쿠버네티스를 간단히 말하자면 Linux 컨테이너 작업을 자동화해주는 오픈소스 플랫폼 입니다. 각 컨테이너별 자원 제한, 문제 발생 시 자동 시작 등 컨테이너를 배포/확장, 제어, 자동화하기 위한 다양한 기능을 지원하는 컨테이너 오케스트레이션 도구

쿠버네티스 기본 용어

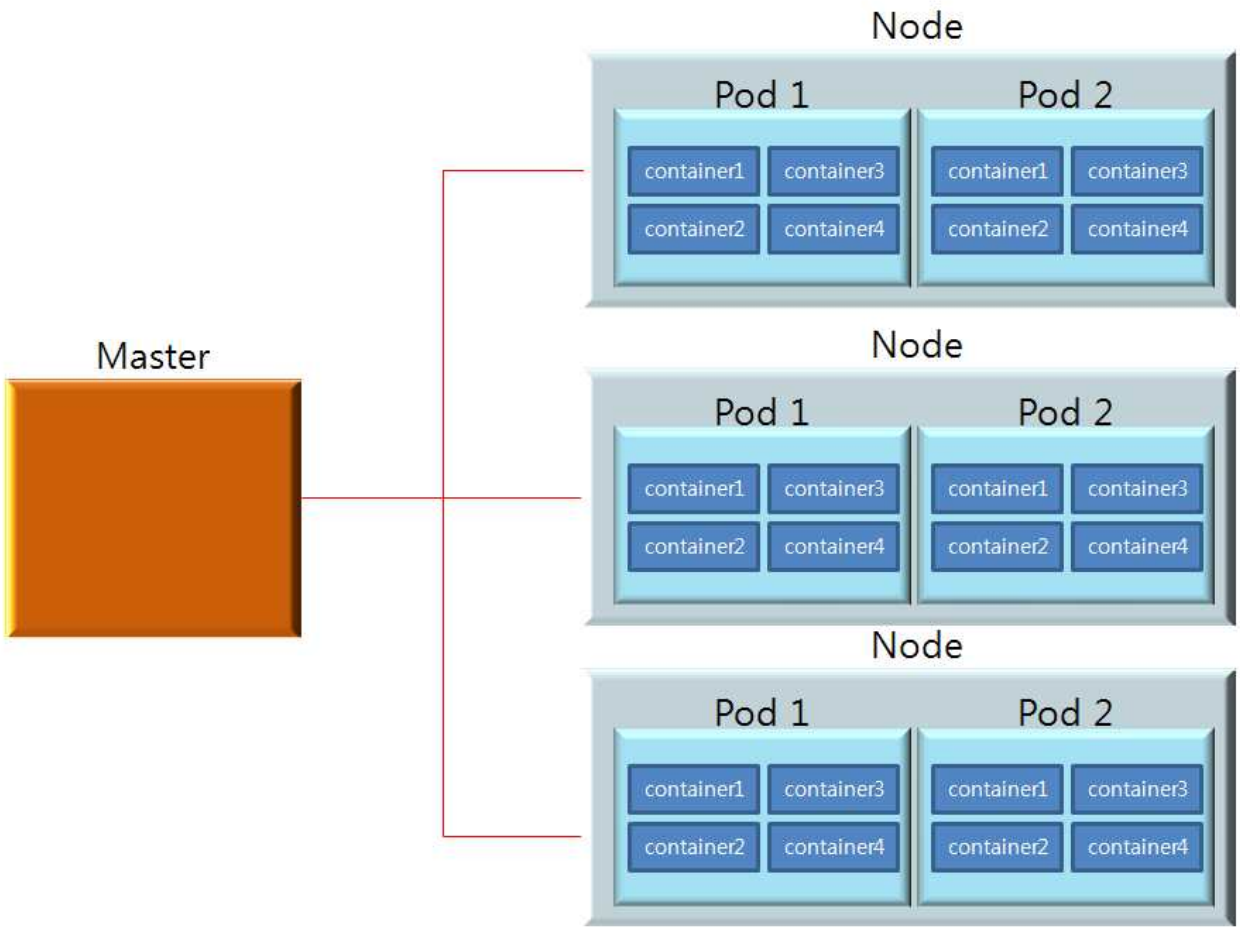
쿠버네티스에서 사용하는 기본 용어들이 있습니다.

마스터(Master) : 노드를 제어하고 전체 클러스터를 관리해주는 컨트롤러 이며, 전체적인 제어/관리를 하기 위한 관리 서버 입니다.

노드(nod) : 컨테이너가 배포될 물리 서버 또는 가상 머신 이며 워커 노드(Worker Node) 라고도 부릅니다.

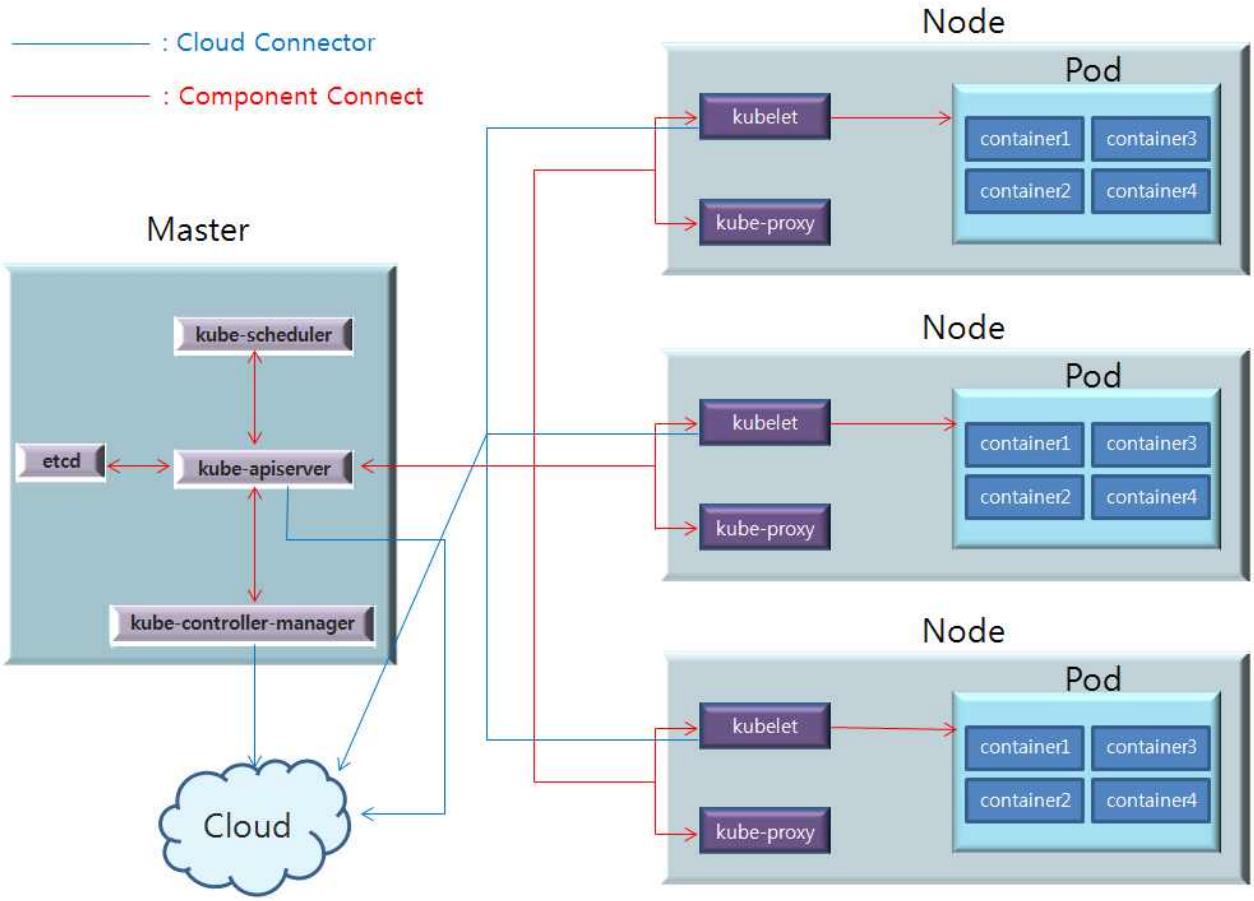
파드(pod) : 단일 노드에 배포된 하나 이상의 컨테이너 그룹이며, 파드라는 단위로 여러개의 컨테이너를 묶어서 파드 단위로 관리 할 수 있게 해줍니다.

대략적인 구조는 아래와 같습니다. (정확히는 Pod 를 묶어서 관리하는게 Docker 같은 툴 입니다.)



master 서버의 경우 고가용성 유지를 위해 여러개로 구성할 수 있으며 실제 관리는 리더 마스터 서버가 하되 나머지는 후보 마스터 서버로 유지 합니다.

만약 리더 마스터 서버가 장애 발생하면 후보 마스터 서버 중 하나가 리더 마스터 서버 역할을 맡아 서비스상의 이슈가 없도록 합니다.



쿠버네티스 컨트롤러 간단한 개념

내가 원하는 상태를 정하고, 그 상태를 컨트롤하기 위한것이 컨트롤러 입니다.

간단한 종류 및 개념은 아래와 같습니다.

데몬셋(DaemonSet) : 모든(또는 일부) 노드가 파드를 실행하도록 하며 클러스터에 노드가 추가되면 자동으로 파드도 같이 추가 됩니다.

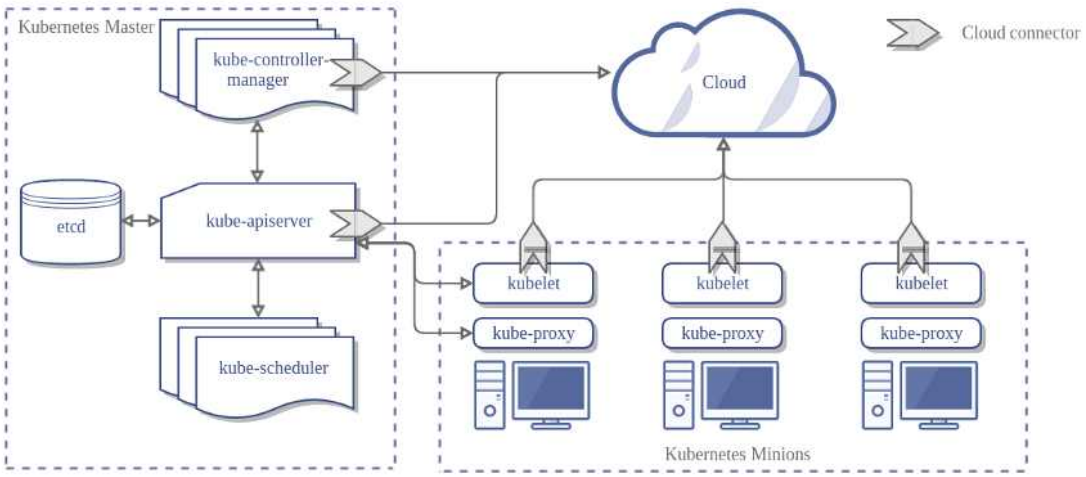
스테이트풀 셋(StatefulSet) : 어플리케이션의 스테이트풀을 관리하며, 여러개의 파드를 실행할 때 첫번째 파드가 정상 실행되어야 다음 파드가 실행된다거나 하는 순차적인 정상 배포 기능, 자동 롤링 업데이트 등의 기능을 제공 합니다.

디플로이먼트(Deployment) : 스테이트풀은 상태가 있는 어플리케이션을 관리했다면, 디플로이먼트는 상태가 없는 어플리케이션을 관리 합니다. 레플리카셋을 관리하며 배포할 때의 기본 단위가 될 수 있습니다.

레플리카셋(ReplicaSet) : 파드 개수에 대한 가용성을 보증하며, 지정한 개수 만큼의 파드가 항상 실행 되도록 관리 합니다.

잡(job) : 지정한 개수의 파드에서 실행할 작업을 정상적으로 수행할 수 있도록 보장 합니다.크론잡(cronjob)을 사용하여 지정한 주기에 따라 주기적으로 작업을 실행시킬 수 있습니다.

Kubernetes Architecture



Master

위의 그림에서 왼쪽의 마스터 컴포넌트는 클러스터의 제어영역(control plane) 을 제공하여, 클러스터에 관한 전반적인 결정을 수행하고 클러스터 이벤트를 감지하고 반응한다. 마스터 컴포넌트는 클러스터 내에 어떤 노드에서든 동작할 수 있지만, 일반적으로 클러스터와 동일한 노드 상에서 구동시킨다. 아래는 마스터 내에서 동작하는 바이너리 컴포넌트들이며 쿠버네티스 초기화시 자동 설치된다.

kube-apiserver

쿠버네티스 API 로, 외부/내부에서 관리자의 원격 명령을 받을 수 있는 컴포넌트이다.

Kubernetes 제어 영역의 프런트 엔드 역할을 합니다.

etcd

모든 클러스터 데이터를 저장하는 고가용성 키-값 저장소로, etcd 데이터에 대한 백업 계획은 필수이다.

kube-scheduler

생성된 파드를 노드에 할당해 주는 컴포넌트. 스케줄링이라 하며, 리소스/하드웨어/소프트웨어/정책/워크로드 등을 모두 참고하여 가장 최적화된 노드에 파드를 배치하게 될 것이다.

kube-controller-manager

컨트롤러 프로세스를 실행하고 클러스터의 실제 상태를 원하는 사양으로 조정합니다.

아래의 컨트롤러들을 구동하는 컴포넌트이다.

- Node Controller : 노드가 다운되었을 때 알림과 대응에 관한 역할
- Replication Controller : 지정된 수의 파드들을 유지시켜 주는 역할
- Endpoints Controller: 서비스와 파드를 연결시켜 엔드포인트 오브젝트를 만든다.
- Service Account & Token Controllers: 새로운 네임스페이스에 대한 기본 계정과 API 접근 토큰을 생성한다.

Kubelet

컨테이너 생성 및 관리를위한 기본 프로그램 인 Docker 엔진과 상호 작용하여 컨테이너가 포드에서 실행되도록합니다. 제공된 PodSpec 세트를 가져와 해당 컨테이너가 완전히 작동하는지 확인합니다.

Kube-proxy

네트워크 연결을 관리하고 노드간에 네트워크 규칙을 유지합니다.
주어진 클러스터의 모든 노드에서 Kubernetes 서비스 개념을 구현합니다.

쿠버네티스에서 사용하는 개념은 크게 객체(Object)와 그걸 관리하는 컨트롤러(Controller)가 있습니다.
객체는 사용자가 쿠버네티스에 바라는 상태(desired state)를 의미하고
컨트롤러는 객체가 원래 설정된 상태를 잘 유지할수있게 관리하는 역할을 합니다.
객체에는 포드(pod), 서비스(service), 볼륨(volume), 네임스페이스(namespace)등이 있습니다.
컨트롤러에는 ReplicaSet, Deployment, StatefulSet, DaemonSet, Job 등이 있습니다.

쿠버네티스 클러스터에 객체나 컨트롤러가 어떤 상태여야 하는지를 제출할때는 yaml 파일형식의 템플릿을 사용합니다.
템플릿의 기본 형식은 다음과 같습니다.

---
apiVersion: v1 <= object 를 생성하기 위한 api version, v1 외에 다른 버전도 있다.
Kind: Pod <= 어떤종류의 object 인지를 명시.
metadata: 해당 쿠버네티스 객체를 유니크하게 식별할수 있는 데이터이름, uid, 네임스페이스를 포함한다.
spec: 해당 객체의 의도

쿠버네티스는 현재 시스템을 사용자가 정의한 상태, 즉 사용자가 원하는 상태(어떤 Pod 이 몇 개가 떠있고, 어떤 Service 가 어떤 포트로 열려있고 등)로 맞춰줍니다. 그러려면 오브젝트의 현재 상태를 지속적으로 체크하고 상태를 제어해야 합니다.

컨트롤러 매니저(Controller Manager)에는 Replication, DaemonSet, Job, Service 등 다양한 오브젝트를 제어하는 컨트롤러가 존재합니다.

스케쥴러(Scheduler)는 노드의 정보와 알고리즘을 통해 특정 Pod 를 어떤 노드에 배포할 지 결정.
대상 노드들을 조건에 따라 걸러내고 남은 노드는 우선 순위(점수)를 매겨서 가장 최적의 노드를 선택합니다.

위의 모듈은 Control Plane 인 Master 노드에 존재하지만, Kubelet 과 Kube-proxy 는 Worker 노드에 존재합니다.
Kubelet 은 API 서버와 통신하며 Worker 노드의 작업을 제어하는 에이전트입니다.
Kube-proxy 는 Pod 에 접근하기 위한 iptables 를 설정합니다.
iptables 는 리눅스 커널의 패킷 필터링 기능을 관리하는 도구입니다.
이전에는 해당 패킷이 Kube-proxy 를 거쳐 지나갔기 때문에 proxy 라는 이름이 붙었지만, 지금은 패킷이 직접 통과하진 않습니다.

쿠버네티스 디플로이먼트

쿠버네티스 클러스터를 구동시키면, 그 위에 컨테이너화된 애플리케이션을 배포할 수 있다.
그러기 위해서, 쿠버네티스 디플로이먼트 설정을 만들어야 한다.
디플로이먼트는 쿠버네티스가 애플리케이션의 인스턴스를 어떻게 생성하고 업데이트해야 하는지를 지시한다.

디플로이먼트가 만들어지면, 쿠버네티스 마스터가 해당 디플로이먼트에 포함된 애플리케이션 인스턴스가 클러스터의 개별 노드에서 실행되도록 스케줄한다.

애플리케이션 인스턴스가 생성되면, 쿠버네티스 디플로이먼트 컨트롤러는 지속적으로 이들 인스턴스를 모니터링한다.
인스턴스를 구동 중인 노드가 다운되거나 삭제되면, 디플로이먼트 컨트롤러가 인스턴스를 클러스터 내부의 다른 노드의 인스턴스로 교체시켜준다.이렇게 머신의 장애나 정비에 대응할 수 있는 자동 복구(self-healing) 메커니즘을 제공한다.

오케스트레이션 기능이 없던 환경에서는, 설치 스크립트가 애플리케이션을 시작하는데 종종 사용되곤 했지만, 머신의 장애가 발생한 경우 복구를 해주지는 않았다. 쿠버네티스 디플로이먼트는 애플리케이션 인스턴스를 생성해주고 여러 노드에 걸쳐서 지속적으로 인스턴스가 구동되도록 하는 두 가지를 모두 하기 때문에 애플리케이션 관리를 위한 접근법에서 근본적인 차이를 가져다준다.

https://kubernetes.io/ko/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ : cluster 구성

Kubectl 이라는 쿠버네티스 CLI 를 통해 디플로이먼트를 생성하고 관리할 수 있다.
Kubectl 은 클러스터와 상호 작용하기 위해 쿠버네티스 API 를 사용한다.

kubectl command

$ kubectl [command] [type] [name] [flags]
[command]
- create
- get
- describe
- delete
- run

[type]
-pod
-service
[name]
resource 이름
[flag] => 옵션

ex)
kubectl run echoserver --generator=run-pod/v1 --image="k8s.gcr.io/echoserver:1.10" --port=8080
Flag --generator has been deprecated, has no effect and will be removed in the future.
pod/echoserver created

namespace 관련명령어
kubectl get namespace
NAME STATUS AGE
default Active 1d
kube-node-lease Active 1d
kube-public Active 1d
kube-system Active 1d

쿠버네티스는 처음에 세 개의 초기 네임스페이스를 갖는다.
default 다른 네임스페이스가 없는 오브젝트를 위한 기본 네임스페이스
kube-system 쿠버네티스 시스템에서 생성한 오브젝트를 위한 네임스페이스
kube-public 이 네임스페이스는 자동으로 생성되며 모든 사용자(인증되지 않은 사용자 포함)가 읽기 권한으로 접근할 수 있다. 이 네임스페이스는 주로 전체 클러스터 중에 공개적으로 드러나서 읽을 수 있는 리소스를 위해 예약되어 있다.

namespace 생성
kubectl create namespace[or ns] namespace 이름
kubectl run nginx --image=nginx --namespace=<insert-namespace-name-here>
kubectl get pods --namespace=<insert-namespace-name-here>

선호하는 네임스페이스 설정하기
이후 모든 kubectl 명령에서 사용하는 네임스페이스를 컨텍스트에 영구적으로 저장할 수 있다.
kubectl config set-context --current --namespace=<insert-namespace-name-here>
확인하기
kubectl config view

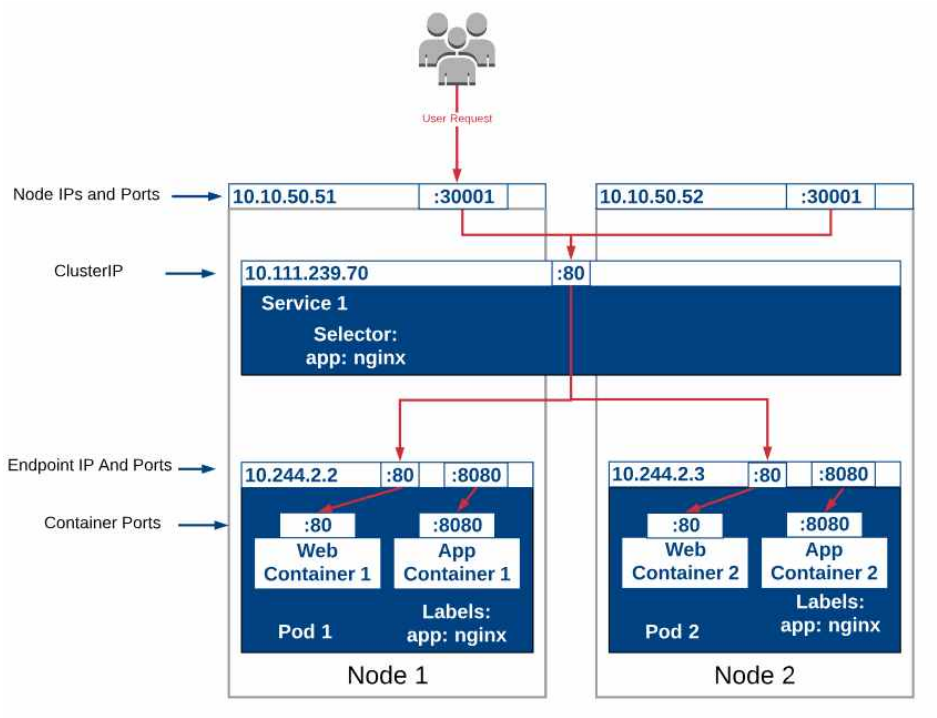
아래처럼 yaml 파일로 생성할수도 있다.

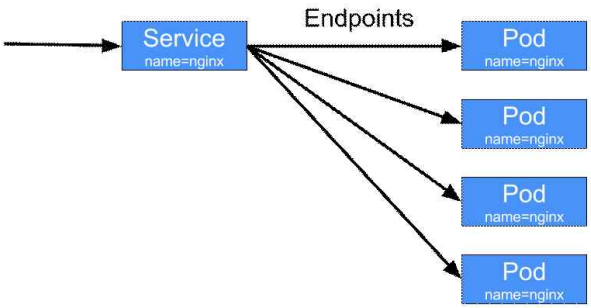
[root@minik test]# cat ns.yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: testns

[root@minik test]# kubectl apply -f ns.yaml

기타 kuberctl 명령어 참고 - 아래 링크.
https://kubernetes.io/ko/docs/reference/kubectl/cheatsheet/

쿠버네티스 ip address 및 port





kubernetes 관리를 위한 계정 생성

- master node 에서 아래와 같이 admin 계정(계정 id 는 달라도 상관없다)생성
useradd admin
passwd admin
echo “admin ALL=(ALL) NOPASSWD: ALL” > /etc/sudoers.d/admin

계정 설정 완료후 admin 계정으로 접속
[admin@master ~]$ mkdir -p $HOME/.kube
[admin@master ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[admin@master ~]$ sudo chown admin:admin /home/admin/.kube/config
[admin@master ~]$ echo "source <(kubectl completion bash)" >> ~/.bashrc
[admin@master ~]$ source .bashrc

kubernetes 주요 object

examples.

1. ns.yaml
```
---
apiVersion: v1
kind: Namespace
metadata:
  name: testns
```

2. apache.yaml
$ cat apache.yaml
```
---
apiVersion: v1
kind: Pod
metadata:
  name: apache-pod
  labels:
    app: myweb
spec:
    containers:
    - name: myweb-container
      image: httpd:2.4
      ports:
        - containerPort: 80
```

$ kubectl create -f apache.yaml
$ kubectl get pods

```
NAME          READY   STATUS    RESTARTS   AGE
apache-pod    1/1     Running   0          9m16s
```

* pod life cycle : pending -> create -> running ; 실패하면 CrashLoopBackOff -> running 상태일때까지 계속 반복될수 있다.
---

3. service

$ cat myweb-service.yaml
```
---
apiVersion: v1
kind: Service
metadata:
  name: myweb-service
spec:
  ports:
  - port: 8001
```

```
    targetPort: 80
  selector:
    app: myweb


$ kubectl create -f myweb-service.yaml
$ kubectl get svc
NAME              TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)     AGE
myweb-service     ClusterIP     10.109.82.232    <none>         8001/TCP    4m19s

$ curl http://10.109.82.232:8001
<html><body><h1>It works!</h1></body></html>
$
```

---

4. replica

```
$ cat replica.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: apache-replica
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache-replica-test
  template:
    metadata:
      labels:
        app: apache-replica-test
    spec:
      containers:
      - name: myweb-container2
        image: httpd:2.4
        ports:
        - containerPort: 80

$ kubectl create -f replica.yaml
replicaset.apps/apache-replica created
$ kubectl get pods
NAME                      READY    STATUS     RESTARTS    AGE
apache-pod                1/1      Running        0          50m
apache-replica-22m9n      1/1      Running        0          4s
apache-replica-cpjb5      1/1      Running        0          4s
apache-replica-tsqk9      1/1      Running        0          4s

kubectl delete pods apache-replica-22m9n
pod "apache-replica-22m9n" deleted
$ kubectl get pods   ; pod 를 삭제하더라도 항상 replica 수만큼 유지한다.
NAME                      READY    STATUS     RESTARTS    AGE
apache-pod                1/1      Running        0          51m
apache-replica-cpjb5      1/1      Running        0          46s
apache-replica-tsqk9      1/1      Running        0          46s
apache-replica-xznnm      1/1      Running        0          24s   <-- pod 가 새로 생성됨

$
* command :
$ kubectl scale --replicas=5 replicaset apache-replica
replicaset.apps/apache-replica scaled

$ kubectl get pods
NAME                      READY    STATUS     RESTARTS    AGE
```

```
apache-pod                1/1     Running   0          54m
apache-replica-26xc2      1/1     Running   0          7s
apache-replica-2jjhf      1/1     Running   0          7s
apache-replica-cpjb5      1/1     Running   0          3m42s
apache-replica-tsqk9      1/1     Running   0          3m42s
apache-replica-xznnm      1/1     Running   0          3m20s

$ kubectl scale --replicas=0 replicaset apache-replica
replicaset.apps/apache-replica scaled

$ kubectl get pods
NAME                      READY   STATUS        RESTARTS   AGE
apache-pod                1/1     Running       0          54m

$ replicas=0 이면 pod 가 전부 삭제된다.
```

---

5. deployment

- 컨테이너 어플리케이션을 배포 및 관리하는 역할
- deployment object 가 replica object 를 대신 할수 있으므로 replica object 는 잘 사용하지 않는다.
- 어플리케이션을 업데이트 할때 replicaset 의 revision 을 기록하여 rollback 이 가능하다.
- rolling update 가 가능하다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80

$ kubectl create -f deployment.yaml
deployment.apps/nginx-test created
$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
apache-pod                        1/1     Running   0          59m
nginx-test-66b6c48dd5-22pn6       1/1     Running   0          5s
nginx-test-66b6c48dd5-4gfrm       1/1     Running   0          5s
nginx-test-66b6c48dd5-j8lr2       1/1     Running   0          5s
$ kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE     IP               NODE
NOMINATED NODE   READINESS GATES
apache-pod                        1/1     Running   0          59m     192.168.39.80    node5.example.com
<none>           <none>
nginx-test-66b6c48dd5-22pn6       1/1     Running   0          13s     192.168.221.21   node2.example.com
<none>           <none>
nginx-test-66b6c48dd5-4gfrm       1/1     Running   0          13s     192.168.206.21   node4.example.com
```

```
<none>              <none>
nginx-test-66b6c48dd5-j8lr2    1/1       Running    0             13s    192.168.11.91      node1.example.com
<none>              <none>

$ kubectl get deployments.apps
NAME           READY    UP-TO-DATE    AVAILABLE    AGE
nginx-test    3/3       3             3            32s

$ kubectl get replicasets.apps
NAME                        DESIRED    CURRENT    READY    AGE
apache-replica               0          0          0        10m
nginx-test-66b6c48dd5       3          3          3        2m24s

update / rollback
$ kubectl set image deployment nginx-test   nginx=nginx:1.19.0 --record
$ kubectl rollout history
$ kubectl rollout history deployment nginx-test
$ kubectl rollout undo deployment --to-revision=2
[admin@master work]$ kubectl set image deployment nginx-test   nginx=nginx:1.18.0 --record
deployment.apps/nginx-test image updated
[admin@master work]$ kubectl get pods
NAME                              READY    STATUS              RESTARTS    AGE
nginx-test-67dfd6c8f9-8hs7d    0/1       ContainerCreating    0           5s
nginx-test-75c7f965d8-5pz6j    1/1       Running              0           7m56s
nginx-test-75c7f965d8-9pq57    1/1       Running              0           7m57s
nginx-test-75c7f965d8-sdccg    1/1       Running              0           7m59s

[admin@master work]$ kubectl get pods
NAME                              READY    STATUS          RESTARTS    AGE
nginx-test-67dfd6c8f9-8hs7d    1/1       Running         0           12s
nginx-test-75c7f965d8-5pz6j    1/1       Terminating     0           8m3s
nginx-test-75c7f965d8-9pq57    1/1       Running         0           8m4s
nginx-test-75c7f965d8-sdccg    1/1       Running         0           8m6s

[admin@master work]$ kubectl get pods
NAME                              READY    STATUS              RESTARTS    AGE
nginx-test-67dfd6c8f9-5dc5h    1/1       Running             0           8s
nginx-test-67dfd6c8f9-66hmt    0/1       ContainerCreating    0           7s
nginx-test-67dfd6c8f9-8hs7d    1/1       Running             0           20s
nginx-test-75c7f965d8-5pz6j    0/1       Terminating         0           8m11s
nginx-test-75c7f965d8-9pq57    0/1       Terminating         0           8m12s
nginx-test-75c7f965d8-sdccg    1/1       Running             0           8m14s
[admin@master work]$ kubectl get pods
NAME                              READY    STATUS              RESTARTS    AGE
nginx-test-67dfd6c8f9-5dc5h    1/1       Running             0           11s
nginx-test-67dfd6c8f9-66hmt    0/1       ContainerCreating    0           10s
nginx-test-67dfd6c8f9-8hs7d    1/1       Running             0           23s
nginx-test-75c7f965d8-5pz6j    0/1       Terminating         0           8m14s
nginx-test-75c7f965d8-sdccg    1/1       Running             0           8m17s
[admin@master work]$ kubectl get pods
NAME                              READY    STATUS          RESTARTS    AGE
nginx-test-67dfd6c8f9-5dc5h    1/1       Running         0           14s
nginx-test-67dfd6c8f9-66hmt    1/1       Running         0           13s
nginx-test-67dfd6c8f9-8hs7d    1/1       Running         0           26s
nginx-test-75c7f965d8-sdccg    1/1       Terminating     0           8m20s

[admin@master work]$ kubectl get pods
NAME                              READY    STATUS          RESTARTS    AGE
nginx-test-67dfd6c8f9-5dc5h    1/1       Running         0           19s
nginx-test-67dfd6c8f9-66hmt    1/1       Running         0           18s
nginx-test-67dfd6c8f9-8hs7d    1/1       Running         0           31s
nginx-test-75c7f965d8-sdccg    0/1       Terminating     0           8m25s
```

```
[admin@master work]$ kubectl get pods
NAME                              READY    STATUS      RESTARTS    AGE
nginx-test-67dfd6c8f9-5dc5h    1/1       Running    0             41s
nginx-test-67dfd6c8f9-66hmt    1/1       Running    0             40s
nginx-test-67dfd6c8f9-8hs7d    1/1       Running    0             53s
[admin@master work]$

[admin@master work]$ kubectl rollout history deployment --revision 11
deployment.apps/nginx-test with revision #11
Pod Template:
  Labels:          app=nginx
          pod-template-hash=56fdbbbdc8
  Annotations:   kubernetes.io/change-cause: kubectl set image deployment nginx-test nginx=nginx:1.17.0
--record=true
  Containers:
   nginx:
    Image:         nginx:1.17.0
    Port:          80/TCP
    Host Port:   0/TCP
    Environment:           <none>
    Mounts:       <none>
  Volumes:        <none>

[admin@master work]$ kubectl rollout history deployment --revision 11^C
[admin@master work]$ kubectl scale deployment nginx-test --replicas=10
deployment.apps/nginx-test scaled
[admin@master work]$ kubectl get pods
NAME                              READY    STATUS      RESTARTS    AGE
nginx-test-66b6c48dd5-75pq9    1/1       Running    0             7s
nginx-test-66b6c48dd5-7jtjn    1/1       Running    0             10m
nginx-test-66b6c48dd5-7l2gs    1/1       Running    0             7s
nginx-test-66b6c48dd5-7wc7c    1/1       Running    0             10m
nginx-test-66b6c48dd5-8xgzl    1/1       Running    0             7s
nginx-test-66b6c48dd5-9qlf9    1/1       Running    0             7s
nginx-test-66b6c48dd5-p7pks    1/1       Running    0             7s
nginx-test-66b6c48dd5-q4zbw    1/1       Running    0             10m
nginx-test-66b6c48dd5-rhpsh    1/1       Running    0             7s
nginx-test-66b6c48dd5-xlsw8    1/1       Running    0             7s
[admin@master work]$
```

---

6. 서비스 타입

a. clusterIP 타입

```
$ cat cluster-ip.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-clusterip
spec:
  ports:
    - name: nginx-port
      port: 8000
      targetPort: 80
  selector:
    app: nginx
  type: ClusterIP
$ kubectl create -f cluster-ip.yaml
```

```
service/nginx-clusterip created
$ kubectl get svc
NAME                TYPE          CLUSTER-IP         EXTERNAL-IP     PORT(S)      AGE
myweb-service       ClusterIP     10.109.82.232      <none>          8001/TCP     102m
nginx-clusterip     ClusterIP     10.110.121.231     <none>          8000/TCP     9s

$ kubectl describe svc nginx-clusterip
Name:                nginx-clusterip
Namespace:           myns
Labels:              <none>
Annotations:         <none>
Selector:            app=nginx
Type:                ClusterIP
IP:                  10.110.121.231
Port:                nginx-port   8000/TCP
TargetPort:          80/TCP
Endpoints:           192.168.206.23:80
Session Affinity:    None
Events:              <none>
$

$ curl http://10.110.121.231   ; 접속안됨

$ curl http://10.110.121.231:8000   ; 접속됨
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    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/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
$
```

---

b. NodePort 타입 서비스

```
[admin@master ~]$ cat node-port.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
spec:
  ports:
```

```
    - name: nginx-port
      port: 8000
      targetPort: 80
  selector:
    app: nginx
  type: NodePort

[admin@master ~]$

$ kubectl create -f nodeport.yaml
service/nginx-nodeport created
$ kubectl get svc
NAME              TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)           AGE
myweb-service     ClusterIP     10.109.82.232    <none>         8001/TCP          56m
nginx-nodeport    NodePort      10.104.4.186     <none>         8000:32245/TCP    6s


$ curl 10.104.4.186
curl: (7) Failed connect to 10.104.4.186:80; 연결이 거부됨
curl 10.104.4.186:8000
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
....
</html>



$ curl 10.104.4.186:32245
curl: (7) Failed connect to 10.104.4.186:32245; 연결이 거부됨

$ curl localhost:32245
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>

------------------------------------------------------------
*. deployment 를 삭제하면 replicaset 및 pod 가 전부 삭제된다.

]$ kubectl get deployments.apps
NAME           READY    UP-TO-DATE    AVAILABLE    AGE
nginx-test     3/3      3             3            8m24s
$ kubectl delete deployments.apps nginx-test
deployment.apps "nginx-test" deleted
$ kubectl get deployments.apps
No resources found in myns namespace.
$ kubectl get pods
NAME           READY    STATUS     RESTARTS    AGE
apache-pod     1/1      Running    0           68m

$ kubectl get svc   ; service 는 삭제되지 않고 남아 있다.
NAME              TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)           AGE
myweb-service     ClusterIP     10.109.82.232    <none>         8001/TCP          61m
nginx-nodeport    NodePort      10.104.4.186     <none>         8000:32245/TCP    5m14s
$ kubectl delete svc nginx-nodeport   ; 서비스 삭제
service "nginx-nodeport" deleted
```

* deployment 를 생성하지 않은 경우에는 replicaset 을 삭제하면 pod 가 전부 삭제된다.
ex)
$ [admin@master ~]$ kubectl delete replicasets.apps apache2-replica
[admin@master ~]$ kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
apache-sample	1/1	Running	0	7h7m
apache2-replica-lvvnf	0/1	Terminating	0	148m
apache2-replica-s44vq	0/1	Terminating	0	148m
apache2-replica-ts4tf	0/1	Terminating	0	148m
apache2-replica-tww56	0/1	Terminating	0	148m
apache2-replica-vgbzk	1/1	Terminating	0	5h5m

c. LoadBalancer type

aws 나 gcp 같은 클라우드 플랫폼에서는 기본적으로 쿠버네티스에서 loadbalancer 를 지원하지만
로컬에 설치된 kubernetes 는 오픈소스 프로젝트인 metal lb 를 설치해야 쿠버네티스에서 loadbalancer type 을 사용할 수 있다.

metal lb 사이트 :https://metallb.universe.tf/installation/

metal lb 설치

```
$ kubectl edit configmap -n kube-system kube-proxy
-->  strictARP: false 를 strictARP: true 로 수정

$ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/namespace.yaml
$ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/metallb.yaml

# On first install only
$ kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"
```

아래처럼 configmap.yaml 파일 작성

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.200.20-192.168.200.100   <-- worknode ip 대역

$ kubectl apply -f configmap.yaml

[admin@master work2]$ kubectl get ns
NAME                STATUS    AGE
default             Active    44h
kube-node-lease     Active    44h
kube-public         Active    44h
kube-system         Active    44h
metallb-system      Active    87m

[admin@master work2]$ kubectl get all -n metallb-system
NAME                                  READY    STATUS     RESTARTS    AGE
pod/controller-65db86ddc6-wtshs    1/1      Running    0           86m
pod/speaker-6vkpw                  1/1      Running    0           86m
pod/speaker-8r42z                  1/1      Running    0           86m
pod/speaker-msmqh                  1/1      Running    0           86m
pod/speaker-r8vnn                  1/1      Running    0           86m
pod/speaker-xvcdl                  1/1      Running    0           86m

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
AGE
daemonset.apps/speaker   5         5         5       5            5           kubernetes.io/os=linux   86m

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/controller    1/1     1            1           86m

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/controller-65db86ddc6    1         1         1       86m
```

```
[admin@master work2]$ cat apache_lb.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: myapache-lb
spec:
  ports:
    - name: myweb-svc
      port: 8001
      targetPort: 80
  selector:
    app: myweb-svc
  type: LoadBalancer
[admin@master work2]$
[admin@master work2]$ cat apache_svc_lb.yaml

$ kubectl apply -f apache_svc_lb.yaml

[admin@master work2]$ kubectl get svc
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)          AGE
myapache-pod8-svc           NodePort       10.97.248.86     <none>           8001:32546/TCP   22h
myapache-pod8-svc-new-lb    LoadBalancer   10.107.34.195    192.168.200.21   8001:32401/TCP   75m
nginx-clusterip             ClusterIP      10.102.15.2      <none>           8000/TCP         16h
nginx-nodeport              NodePort       10.99.62.224     <none>           8000:32366/TCP   16h
[admin@master work2]$
```

---

8. 쿠버네티스 영구볼륨 설정

참고 : https://kubernetes.io/ko/docs/concepts/storage/volumes/

컨테이너가 삭제되면 컨테이너 내의 데이터도 삭제된다. 데이터베이스 처럼 일반적으로 컨테이너가 삭제되더라도 데이터를 영구적으로 보존해야 하는 경우가 있을 수 있다.
이런 경우 영구볼륨설정이 필요하다.

apache.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: myapache
  labels:
    app: myweb-svc
spec:
    containers:
    - name: myapache-container
      image: httpd:2.4
      ports:
        - containerPort: 80
      volumeMounts:
      - name: hostpath-volume
        mountPath: /usr/local/apache2/htdocs
    volumes:
    - name: hostpath-volume
      # hostPath: pod 가 실행중인 node 에 디렉토리가 자동으로 생성이 되고
      # 컨테이너내의 mountPath 디렉토리가 바인딩된다
      hostPath:
        path: /var/tmp/web_docs
```

```
-----------------------------------------------------------------
apiVersion: v1
kind: Pod
metadata:
  name: nfs-storage-test
spec:
  containers:
    - name: nfs-container-test
      image: centos:7
      # container 실행시 즉시 종료되지 않도록 컨테이너 내에서 명령어 실행
      command: [ 'sh', '-c', '/usr/bin/sleep 3600s' ]
      volumeMounts:
      - name: nfs-volume
        # 컨테이너 내의 nfs 공유디렉토리의 마운트 포인터
        mountPath: /mnt
  volumes:
    - name: nfs-volume
      persistentVolumeClaim:
        claimName: nfs-pvc
-----------------------------------------------------------------------
아래처럼 pv 와 pvc 객체를 사용하여 영구스토리지 볼륨을 관리하는게 일반적이다.

pv - 영구 스토리지 볼륨을 설정하기 위한 객체
pvc - 영구 스토리지 볼륨 사용을 요청하기 위한 객체

accessModes:
  ReadWriteOnce -- 하나의 노드에서 볼륨을 읽기-쓰기로 마운트할 수 있다
  ReadOnlyMany -- 여러 노드에서 볼륨을 읽기 전용으로 마운트할 수 있다
  ReadWriteMany -- 여러 노드에서 볼륨을 읽기-쓰기로 마운트할 수 있다

CLI 에서 접근 모드는 다음과 같이 약어로 표시된다.
  RWO - ReadWriteOnce
  ROX - ReadOnlyMany
  RWX - ReadWriteMany

* reclaim policy - pvc 를 삭제했을때 스토리지 볼륨을 처리하는 방법에 대한 정책
  retain, delete, recycle 이 있다.
디폴트 정책은 retain.
retain - pvc 사용이 끝난후에도 스토리지 볼륨의 데이터를 보존
delete - pvc 사용이 끝난후에는 스토리지 볼륨 삭제 및 pv 도 삭제
recycle - pvc 사용이 끝난후에 스토리지 볼륨 데이터 삭제후 스토리지 볼륨을 사용가능한 상태로 설정


apiVersion: v1
kind: PersistentVolume
metadata:
  # 아래 name 은 어떤 이름이라도 상관 없다.
  name: nfs-pv
  labels:
    volume: nfs-pv-volume
spec:
  capacity:
    # 스토리지 크기 결정
    storage: 5Gi
  accessModes:
    # ReadWriteMany - multi node 에서 읽고쓰기가 가능하다.
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    # retain even if pods terminate
    Retain
  nfs:
    # NFS server's definition
```

```
    # nfs 공유 디렉토리
    path: /var/nfs_storage
   # nfs 서버의 주소
    server: 192.168.200.90
   # 공유디렉토리에 대해서 읽고 쓰기 권한 부여
    readOnly: false
---------------------------------------------------------------
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  # pvc 이름은 어떤 이름이라도 상관없다.
  name: nfs-pvc
spec:
  selector:
    matchLabels:
    # nfs pv 의 라벨이름과 일치해야 한다.
      volume: nfs-pv-volume
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       # storage size to use
       storage: 1Gi
-----------------------------------------------------------
```