Section 5: Application Lifecycle Management

97-99. Rolling Updates and Rollbacks

Rollout 명령어

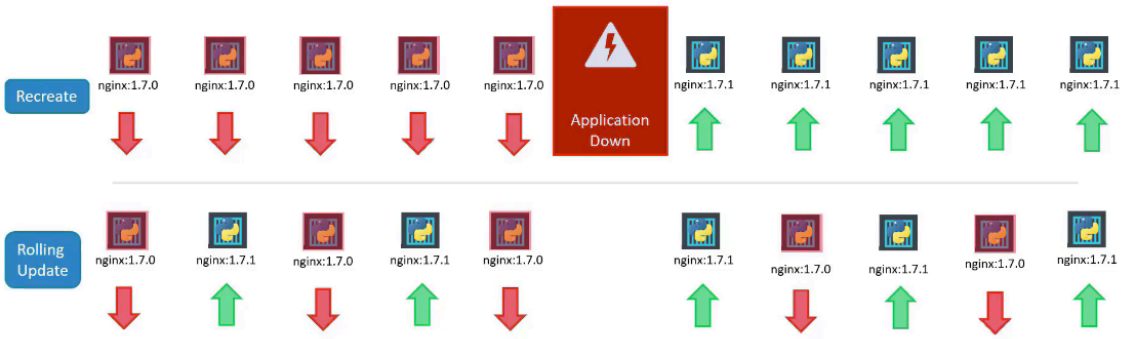
```
# rollout status를 확인
kubectl rollout status deployment/myapp-deployment

# rollout의 리비전과 history 확인
kubectl rollout history deployment/myapp-deployment

# rollout 취소 (이전 리비전으로 roll back)
kubectl rollout undo deployment/myapp-deployment
```

배포 전략

- Recreate : 모두 destroy 후 새 버전의 인스턴스 새로 배포
  - 구 버전이 다운되고 새 버전이 배포되기 전까지 사용자가 접근 불가능
- (default) Rolling Update : 하나씩 내리고 하나씩 새 버전 올림



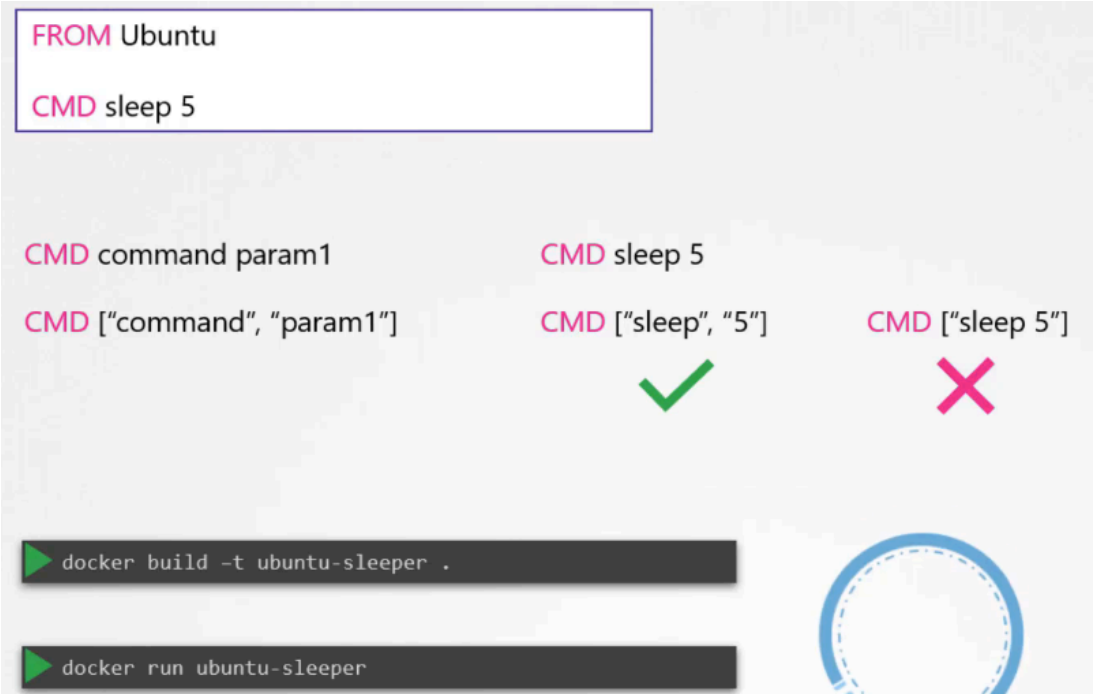
100. Configure Applications

애플리케이션 구성(Configuring)은 다음 개념을 이해하는 것으로 구성된다.

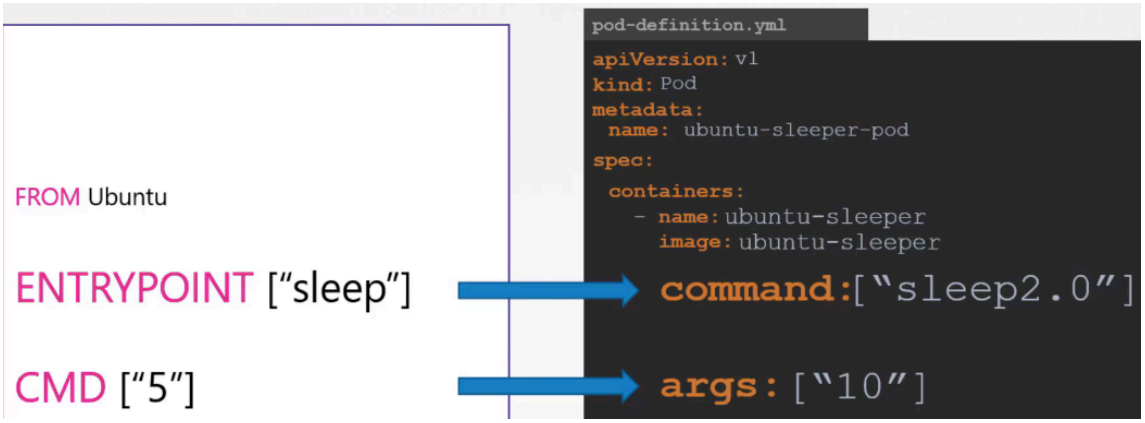
- Configuring Command and Arguments on applications
- Configuring Environment Variables
- Configuring Secrets

101-104. Commands and Arguments

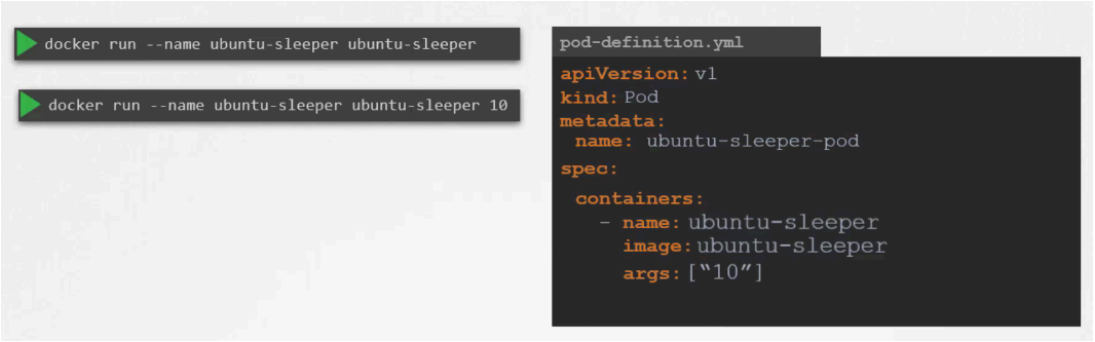
- JSON 배열 포맷으로 명령을 지정할 때 주의 : 명령어와 매개변수를 함께 지정하지 말 것



- 왼쪽은 docker file, 오른쪽은 쿠버네티스 pod 정의 yaml



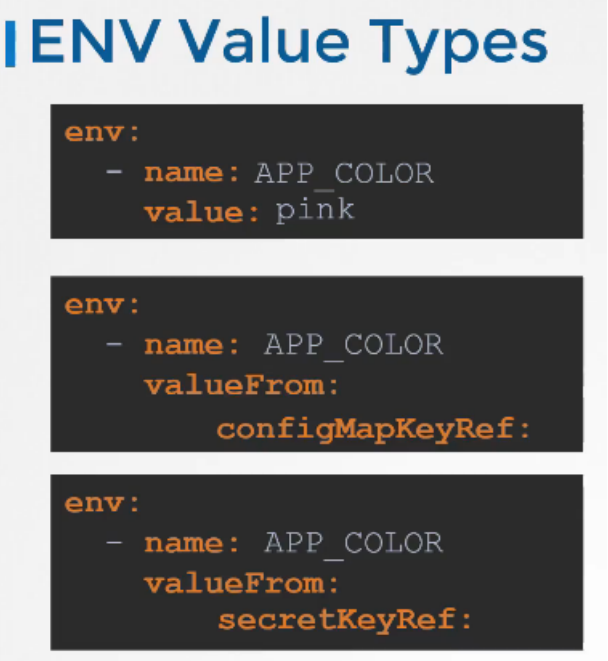
- docker file의 CMD 는 쿠버네티스에서 args로 재정의 가능 → ["10"] 형식으로 매개변수 명시



- docker 파일의 ENTRYPOINT 는 쿠버네티스에서는 command 필드 사용
  - docker file의 ENTRYPOINT를 sleep2.0으로 재정의하고 싶다면? → command: ["sleep2.0"]

105. Configure Environment Variables in Applications

- 환경변수 설정 : env 필드 사용
  - configMap, secret 을 이용해 환경 변수 설정 가능: value 대신 valueFrom 이라고 기입



106. Configuring ConfigMaps in Applications

- 각 pod 정의 yaml에 env 필드로 환경변수를 정의할 수 있지만, 파일이 많아질 경우 환경변수 관리가 어려워짐 → ConfigMap 사용하여 해결 가능!
  - ConfigMap 생성 후, 포드 정의 yaml의 `envFrom` 필드를 통해 ConfigMap을 적용해준다.
- ConfigMap 생성하는 두가지 방법:
  - Imperative (명령적 방법) - `kubectl create configmap`
    - `kubectl create configmap <config-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>`
    - `kubectl create configmap <config-name> --from-file=<path-to-file>`
  - Declarative (선언적 방법) - `kubectl create -f config-map.yaml`
    - config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: app-config
```

```
data:
  APP_COLOR: blue
  APP_MODE: prod
```

- ConfigMap 목록 조회 : `kubectl get configmaps`
- ConfigMap 상세 조회 : `kubectl describe configmaps`

109. Configure Secrets in Applications

- 비밀번호 등 민감 정보를 저장하는데 쓰임
- 데이터를 base64 형식으로 인코딩함
- Secret 생성하는 두가지 방법:
  - Imperative (명령적 방법) - `kubectl create secret generic app-secret`
    - `kubectl create secret generic <secret-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>`
    - `kubectl create secret generic <secret-name> --from-file=<path-to-file>`
  - Declarative (선언적 방법) - `kubectl create -f secret-data.yaml`
    - secret-data.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bXlzcWw= # "mysql" 을 인코딩한 값
  DB_User: cm9vdA== # "root"를 인코딩한 값
  DB_Password: cGFzd3Jk # "paswrd"를 인코딩한 값
```

base64로 인코딩 : `echo -n 'mysql' | base64`

base64로 인코딩된 값 디코딩 : `echo -n 'bXlzcWw=' | base64 --decode`

- Secret 목록 조회 : `kubectl get secrets`
- Secret 상세 조회 : `kubectl describe secrets`
- 파드에 적용 : pod.definition.yaml 의 spec.containers.envFrom 에 - secretRef: 정의

- 주의:
  - Secret은 암호화되어있지 않음! 그냥 인코딩되어있어서 누구나 해독해서 데이터를 확인할 수 있음
  - Secret은 etcd에 암호화되어있지 않음
    - EncryptionConfiguration 을 통해 저장된 데이터(at rest) 암호화 가능
  - 동일한 namespace에 pod 나 deployment를 생성할 수 있는 모든 사람은 secret에 접근하여 기밀 정보를 볼 수 있다.

        - RBAC를 구성해 Scret에 접근하기 위한 최소한의 특별 권한을 구성하도록 한다.
    - third-part secret store provider를 고려하자. ex) AWS Provider, Azure Provider, GCP Provider, Vault Provider

120. InitContainers

파드의 주 컨테이너가 시작되기 전에 실행되는 특수 컨테이너

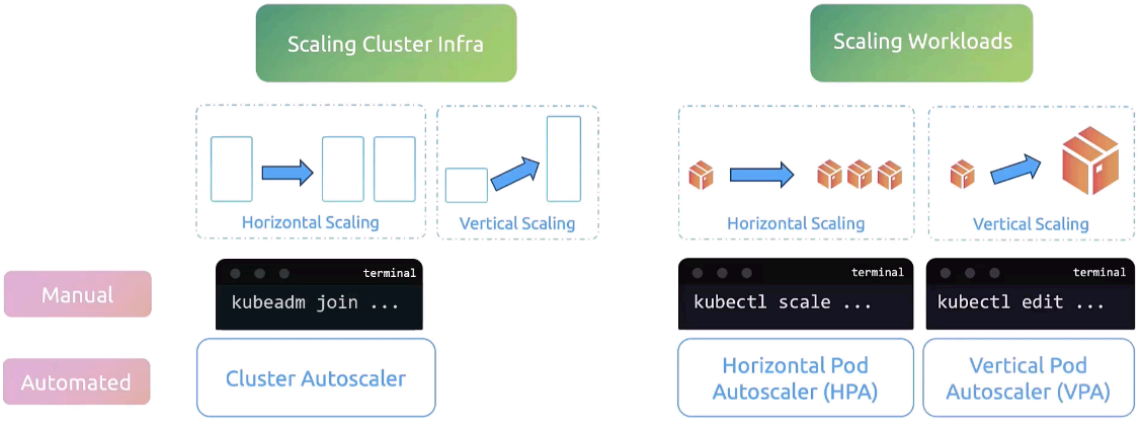
- 파드 생성 시 한 번만 실행되고 완료되어야 하는 작업에 사용됨
    - ex) 애플리케이션에 필요한 코드나 바이너리 다운로드, 외부 서비스나 데이터베이스가 준비될 때까지 대기
- initContainer가 반드시 완료되어야 주 컨테이너가 시작됨
- 여러 개의 initContainer가 있을 경우, 순차적으로 하나씩 실행됨
- initContainer가 실패하면 쿠버네티스는 성공할 때까지 파드를 반복해서 재시작함

일반 컨테이너 vs initContainers

- 일반 컨테이너: 파드 수명 주기 동안 계속 실행 예상 (ex. web application, logging agent)
- initContainers: 작업 완료 후 종료되도록 설계됨 (ex. 초기 설정 작업)

124. Autoscaling

- **Horizontal Scaling vs Vertical Scaling**
    - Vertical Scaling (수직 확장) : 기존 application에 더 많은 resource를 추가하는 것
    - Horizontal Scaling (수평 확장) : 서버를 추가하여 인스턴스의 개수를 늘리는 것
- 쿠버네티스와 같은 컨테이너 오케스트레이터의 주요 목적 중 하나는 컨테이너 형태로 애플리케이션을 호스팅하고 수요에 따라 확장 및 축소하는 것
- 쿠버네티스의 Scaling 유형 두가지
    - **워크로드** 확장 : 클러스터에 컨테이너나 파드를 추가/제거
        - 수평확장: 더 많은 파드 생성
        - 수직확장: 기존 파드에 할당된 리소스를 늘림
    - **기본 클러스터 자체를** 확장 : 클러스터에 서버나 인프라를 추가/제거
        - 수평확장: 클러스터에 노드를 추가
        - 수직확장: 클러스터의 기존 노드에서 리소스를 늘림
- 지금까지는 수동으로 (in Manual) 수평/수직 확장해줬는데, 이제 자동으로 하는 법을 배울 것



125. HPA (Horizontal Pod Autoscaler)

- workload를 수동으로 확장하는 방법
  1. `kubectl top pod my-app-pod` 를 통해 파드의 리소스 사용량 모니터링 후,
  2. yaml에 정의한 limit 값에 도달했거나 가까워지면 `kubectl scale deployment my-app --replicas=3` 등을 통해 파드를 추가
- 문제점: 관리자가 컴퓨터 앞에 앉아 리소스 사용량을 지속적으로 모니터링해줘야 하고, 수동으로 scale up/down 명령어를 실행해야 하며, 트래픽의 급증을 감당할만큼 빠르게 대응하지 못할 수 있음.
- 이를 해결하기 위해 HPA 사용한다! HPA는 수동으로 했던 1~2번을 대신 해준다.
  - `kubectl autoscale deployment my-app --cpu-percent=50 --min=1 --max=10`
  - hpa 목록 확인 : `kubectl get hpa`
  - hpa 삭제 : `kubectl delete hpa my-app`
  - 선언적 방법:
    - apiVersion: autoscaling/v2
    - kind: HorizontalPodAutoscaler
    - spec.scaleTargetRef 지정 : HPA가 모니터링하고자 하는 target resource
    - spec.minReplicas, spec.maxReplicas : 최소/최대 복제본 수
    - spec.metrics 지정 : 모니터링할 메트릭과 리소스

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
```

```
maxReplicas: 10
metrics:
- type: Resource
  resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 50   # 목표 사용률
```

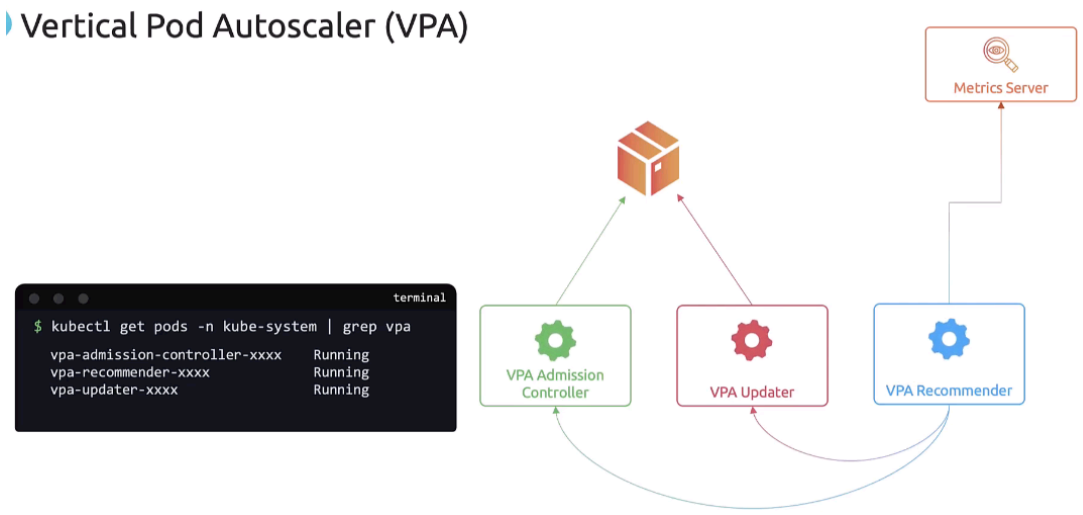
- 참고로, HPA는 쿠버네티스 버전 1.23부터 기본제공됨. 별도의 설치 필요 없음~
- 메트릭 서버 : 메트릭 서버는 리소스 사용 데이터를 클러스터 전반에 걸쳐 집계하는 시스템으로, HPA가 확장 결정을 내리는 데 필요한 메트릭을 제공한다.

128. In-place Resize of Pods

- `spec.template.spec.containers[0].resizePolicy` 로 각 리소스에 대한 restart 정책 지정 가능
  - 1.23 버전에서는 FEATURE_GATES=InPlacePodVerticalScaling=true 로 설정해야함.
  - 이를 설정하면 CPU 리소스를 변경해도 파드를 삭제 후 재시작해줄 필요 없이, 자동으로 재시작해줌. 그 다음, CPU를 하나로 업데이트 하는 등 리소스를 변경함.

129-131. VPA (Vertical Pod Autoscaling)

- HPA와 달리, VPA는 기본 제공되지 않음. 별도로 설치 필요



- VPA 구성요소
  1. VPA Recommender: 정보 수집
     - 컨테이너의 현재 및 과거 resource 소비(CPU 및 memory)를 지속적으로 모니터링
     - 관찰된 사용량을 바탕으로 컨테이너의 CPU 및 메모리 요청에 대한 권장 값을 제공한다
       - 이러한 권장 사항은 다른 구성 요소들이 컨테이너의 자원 할당을 조정하는 데 사용됨

- 목적: 컨테이너의 리소스 요청이 항상 실제 사용량을 기준으로 최적으로 설정되도록 하여 리소스의 over-provisioning과 under-provisioning을 방지하는 데 도움이 됨

2. VPA Updater: Recommender로부터 얻은 권장값과 실제 파드를 비교
   - Recommender의 제안에 따라 실행 중인 pod에 올바른 리소스 요청이 있는지 확인하는 역할
   - 관리되는 포드 중 어느 것이 오래되었거나 잘못된 리소스 설정을 가지고 있는지 확인
   - pod의 리소스를 업데이트해야 하는 경우, 업데이트된 리소스 요청을 통해 컨트롤러(ex. deployment, ReplicaSet)가 pod을 다시 생성할 수 있도록 Updater는 해당 pod을 삭제
   - 목적: 필요한 경우 업데이트된 요청으로 pod를 재시작하여 실행 중인 pod가 항상 권장 리소스를 확보할 수 있다
3. VPA Admission Plugin : 파드 생성 프로세스에 개입
   - 새 파드가 처음 생성되거나 VPA Updater의 작업으로 인해 다시 생성될 때 올바른 리소스 요청을 설정한다.
   - pod 생성 과정에서 작동하며, pod가 VPA에 의해 관리되는지 확인한다.
   - pod가 VPA에 의해 관리되는 경우, Recommender가 제공하는 권장 값을 반영하여 pod의 리소스 요청을 수정한다.
   - 목적: 새로 생성되거나 recreate된 pod가 처음부터 최적의 리소스 요청으로 시작하도록 보장

   이 세 가지 구성 요소는 함께 작동하여 Kubernetes pod에 동적 리소스 할당을 제공하여 리소스 사용을 최적화하고 클러스터 효율성을 향상시킨다.

- spec.resourcePolicy.containerPolicies : 모니터링 대상을 정의. CPU 최소 허용, CPU 최대 허용 설정 등
- spec.updatePolicy
  - 4가지 모드 : Off, Initial, Recreate, Auto

VPA vs HPA

	VPA	HPA
스케일링 방법	기존 파드의 CPU와 메모리를 늘리거나 새 리소스로 해당 파드를 다시 생성하는 방식	수요에 따라 파드를 추가하거나 제거
pod 동작	파드를 재시작하여 새 리소스 값을 적용	기존 포드를 계속 실행하고 새 포드를 간단히 스핀업하여 지속적인 가용성 보장
트래픽 급증 처리 방법	👎 일반적으로 파드를 다시 시작해야 하므로 지연이 발생하기 때문에 갑작스러운 스파이크를 효율적으로 처리할 수 없음	👍 수요가 증가하면 즉시 더 많은 파드를 추가할 수 있어 빠른 확장이 필요한 애플리케이션에 선호됨
비용 최적화 관점	👍 실제 사용량에 맞게 CPU 및 memory 할당을 조정하여 over-provisioning을 방지함	👍 불필요한 idle 포드를 방지하여 과도한 인스턴스를 계속 실행하지 않고, 리소스를 효율적으로 사용할 수 있도록 함
어떤 경우에 적합?	Stateful 워크로드 및 CPU 또는 메모리를 많이 사용하는 애플리케이션	변동하는 트래픽을 처리하기 위해 빠른 확장이 필요한 상태 stateless 서비스에 이상적
사용 예시	데이터베이스, JVM 기반 애플리케이션, 미세 조정된 리소스가 필요한 AI ML 워크로드 ex) 초기 시작 시에는 많은 CPU를 필요로 하지만 나중에는 많은 CPU가 필요하지 않은 어플리	웹 애플리케이션, 마이크로 서비스, 웹 서버(nginx, API services), message queues, API 기반 애플리케이션 등

	VPA	HPA
	케이션의 경우, 많은 CPU로 인스턴스를 시작하고 초기화 프로세스가 끝나면 할당된 CPU와 memory 사용량을 줄여야 할 때 VPA 사용	
요약	개별 파드에 대한 리소스 할당을 최적화	수요에 따라 동적으로 파드 수 확장