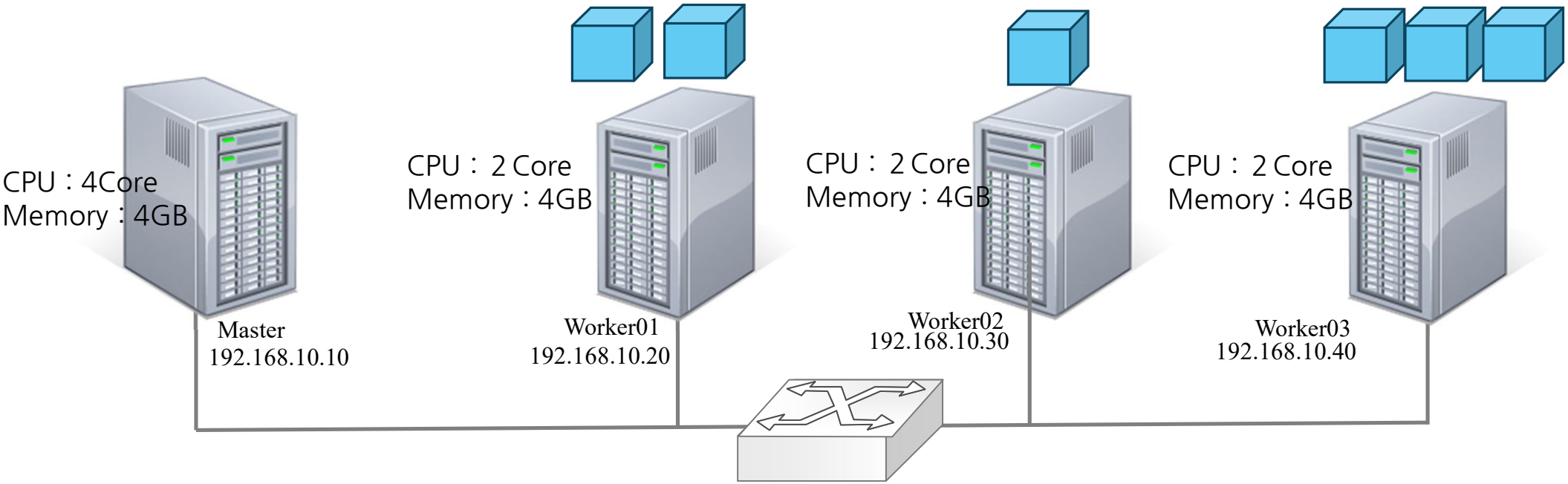
Pod Resource Request와 limit

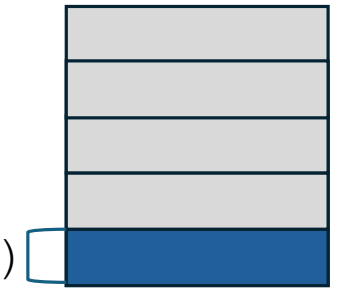
Pod Resource Request와 limit



Pod의 CPU 단위

- 쿠버네티스 CPU는 vCPU(core) 기준
- 소수점 또는 밀리코어(millicore, m) 단위를 사용
- vCPU = Virtual CPU(가상 CPU)
- 물리 코어 1개를 여러 조각으로 나누어 여러 VM 또는 컨테이너가 공유하도록 만든 것이 개념
- 쿠버네티스에서 표현하는 CPU 1은 1 vCPU = 1 logical core

  - cpu: 1000m → vCPU 1개
  - cpu: 500m → vCPU 0.5개
  - cpu: 200m → vCPU 0.2개



Pod의 Memory 단위

- 1MB = 1,000,000 Bytes (10^6)
  - 국제 표준 단위(SI, International System of Units)
  - 10진수(SI 단위계)
  - 저장장치 제조사(SSD/HDD), 네트워크 속도 표기에서 주로 사용
- 1MiB = 1,048,576 Bytes (2^{20})
  - 2진수(바이너리 단위)
  - 실제 메모리(RAM), OS, Kubernetes 등에서 사용
  - 바이너리 기반 실제 메모리 단위와 정확히 일치

KiB키-비-바이트키비바이트

MiB메-비-바이트메비바이트

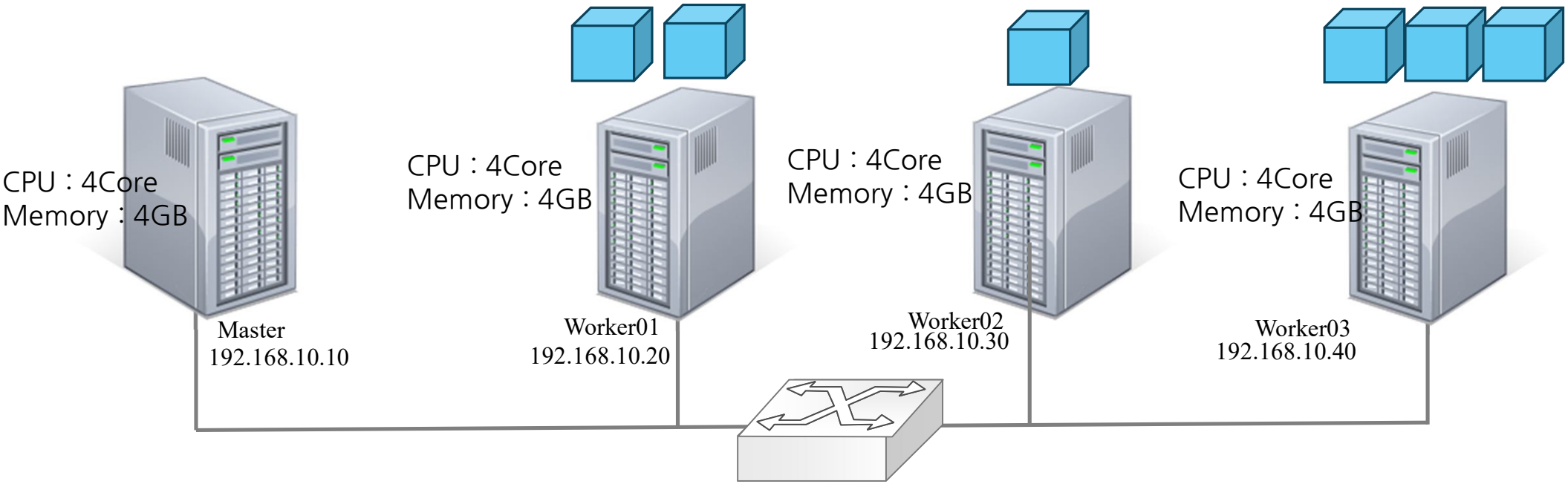
Pod Resource Request

- Pod의 container가 안정적으로 동작하기 위해 최소한으로 필요한 자원량
- 스케줄러가 Pod를 어느 노드에 배치할지 결정하는 기준
- 파드를 실행하기 위한 최소 리소스 양을 요청
- 너무 높게 설정하면 Pending 발생, 너무 낮게 설정하면 성능 부족 발생

```
requests:
  cpu: "200m"
  memory: "512Mi"
```

→ Pod가 동작하려면 최소 0.2 vCPU, 512MiB 메모리가 필요함

→ 스케줄러는 이 값을 기준으로 배치 가능 여부를 판단


CPU : 4Core
Memory : 4GB
Master
192.168.10.10
CPU : 4Core
Memory : 4GB
Worker01
192.168.10.20
CPU : 4Core
Memory : 4GB
Worker02
192.168.10.30
CPU : 4Core
Memory : 4GB
Worker03
192.168.10.40

Pod Resource Limits

- Pod의 container가 사용할 수 있는 최대 리소스 양을 제한
- CPU Limit 초과 → Throttle(속도 제한)
- Memory Limit 초과 → OOMKilled(메모리 초과로 강제 종료)
- Memory limit을 초과해서 사용되는 파드는 종료(OOM Kill)되며 다시 스케줄링

```
limits:
  cpu: “500m"
  memory: “1Gi"
```

→ Pod는 절대 500m 이상 CPU 사용 불가

→ 1GiB 메모리 초과 시 OOM Killed

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource1
spec:
  containers:
  - name: nginx-container
    image: nginx:1.14
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      requests:
        memory: 500Mi
        cpu: 1
```

<resource1.yaml>

```
kubectl create -f  resource1.yaml
kubectl describe pod resource1
kubectl delete pod --all
```

Nginx-containe가 실행 시
최소한 메모리는 500mi(메비바이트)
cpu는 1코어가 보장하는 node에 해당 파드를 생성

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource2
spec:
  containers:
  - name: nginx-container
    image: nginx:1.14
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      limits:
        memory: 500Mi
        cpu: 1
```

<resource2.yaml>

kubectl create -f resource2.yaml

kubectl describe pod resource2

kubectl delete pod --all

Nginx-containe가 실행 시

최대 사용할 수 있는 최대 메모리는 500mi(메비바이트),

cpu는 1코어가 보장하는 node에 해당 파드를 생성

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource3
spec:
  containers:
  - name: nginx-container
    image: nginx:1.14
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      limits:
        memory: 500Mi
        cpu: 8
```

<resource3.yaml>

kubectl create -f resource3.yaml

kubectl describe pod resource3

kubectl delete pod --all

Nginx-containe가 실행 시

최대 사용할 수 있는 최대 메모리는 500mi(메비바이트),

cpu는 8코어가 보장하는 node에 해당 파드를 생성

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource4
spec:
  containers:
  - name: nginx-container
    image: nginx:1.14
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      request:
        memory: 500Mi
        cpu: 200m
      limits:
        memory: 1Gi
        cpu: 1
```

<resource4.yaml>

```
kubectl create -f  resource4.yaml
kubectl describe pod resource4
kubectl delete pod --all
```

kube-system의 주요 컴포넌트

CoreDNS	내부 DNS 서비스 Pod 이름를 기반으로 IP 변환
kube-proxy	노드 내에서 서비스 트래픽 iptables/ipvs 라우팅 담당 Service를 이용하여 Pod로 패킷 전달 기능
etcd	Kubernetes의 모든 상태가 저장되는 분산 Key-Value DB
kube-apiserver, scheduler, controller-manager	Control Plane에서 실행되는 Kubernetes 핵심 컴포넌트들
CNI 플러그인	Pod 간 네트워크 연결 관리

K8S AutoScale

오토스케일(AutoScale)

- 시스템의 부하(트래픽, CPU 사용량, 메모리 사용량 등)에 따라 자동으로 컴퓨팅 자원의 수를 조정하는 기능
- 주요 목적

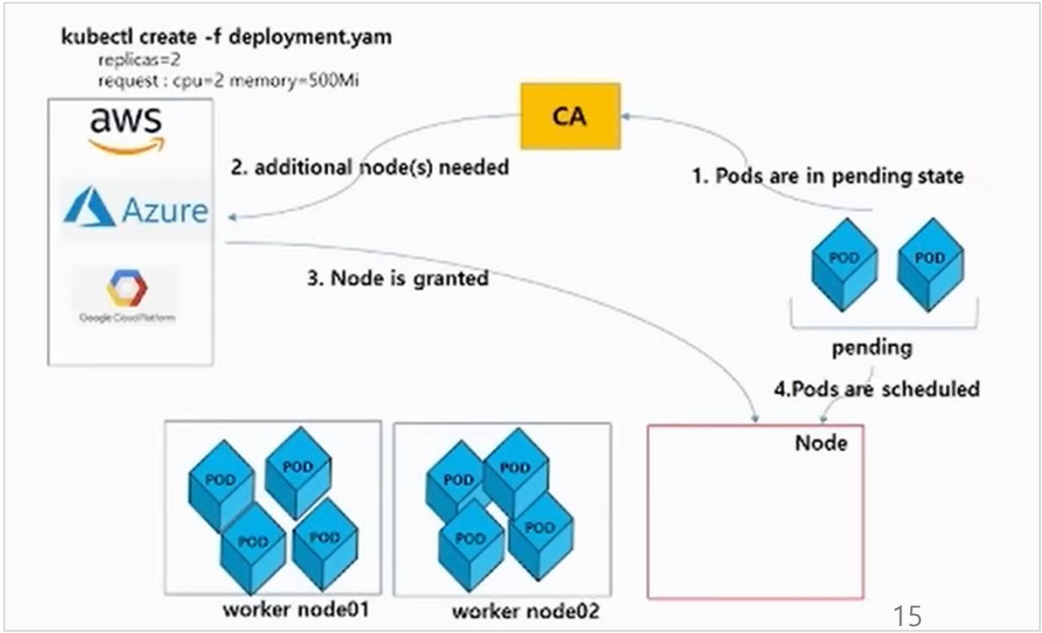
가용성 확보	갑작스러운 트래픽 급증에도 서비스가 중단되지 않도록 자동으로 Pod를 추가
비용 절감	사용량이 줄어들면 불필요한 Pod를 줄여 클라우드 비용을 절약
운영 효율화	관리자가 일일이 서버를 증설하거나 삭제하지 않아도 됨

- 종류

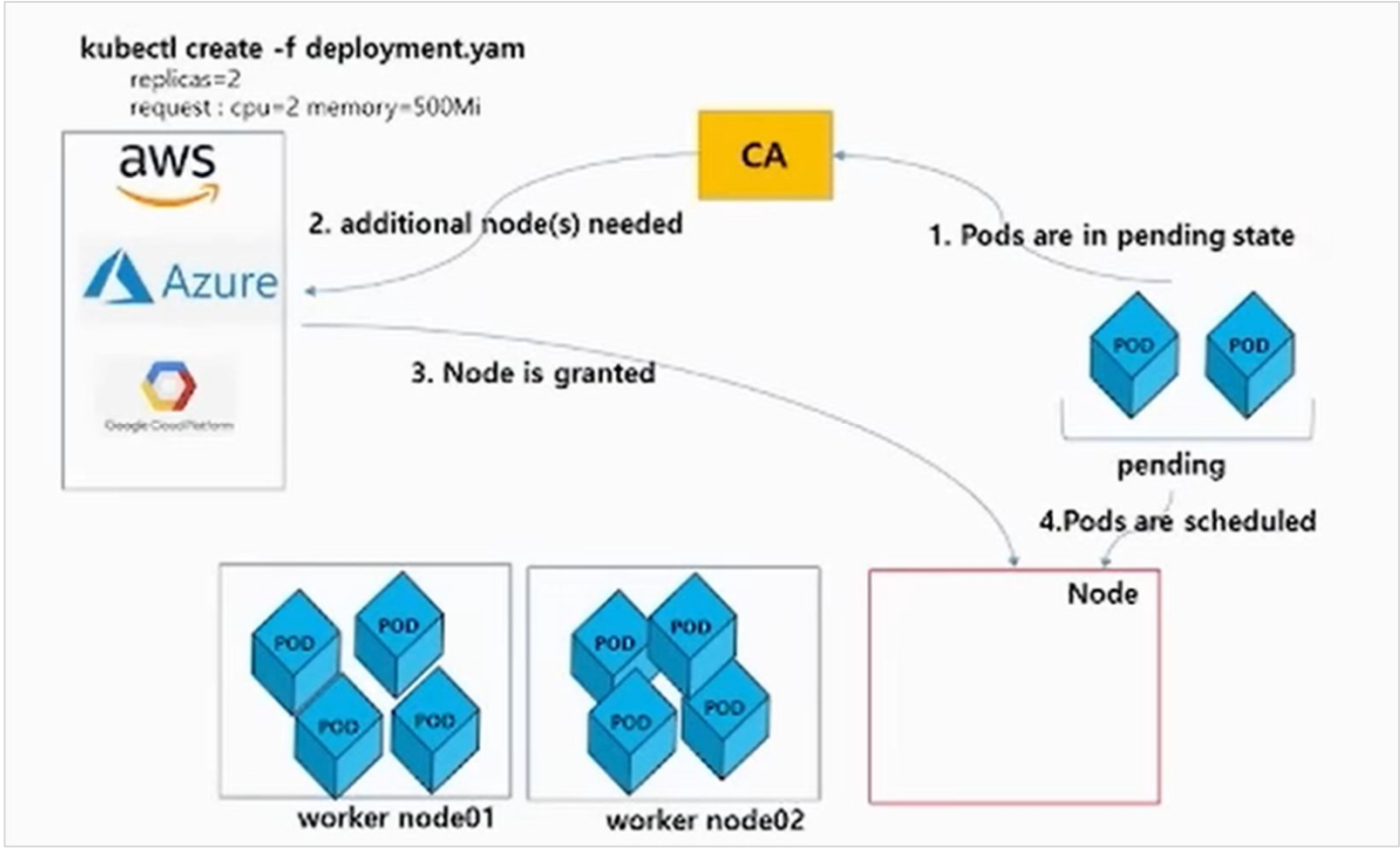
Cluster level scalability	노드(Worker Node) 자체를 증감
Pods layer auto scale	Horizontal Pod Autoscaler(HPA) 인스턴스나 Pod 개수를 늘리거나 줄임
	Vertical Pod Autoscaler(VPA) 관리자가 일일이 서버를 증설하거나 삭제하지 않아도 됨

Cluster Autoscaler(CA)

- GCP, AWS 및 Azure와 같은 cloud infrastructure를 통해서 사용
- Pode가 node 리소스를 할당 받지 못해 pending 될 때 worker node를 확장
- Node pool의 min/max를 기준으로 그 범위내로 노드 확장
- 할당된 node가 장시간 충분히 활용되지 못하면 node를 해제
- 10초마다 불필요한 노드 확인, 10분간 적은 리소스 유지하면 scale down



Cluster Autoscaler(CA)



	Scale-Out(스케일 아웃)	Scale-Up(스케일 업)
K8S Auto Scale	Horizontal Auto Scale (HAS)	Vertical Auto Scale(VAS)
개념	서버 수를 늘림 (수평 확장) (예) 웹 서버 1대 → 3대	한 서버의 성능을 향상 (수직 확장) (예) CPU 4core → 16core
장점	유연한 확장, 장애 분산, 비용 효율적	설정 간단, 애플리케이션 변경 불필요
단점	로드밸런싱 필요, 시스템 복잡도 증가	확장 한계, 고성능 장비 비용 높음

Horizontal Pod Autoscaler(HPA)

- Metrics Server

  - 각 Pod와 node의 사용량을 모니터링하고 API를 통해 볼 수 있게 제공

- Pod의 replicas 수를 관리

  - 구독 중인 Pod의 CPU/Memory 사용률을 기반으로 Pod를 Scale out

  - 확장/축소 할 최소/최대 Pods 수량은 Pods의 Deployment에 의해 제어

- HPA 동작 조건

  - HPA는 기본 30초 간격으로 Pod 리소스 사용량을 check HPA에 설정한 임계 값을 초과할 경우 Pod를 확장

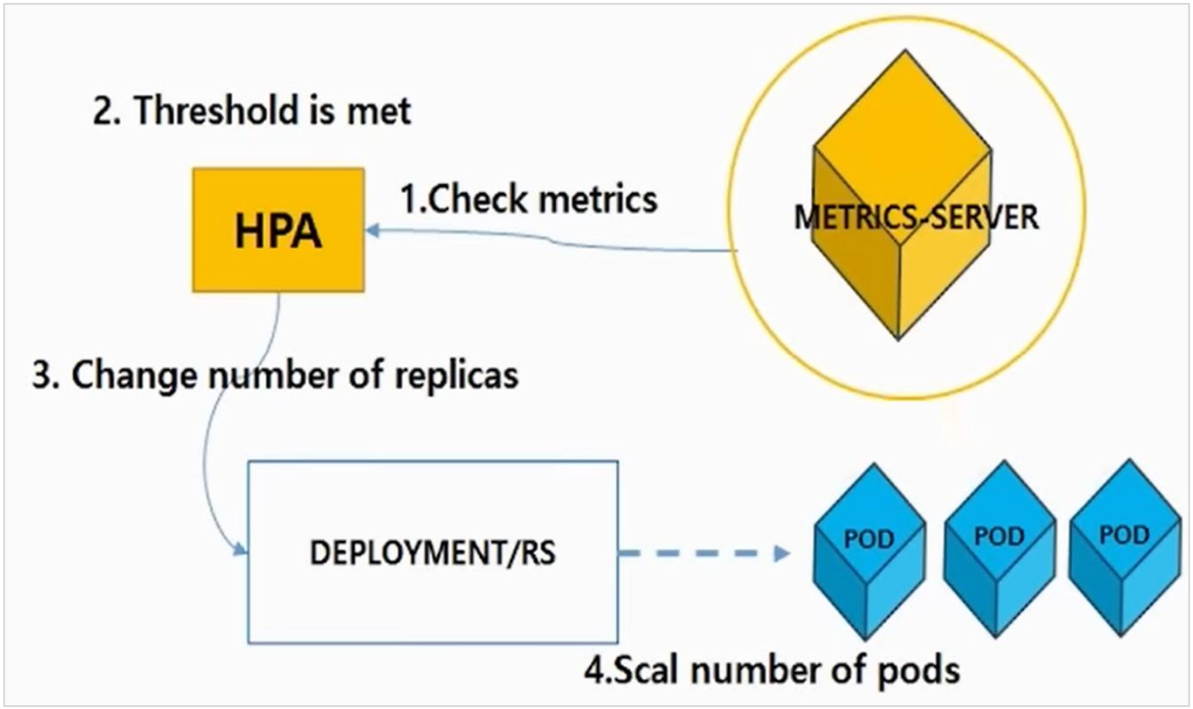
  - Scale-out 이후 3분 대기,Scale-in 이후 5분 대기

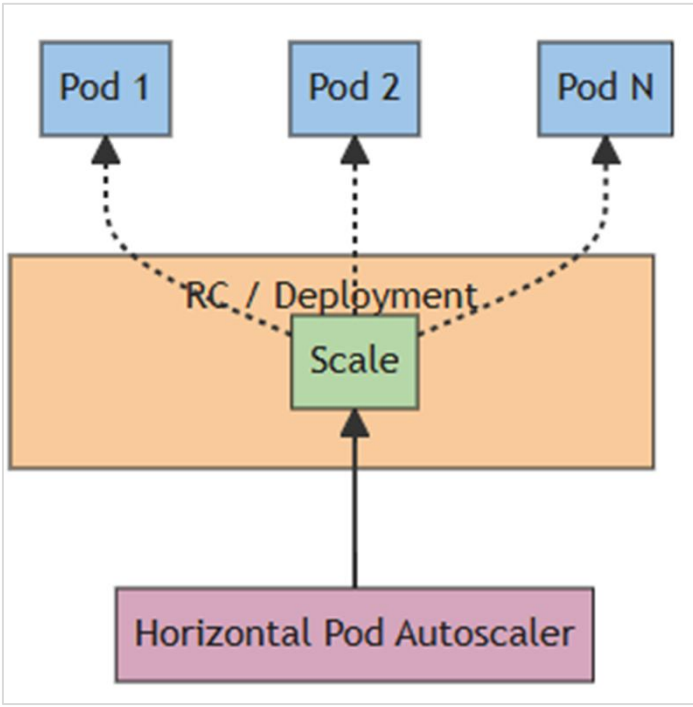
Metric Server

- Pod의 리소스 사용량(CPU, 메모리 등)을 수집하고 제공하는 핵심 컴포넌트
- 오토스케일링(HPA, VPA 등)이 동작하기 위한 기반 데이터를 제공
- Pod와 Node들의 CPU/Memory 사용량을 주기적으로 모니터링하고 metric 정보를 수집하여 API 제공
- Horizontal Pod Autoscaler의 Replicas 수 결정 계산법

원하는 레플리카 수 = ceil[현재 레플리카 수 * (현재 메트릭 값 / 원하는 메트릭 값)]

Metric Server





Vertical Pod Autoscaler(HPA)

- Pod의 리소스를 관리

  - Pod에 대한 CPU/Memory 리소스를 추천

  - Pod 대한 CPU/Memory 리소스를 자동으로 조정

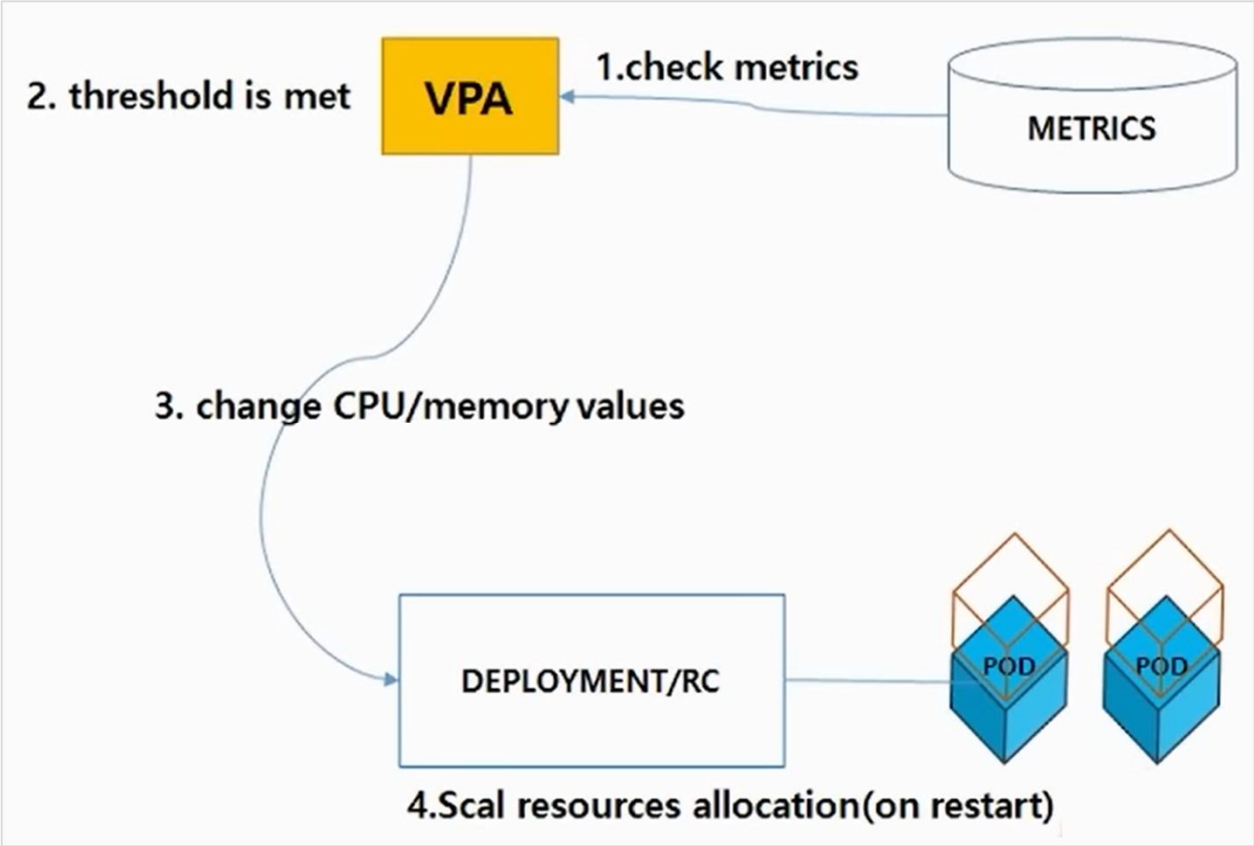
- 동작방식

  - Metric를 10초 간격으로 지속적으로 확인

  - 할당된 CPU/Memory의 임계치를 넘으면 Pod 템플릿을 변경하여 Pod의 리소스 할당 값을 변경한 후 Pod를 다시 시작

- Vertical Pod Autoscaler라는 사용자 정의 리소스로 구성

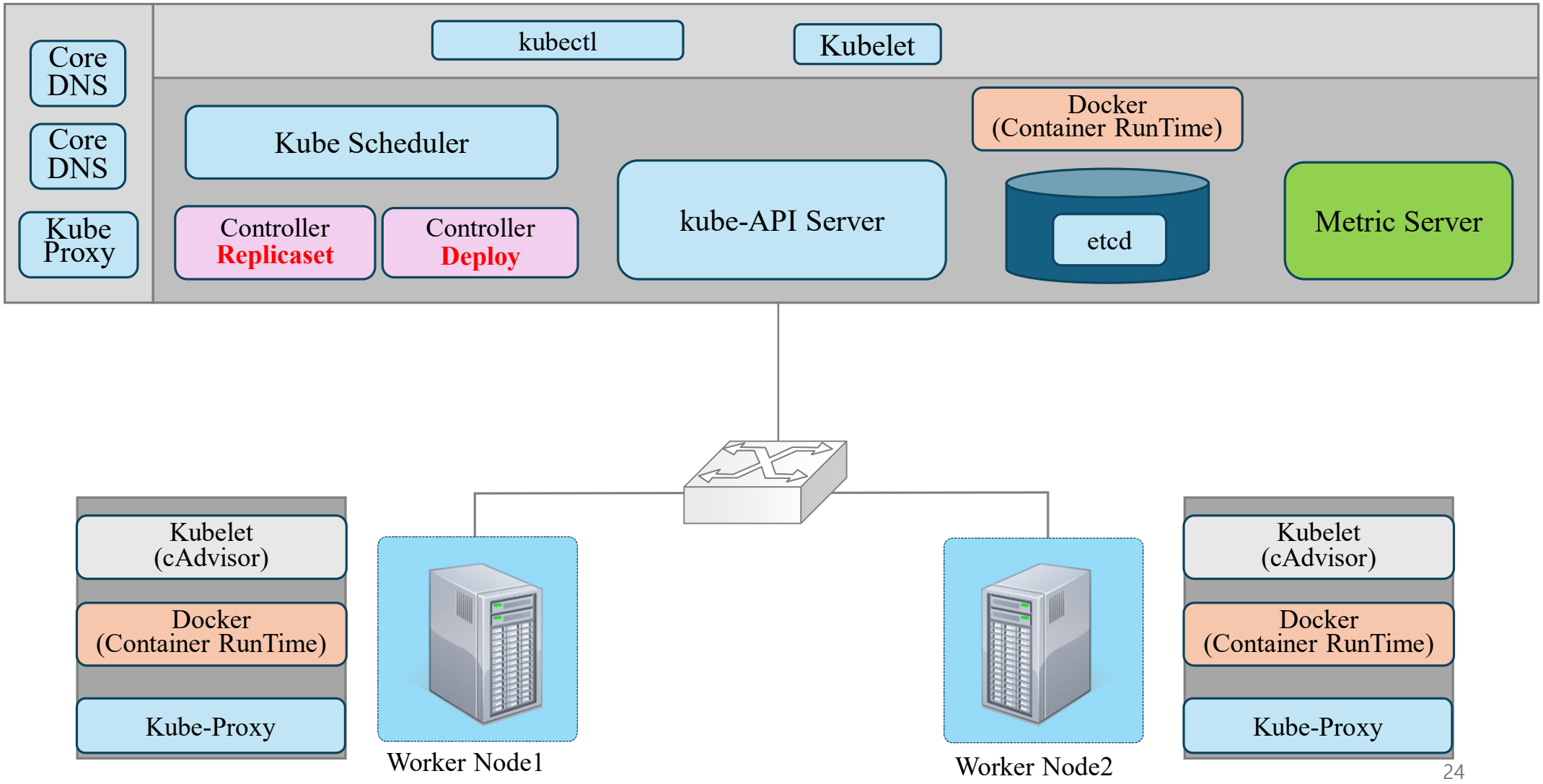
Vertical Pod Autoscaler(HPA)



HPA Autoscaling 실습

- Metric Server 설치
- Deploy 설치
- Service 설치

kubectl autoscale deployment webserver --cpu-percent=70 --min=2 --max=10



Metric Server 설치 방법(1)

① wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

-O metrics-server.yaml

② nano metrics-server.yaml

```
containers:
- name: metrics-server
  image: registry.k8s.io/metrics-server/metrics-server:v0.7.2
  imagePullPolicy: IfNotPresent
  args:
    - --cert-dir=/tmp
    - --secure-port=4443                                          //수정
    - --kubelet-preferred-address-types=InternalIP,Hostname,InternalDNS  //추가
    - --kubelet-use-node-status-port
    - --metric-resolution=15s
    - --kubelet-insecure-tl    //추가

  ports:
  - containerPort: 4443        //추가
```

③ kubectl apply -f metrics-server.yaml

Metric Server 설치 방법(2)

① git clone https://github.com/soraddang/kubernetes-metrics-server.git

② cd kubernetes-metrics-server

③ kubectl create -f .

Metric Server 설치 확인

kubectl get pod -n kube-system

```
root@master:~# kubectl get pod -n kube-system
NAME                                READY   STATUS    RESTARTS          AGE
coredns-7c65d6cfc9-wc8ts            1/1     Running   3 (6h48m ago)     4d7h
coredns-7c65d6cfc9-x68p2            1/1     Running   3 (6h48m ago)     4d7h
etcd-master                         1/1     Running   17 (6h48m ago)    4d7h
kube-apiserver-master               1/1     Running   21 (6h48m ago)    4d7h
kube-controller-manager-master      1/1     Running   25 (6h48m ago)    4d7h
kube-proxy-jkvlf                    1/1     Running   14 (6h48m ago)    4d7h
kube-proxy-p7xtm                    1/1     Running   3 (6h48m ago)     4d6h
kube-proxy-qhtlv                    1/1     Running   3 (6h48m ago)     4d6h
kube-proxy-vmx9w                    1/1     Running   4 (6h48m ago)     4d6h
kube-scheduler-master               1/1     Running   27 (6h48m ago)    4d7h
metrics-server-6f66bcb644-xxf6l     1/1     Running   7 (6h46m ago)     3d20h
root@master:~#
```

kube-system

- K8S를 운영하는 핵심 파드들이 들어있는 시스템 영역
- K8s를 운영하는 필수적인 모든 구성요소가 들어 있는 공간

Metric Server 설치 확인

kubectl get deployment metrics-server -n kube-system

Kubectl top nodes

```
root@master:~# kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           3d20h
root@master:~#
root@master:~# kubectl top nodes
NAME       CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master     130m         3%     2428Mi          64%
worker01   39m          1%     1514Mi          40%
worker02   21m          1%     1544Mi          40%
worker03   32m          1%     1627Mi          43%
root@master:~#
```

Deploy 설치

```
root@master:/k8s# cat deploy_web.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: soraland/test01
        name: web
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 200m
```

kubectl apply -f deploy_web.yaml

kubectl get deploy

```
root@master:/k8s# kubectl get deploy
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
deploy-web   1/1     1            1           28h
root@master:/k8s#
```

Service 설치

```
root@master:/k8s# cat deploy_web_1.yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-web
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: web
```

kubectl apply -f deply_web_1.yaml

kubectl get service

```
root@master:/k8s# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   4d8h
svc-web      ClusterIP   10.106.249.20   <none>        80/TCP    28h
```

WebServer 접속 확인

```
root@master:/k8s# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   4d8h
svc-web      ClusterIP   10.106.249.20   <none>        80/TCP    28h
root@master:/k8s#
root@master:/k8s# curl 10.106.249.20
okroot@master:/k8s#
```

kubectl get service

curl 10.106.249.20

WebServer 사용률 증가 실습

while true

do

curl 10.106.249.20

done

WebServer 사용률 증가 시 Scale out 확인 (HPA 확인)

1 Master

watch kubectl get pods –o wide

```
Every 2.0s: kubectl get pods -o wide

NAME                          READY   STATUS    RESTARTS         AGE   IP               NODE       NOMINATED NODE   READINESS GATES
deploy-web-6c5c56786c-2qq2d   1/1     Running   0                9s    192.168.30.77    worker02   <none>           <none>
deploy-web-6c5c56786c-54pwc   1/1     Running   1 (7h34m ago)    28h   192.168.5.20     worker01   <none>           <none>
deploy-web-6c5c56786c-5vksk   1/1     Running   0                9s    192.168.5.21     worker01   <none>           <none>
deploy-web-6c5c56786c-69rwg   1/1     Running   0                9s    192.168.19.83    worker03   <none>           <none>
deploy-web-6c5c56786c-lrdw5   1/1     Running   0                24s   192.168.30.76    worker02   <none>           <none>
deploy-web-6c5c56786c-n5zvk   1/1     Running   0                24s   192.168.19.82    worker03   <none>           <none>
```

1 Master

watch kubectl top pods

```
Every 2.0s: kubectl top pods

NAME                          CPU(cores)   MEMORY(bytes)
deploy-web-6c5c56786c-54pwc   704m         39Mi
deploy-web-6c5c56786c-pfv8t   347m         37Mi
```

1 Master

```
root@master:~# while true
> do
> curl 10.106.249.20
> done
okokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokokok
```