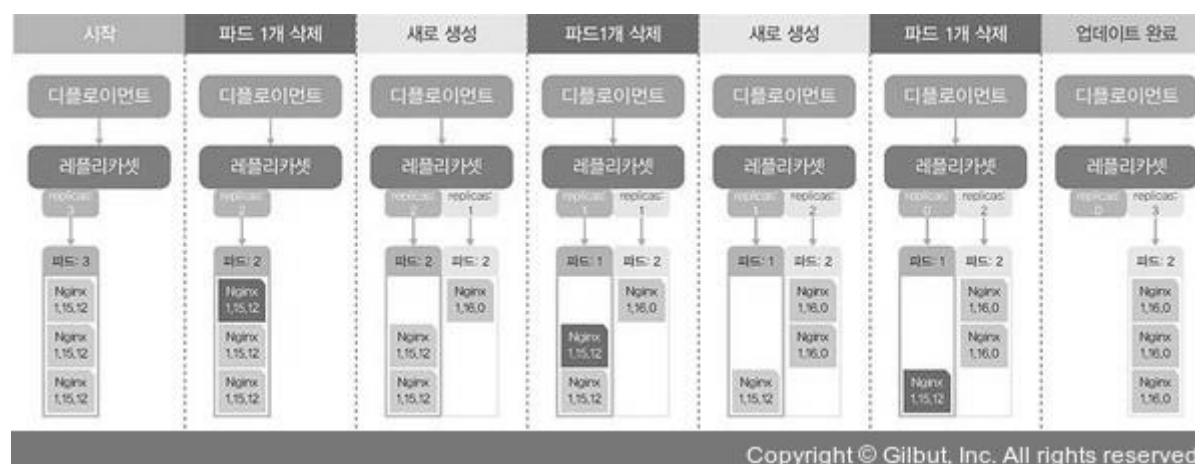
4. 배포된 파드에 속해 있는 nginx 컨테이너 버전 확인

```
curl -I -s [IP] | grep server
```

5. 파드의 nginx 컨테이너 버전을 업데이트. record로 명령 기록

```
kubectl set image deployment rollout-nginx nginx=nginx:1.16.0 --record
```

6. 업데이트 후 파드의 상태 확인 > IP 변경되었음 why? 파드는 언제라도 지우고 다시만들수있으므로 파드에 속한 nginx 컨테이너를 업데이트하는 가장 쉬운 방법은 파드를 관리하는 replicas의 수를 줄이고 늘려 파드를 새로 생성하는 것!



rollout으로 파드 업데이트 시 구성 변화

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

7. nginx 컨테이너가 모두 업데이트된 후 Deployment의 상태 확인

```
kubectl rollout status deployment rollout-nginx
```

8. rollout-nginx에 적용된 명령들을 확인

```
kubectl rollout history deployment rollout-nginx
```

9. 업데이트가 제대로 이루어졌는지 확인

```
curl -I -s [IP] | grep Server
```

업데이트 실패 시 파드 복구하기

업데이트 시 실수로 잘못된 버전을 입력했을 때 복구방법

1. nginx 컨테이너 버전을 의도와 다르게 입력

```
kubectl set image deployment rollout-nginx nginx=nginx:1.17.23 --record
```

2. 파드가 삭제되지 않고 pending(대기) 상태에서 넘어가지 않음

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

3. 문제 확인을 위해 rollout status를 실행. 새로운 replicas는 생성했으나 디플로이먼트를 배포하는 단계에서 대기 중으로 더 이상 진행되지 않을 것을 확인할 수 있음

```
kubectl rollout status deployment rollout-nginx
```

4. Deployment를 생성하려고 여러 번 시도했지만, 끝내 생성되지 않았다는 메시지 출력

```
kubectl rollout status deployment rollout-nginx
```

5. describe 명령으로 문제점 좀 더 살펴보기 > replicas가 새로 생성되는 과정에서 멈춰 있음. why? 1.17.23 버전의 nginx 컨테이너가 없기 때문.

```
kubectl describe deployment rollout-nginx
```

정상적인 상태로 복구하자

6. 업데이트 시 사용하던 명령들을 확인

```
kubectl rollout history deployment rollout-nginx
```

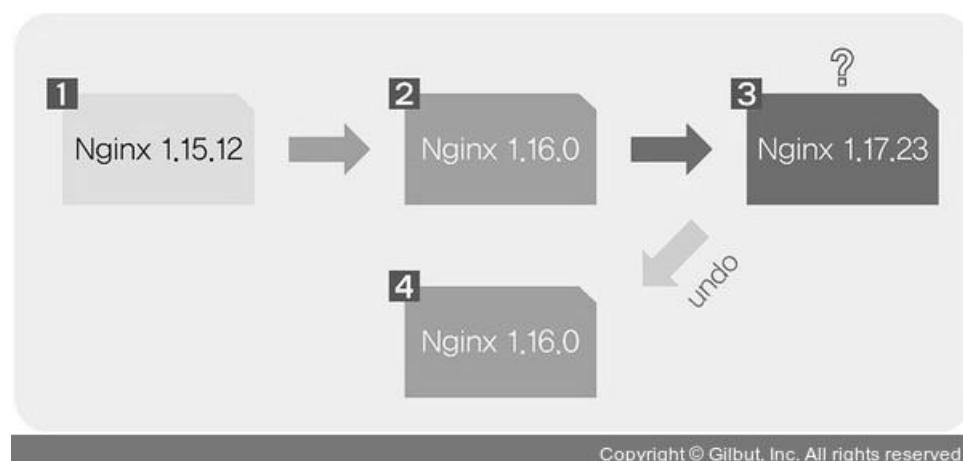
7. 명령 실행을 취소해 마지막 단계(revision 3)에서 전 단계(revision 2)로 상태를 되돌림

```
kubectl rollout undo deployment rollout-nginx
```

8. 파드 상태를 다시 확인

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

9. 실행된 명령 확인. 가장 최근 상태는 revision4가 된다. (2는 삭제됨)



```
kubectl rollout history deployment rollout-nginx
```

10. 배포된 컨테이너의 nginx 버전 확인 > 1.16.0로 되돌려 졌음을 확인 가능

```
curl -I -s [IP] | grep Server
```

11. 변경이 정상 적용되었는지 확인

```
kubectl rollout status deployment rollout-nginx
```

12. 현재 디플로이먼트 상태도 세부 점검 후 넘어가기

```
kubectl describe deployment rollout-nginx
```

특정 시점으로 파드 복구하기

1. 처음 상태인 revision 1로 돌아가기

```
kubectl rollout undo deployment rollout-nginx --to-revision=1
```

2. 새로 생성된 파드들의 IP 확인

```
kubectl get pods \
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName
```

3. 컨테이너 버전 확인 > 1.15.12 이므로 처음 상태로 복구 되었음

```
curl -I --silent [IP] | grep Server
```

4. 다음 단계 진행을 위해 디플로이먼트 삭제

```
kubectl delete -f ~/Book_k8sInfra/ch3/3.2.10/rollout-nginx.yaml
```

5. 배포된 파드 있는지 확인

```
kubectl get pods
```