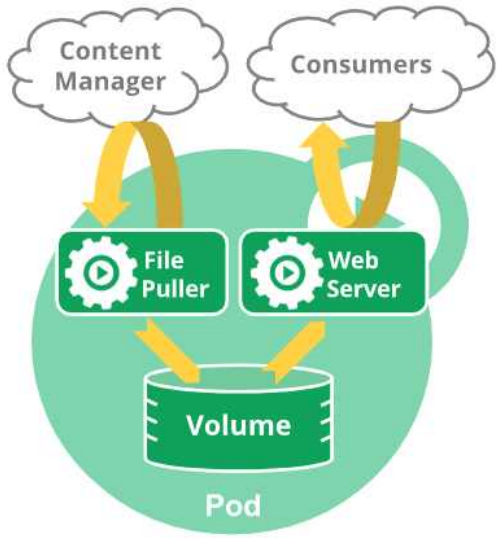
Kubernetes Controllers : Replication, Deployment, DaemonSet

1. Prerequisites
OS : CentOS v7.6
Arch : x86

2. k8s클러스터는 1마스터 2노드로 구성
Master : 4cpu, ram16G
Node : 2cpu, ram8G

3. Controller?
쿠버네티스는 크게 객체(object)와 그것을 관리하는 컨트롤러(controller)로 구성
객체(object)에는 pod, service, volume, namespace 등이 있다.
컨트롤러(Controller)에는 Replication, Deployment, StatefulSet, DaemonSet, Job 등
컨트롤러를 통해 객체를 사용자가 미리 설정했던 상태로 유지할 수 있도록 관리

Pod?
pod은 쿠버네티스가 배포, 관리할 수 있는 최소단위이며 컨테이너들의 집합



하나의 pod안에 있는 컨테이너들은 자원을 공유하게 되기 때문에, 동일한 목적을 가진 컨테이너들로 구성하는것이 좋다.
하나의 pod안에 File puller, Web server와 같이 역할은 다르지만 동일한 목적을 가진 컨테이너끼리 뭉쳐있으며 이들은 같은 volume을 공유하고 있는 것을 확인할 수 있다.

Replication Controller
지정된 숫자 만큼의 pod이 항상 클러스터내에서 실행되고 있도록 관리하는 역할
Controller를 사용하지 않고 pod을 직접 실행했을 때는 pod이 비정상적으로 종료되었을 때 재시작이 어려운데, Replication Controller를 사용해서 띄운 pod이라면 노드에 장애가 생겨 pod이 내려갔을때 클러스터 내의 다른 노드에 다시 pod을 실행

실습1 : 기본동작
$ vim replicationcontroller-nginx.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx        # replication controller의 이름
spec:
```

```
  replicas: 3        # pod의 복제본은 3개(원본포함)
  selector:          # nginx라는 label을 가진 pod을 select
    app: nginx
  template:
    metadata:
      name: nginx
      labels:        # spec.selector와 spec.template.metadata.labels는 같아야함
        app: nginx
    spec:             # 컨테이너의 정보
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

다음 명령어를 통해 클러스터에 pod을 배포
$ kubectl apply -f replicationcontroller-nginx.yaml
replicationcontroller/nginx created

배포 후 실행되고 있는 pod들
$ kubectl get pods

```
[root@kube-m yaml]# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-cntjd   1/1     Running   0          10s
nginx-csq5p   1/1     Running   0          10s
nginx-mf4h5   1/1     Running   0          10s
```

nginx의 복제본이 3개 만들어진것을 확인
첫번째 pod을 삭제
$ kubectl delete pod nginx-cntjd
pod "nginx-cntjd" deleted

pod을 확인해보면,
$ kubectl get pods

```
[root@kube-m yaml]# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-csq5p   1/1     Running   0          4h1m
nginx-mf4h5   1/1     Running   0          4h1m
nginx-zp7nz   1/1     Running   0          6s
```

첫번째 pod이 사라짐과 동시에 새로운 복제본이 생긴것을 확인할 수 있다.

실습2 : controller 삭제해보기
이번에는 컨트롤러를 삭제했다가 다시 생성
먼저 controller와 pod 확인
$ kubectl get rc,pods

```
[root@kube-m yaml]# kubectl get rc,pods
NAME                          DESIRED   CURRENT   READY   AGE
replicationcontroller/nginx   3         3         3       5h9m

NAME              READY   STATUS    RESTARTS   AGE
pod/nginx-csq5p   1/1     Running   0          5h9m
pod/nginx-mf4h5   1/1     Running   0          5h9m
pod/nginx-zp7nz   1/1     Running   0          68m
```

replication controller를 삭제.
# cascade=true이면 controller에 딸린 pod까지 전부 삭제
# default는 cascade=true
$ kubectl delete rc nginx --cascade=false

controller를 삭제한 상태에서 pod을 지우게되면 :

```
[root@kube-m yaml]# kubectl delete pod nginx-csq5p
pod "nginx-csq5p" deleted
[root@kube-m yaml]# kubectl get rc,pods
NAME              READY   STATUS    RESTARTS   AGE
pod/nginx-mf4h5   1/1     Running   0          5h12m
pod/nginx-zp7nz   1/1     Running   0          70m
```

pod 복제본3개가 유지되지 않는 것을 확인.

그럼 이번엔 replication controller를 다시 생성

```
[root@kube-m yaml]# kubectl apply -f replicationcontroller-nginx.yaml
replicationcontroller/nginx created
[root@kube-m yaml]# kubectl get rc,pods
NAME                          DESIRED   CURRENT   READY   AGE
replicationcontroller/nginx   3         3         3       5s

NAME              READY   STATUS    RESTARTS   AGE
pod/nginx-mf4h5   1/1     Running   0          5h13m
pod/nginx-xbt6f   1/1     Running   0          5s
pod/nginx-zp7nz   1/1     Running   0          72m
```

새롭게 3개의 복제본이 생기는 것이 아니라, **이미 있는 label을 체크해서 1개의 복제본만 추가**하는 모습을 확인

실습3 : label 바꿔보기
이번엔 실행중인 pod의 label을 변경
$ kubectl edit pod nginx-mf4h5

```
# Please edit the object below. Lines beginning with
# and an empty file will abort the edit. If an error
# reopened with the relevant failures.
#
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/podIP: 192.168.150.4/32
  creationTimestamp: "2019-11-12T01:01:05Z"
  generateName: nginx-
  labels:
    app: nginx-test
  name: nginx-mf4h5
  namespace: default
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicationController
```

빨갛게 표시해둔 부분을 nginx에서 nginx-test라는 label로 변경
변경 후에, pod 리스트를 출력함과 동시에 label만 필터링해서 출력.
$ kubectl get pods -o jsonpath='{range .items[*]}{.metadata.name}{"₩t"}{.metadata.labels}{"₩n"}{end}'

```
[root@kube-m yaml]# kubectl get pods -o jsonpath='{
{"\n"}{end}'
nginx-mf4h5     map[app:nginx-test]
nginx-mnq9j     map[app:nginx]
nginx-xbt6f     map[app:nginx]
nginx-zp7nz     map[app:nginx]
```

label을 변경했던 pod은 replication controller의 영향권내에서 벗어나게 된다.
nginx라는 label을 가진 pod이 하나 없어졌으므로 3개를 맞추기 위해 새로 생성된 모습을 확인
이렇게 label을 활용해 실행중인 pod을 재부팅없이 실제 서비스 상태에서 떼어내 다른 용도로 활용할 수 있다.

실습4 : rolling update
실행되고 있는 pod들을 업데이트시킬때 사용하는 방법
$ kubectl rolling-update {label }--image=nginx:latest

```
[root@kube-m yaml]# kubectl rolling-update nginx --image=nginx:latest
Command "rolling-update" is deprecated, use "rollout" instead
Created nginx-de92d601e9a36502627bd35472ed3654
Scaling up nginx-de92d601e9a36502627bd35472ed3654 from 0 to 3, scaling do
available, don't exceed 4 pods)
Scaling nginx-de92d601e9a36502627bd35472ed3654 up to 1
Scaling nginx down to 2
Scaling nginx-de92d601e9a36502627bd35472ed3654 up to 2
```

새로운 버전의 pod이 뜨고, 옛날버전은 그다음에 삭제되는 것을 확인할 수 있다.
즉, 1개씩 pod이 교체된다.

```
[root@kube-m ~]# kubectl get pod
NAME                                          READY   STATUS
nginx-de92d601e9a36502627bd35472ed3654-7nthl  0/1     ContainerCreating
nginx-de92d601e9a36502627bd35472ed3654-k9m77  1/1     Running
nginx-xbt6f                                   1/1     Running
nginx-zp7nz                                   1/1     Running
```

Replication controller의 다음 버전인 **ReplicaSet**에서는 rolling update가 지원되지 않는다. 그래서 일반적으로는 아래에서 언급할 Deployments controller를 사용합니다.

Deployments Controller
Deployments Controller는 일반적인 상태가 없는 application을 배포할때 사용하는 컨트롤러
처음에는 Replication Controller가 이 역할을 했지만 이제는 Deployments Controller가 기본적인 방법으로 사용되고 있다.
Deployments Controller는 기존에 Replication Controller가 했던 역할인 복제 세트 관리를 보다 세밀하게 관리할 수 있으며, 배포에 관한 다양한 기능을 가지고 있다.

실습1 : 기본동작
샘플 yaml파일을 생성.
$ vim deployments-nginx.yaml

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

생성한 yaml파일을 배포해보면, Replication Controller와 ReplicaSet을 포함해서 배포되는 것을 확인할 수 니다.
$ kubectl apply -f deployments-nginx.yaml

$ kubectl get deploy,rs,rc,pods

```
[root@kube-m yaml]# kubectl apply -f deployments-nginx.yaml
deployment.apps/nginx-deployment created
[root@kube-m yaml]# kubectl get deploy,rs,rc,pods
NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           33s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-85ff79dd56   3         3         3       33s

NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-85ff79dd56-4vxhx   1/1     Running   0          33s
pod/nginx-deployment-85ff79dd56-8sbjt   1/1     Running   0          33s
pod/nginx-deployment-85ff79dd56-p6nqn   1/1     Running   0          33s
```

실습2 : rolling-update
pod들이 생성되었으니 업데이트를 해보도록 하겠다.
기본적으로는 replication controller와 유사하게 동작한다.
# 현재 pod의 이미지 정보 출력
$ kubectl get deploy -o jsonpath
='{.items[0].spec.template.spec.containers[0].image}{"₩n"}'

# 업데이트
$ kubectl set image deployment/nginx-deployment nginx=nginx:latest

```
[root@kube-m yaml]# kubectl get deploy -o jsonpath='{.items[0].spec.template.spec.containers[0].image}{"\n"}'
nginx
[root@kube-m yaml]# kubectl set image deployment/nginx-deployment nginx=nginx:latest
deployment.apps/nginx-deployment image updated
[root@kube-m yaml]# kubectl get deploy -o jsonpath='{.items[0].spec.template.spec.containers[0].image}{"\n"}'
nginx:latest
```

nginx이미지가 latest버전으로 업데이트된것을 확인할 수 있다.
이전과 동일하게 새로운 pod을 먼저 만들고 그 후 기존의 pod을 삭제하는 방식으로 진행.

```
[root@kube-m yaml]# kubectl set image deployment/nginx-deployment ng
deployment.apps/nginx-deployment image updated
[root@kube-m yaml]# kubectl get pods
NAME                                READY   STATUS              REST
nginx-deployment-59c9f8dff-8trkk    1/1     Running             0
nginx-deployment-59c9f8dff-pknq4    1/1     Running             0
nginx-deployment-59c9f8dff-szblh    1/1     Running             0
nginx-deployment-85ff79dd56-f2f7g   0/1     ContainerCreating   0
```

실습3 : rollout(롤백)
배포한 버전을 롤백시킬수도 있다.
# 이미지 명 및 변경내역 확인
$ kubectl rollout history deploy nginx-deployment

# 바로 직전버전으로 롤백(undo)
$ kubectl rollout undo deploy nginx-deployment

# 특정 버전으로 롤백
$ kubectl rollout undo deploy nginx-deployment --to-revision=3

되돌릴 수 있는 revision 개수는 .spec.revisionHistoryLimit 옵션을 이용해서 조정할수 있다.

실습4 : pod 개수 조정
실행하고있는 pod들의 개수를 조절할수도 있다.
원래 3개의 복제본을 가지고 있었는데 5개로 늘리는 경우 :

$ kubectl scale deploy nginx-deployment --replicas=5

```
[root@kube-m yaml]# kubectl scale deploy nginx-deployment --replicas=5
deployment.apps/nginx-deployment scaled
[root@kube-m yaml]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-85ff79dd56-7c26t   1/1     Running   0          3m58s
nginx-deployment-85ff79dd56-9c8mw   1/1     Running   0          7s
nginx-deployment-85ff79dd56-fjkll   1/1     Running   0          3m53s
nginx-deployment-85ff79dd56-ptvsv   1/1     Running   0          3m56s
nginx-deployment-85ff79dd56-vrh8h   1/1     Running   0          7s
```

2개의 pod이 새로 추가된 모습을 확인할 수 있다.

DaemonSet Controller
다음은 클러스터 전체에 pod을 띄울 때 사용하는 controller이다.
클러스터 내부에 새로운 노드가 추가되었을 때, 자동으로 그 노드에 pod을 실행시켜주고
노드가 사라지면 노드속의 pod은 그대로 사라지게 된다.
보통 로그수집이나, 모니터링 등 항상 실행시켜두어야 하는 pod에 사용.

실습1 : 기본동작
이번에는 node가 1개인 상태에서 실습을 진행.

```
[root@kube-m yaml]# kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
kube-m     Ready    master   37m     v1.16.2
kube-n01   Ready    <none>   8m42s   v1.16.2
```

이번엔 nginx말고 elasticsearch를 배포
$ vim daemonset-elastic.yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      app: fluentd-elasticsearch
  template:
    metadata:
      labels:
        app: fluentd-elasticsearch
    spec:
      tolerations:
      # this toleration is to have the daemonset runnable on master nodes
      # remove it if your masters can't run pods
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
```

```
        #image: k8s.gcr.io/fluentd-elasticsearch:1.20
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

$ kubectl apply -f daemonset-elastic.yaml
혹은
$ kubectl apply -f https://k8s.io/examples/controllers/daemonset.yaml

daemonset.apps/fluentd-elasticsearch created

마스터에서 pod 목록을 확인해보면
$ kubectl get pods -A

```
[root@kube-m yaml]# kubectl get pods -A |grep fluentd
kube-system   fluentd-elasticsearch-5f6c5                1/1
kube-system   fluentd-elasticsearch-dlntl                1/1
```

정상적으로 실행된 것을 확인할 수 있고, 현재 두개의 pod이 올라온 것을 확인할 수 있다.

실제로 각각의 pod이 어느 노드에 올라가 있는지 확인.
$ kubectl describe pod fluentd-elasticsearch --namespace=kube-system |grep Node:

```
[root@kube-m yaml]# kubectl describe pod flue
ode:
Node:           kube-n01/
Node:           kube-m/
```

마스터와 1번노드에 올라가있는것을 확인할 수 있다.

이제 2번노드를 클러스터에 추가한 뒤, 바로 확인

```
[root@kube-m yaml]# kubectl describe pod fluentd-el
ode:
Node:         kube-n01/
Node:           kube-n02/
Node:         kube-m/
```

정상적으로 2번노드에도 배포된 것을 보실 수 있다.