애플리케이션 스케줄링과 라이프사이클 관리

- 애플리케이션 환경 변수 관리
- 초기 명령어 및 아규먼트 전달과 실행
- 한 포드에 멀티 컨테이너
- Init 컨테이너
- 시스템 리소스 요구사항과 제한 설정
- 데몬셋
- 포드 매뉴얼 스케줄링
- 스태틱 포드
- 멀티플 스케줄러
- 쿠버네티스 스케줄러 설정
- 애플리케이션 롤링 업데이트와 롤백

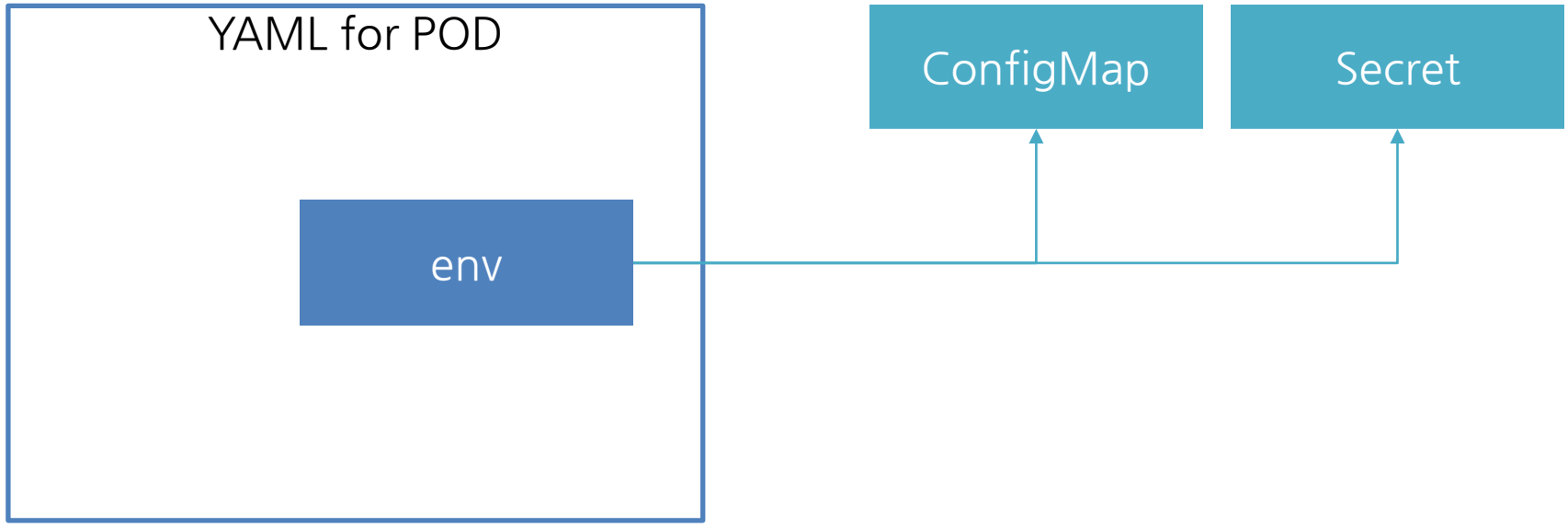
애플리케이션 변수 관리

환경 변수 설정, 컨피그맵 설정, 시크릿 설정

애플리케이션 변수 관리

도커 컨테이너의 환경 설정

- 쿠버네티스의 환경 변수는 YAML 파일이나 다른 리소스로 전달
- 하드 코딩된 환경 변수는 여러 환경에 데이터를 정의하거나 유지, 관리가 어려움
- ConfigMap은 외부에 컨테이너 설정을 저장할 수 있음



애플리케이션 변수 관리

도커 컨테이너의 환경 설정

```
env:
  - name: DEMO_GREETING
    value: "Hello from the environment"
```

일반적인 키와 값

```
env:
  - name: DEMO_GREETING
    valueFrom:
      configMapKeyRef: confimap-name
```

ConfigMap

```
env:
  - name: DEMO_GREETING
    valueFrom:
      secretKeyRef: secret-name
```

Secrets

애플리케이션 변수 관리

환경변수를 포드에 저장하는 방법

node-env.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

```
$ kubectl exec -it envar-demo -- /bin/bash
root@envar-demo:/# printenv
NODE_VERSION=4.4.2
HOSTNAME=envar-demo
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT=tcp://10.12.0.1:443
TERM=xterm
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.12.0.1
NPM_CONFIG_LOGLEVEL=info
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/u
PWD=/
DEMO_FAREWELL=Such a sweet sorrow
SHLVL=1
HOME=/root
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_ADDR=10.12.0.1
KUBERNETES_PORT_443_TCP=tcp://10.12.0.1:443
DEMO_GREETING=Hello from the environment
_=/usr/bin/printenv
```

애플리케이션 변수 관리

환경변수를 ConfigMap에 저장하는 방법

- kubectl 명령어를 사용하여 바로 저장할 수 있음

```
$ kubectl create configmap <map-name> <data-source>
```

- test 파일을 생성하고 그 파일을 환경 변수로 저장

```
$ echo  -n 1234 > test
$ kubectl create configmap map-name --from-file=test
configmap/map-name created
```

- 저장한 내용 확인

```
$ kubectl get configmap map-name -o yaml
apiVersion: v1
data:
  test: |
    1234
kind: ConfigMap
[중략]
```

애플리케이션 변수 관리

YAML을 사용하여 환경변수를 ConfigMap에 저장하는 방법

node-env.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        # Define the environment variable
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to assign to SPECIAL_LEVEL_KEY
              name: special-config
              # Specify the key associated with the value
              key: special.how
  restartPolicy: Never
```

configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config
  namespace: default
data:
  log_level: INFO
```

애플리케이션 변수 관리

환경변수를 Secret에 저장하는 방법

- 비밀번호, OAuth 토큰 및 ssh 키와 같은 민감한 정보
- kubectl 명령어로 secret 설정하는 방법

```
$ echo -n 'admin' > ./username
$ echo -n '1f2d1e2e67df' > ./password
$ kubectl create secret generic db-user-pass --from-file=./username --from-
file=./password
secret/db-user-pass created
```

- echo -n 명령어를 실행하면 데이터가 개행문자(엔터)와 함께 출력됨
- 마지막 명령어를 실행하면 db-user-pass 유저가 생성됨
- secret 에는 DB의 user와 password와 같이 민감한 파일들이 base64 인코딩돼서 저장
- 이 값이 안전한 것은 아니지만 공격자에게 혼란을 줄 수는 있음

```
$ kubectl get secret db-user-pass -o yaml
apiVersion: v1
data:
  password: MWYyZDFlMmU2N2Rm
  username: YWRtaW4=
kind: Secret
…
```

애플리케이션 변수 관리

YAML 파일로 환경변수를 Secret에 저장하는 방법

- 수동으로 데이터를 만들어야 하는 경우에는 base64 인코딩을 수동으로 해줘야 한다.

```
$ echo -n 'admin' | base64
YWRtaW4=

$ echo -n '1f2d1e2e67df' | base64
MWYyZDFlMmU2N2Rm
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

애플리케이션 변수 관리

연습문제

- Kube-system에 존재하는 secret의 개수는 몇 개인가?
- default-token의 개수는 몇 개인가?
- default-token의 타입은 무엇인가?
- Secret 데이터에는 어떤 secret data가 포함돼 있는가?
- 다음과 같이 Mysql 서버를 지원하는 secret-mysql.yaml을 생성하자.
  - `Secret Name: db-secret`
  - `Secret Data 1: DB_Password=Passw0rd!0`

- Mysql 이미지를 하나 생성하고 앞서 만든 secret을 환경 변수 이름과 연결하자.
  - Image: mysql:5.6
  - Port 번호: 3306
  - 환경 변수 name: MYSQL_ROOT_PASSWORD

  - 잘 적용됐는지 확인할 수 있는 명령어:
    - ✓ kubectl exec -it mysql -- mysql -u root -p
    - ✓ password: `Passw0rd!0`

초기 명령어 및 아규먼트 전달과 실행

초기 명령어 및 아규먼트 전달과 실행

초기 실행시 명령어와 아규먼트를 전달

- Pod을 생성할 때 spec.containers.command와 args에 실행하기 원하는 인자를 전달하면 컨테이너가 부팅된 뒤 실행

pod-command-args.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

초기 명령어 및 아규먼트 전달과 실행

초기 실행시 명령어와 아규먼트를 전달

- 환경 변수를 활용하여 출력할 때는 $를 사용하여 명령 내용 변경 가능

```
env:
- name: MESSAGE
  value: "hello world"
command: ["/bin/echo"]
args: ["$(MESSAGE)"]
```

초기 명령어 및 아규먼트 전달과 실행

연습문제

- busybox 이미지를 사용하는 busybox 포드를 만들어라.
- busybox 포드가 유지되는가? 그렇지 않다면 그 이유는 무엇인가?
- busybox를 장시간 유지하기 위해 장시간 sleep하는 명령어와 아규먼트를 추가하여 실행하라.
- busybox가 계속 유지될 수 있는가? 셸을 접속하여 확인하자.

한 포드에 멀티 컨테이너

한 포드에 멀티 컨테이너

하나의 포드에 다수의 컨테이너를 사용

- 하나의 포드를 사용하는 경우 같은 네트워크 인터페이스와 IPC, 볼륨 등을 공유
- 이 포드는 효율적으로 통신하여 데이터의 지역성을 보장하고 여러 개의 응용프로그램이 결합된 형태로 하나의 포드를 구성할 수 있음

```
$ kubectl exec -it two-containers -- cat /usr/share/nginx/html/index.html
Defaulting container name to nginx-container.
Use 'kubectl describe pod/two-containers -n default' to see all of the containers in this pod.
Hello from the debian container
```

pod-mutil-continaer.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:

  restartPolicy: Never

  volumes:
  - name: shared-data
    emptyDir: {}

  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"]
```

한 포드에 멀티 컨테이너

연습문제

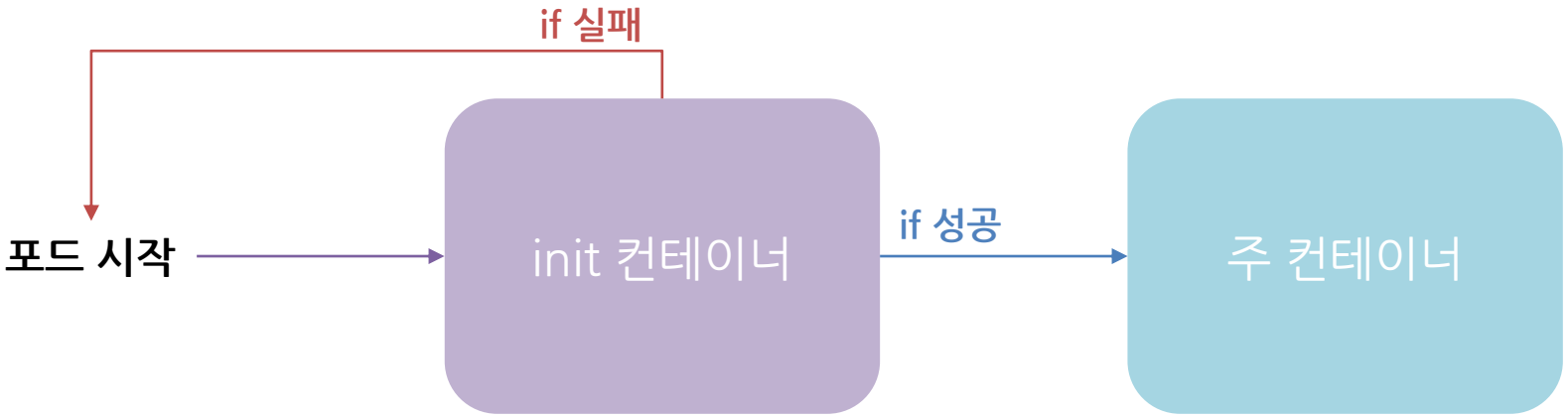
- 하나의 포드에서 nginx와 redis 이미지를 모두 실행하는 yaml을 만들고 실행하라.

Init 컨테이너

Init 컨테이너

init 컨테이너의 특징

- 포드 컨테이너 실행 전에 초기화 역할을 하는 컨테이너
- 완전히 초기화가 진행된 다음에야 주 컨테이너를 실행
- Init 컨테이너가 실패하면, 성공할때까지 포드를 반복해서 재시작
- restartPolicy에 Never를 하면 재시작하지 않음



Init 컨테이너

init 컨테이너의 특징

- 이 yaml은 mydb와 myservice가 탐지될 때까지 init 컨테이너가 멈추지 않고 돌아감

pod-init-container.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myse
rvice; sleep 2; done;']
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sle
ep 2; done;']
```

Init 컨테이너

init 컨테이너의 특징

- init 프로세스를 끝낼 수 있는 종결자 등장!

svc-pod-mydb.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
---
apiVersion: v1
kind: Service
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

Init 컨테이너

함께하기

- pod-init-container.yaml를 작성하여 리소스를 생성하라.
- my-app-pod 포드에서 주 컨테이너가 실행되지 않는 현상을 관찰하라.
- 주 컨테이너가 실행되지 않는 이유는 무엇인가?
- svc-pod-mydb.yaml을 작성 및 실행하고 주 컨테이너의 반응을 관찰하라.
- 주 컨테이너가 정상적으로 실행되었는가? 그렇다면 그 이유는 무엇인가?

시스템 리소스 요구사항과 제한 설정

시스템 리소스 요구사항과 제한 설정

컨테이너에서 리소스 요구사항

- CPU와 메모리는 집합적으로 컴퓨팅 리소스 또는 리소스로 부름
- CPU 및 메모리 는 각각 자원 유형을 지니며 자원 유형에는 기본 단위를 사용

- 리소스 요청 설정 방법
  - spec.containers[].resources.requests.cpu
  - spec.containers[].resources.requests.memory

- 리소스 제한 설정 방법
  - spec.containers[].resources.limits.cpu
  - spec.containers[].resources.limits.memory

시스템 리소스 요구사항과 제한 설정

컨테이너에서 리소스 요구사항

- CPU는 코어 단위로 지정되며 메모리는 바이트 단위로 지정

자원 유형	단위
CPU	m(millicpu),
Memory	…… Ti, Gi, Mi, Ki, T, G, M, K

※ CPU 0.5가 있는 컨테이너 는 CPU 1 개를 요구하는 절반의 CPU
※ CPU 0.1은 100m과 동일한 기능
※ K, M, G의 단위는 1000씩 증가
※ Ki, Mi, Gi의 단위는 1024씩 증가

- 환경에 따른 CPU의 의미
  - 1 AWS vCPU
  - 1 GCP 코어
  - 1 Azure vCore
  - 1 IBM vCPU
  - 1 하이퍼 스레딩 기능이 있는 베어 메탈 인텔 프로세서의 하이퍼 스레드

시스템 리소스 요구사항과 제한 설정

- 컨테이너에서 리소스 요구사항 yaml 작성 요령

pod-resource-frontend.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

각각의 컨테이너마다
리소스 작성

시스템 리소스 요구사항과 제한 설정

연습문제

- **다음 요구사항에 맞는 deploy를 구현하라.**
  - Deploy name: nginx
  - Image: nginx
  - 최소 요구사항
    - ✓ CPU: 1m
    - ✓ 메모리: 200Mi
  - 리소스 제한
    - ✓ CPU: 2m
    - ✓ 메모리: 400Mi
  - Port: 80

시스템 리소스 요구사항과 제한 설정

limitRanges

- https://kubernetes.io/docs/concepts/policy/limit-range/
- 네임 스페이스에서 포드 또는 컨테이너별로 리소스를 제한하는 정책

- 리미트 레인지의 기능
  - 네임 스페이스에서 포드나 컨테이너당 최소 및 최대 컴퓨팅 리소스 사용량 제한
  - 네임 스페이스에서 PersistentVolumeClaim 당 최소 및 최대 스토리지 사용량 제한
  - 네임 스페이스에서 리소스에 대한 요청과 제한 사이의 비율 적용
  - 네임 스페이스에서 컴퓨팅 리소스에 대한 디폴트 requests/limit를 설정하고 런타임 중인 컨테이너에 자동으로 입력

- LimitRange 적용 방법
  - Apiserver 옵션에 --enable-admission-plugins=LimitRange를 설정

시스템 리소스 요구사항과 제한 설정

limitRanges

- **컨테이너 수준의 리소스 제한**
  - 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

- **예제 해석**
  - 각 컨테이너에 설정

기준	CPU	메모리
최대	800m	1Gi
최소	100m	99Mi
기본 제한	700m	900Mi
기본 요구사항	110m	111Mi

- **리소스 조회**
  - kubectl describe limitrange -n 네임스페이스

limit-mem-cpu-per-container.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-container
spec:
  limits:
  - max:
      cpu: "800m"
      memory: "1Gi"
    min:
      cpu: "100m"
      memory: "99Mi"
    default:
      cpu: "700m"
      memory: "900Mi"
    defaultRequest:
      cpu: "110m"
      memory: "111Mi"
    type: Container
```

시스템 리소스 요구사항과 제한 설정

limitRanges

- **포드 수준의 리소스 제한**
  - 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

- **예제 해석**
  - 각 포드에 설정

기준	CPU	메모리
최대	2	2Gi
최소	-	-
기본	-	-
최소 요구사항	-	-

limit-mem-cpu-per-pod.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-pod
spec:
  limits:
  - max:
      cpu: "2"
      memory: "2Gi"
    type: Pod
```

- **리소스 조회**
  - kubectl describe limitrange -n 네임스페이스

시스템 리소스 요구사항과 제한 설정

limitRanges

- **스토리지 리소스 제한**
  - 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

- **예제 해석**
  - 각 PVC에 설정

기준	용량
최대	2Gi
최소	1Gi

storagelimits.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimits
spec:
  limits:
  - type: PersistentVolumeClaim
    max:
      storage: 2Gi
    min:
      storage: 1Gi
```

- **리소스 조회**
  - kubectl describe limitrange -n 네임스페이스

시스템 리소스 요구사항과 제한 설정

ResourceQuata

- https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/
- **네임스페이스별 리소스 제한**
  - 제한하기 원하는 네임스페이스에 ResourceQuata 리소스 생성
  - 모든 컨테이너에는 CPU, 메모리에 대한 최소요구사항 및 제한 설정이 필요

mem-cpu-demo.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

- **예제 해석**
  - 네임스페이스 내의 모든 컨테이너의 합이 다음을 넘어서는 안됨

기준	CPU	메모리
최대	2	2Gi
최소 요구사항	1	1Gi

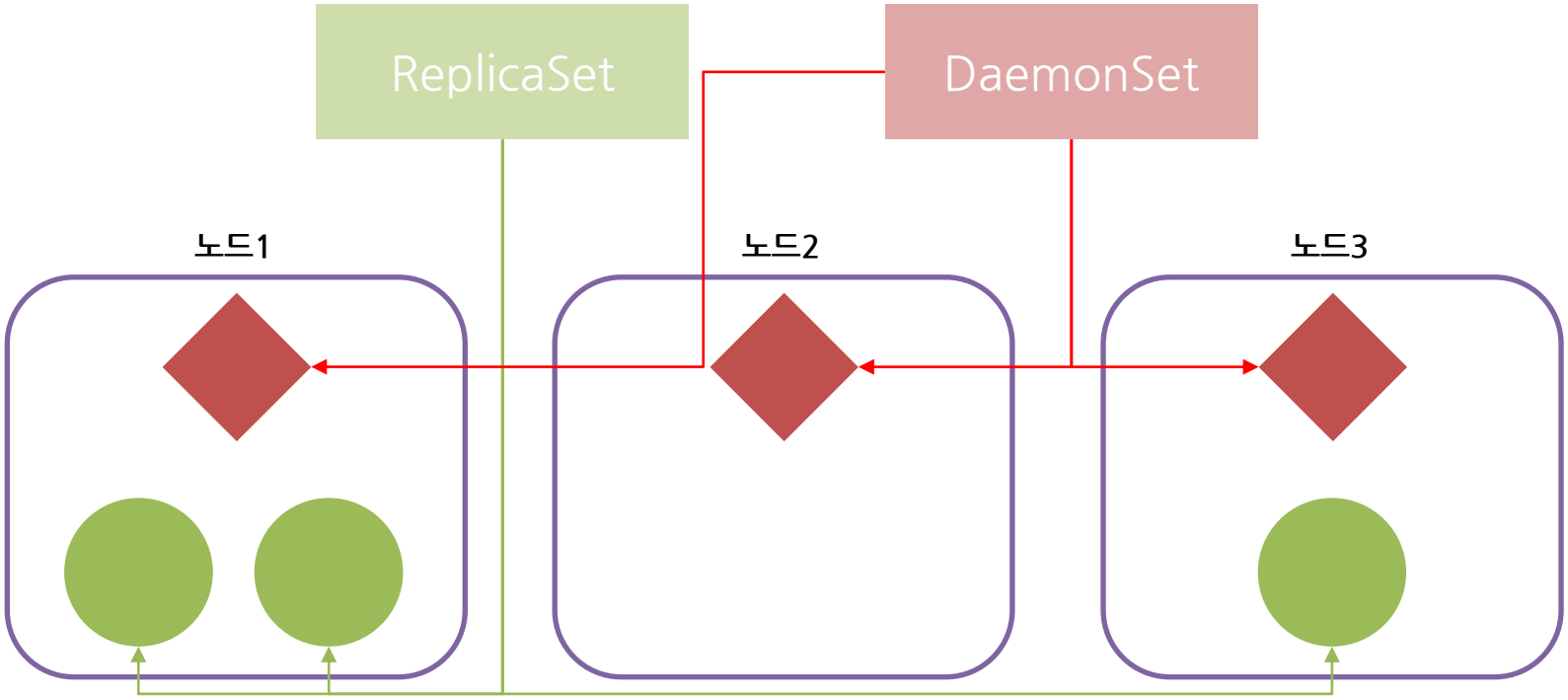
- **리소스 조회**
  - kubectl describe resourcequota -n 네임스페이스

데몬셋

데몬셋, 노드당 포드 하나씩!

데몬셋

- 레플리케이션컨트롤러와 레플리카셋은 무작위 노드에 포드를 생성
- 데몬셋은 각 하나의 노드에 하나의 포드만을 구성
- kube-proxy가 데몬셋으로 만든 쿠버네티스에서 기본적으로 활동중인 포드



데몬셋, 노드당 포드 하나씩!

데몬셋 예제

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

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: gcr.io/fluentd-elasticsearch/fluentd:v2.5.1
        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
```

데몬셋, 노드당 포드 하나씩!

연습문제

- 데몬셋으로 각 노드에 http-go 배치하기
- 이미지: gasbugs/http-go

스태틱 포드

스태틱 포드

스태틱 포드의 필요성

- 스태틱 포드: kubelet이 직접 실행하는 포드
- 각각의 노드에서 kubelet에 의해 실행
- 포드들을 삭제할때 apiserver를 통해서 실행되지 않은 스태택 포드(static pod)는 삭제 불가
- 즉, 노드의 필요에 의해 사용하고자 하는 포드는 스태틱 포드로 세팅
- 다음 명령어 들을 사용하여 실행하고자 하는 static pod의 위치를 설정 가능

```
# sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since Sun 2019-07-07 12:00:33 KST; 13min ago
     Docs: https://kubernetes.io/docs/home/
 Main PID: 27987 (kubelet)
    Tasks: 18 (limit: 4680)
   CGroup: /system.slice/kubelet.service
           └─27987 /usr/bin/kubelet --bootstrap-kube config=/etc/kubernetes/bootstrap-
kubelet.conf --k
```

```
# vim /lib/systemd/system/kubelet.service
# systemctl daemon-reload
# systemctl restart kubelet
```

ExecStart= 부분에 --pod-manifest-path=/etc/kubelet.d/ 추가

스태틱 포드

스태틱 포드의 기본 경로

- 기본 경로는 /etc/kubernetes/manifests
- 이미 쿠버네티스 시스템에서는 필요한 기능을 위해 다음과 같은 스태틱 포드를 사용
- 각각의 컴포넌트의 세부 사항을 설정할 때는 여기 있는 파일들을 수정하면 자동으로 업데이트돼 포드를 재구성
- 포드의 작성 요령은 기존의 포드와 동일

```
$ ls /etc/kubernetes/manifests/
etcd.yaml  kube-apiserver.yaml  kube-controller-manager.yaml  kube-scheduler.yaml
```

스태틱 포드

간단한 스태틱 포드의 작성

- 일반적인 포드와 동일하게 작성
- 작성된 파일은 반드시 해당 경로에 위치
- 실행을 위해 별도의 명령은 필요하지 않음
- 작성 후 바로 get pod를 사용하여 확인

```yaml
# path: /etc/kubenetes/manifests/static-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

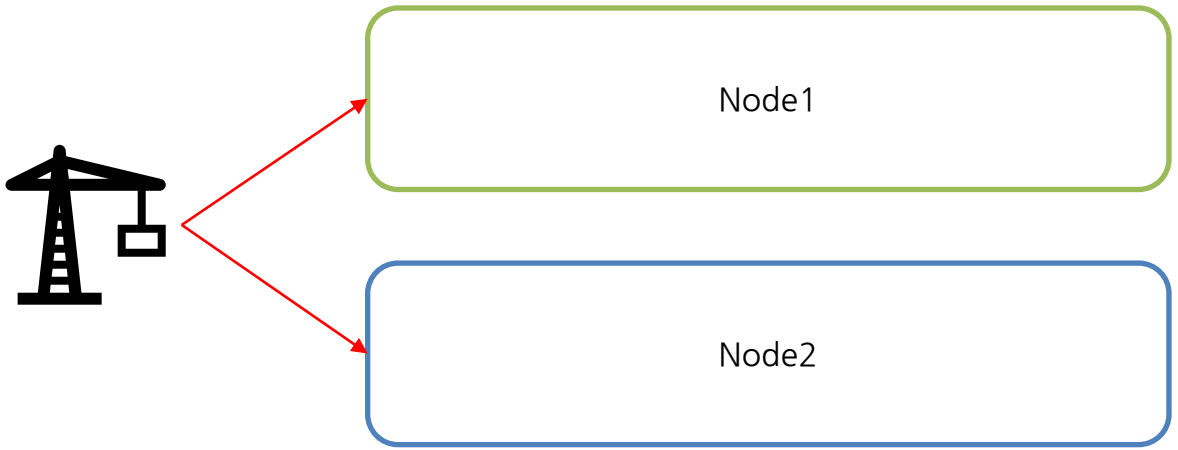
```
$ k get pod
NAME                READY     STATUS    RESTARTS   AGE
static-web-master   1/1       Running   0          14s
```

수동 스케줄링

수동 스케줄링: 원하는 포드를 원하는 노드에

포드 매뉴얼 스케줄링

- 특수한 환경의 경우 특정 노드에서 실행되도록 선호하도록 포드를 제한
- 일반적으로 스케줄러는 자동으로 합리적인 배치를 수행하므로 이러한 제한은 필요하지 않음
- 더 많은 제어가 필요할 수 있는 몇 가지 케이스
  - SSD가 있는 노드에서 포드가 실행하기 위한 경우
  - 블록체인이나 딥러닝 시스템을 위해 GPU 서비스가 필요한 경우
  - 서비스의 성능을 극대화하기 위해 하나의 노드에 필요한 포드를 모두 배치



수동 스케줄링: 원하는 포드를 원하는 노드에

- nodeName 필드를 사용한 매뉴얼 스케줄링
  - 포드를 강제로 원하는 node를 스케줄링 할 수 있음
  - spec 아래에 nodeName: work1과 같이 노드 이름 설정

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: http-go
spec:
  containers:
  - name: http-go
    image: gasbugs/http-go
  nodeName: work1
```

수동 스케줄링: 원하는 포드를 원하는 노드에

노드 셀렉터를 활용한 스케줄링

- 특정 하드웨어를 가진 노드에서 포드를 실행하고자 하는 경우에 사용
- GPU, SSD 등의 이슈를 가진 사항을 적용

- 다음 명령어로 gpu=true를 적용

```
$ kubectl label node <node_name> gpu=true

$ kubectl get node
NAME                                                  STATUS   ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-b1e2cd6b-3r3q     Ready    <none>   66m   v1.12.8-gke.6
gke-standard-cluster-1-default-pool-b1e2cd6b-cb83     Ready    <none>   66m   v1.12.8-gke.6
gke-standard-cluster-1-default-pool-b1e2cd6b-z2nd     Ready    gpu      66m   v1.12.8-gke.6
```

수동 스케줄링: 원하는 포드를 원하는 노드에

레이블링을 활용한 스케줄링

- 다음 http-go-gpu.yaml을 작성하고 포드 생성

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: http-go
spec:
  containers:
  - name: http-go
    image: gasbugs/http-go
  nodeSelector:
    gpu: "true"
```

수동 스케줄링: 원하는 포드를 원하는 노드에

연습문제

- 노드 중 하나를 ssd용 레이블을 갖도록 설정하라.
- ssd 레이블을 가진 노드에만 포드를 할당하는 디플로이먼트를 생성하라.
- Image는 nginx를 사용한다.

멀티플 스케줄러

멀티플 스케줄러

멀티 스케줄러의 필요성

- 기본 스케줄러가 사용자의 필요에 맞지 않으면 사용자 고유의 스케줄러를 구현 가능
- 기본 스케줄러와 함께 여러 스케줄러를 동시에 실행 가능
- 각 포드에 사용할 스케줄러를 지정하는 방식도 가능

- 스케줄러의 사용 옵션
  - https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/

- 스케줄러를 실행하는 yaml 파일이 너무 크니 복붙으로 해결
  - https://Kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
  - 이미지의 경우 에러가 발생
  - 다음 이미지를 사용
  - k8s.gcr.io/kube-scheduler-amd64:v1.11.3

```yaml
# my-scheduler.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-scheduler
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-scheduler-as-kube-scheduler
subjects:
- kind: ServiceAccount
  name: my-scheduler
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:kube-scheduler
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

멀티플 스케줄러

두 포드 스케줄링

- 멀티 스케줄러를 사용해 두 포드를 스케줄
- schedulerName 에 default-scheduler 를 작성하거나 아무 서술이 없는 경우 default-scheduler로 실행됨
- 두 번째 포드는 my-scheduler를 사용

```
$ kubectl get pod | grep sche
annotation-default-scheduler    1/1     Running
annotation-second-scheduler     0/1     Pending
```

sched-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: annotation-default-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: default-scheduler
  containers:
  - name: pod-with-default-annotation-container
    image: k8s.gcr.io/pause:2.0
---
apiVersion: v1
kind: Pod
metadata:
  name: annotation-second-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: my-scheduler
  containers:
  - name: pod-with-second-annotation-container
    image: k8s.gcr.io/pause:2.0
```

멀티플 스케줄러

클러스터롤에 권한이 부재

- 클러스터에 API 권한을 부여하면 my-scheduler도 활성화 가능
- 다음 명령어로 클러스터 롤을 편집

```
$ kubectl edit clusterrole system:kube-scheduler
```

- ResourceNames에 my-scheduler 추가

```
resourceNames:
- kube-scheduler
- my-scheduler
```

오토 스케일링 HPA

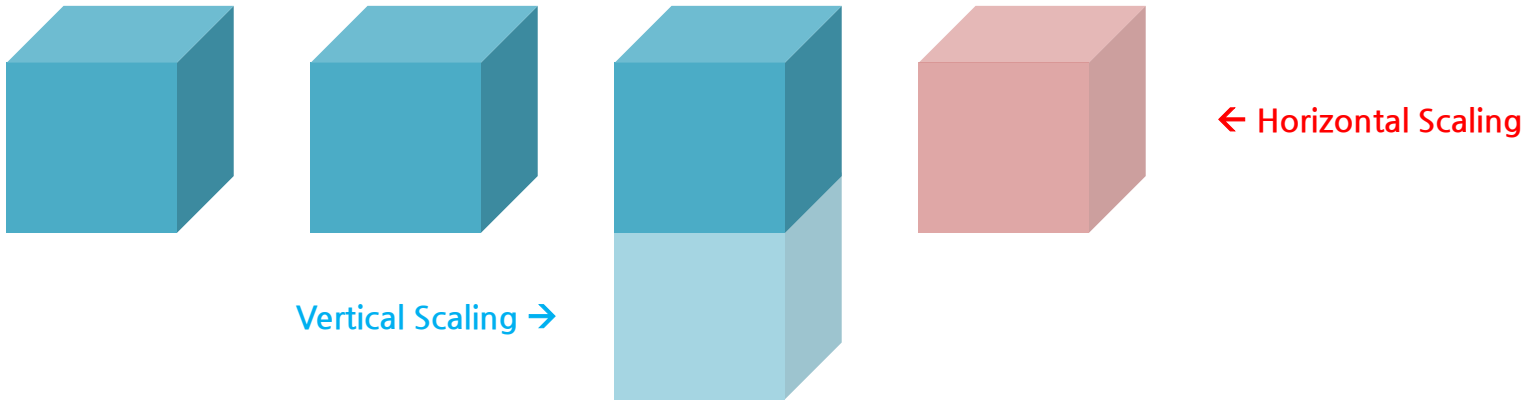
오토 스케일링 HPA

포드 스케일링의 두 가지 방법

- HPA: 포드 자체를 복제하여 처리할 수 있는 포드의 개수를 늘리는 방법
- VPA: 리소스를 증가시켜 포드의 사용 가능한 리소스를 늘리는 방법
- CA: 번외로 클러스터 자체를 늘리는 방법(노드 추가)

HPA(Horizontal Pod Autoscaler)

- 쿠버네티스에는 기본 오토스케일링 기능 내장
- CPU 사용률을 모니터링하여 실행된 포드의 개수를 늘리거나 줄임



오토 스케일링 HPA

VPA와 CA는 어떻게!?

- 공식 쿠버네티스에서 제공하지는 않으나 클라우드 서비스에서 제공

- 클러스터 자동 확장 처리(CA)
  - https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler

```
gcloud container clusters create example-cluster \
  --zone us-central1-a \
  --node-locations us-central1-a,us-central1-b,us-central1-f \
  --num-nodes 2 --enable-autoscaling --min-nodes 1 --max-nodes 4
```

- 수직형 pod 자동 확장(VPA)
  - https://cloud.google.com/kubernetes-engine/docs/how-to/vertical-pod-autoscaling?authuser=1&_ga=2.95397596.-1524441062.1575378592&_gac=1.156945225.1583154351.CjwKCAiA-vLyBRBWEiwAzOkGVAWhHWKuFV0kIRfpVDRA5yaGh1wIUZOHQZ6Z9lZYS2lZtzlyC5lBbhoCT3AQAvD_BwE

```
gcloud container clusters create [CLUSTER_NAME] --enable-vertical-pod-autoscaling
gcloud container clusters update [CLUSTER-NAME] --enable-vertical-pod-autoscaling
```

오토 스케일링 HPA

HPA(Horizontal Pod Autoscaler) 설정 방법

- 명령어를 사용하여 오토 스케일 저장하기

```
$ kubectl autoscale deployment my-app --max 6 --min 4 --cpu-percent 50
```

- HPA yaml을 작성하여 타겟 포드 지정

autoscaling.yaml

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
  namespace: default
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: myapp
  targetCPUUtilizationPercentage: 30
```

오토 스케일링 HPA

php-apache 서버 구동 및 노출

- https://kubernetes.io/ko/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/
- 이미지는 이미 구축돼 있으므로 바로 kubectl 명령어를 실행

```
$ kubectl apply -f
https://k8s.io/examples/application/php-apache.yaml
deployment.apps/php-apache created
service/php-apache created

$ kubectl autoscale deployment php-apache --cpu-
percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache
autoscaled

$ kubectl get hpa
NAME          REFERENCE                  TARGETS
MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache   <unknown>/50%   1
10        0          5s
```

dockerfile

```
FROM php:5-apache
ADD index.php /var/www/html/index.php
RUN chmod a+rx index.php
```

index.php

```
<?php
  $x = 0.0001;
  for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
  }
  echo "OK!";
?>
```

오토 스케일링 HPA

오토스케일링을 적용한 서비스에 무한 루프 쿼리를 통해 공격 수행

```
$ kubectl run --generator=run-pod/v1 -it --rm load-generator --image=busybox /bin/sh
Hit enter for command prompt

/ # while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!...
```

1분 후 확인(중간에 unknown이 발생할 수도 있음, 수집까지 시간이 걸림)

```
$ kubectl get hpa
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   458%/50%   1         10        1          75s

$ kubectl get hpa
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   123%/50%    1         10        10         4m29s`
```

오토 스케일링 HPA

연습문제

- **이미지 nginx를 생성하고 HPA가 적용하라.**
  - 최대 스케일링 개 수: 10
  - 최소 스케일링 개 수: 2
  - 스케일링을 수행할 CPU 활동량: 30%