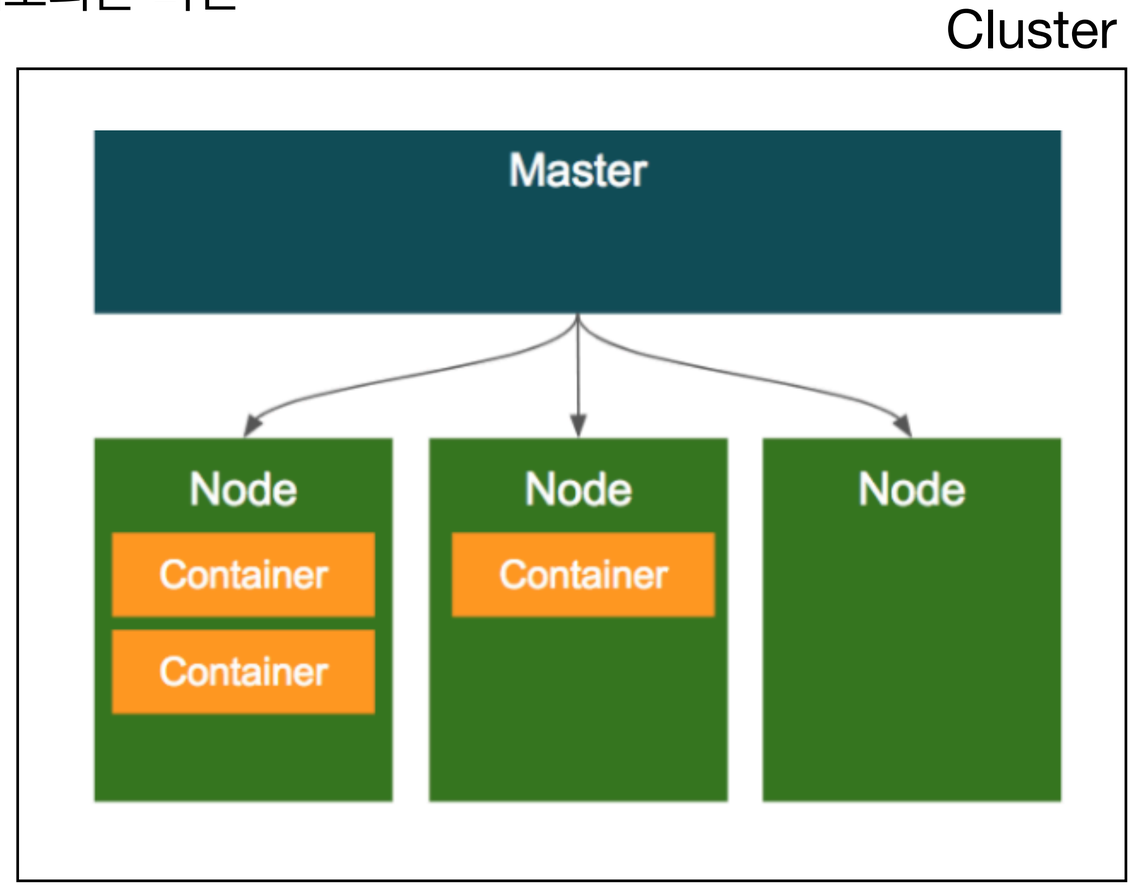
Chapter 5. Objects and Pod

Kubernetes Study week #4

Sieun Park / 2021.06.22

Review: Cluster 구성

- Object: 쿠버네티스에 의해서 배포 및 관리되는 가장 기본적인 요소들. (Pod, Service, Volume, Namespace)
- Master: Cluster 전체를 관리하는 controller. Object 들을 생성하고 관리하는 역할
- Worker: Container가 배포되는 머신



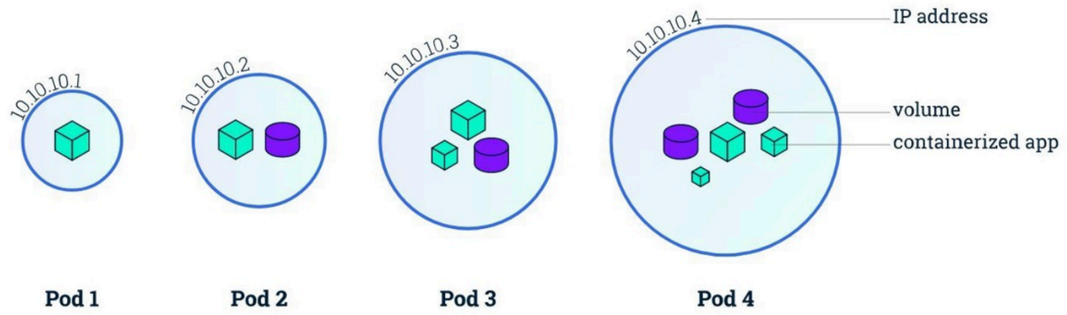
Basic Object #1 Pod

- Pod: k8s 의 가장 기본 배포 단위
- k8s 는 컨테이너를 개별적으로 하나씩 배포하지 않고 Pod 단위로 배포하는데,
  Pod 는 하나 이상의 컨테이너를 포함한다. (주로 1개, 최대 3개)

```
$ cat mynginx.yaml

apiVersion: v1
kind: Pod
metadata:
 labels:
    run: mynginx
 name: mynginx
spec:
  containers:
  - image: nginx
    name: mynginx
restartPolicy: Never
```

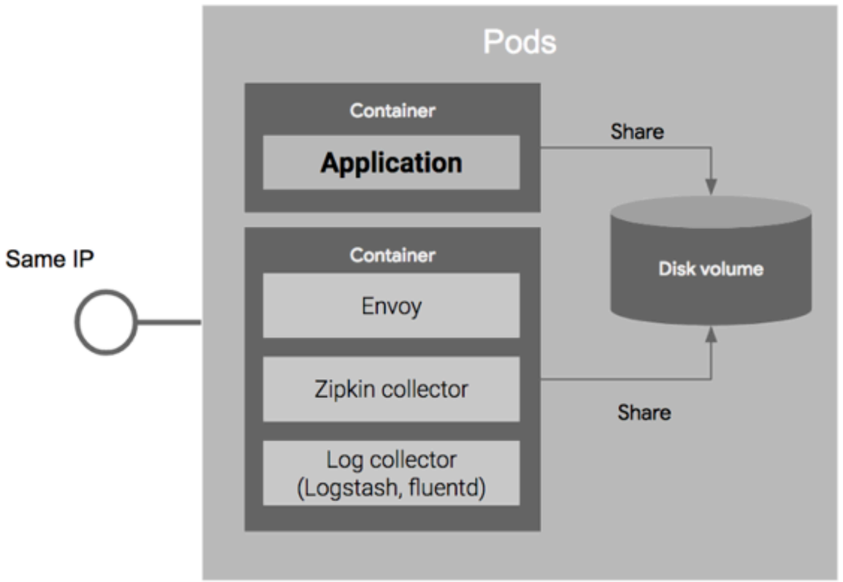
- apiVersion: script 실행을 위한 k8s API 버전 (보통 v1)
- kind: resource의 종류 정의
- Metadata: 각종 메타데이터 정의
- Spec: 리소스에 대한 상세한 스펙 정의
  - Pod는 container를 가지고 있으므로 container 정의
  - Container 가 여러 개일 경우 여러 개의 container 정의



왜 여러개의 컨테이너를 Pod 단위로 묶어서 배포할까?

1. **동일 Pod 내의 컨테이너들은 IP를 공유한다.**
   - 동일 Pod 내 container들은 localhost로 통신 가능
   - Container A (Port 8080) Container B (Port 7001)가 배포될 경우, B가 A와 통신할 때 localhost:8080으로 통신

2. **동일 Pod 내의 컨테이너들은 Disk volume을 공유할 수 있다.**
   - Application을 배포할 때, 로그 수집기 등 보조 솔루션이 함께 배포되는 경우가 많음. 보조 솔루션을 다른 Pod에 배포하면 Application의 파일 시스템에 접근할 수 없다.
   - 하지만, 동일 Pod에서 컨테이너만 분리할 경우 다른 컨테이너의 파일을 읽을 수 있는 장점이 있음
   - Host volume 또는 emptyDir 이용하여 디스크 공유 가능



Basic Object #2 Volume

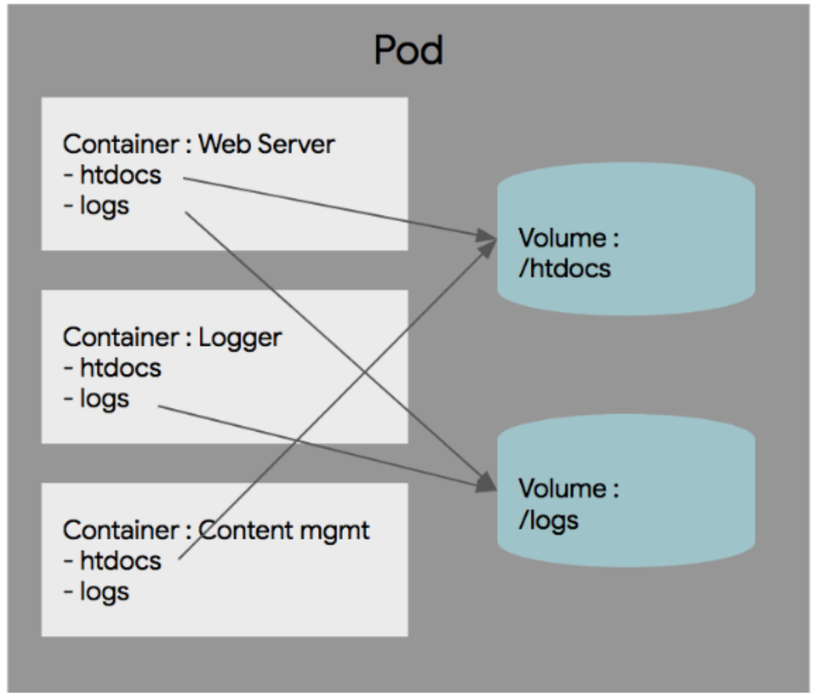
- Pod 내부 스토리지의 생명주기는 Pod와 동일
  - Pod 를 재시작하거나 새로 배포할 경우 기존 데이터 유실
- Pod 내에서 생성된 데이터를 Pod 생명주기와 관계 없이 저장해두고 싶을 경우, volume 을 따로 연결해야 한다. (외장 디스크!)
- Pod 내 컨테이너끼리 데이터 공유를 하고 싶은 경우에도 volume을 연결해야 한다.
  - host: host node의 volume 공간에 저장

    만약 container restart 후 다른 node에 배포될 경우 데이터 접근 불가능
  - emptyDir: Pod 내 컨테이너끼리 데이터 공유

    emptyDir의 생명주기는 Pod 단위 (container X)

- Example
  - Web Server 배포하는 Pod
    - WebServer: 웹서비스 서비스
    - Content mgmt: 컨텐츠 내용 (/htdocs) 관리
    - Logger: 로그 메세지 관리
  - /htdocs, /logs volume을 각각 생성 후

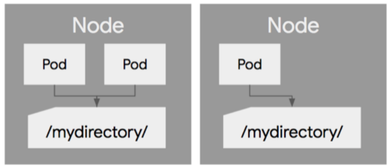
    각 container에 mount 하여 공유



Basic Object #2 Volume

- **Host volume에 연결**

```
$ cat volume.yaml
apiVersion: v1
kind: Pod
metadata:
        name: volume
spec:
        containers:
                - name: nginx
                  image: nginx
                  volumeMounts:
                          - mountPath: /container-volume
                            name: my-volume
        volumes:
                - name: my-volume
                  hostPath:
                          path: /home
```



- **emptyDir에 연결**

```
apiVersion: v1
kind: Pod
metadata:
  name: shared-volumes
spec:
  containers:
  - name: redis
    image: redis
    volumeMounts:
    - name: shared-storage
      mountPath: /data/shared
  - name: nginx
    image: nginx
    volumeMounts:
    - name: shared-storage
      mountPath: /data/shared
  volumes:
  - name: shared-storage
    emptyDir: {}
```

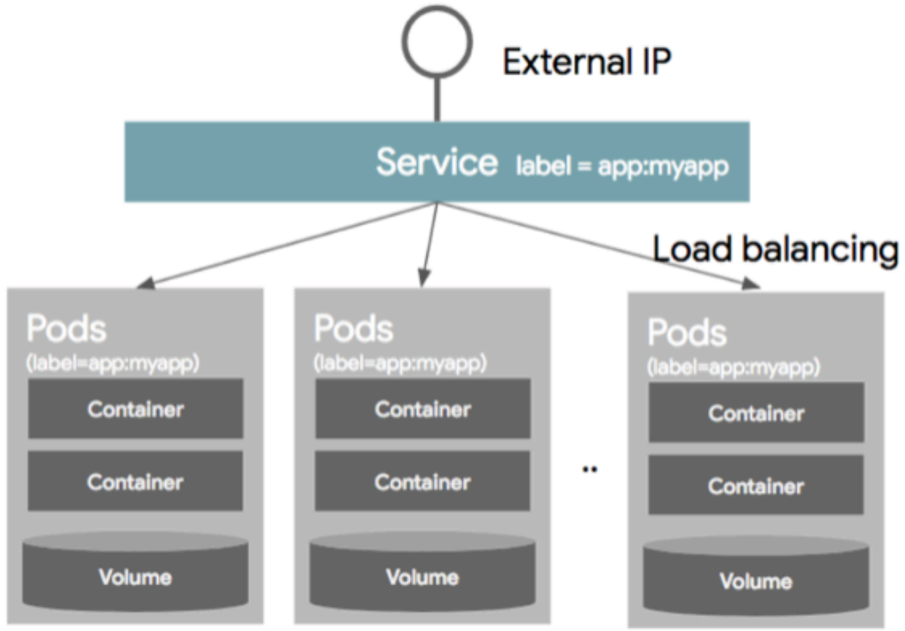
Basic Object #3 Service (Pod: Label)

- 실제 Pod 를 배포하여 서비스를 제공할 때 단일 Pod 만으로 서비스 하는 경우는 드물고,
  여러 개의 Pod를 서비스할 때 load balancer를 이용하여 하나의 IP와 포트를 이용한다.

- Pod의 경우 Controller에 의해 동적으로 생성되고 restart가 될 경우 IP가 바뀌기 때문에,
  load balancer 에서 함께 서비스할 Pod 목록을 지정할 때 IP 주소를 사용할 수 없다.

- 이 때 원하는 Pod 리스트를 grouping 하기 위해 사용하는 것이
  Label 과 Label selector !
  - Label 은 단순 key-value 형태의 문자열
  - Node 에도 라벨을 설정하여 container 생성 시 라벨을 이용하여
    원하는 node에 container를 배포할 수 있다.

- 그 외에도 Label을 사용하면 특정 리소스만 배포하거나 업데이트,
  네트워크 접근 권한을 부여하는 등의 설정이 가능하다.



Basic Object #3 Service (Pod: Label)

- Service 맛보기

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

- Label 설정

```
$ cat mynginx.yaml

apiVersion: v1
kind: Pod
metadata:
 labels:
    run: mynginx
 name: mynginx
spec:
  containers:
  - image: nginx
    name: mynginx
restartPolicy: Never
```

- Node-selector

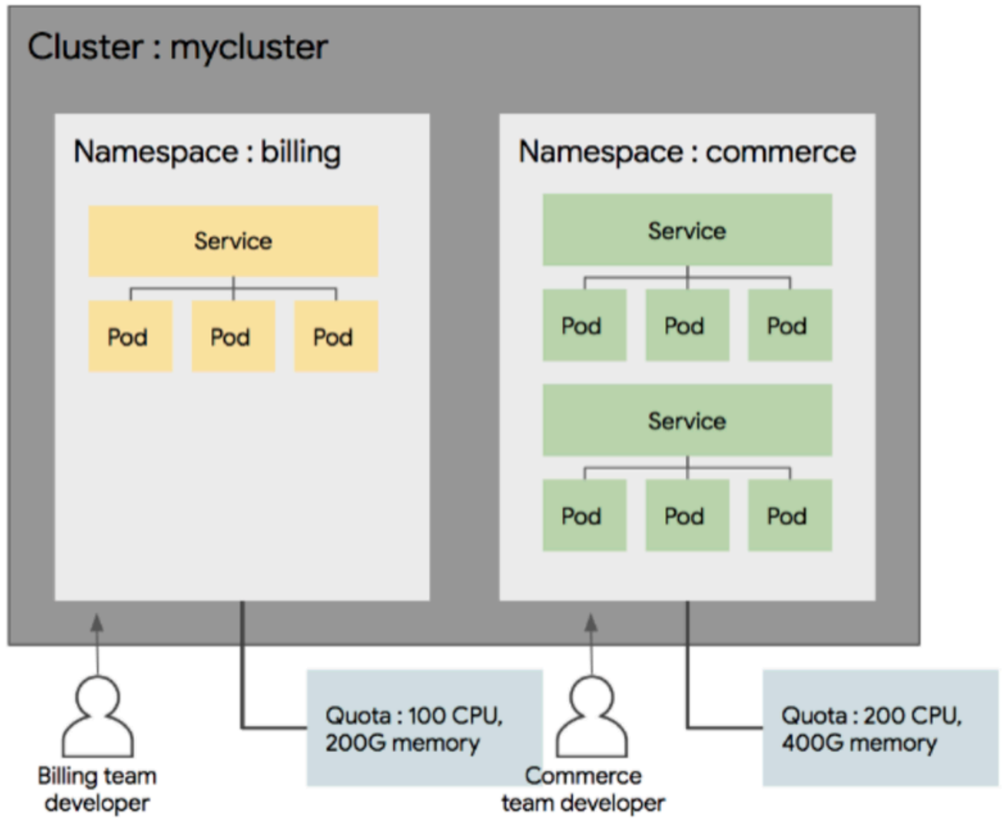
```
$ kubectl label node master disktype=ssd
$ kubectl label node worker disktype=hdd
```

```
$ cat node-selector.yaml
apiVersion: v1
kind: Pod
metadata:
        name: node-selector
spec:
        containers:
                - name: nginx
                  image: nginx
        nodeSelector:
                disktype: hdd
```

```
sieunpark95@master:~/script$ kubectl get node --show-labels | grep disktype
master   Ready    master   3d7h   v1.18.6+k3s1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=k3s,beta.kubernetes.io/os=linux,disktype=ssd,diskype=ssd,k3s.io/hostname=master,k3s.io/internal-ip=10.178.0.2,kubernetes.io/arch=amd64,kubernetes.io/hostname=master,kubernetes.io/os=linux,node-role.kubernetes.io/master=true,node.kubernetes.io/instance-type=k3s
worker   Ready    <none>   3d7h   v1.18.6+k3s1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=k3s,beta.kubernetes.io/os=linux,disktype=hdd,k3s.io/hostname=worker,k3s.io/internal-ip=10.178.0.4,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker,kubernetes.io/os=linux,node.kubernetes.io/instance-type=k3s
```

Basic Object #4 Namespace

- Namespace: Cluster 내의 논리적인 분리 단위
  - 사용자별로 namespace별 접근 권한을 다르게 운영할 수 있다.
  - Namespace별로 리소스 할당량을 지정할 수 있다.
  - Namespace별로 Service, Pod 를 나누어 관리할 수 있다.



Pod #1 Resource 관리

- Request: Pod 가 보장받을 수 있는 최소 리소스 사용량 정의
- Limits: Pod 가 최대로 사용할 수 있는 최대 리소스 사용량 정의
  - cpu 1000m = 1core
  - Memory Mi = 1MiB (2^20 bytes)

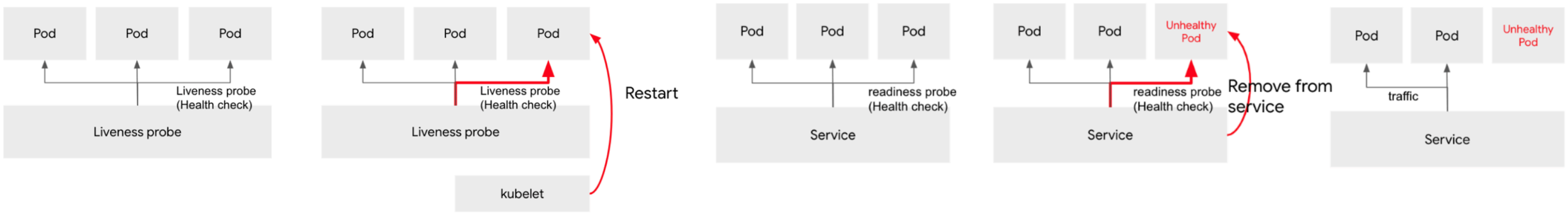
바이트 크기					v · d · e · h
SI 접두어		전통적 용법		이진 접두어	
기호(이름)	값	기호	값	기호(이름)	V값
kB (킬로바이트)	$1000^1 = 10^3$	KB	$1024^1 = 2^{10}$	KiB (키비바이트)	2^{10}
MB (메가바이트)	$1000^2 = 10^6$	MB	$1024^2 = 2^{20}$	MiB (**메비바이트**)	2^{20}
GB (기가바이트)	$1000^3 = 10^9$	GB	$1024^3 = 2^{30}$	GiB (기비바이트)	2^{30}
TB (테라바이트)	$1000^4 = 10^{12}$	TB	$1024^4 = 2^{40}$	TiB (테비바이트)	2^{40}
PB (페타바이트)	$1000^5 = 10^{15}$	PB	$1024^5 = 2^{50}$	PiB (페비바이트)	2^{50}
EB (엑사바이트)	$1000^6 = 10^{18}$	EB	$1024^6 = 2^{60}$	EiB (엑스비바이트)	2^{60}
ZB (제타바이트)	$1000^7 = 10^{21}$	ZB	$1024^7 = 2^{70}$	ZiB (제비바이트)	2^{70}
YB (요타바이트)	$1000^8 = 10^{24}$	YB	$1024^8 = 2^{80}$	YiB (요비바이트)	2^{80}

```
$ cat requests.yaml
apiVersion: v1
kind: Pod
metadata:
        name: requests
spec:
        containers:
                - name: nginx
                  image: nginx
                  resources:
                          requests:
                                  cpu: "250m"
                                  memory: "500Mi"
```

```
$ cat limits.yaml
apiVersion: v1
kind: Pod
metadata:
        name: limits
spec:
        restartPolicy: Never
        containers:
                - name: nginx
                  image: nginx
                  command: ["python"]
                  args: ["-c", "arr=[]\nwhile True: arr.append(range(1000))"]
                  resources:
                          limits:
                                  cpu: "500m"
                                  memory: "1Gii"
```

Pod #2 Health check

- kubelet 은 각 container의 상태를 주기적으로 체크해서, 문제가 있는 pod/container를 자동으로 재시작 하거나 또는 문제가 있는 Pod를 서비스에서 제외할 수 있다. 이러한 기능을 health check라고 한다.
  - Liveness probe: container가 살아 있는지 아닌지를 체크. 비정상일 경우 재시작
  - Readiness probe: container의 서비스가 가능한 상태인지를 체크. 비정상일 경우 서비스 목록에서 제외



- Probe types: Command probe / HTTP probe
  - Command probe: shell 명령을 수행하고 그 결과값으로 정상 여부 체크. 0이면 정상, 0이 아니면 실패로 간주
  - HTTP probe: HTTP GET 요청을 보내 return되는 응답 코드가 200~300이면 정상, 그 외의 경우 실패로 간주
  - TCP probe: container의 지정된 IP와 TCP 상태 확인을 하고, 포트가 열려 있으면 정상이라고 판단

Pod #2 Health check

- **LivenessProbe (HTTP Probe)**

```
$ cat liveness.yaml
apiVersion: v1
kind: Pod
metadata:
        name: liveness
spec:
        containers:
                - name: nginx
                  image: nginx
                  livenessProbe:
                        httpGet:
                                path: /live
                                port: 80
```

- /usr/share/nginx/html/live 파일 생성 필요

- **ReadinessProbe (CMD Probe)**

```
$ cat readiness-cmd.yaml
apiVersion: v1
kind: Pod
metadata:
        name: readiness-cmd
spec:
        containers:
                - name: nginx
                  image: nginx
                  readinessProbe:
                        exec:
                                command:
                                        - cat
                                        - /tmp/ready
```

- /tmp/ready 파일 생성 필요
- /tmp/ready 파일이 있어야함 !

Pod #3 2개 Container 실행

- **Container 2개 생성**

```
$ cat second.yaml
apiVersion: v1
kind: Pod
metadata:
        name: second
spec:
        containers:
                - name: nginx
                  image: nginx
                - name: curl
                  image: curlimages/curl
                  command: ["/bin/sh"]
                  args: ["-c", "while true; do sleep 5; curl
localhost; done"]
```

- Container 실행 순서 보장되지 않음

- **초기화 container**

```
$ cat init-container.yaml
apiVersion: v1
kind: Pod
metadata:
        name: init-container
spec:
        restartPolicy: OnFailure
        containers:
                - name: busybox
                  image: k8s.gcr.io/busybox
                  command: ["ls"]
                  args: ["/tmp/moby"]
                  volumeMounts:
                          - name: workdir
                            mountPath: /tmp
        initContainers:
                - name: git
                  image: alpine/git
                  command: ["sh"]
                  args:
                          - "-c"
                          - "git config --global --unset https.proxy"
                          - "git clone https://github.com/moby/moby.git /tmp/moby"
                  volumeMounts:
                          - name: workdir
                            mountPath: /tmp
        volumes:
                - name: workdir
                  emptyDir: {}
```

Pod #4 ConfigMap

- 애플리케이션을 배포할 때, 환경에 따라서 다른 설정값을 사용하는 경우가 발생한다.
  - Database IP, API KEY, 개발/운영에 따른 디버그 모드, 환경 설정 파일 등
- 애플리케이션 이미지가 같은데 환경 변수가 차이로 매번 다른 컨테이너 이미지를 만드는 것은 관리가 불편하다.
- ConfigMap/Secret 사용 시, 환경 변수나 설정값들을 변수로 관리해서 Pod가 생성될때 이 값을 넣어줄 수 있다.

- **ConfigMap 리소스 생성**

```
$ cat game.properties
weapon=gun
health=3
potion=5

$ kubectl create configmap game-config —from-file.properties
```

```
$ kubectl create configmap game-config \
                    —from-literal=weapon=gun \
                    —from-literal=health=3 \
                    —from-literal=potion=5
```

```
$ kubectl get configmap game-config -o yaml
apiVersion: v1
data:
  game.properties: |
    weapon=gun
    health=3
    potion=5
kind: ConfigMap
```

Pod #4 Secret

- ConfigMap 항목 중 보안이 중요한 요소들을 secret 으로 저장할 경우, node나 API Server 로 공유되지 않는다.
- ConfigMap 과 거의 유사하나, secret 생성 시 base64 포맷으로 value를 인코딩하여 저장해줘야한다.
- 평문을 k8s 에서 알아서 인코딩해줬으면 한다면 stringData property 사용

```
$ cat user-info.yaml
apiVersion: v1
kind: Secret
metadata:
        name: user-info
type: Opaque
data:
        username: YWRtaW4=
        password: cGFzc3dvcmQxMjM=
```

```
$ cat user-info-stringdata.yaml
apiVersion: v1
kind: Secret
metadata:
        name: user-info-stringdata
type: Opaque
stringData:
        username: admin
        password: password123
```

Pod #4 ConfigMap / Secret 연결

- ConfigMap / Secret 을 Pod로 넘기는 방법은 크게 두가지
  - Pod의 환경 변수 (Environment variable)로 넘기는 방법
  - Pod의 디스크 볼륨으로 마운트 하는 방법

```
$ cat game-volume.yaml
apiVersion: v1
kind: Pod
metadata:
        name: game-volume
spec:
        restartPolicy: OnFailure
        containers:
                - name: game-volume
                  image: k8s.gcr.io/busybox
                  command: ["/bin/sh", "-c", "cat /etc/config/game.properties" ]
                  volumeMounts:
                        - name: game-volume
                          mountPath: /etc/config
        volumes:
                - name: game-volume
                  configMap:
                        name: game-config
```

```
$ cat monster-env.yaml
apiVersion: v1
kind: Pod
metadata:
        name: monster-env
spec:
        restartPolicy: OnFailure
        containers:
                - name: monster-env
                  image: k8s.gcr.io/busybox
                  command: [ "printenv" ]
                  envFrom:
                        - configMapRef:
                                name: monster-config
```

Pod #4 ConfigMap / Secret 연결

- ConfigMap / Secret 을 Pod로 넘기는 방법은 크게 두가지
  - Pod의 환경 변수 (Environment variable)로 넘기는 방법
  - Pod의 디스크 볼륨으로 마운트 하는 방법

```
$ cat game-volume.yaml
apiVersion: v1
kind: Pod
metadata:
        name: game-volume
spec:
        restartPolicy: OnFailure
        containers:
                - name: game-volume
                  image: k8s.gcr.io/busybox
                  command: ["/bin/sh", "-c", "cat /etc/config/game.properties" ]
                  volumeMounts:
                        - name: game-volume
                          mountPath: /etc/config
        volumes:
                - name: game-volume
                  configMap:
                        name: game-config
```

```
$ cat monster-env.yaml
apiVersion: v1
kind: Pod
metadata:
        name: monster-env
spec:
        restartPolicy: OnFailure
        containers:
                - name: monster-env
                  image: k8s.gcr.io/busybox
                  command: [ "printenv" ]
                  envFrom:
                        - configMapRef:
                                name: monster-config
```

Pod #5 Downward API

- Pod의 메타데이터를 컨테이너에 전달하는 메커니즘
- ConfigMap, Secret 과 마찬가지로 volume 연결, 환경변수를 통해 컨테이너에 정보 전달 가능

```
$ cat downward-volume.yaml
apiVersion: v1
kind: Pod
metadata:
        name: downward-volume
        labels:
                zone: ap-north-east
                cluster: cluster1
spec:
        restartPolicy: OnFailure
        containers:
                - name: downward
                  image: k8s.gcr.io/busybox
                  command: [ "sh", "-c" ]
                  args: ["cat /etc/podinfo/labels"]
                  volumeMounts:
                        - name: podinfo
                          mountPath: /etc/podinfo
        volumes:
                - name: podinfo
                  downwardAPI:
                        items:
                                - path: "labels"
                                  fieldRef:
                                        fieldPath: metadata.labels
```

```
$ cat downward-env.yaml
apiVersion: v1
kind: Pod
metadata:
        name: downward-env
spec:
        restartPolicy: OnFailure
        containers:
                - name: downward
                  image: k8s.gcr.io/busybox
                  command: ["printenv"]
                  env:
                        - name: NODE_NAME
                          valueFrom:
                                fieldRef:
                                        fieldPath: spec.nodeName
                        - name: POD_NAME
                          valueFrom:
                                fieldRef:
                                        fieldPath: metadata.name
                        - name: POD_NAMESPACE
                          valueFrom:
                                fieldRef:
                                        fieldPath: metadata.namespace
                        - name: POD_IP
                          valueFrom:
                                fieldRef:
                                        fieldPath: status.podIP
```