올인원 패키지 Online.

# CKA 쿠버네티스 자격증 과정

PART1 | CKA 자격증 소개
자격증 등록과 Hands-On 실습 환경 만들기

PART2 | Kubernetes 아키텍쳐
Etcd 백업/복구, Kubernetes 업그레이드, RBAC 인증 실습

**PART3 | Workloads & Scheduling**
**Application deploy 후 scale/rollback, pod 스캐줄링 실습**

PART4 | Services & Networking
Service 운영과 접근제한(NetworkPolicy),Ingress 운영 실습

PART5 | Storage
StorageClass 적용한 PV, PVC 생성하여 pod에 적용하기 실습

PART6 | Troubleshooting
controller component 설정 정보 수정및 node/pod 문제해결

PART7 | 실전문제풀이
실전 문제를 직접풀어보고, 답안 리뷰 확인

# Part3 Workloads & Scheduling

# Part3 Workloads & Scheduling

## 01 Pod 기본

# Pod 란

Pod란 컨테이너를 표현하는 k8s API의 최소 단위

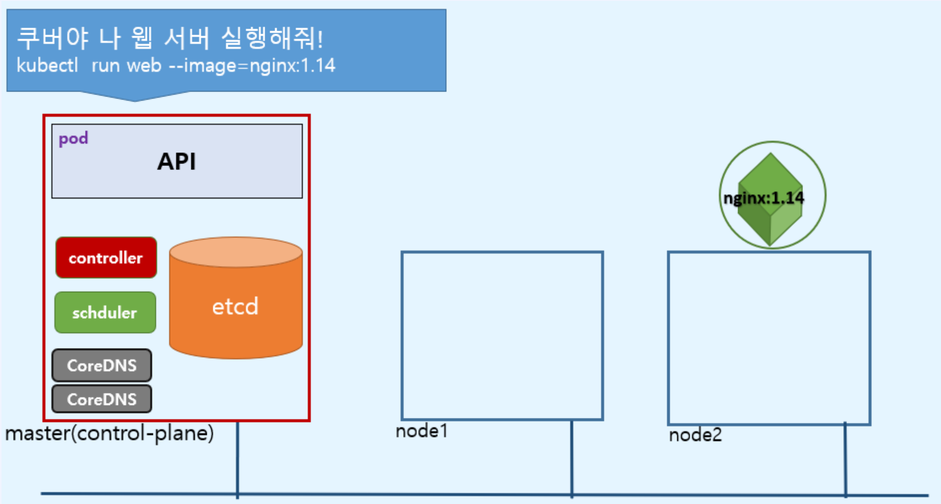
Pod에는 하나 또는 여러 개의 컨테이너가 포함될 수 있음

# Pod 생성(1)



## 명령어로 CLI 모드에서 생성

nginx 웹 서버 컨테이너를 pod로 동작시키기
- pod name: **web**
- image : **nginx:1.14**
- port : **80**




```
$ kubectl run  web  --image=nginx:1.14 --port=80

$ kubectl get pod -n devops
NAME   READY   STATUS    RESTARTS   AGE
web    1/1     Running   0          18s

$ kubectl delete pod web
```

# Pod 생성(2)

## Yaml 템플릿으로 생성

nginx 웹 서버 컨테이너를 pod로 동작시키기
- pod name: **web**
- image : **nginx:1.14**
- port : **80**

```
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  containers:
  - image: nginx:1.14
    name: web
    ports:
    - containerPort: 80
```

```
$ kubectl run web  --image=nginx:1.14 --port=80 --dry-run -o yaml > web.yaml
$ vi web.yaml

$ kubectl apply -f web.yaml
$ kubectl get pod -n devops
$ kubectl delete pod -n devops web
```

# 문제1:
# Pod 생성하기

■ 작업 클러스터 : **k8s**

'**cka-exam**'이라는 namespace를 만들고, 'cka-exam' namespace에 아래와 같은 Pod를 생성하시오.

- pod Name: **pod-01**
- image: **busybox**
- 환경변수 : **CERT = "CKA-cert"**
- command: **/bin/sh**
- args: **-c "while true; do echo $(CERT); sleep 10;done"**

```
$ kubectl config use-context k8s
$ kubectl create namespace cka-exam
$ kubectl get namespaces cka-exam

$ kubectl run pod-01 --image=busybox  --dry-run -o yaml > 2-1.yaml
$ vi 2-1.yaml
...
$ kubectl apply -f 2-1.yaml
$ kubectl get pod -n cka-exam
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-01
  namespace: cka-exam
spec:
  containers:
  - env:
    - name: CERT
      value: CKA-cert
    image: busybox
    name: pod-01
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(CERT); sleep 10;done"]
```

문제2:
pod의 로그 확인해서 결과 추출하기

## ■ 작업 클러스터 : hk8s

Pod "**custom-app**"의 log를 모니터링하고 '**file not found**'메세지를 포함하는 로그 라인인 추출하세요.
추출된 결과는 **/opt/REPORT/2022/custom-app-log**에 기록하세요.

```
$ kubectl config use-context hk8s
$ kubectl get pod custom-app

$ kubectl logs custom-app | grep 'file not found' >
$ cat /opt/REPORT/2022/custom-app-log
```

# Static Pod 만들기

- API 서버 없이 특정 노드에 있는 **kubelet** 에 의해 직접 관리
- **/etc/kubernetes/manifests/** 디렉토리에 **pod yaml** 파일을 저장 시 적용됨

- **static pod** 디렉토리 구성
  # cat **/var/lib/kubelet/config.yaml**
  …
  **staticPodPath: /etc/kubernetes/manifests**

- 디렉토리 수정시 kubelet 데몬 재실행
  # **systemctl restart kubelet**

- hk8s-w1에 static Pod 만들기
  ① ssh로 hk8s-w1 서버에 접속
  ② sudo -i로 root 권한 얻기
  ③ /etc/kubernetes/manifests 디렉토리에 pod  yaml파일을 저장하기

# 문제3:
# static pod 생성하기

hk8s-w1 노드에 **nginx-static-pod.yaml** 라는 이름의 Static Pod를 생성하세요.

- pod name: **nginx-static-pod**
- image: **nginx**
- port : **80**

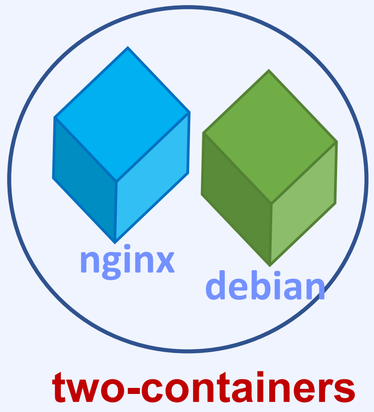
```
$ ssh hk8s-w1
$ sudo -i
# grep staticPod /var/lib/kubelet/config.yaml
staticPodPath: /etc/kubernetes/manifests

# cd /etc/kubernetes/manifests/
# cat > nginx-static-pod.yaml
...

# exit
$ exit

$ kubectl get pods
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-pod
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      protocol: TCP
```

# Part3 Workloads & Scheduling

## 02 Pod 응용

- 하나의 Pod에 여러 개의 컨테이너가 포함되어 함께 실행됨




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  containers:
  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html
  - name: debian-container
    image: debian
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
volumes:
  - name: shared-data
    emptyDir: {}
```

# 문제4:
multi container Pod 생성하기

## ■ 작업 클러스터 : k8s

4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성하시오.
- pod image: nginx, redis, memcached, consul

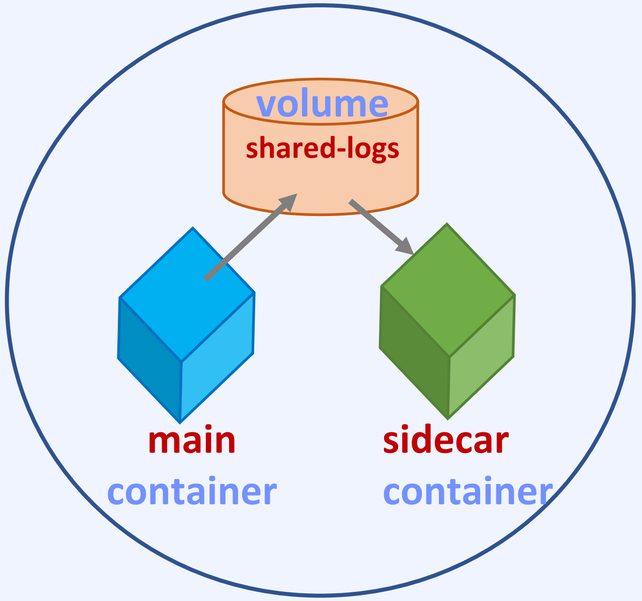
```
$ kubectl run eshop-frontend --image=nginx  --dry-run=client -o
yaml > multi.yaml

$ vi multi.yaml
...

$ kubectl apply -f multi.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-frontend
spec:
  containers:
  - image: nginx
    name: nginx
  - image: redis
    name: redis
  - image: memcached
    name: memcached
  - image: consul
    name: consul
```

# sidecar-container

기본 컨테이너 기능을 확장하기 위해 사용
본래의 컨테이너는 기본 서비스에 충실하고,
추가 기능을 별도의 컨테이너를 이용해 적용

**웹서버는 기본서비스에 충실**하고, **추가로 생성되는 웹서버 로그는 별도의 사이드 카 컨테이너가 수집하여 외부 log aggregator로 전송**하는 형태의 서비스




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  volumes:
  - name: shared-logs
    emptyDir: {}

  containers:
  - name: main
    image: nginx
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log/nginx

  - name: sidecar
    image: busybox
    command: ["sh","-c","while true; do cat /log/access.log /log/error.log; sleep 10; done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /log
```

문제5:
sidecar container Pod 생성하기

## ■ 작업 클러스터 : k8s

현재 운영중인 eshop-cart-app Pod의 로그를 Kubernetes built-in logging 아키텍처(예: kubectl logs)에 통합하는 로그 스트리밍 사이드카 컨테이너를 운영하시오.

- **busybox** 이미지를 사용하여 **price**라는 이름의 사이드카 컨테이너를 기존 eshop-cart-app에 추가합니다.
- 새 **price** 컨테이너는 다음과 같은 command를 실행해야 합니다.
  Command: **/bin/sh, -c, “tail -n+1 -f /var/log/cart-app.log”**
- **/var/log**에 마운트 된 볼륨을 사용하여 사이드카 컨테이너에서 로그 파일 cart-app.log를 사용해야 합니다.
- eshop-cart-app Pod와 cart-app 컨테이너를 수정하지 마시오.

```
$ kubectl get pod eshop-cart-app -o yaml > 3-3.yaml

$ vi 3-3.yaml
...

$ kubectl apply -f multi.yaml
```

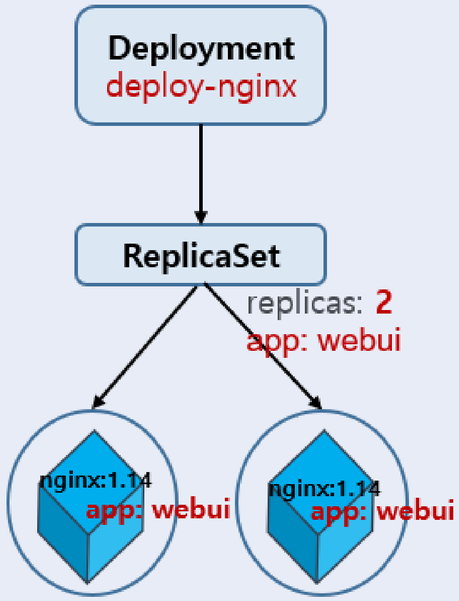
```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-cart-app
spec:
....
  - name: price
    image: busybox
    args: [/bin/sh, -c, "tail -n+1 -f /var/log/cart-app.log"]
    volumeMounts:
    - name: varlog
      mountPath: /var/log
```

# Part3 Workloads & Scheduling

## 03 Deployment

# Deployment

- 쿠버네티스 클러스터에서 애플리케이션을 배포하는 가장 일반적인 방식
- ReplicaSet 컨트롤러를 통해 replica 수 보장 및 scaling 가능
- Rolling update 또는 Roll back 지원




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webui
  template:
    metadata:
      labels:
        app: webui
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.14
```

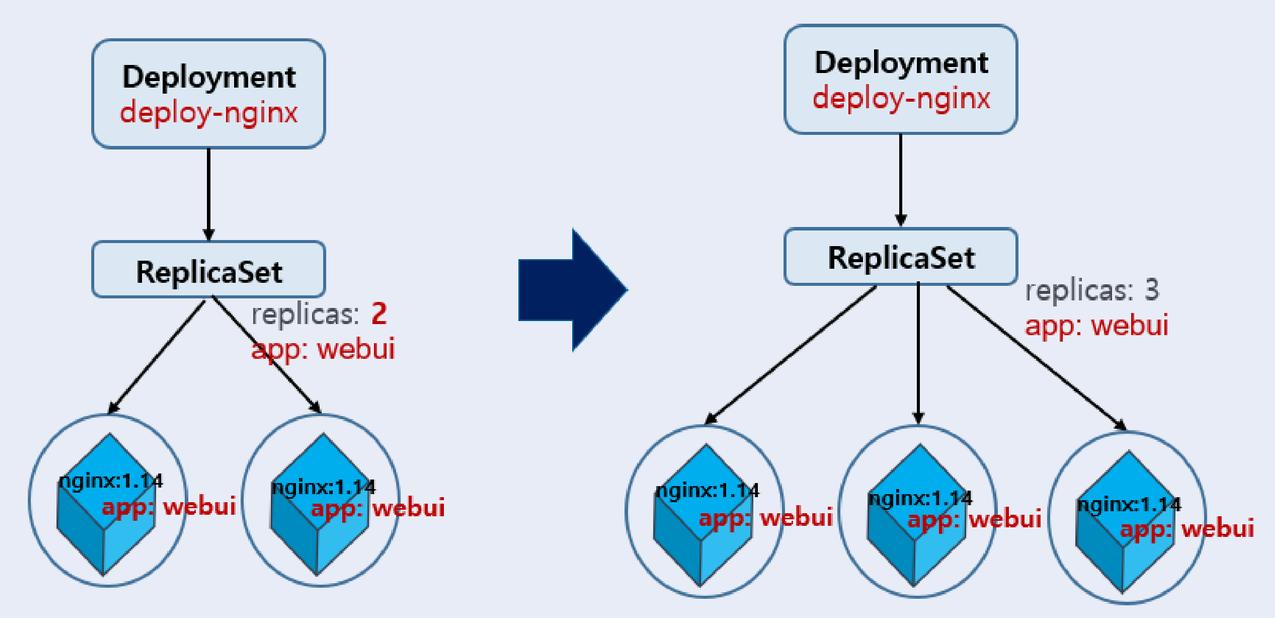
# 애플리케이션 Scaling



- Deployment는 ReplicaSet 컨트롤러를 이용해 Pod scaling을 지원
  kubectl scale deployment <deploy_name>  --replicas=N

  - **Scale-out**: 애플리케이션 Pod수를 확장하여 처리능력을 향상
  - **Scale-in**: 애플리케이션 Pod수를 줄여서 리소스 낭비 최소화

kubectl scale deployment deploy-nginx  --replicas=3



문제6:
Deployment & Scaling

## ■ 작업 클러스터 : k8s

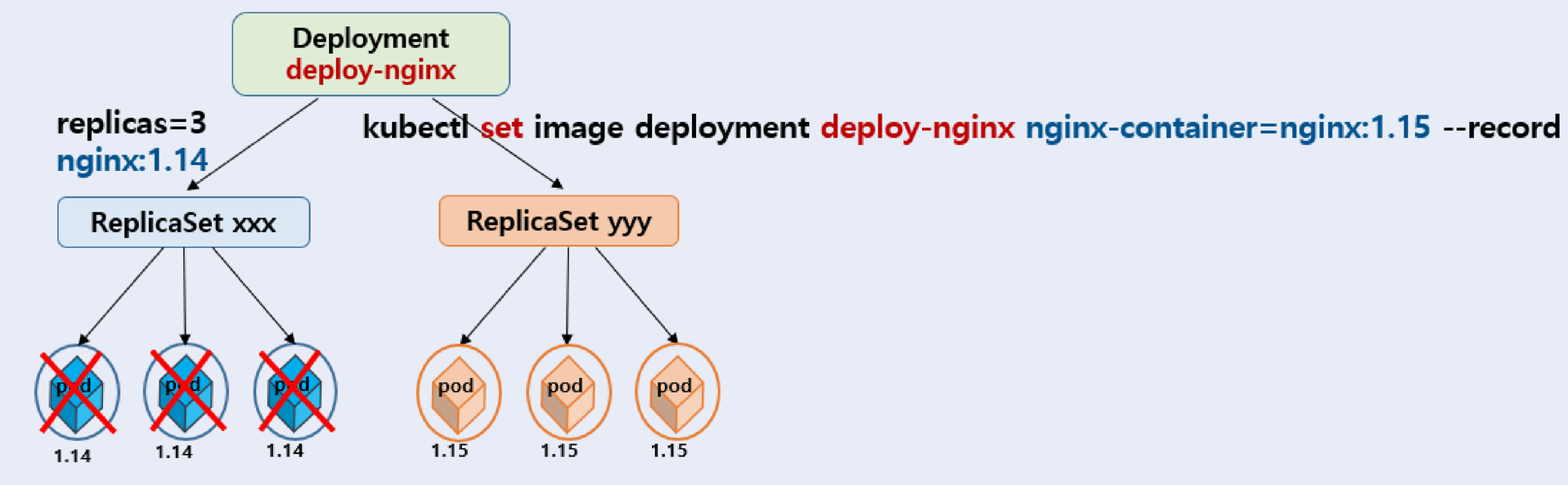
a. webserver 라는 이름으로 deployment를 생성하시오.
- Name: **webserver**
- **2** replicas
- label: app_env_stage=dev
- container name: **webserver**
- container image: **nginx:1.14**

b. 다음, webserver Deployment의 pod 수를 3개로 확장하시오.

```
$ kubectl create deployment webserver --image=nginx:1.14 --
replicas=2 --port=80 --dry-run=client -o yaml > webserver.yaml

$ vi webserver.yaml
...
$ kubectl apply -f webserver.yaml
$ kubectl get deployments.apps webserver
$ kubectl get all

$ kubectl scale deployment webserver --replicas=3
$ kubectl get deployments.apps webserver
$ kubectl get pods --show-lables
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
spec:
  replicas: 2
  selector:
    matchLabels:
      app_env_stage: dev
  template:
    metadata:
      labels:
        app_env_stage: dev
    spec:
      containers:
      - image: nginx:1.14
        name: webserver
        ports:
        - containerPort: 80
```

- Rolling Update : 동작중인 애플리케이션의 **서비스 중단 없이 점진적으로 Pod 업데이트**
  kubectl set image deployment <deploy_name> <container_name>=<new_version_image> --record



- Rollback: 동작중인 애플리케이션 서비스 중단 없이 **이전 버전으로 되돌리기**
  kubectl rollout **history** deployment <deploy_name>

  kubectl rollout **undo** deployment <deploy_name>
  kubectl rollout **undo** deployment <deploy_name> --to-revision=NUMBER

문제7:
Rolling update & Roll back

## ■ 작업 클러스터 : k8s

Deployment를 이용해 nginx 파드를 3개 배포한 다음 컨테이너 이미지 버전을 rolling update하고 update record를 기록합니다. 마지막으로 컨테이너 이미지를 previous version으로 roll back 합니다.

- name: eshop-payment
- Image : nginx
- Image version: 1.16
- update image version: 1.17
- label: app=payment, environment=production

```
$ kubectl create deployment eshop-payment --image=nginx:1.16 --
dry-run=client -o yaml > 3-4.yaml

$ vi 3-4.yaml
$ kubectl apply -f 3-4.yaml --record

# rolling update
$ kubectl set image deployment web nginx=nginx:1.17 --record
kubectl rollout history deployment web

#Roll back to previous version
kubectl rollout undo deployment web
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eshop-payment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: payment
      environment: production
  template:
    metadata:
      labels:
        app: payment
        environment: production
    spec:
      containers:
      - image: nginx:1.16
        name: nginx
```

# 02 Workloads & Scheduling

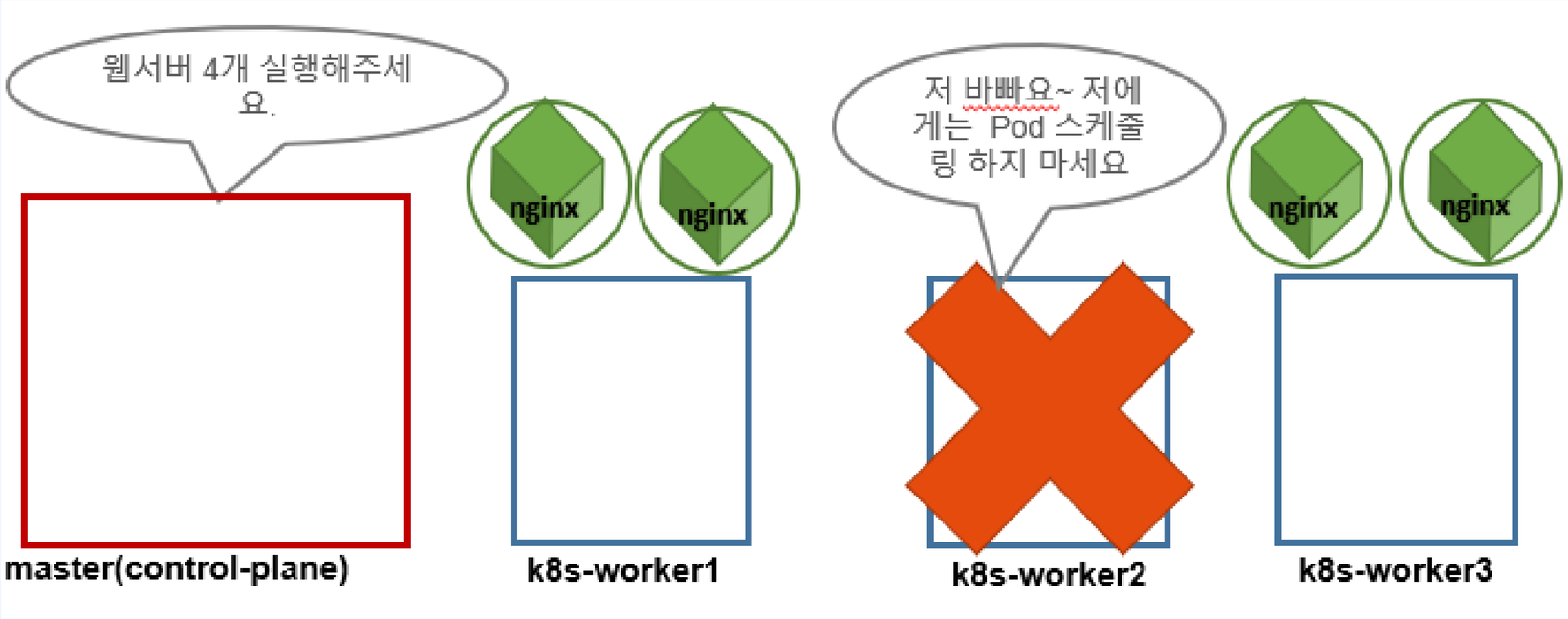
## 04 node 관리

# Node 스케줄링
# 중단 및 허용



- 컨테이너를 포함한 Pod는 node에서 실행됨
- node는 Control-plane에 의해 관리

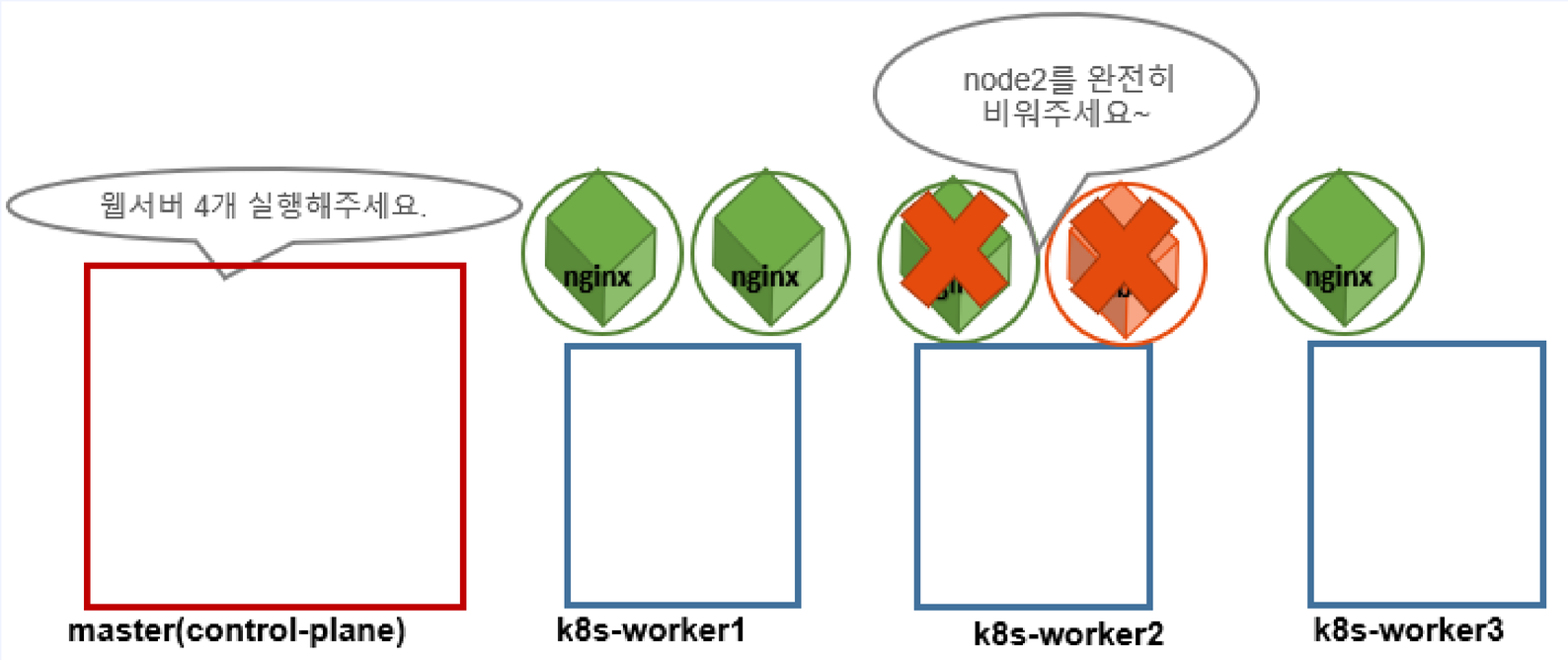
- 특정 node의 스케줄링 중단(cordon) 및 허용(uncordon)
  kubectl cordon <NODE_NAME>
  kubectl uncordon <NODE_NAME>



# Node 비우기



- 노드 비우기(drain)
  특정 node 에서 실행중인 pod 비우기(drain) 및 제거(delete)
  **kubectl drain <NODE_NAME> --ignore-daemonsets –force**
  --ignore-daemonsets  DaemonSet-managed pod들은 ignore.
  --force=false  RC, RS, Job, DaemonSet 또는 StatefulSet에서 관리하지 않는 Pod까지 제거.



# 문제8:
# 노드 비우기

■ 작업 클러스터 : k8s

k8s-worker2 노드를 스케줄링 불가능하게 설정하고, 해당 노드에서 실행 중인 모든 Pod을 다른 node로 reschedule 하세요.

```
$ kubectl drain k8s-worker2 --ignore-daemonsets  --force

$ kubectl get nodes

$ kubectl get pods
```

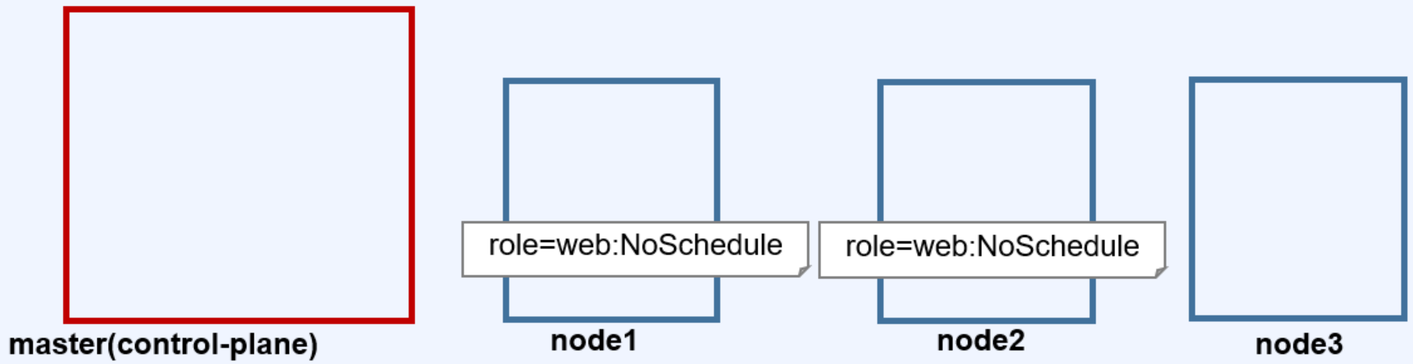
# 02 Workloads & Scheduling

## 05 node 정보보기

# Node Taint & Toleration



- node taint, Pod toleration
- worker node에 taint가 설정된 경우 동일 값의 toleration이 있는 Pod만 배치된다.
- toleration이 있는 Pod는 동일한 taint가 있는 node를 포함하여 모든 node에 배치된다.
- effect 필드 종류
  - NoSchedule : toleration이 맞지 않으면 배치 되지 않는다.
  - PreferNoSchedule : toleration이 맞지 않으면 배치되지 않으나, 클러스터 리스소가 부족하면 할당된다.
  - NoExecute : toleration이 맞으면 동작중인 pod를 종료



문제9:
Ready 노드 확인하기

■ 작업 클러스터 : hk8s

Ready 상태(NoSchedule로 taint된 node는 제외)인 node를 찾아 그 수를 /var/CKA2022/notaint_ready_node에 기록하세요..

```
NoSchedule로 taint된 node check
$ kubectl get nodes
$ kubectl get node hk8s-m -o yaml | grep -i noschedule
$ kubectl get node hk8s-w1 -o yaml |  grep -i noschedule
$ kubectl get node hk8s-w2 -o yaml |  grep -i noschedule
$ kubectl get nodes | grep -i ready
$ echo "2" > / var/CKA2022/notaint_ready_node
```

# 02 Workloads & Scheduling

## 06 pod scheduling

# Node Selector

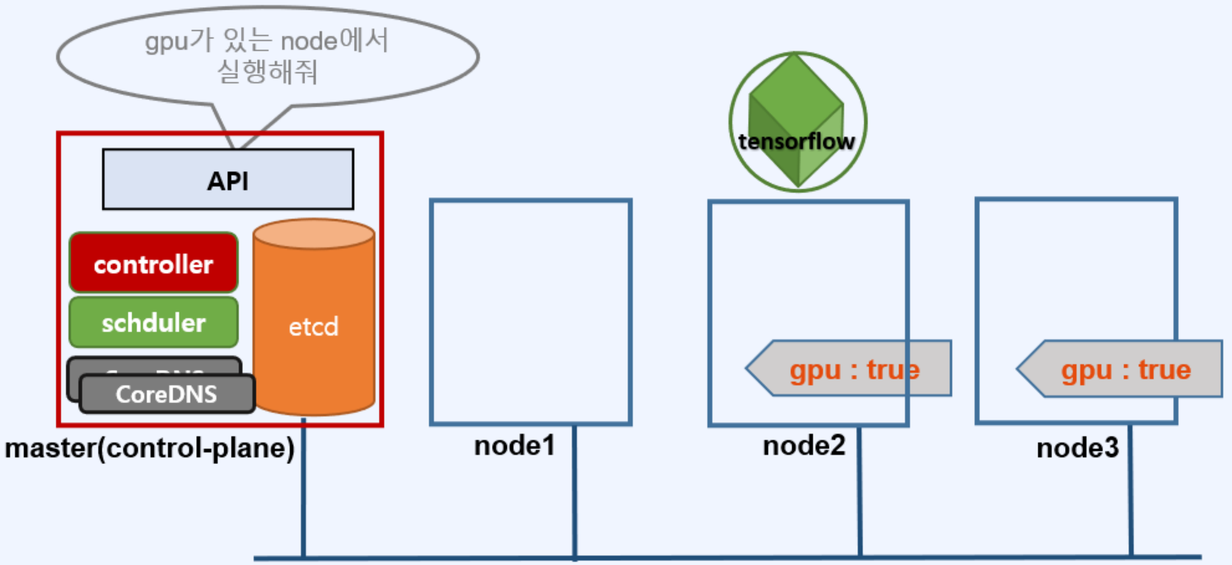
- Worker node에 할당된 label을 이용해 node를 선택
- node Label 설정

  kubectl label nodes <노드 이름> <레이블 키>=<레이블 값>

  kubectl label nodes node1.example.com gpu=true

  kubectl get  nodes -L gpu

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow
spec:
  containers:
  - name: tensorflow
    image: tensorflow/tensorflow:nightly-jupyter
    ports:
    - containerPort: 8888
      protocol: TCP
  nodeSelector:
    gpu: "true"
```



# 문제10:
Pod Scheduling

■ 작업 클러스터 : k8s
다음의 조건으로 pod를 생성하세요.
Name: eshop-store
Image: nginx
Node selector: disktype=ssd

Node Label 확인
$ kubectl get nodes –L disktype

$ kubectl run eshop-store --image=nginx --dry-run=client -o yaml > 10.yaml

$ vi 10.yaml
-- nodeSelector 추가 –

$ kubectl apply -f 10.yaml
$ kubectl get pods eshop-store -o wide

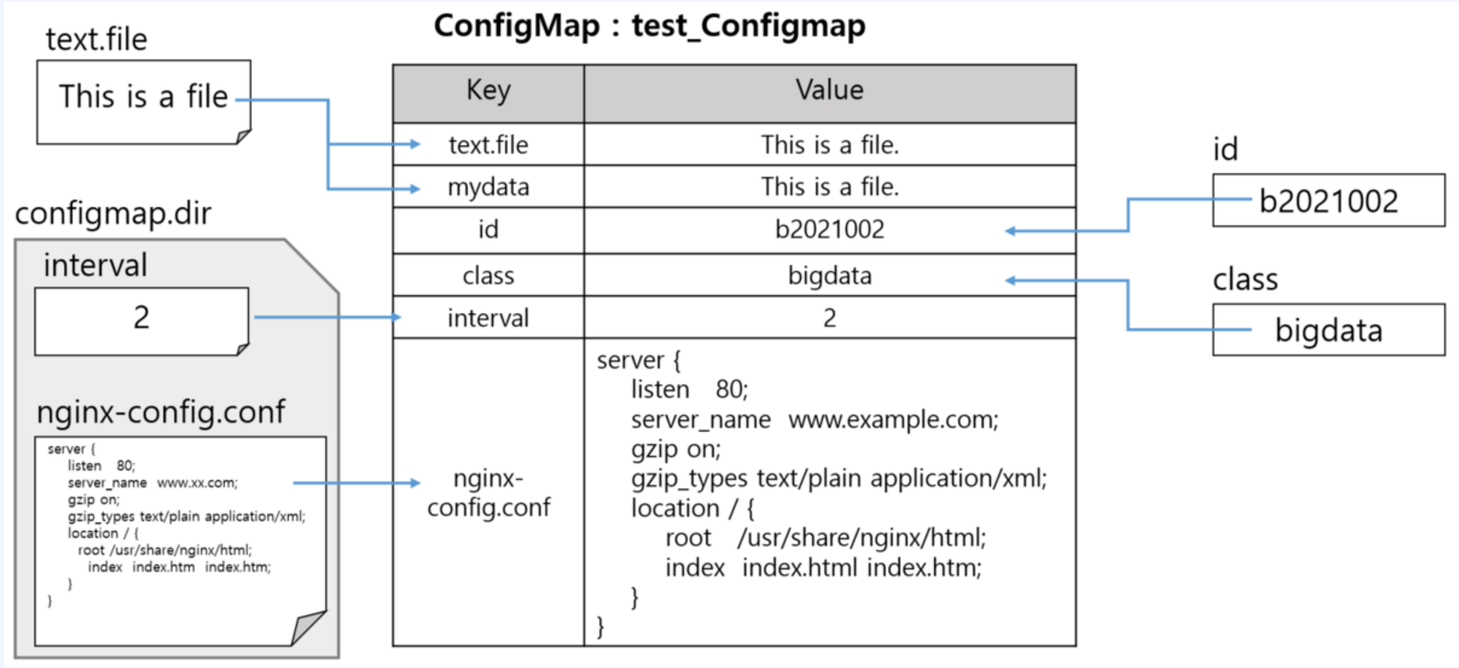
```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-store
spec:
  nodeSelector:
    disktype: ssd
  containers:
  - image: nginx
    name: eshop-store
```

# 02 Workloads & Scheduling

## 07 ConfigMap

- 컨테이너 구성 정보를 한곳에 모아서 관리

**kubectl create configmap NAME [--from-file=source] [--from-literal=key1=value1]**




```
$ kubectl create configmap CONFIG_NAME --from-literal=id=b2021002  --from-literal=class=bigdata
$ kubectl create configmap CONFIG_NAME --from-file=text.file
$ kubectl create configmap CONFIG_NAME --from-file=mydata=text.file
$ kubectl create configmap CONFIG_NAME  --from-file=/configmap.dir/
```

# 문제11:
# ConfigMap으로 환경변수 전달

## ■ 작업 클러스터 : k8s

다음의 변수를 configMap eshop으로 등록하세요.

- DBNAME: mysql
- USER: admin

등록한 eshop configMap의 DBNAME을  eshop-configmap라는 이름의 nginx 컨테이너에 DB라는 환경변수로 할당하세요.

```
$ kubectl create configmap eshop --from-literal=DBNAME=mysql --from-literal=USER=admin

$ kubectl get configmaps eshop
$ kubectl describe configmaps eshop

$  kubectl run eshop-configmap --image=nginx --dry-run=client -o yaml > eshop.yaml
$ vi eshop.yaml

$ kubectl apply -f eshop.yaml
$ kubectl exec -it eshop-configmap -- /bin/bash
root@eshop-configmap:/# echo $DB
root@eshop-configmap:/# exit
```

```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-configmap
spec:
  containers:
  - image: nginx
    name: eshop-configmap
    env:
    - name: DB
      valueFrom:
        configMapKeyRef:
          name: eshop
          key: DBNAME
```

# 02 Workloads & Scheduling

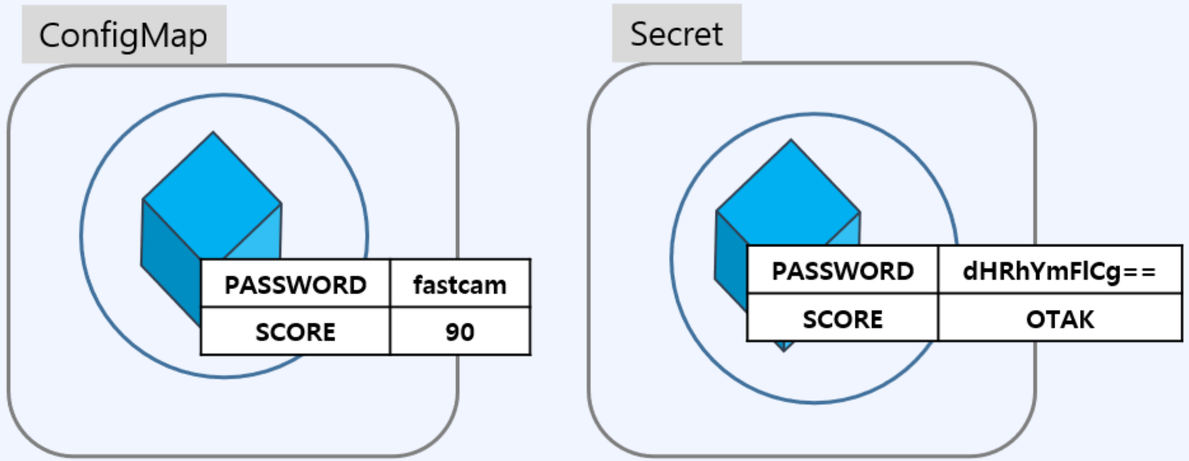
## 08 Secret

# Kubernetes Secret

- ConfigMap : 컨테이너 구성 정보를 한곳에 모아서 관리
- Secret : 컨테이너가 사용하는 password, auth token, ssh key와 같은 중요한 정보를 저장하고 민감한 구성정보를 base64로 인코딩해서 한 곳에 모아서 관리
- 민감하지 않은 일반 설정파일 configMap을 사용하고 민감한 데이터는 secret을 사용

```
$ kubectl create secret generic eshop-secret \
        --from-literal=PASSWORD=fastcam \
        --from-literal=SCORE=90
```

| Key | Value |
|---|---|
| PASSWORD | fastcam |
| SCORE | 90 |



문제12:
Create a Kubernetes secretReady

■ 작업 클러스터 : k8s

Name: super-secret
password: bob

Create a pod named pod-secrets-via-file, using the redis Image, which mounts a secret named super-secret at/secrets.
Create a second pod named pod-secrets-via-env, using the redis Image, which exports password as CONFIDENTIAL