7단원. 쿠버네티스 기초 개념

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무(네트워크·클라우드·시스템·보안) 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

Q1. 쿠버네티스란 무엇이며 왜 필요한가요? ★★★

답안 쿠버네티스(**Kubernetes**)는 다수의 컨테이너를 여러 서버(노드)에 걸쳐 자동으로 배포, 확장, 관리해주는 컨테이너 오케스트레이션 플랫폼입니다. docker-compose가 단일 호스트에서의 멀티 컨테이너 관리에 초점을 둔다면, 쿠버네티스는 여러 대의 서버를 하나의 클러스터로 묶어 컨테이너가 어느 노드에서 실행될지 자동으로 스케줄링하고, 장애가 발생한 컨테이너를 자동으로 재시작하며, 트래픽 증가에 따라 컨테이너 수를 자동으로 조절합니다. 대규모 마이크로서비스 아키텍처에서는 수백, 수천 개의 컨테이너를 수동으로 관리하는 것이 불가능하기 때문에, 쿠버네티스가 제공하는 자동 복구(**Self-healing**), 선언적 배포, 오토스케일링 기능이 필수적입니다.

관련 개념 컨테이너 오케스트레이션, 선언적 구성(Declarative), Self-healing, 클러스터

Q2. Pod란 무엇이며 왜 컨테이너가 아닌 **Pod**가 쿠버네티스의 최소 배포 단위인가요? ★★★

답안 **Pod**는 쿠버네티스에서 배포할 수 있는 최소 단위로, 하나 이상의 컨테이너를 포함하며 같은 Pod 안의 컨테이너들은 네트워크(같은 IP, 포트 공간)와 스토리지(볼

륨)를 공유합니다. 컨테이너 자체가 아닌 Pod를 최소 단위로 삼은 이유는, 서로 밀접하게 연관되어 항상 함께 배포되고 함께 스케일링되어야 하는 컨테이너들(예: 메인 애플리케이션과 로그 수집 사이드카)을 하나의 논리적 단위로 묶어 관리할 필요가 있기 때문입니다. 대부분의 Pod는 하나의 컨테이너만 포함하지만, 사이드카 패턴처럼 보조 컨테이너를 함께 배치해야 하는 경우 멀티 컨테이너 Pod를 구성합니다. Pod는 기본적으로 일시적(Ephemeral)인 존재로, 삭제되면 새로운 IP를 가진 새 Pod로 교체된다는 특징이 있습니다.

관련 개념 사이드카 패턴, Pod 네트워크, 일시성(Ephemeral), 컨테이너 그룹

> **Q3. Node**와 **Cluster**의 개념을 설명하세요. ★★★

답안 노드(**Node**)는 쿠버네티스 클러스터를 구성하는 개별 서버(물리 서버 또는 VM)로, 실제 Pod가 배치되어 실행되는 작업 단위입니다. 노드는 역할에 따라 클러스터 전체를 관리하는 컨트롤 플레인(마스터) 노드와 실제 애플리케이션 워크로드가 실행되는 워커 노드로 구분됩니다. 클러스터(**Cluster**)는 이러한 여러 노드들을 하나로 묶어 관리하는 전체 시스템으로, 사용자는 개별 노드를 직접 다루기보다 클러스터 전체를 하나의 대상으로 삼아 애플리케이션을 배포하고 관리합니다. 워커 노드에는 컨테이너 런타임, kubelet, kube-proxy 같은 컴포넌트가 실행되어 컨트롤 플레인의 지시에 따라 실제 Pod를 실행하고 상태를 보고합니다.

관련 개념 컨트롤 플레인, 워커 노드, kubelet, kube-proxy

> **Q4.** 쿠버네티스 컨트롤 플레인의 주요 구성 요소를 설명하세요. ★★★

답안 컨트롤 플레인은 클러스터의 두뇌 역할을 하며 여러 컴포넌트로 구성됩니다. **API** 서버(**kube-apiserver**)는 모든 요청이 거쳐가는 유일한 진입점으로, 사용자와 다른 컴포넌트 간의 통신을 중개합니다. **etcd**는 클러스터의 모든 상태 정보(어떤 Pod가 어디에 있는지 등)를 저장하는 분산 키-값 저장소입니다. 스케줄러(**kube-scheduler)**는 새로 생성된 Pod를 자원 상황에 맞는 적절한 노드에 배치하는 역할

을 하고, 컨트롤러 매니저(**kube-controller-manager**)는 현재 상태를 원하는 상태(Desired State)와 지속적으로 비교하며 차이가 있으면 이를 조정하는 여러 컨트롤러(레플리케이션 컨트롤러 등)를 실행합니다. 이 컴포넌트들이 함께 동작하며 "선언한 상태를 계속 유지"하는 쿠버네티스의 핵심 원리를 구현합니다.

관련 개념 kube-apiserver, etcd, kube-scheduler, controller-manager, Desired State

---

> **Q5. Deployment**란 무엇이며 **ReplicaSet**과의 관계를 설명하세요. ★★★

답안 **ReplicaSet**은 지정된 개수의 Pod 복제본이 항상 유지되도록 보장하는 컨트롤러로, Pod가 예기치 않게 종료되면 자동으로 새 Pod를 생성해 개수를 맞춥니다. **Deployment**는 이 ReplicaSet을 한 단계 더 추상화한 상위 오브젝트로, 애플리케이션의 버전 관리와 배포 전략(롤링 업데이트, 롤백)을 담당합니다. 사용자가 Deployment의 이미지 버전을 변경하면, Deployment는 새로운 ReplicaSet을 생성하고 기존 ReplicaSet의 Pod 수를 점진적으로 줄이면서 새 ReplicaSet의 Pod 수를 늘리는 방식으로 무중단 롤링 업데이트를 수행합니다. 실무에서는 대부분 ReplicaSet을 직접 다루기보다 Deployment를 통해 애플리케이션을 배포하고 관리합니다.

관련 개념 ReplicaSet, 롤링 업데이트, 롤백(Rollback), Desired State 관리

> **Q6. Service**의 역할과 필요성을 설명하세요. ★★★

답안 Pod는 재시작되거나 재배치될 때마다 **IP** 주소가 바뀌는 일시적인 존재이기 때문에, 클라이언트가 특정 Pod의 IP를 직접 참조하면 연결이 끊어질 위험이 큽니다. **Service**는 여러 Pod 앞에 위치해 고정된 가상 **IP**와 **DNS** 이름을 제공하고, 그 뒤에 있는 Pod들에게 트래픽을 자동으로 분산해주는 오브젝트입니다. 즉 Service는 쿠버네티스 내부의 로드밸런서 역할을 하며, Label Selector를 통해 어떤 Pod들이 이 Service의 대상이 되는지를 지정합니다. 클라이언트는 개별 Pod의 생성·삭제와

무관하게 항상 동일한 Service 이름이나 IP로 접근할 수 있어, Pod의 동적인 생명주기로부터 클라이언트를 격리시키는 추상화 계층 역할을 합니다.

관련 개념 Label Selector, ClusterIP, NodePort, LoadBalancer 타입, 쿠버네티스 DNS

> **Q7. Service**의 타입(**ClusterIP, NodePort, LoadBalancer**)을 비교 설명하세요. ★★

답안 **ClusterIP**는 기본 타입으로 클러스터 내부에서만 접근 가능한 가상 IP를 부여하며, 주로 애플리케이션 계층 간 내부 통신에 사용됩니다. **NodePort**는 모든 워커 노드의 특정 포트를 열어 외부에서 `노드IP:포트`로 접근할 수 있게 해주는 타입으로, 간단하게 외부 노출이 가능하지만 노드 IP가 바뀌거나 포트 범위 제약이 있어 프로덕션에서는 잘 사용되지 않습니다. **LoadBalancer**는 클라우드 제공업체의 외부 로드밸런서(예: AWS ALB/NLB)를 자동으로 프로비저닝해 외부 트래픽을 클러스터로 유입시키는 타입으로, 프로덕션 환경에서 외부에 서비스를 노출할 때 가장 일반적으로 사용됩니다. 이 외에 여러 서비스를 경로 기반으로 라우팅하는 **Ingress**를 함께 사용해 하나의 진입점으로 여러 서비스를 관리하기도 합니다.

관련 개념 Ingress, 클라우드 로드밸런서 연동, 외부 노출 전략, 포트 범위

> **Q8.** 쿠버네티스 스케줄링의 기본 원리를 설명하세요. ★★

답안 새로운 Pod가 생성되면 아직 어떤 노드에도 배정되지 않은 상태(Pending)이며, **kube-scheduler**가 이를 감지해 적절한 노드를 선택하는 과정을 거칩니다. 스케줄링은 크게 두 단계로 이루어지는데, 먼저 필터링(**Filtering**) 단계에서 자원이 부족하거나 조건에 맞지 않는 노드를 후보에서 제외하고, 이후 스코어링(**Scoring**) 단계에서 남은 후보 노드들에 점수를 매겨 가장 적합한 노드를 선택합니다. 이때 Pod가 요청한 CPU/메모리 자원(`requests`), 특정 노드에만 배치되도록 지정하는 **nodeSelector/Affinity**, 특정 노드에는 배치되지 않도록 하는 **Taint/**

Toleration 같은 설정이 스케줄링 결과에 영향을 줍니다. 스케줄러가 노드를 결정하면 해당 노드의 kubelet이 실제로 컨테이너를 실행합니다.

관련 개념 requests/limits, Node Affinity, Taint/Toleration, kubelet

---

> **Q9. Namespace**의 역할을 설명하세요. ★★

답안 **Namespace**는 하나의 물리적인 쿠버네티스 클러스터 안에서 리소스를 논리적으로 격리된 여러 가상 클러스터처럼 분리해 관리할 수 있게 해주는 개념입니다. 예를 들어 개발, 스테이징, 운영 환경을 하나의 클러스터 안에서 각각 다른 네임스페이스로 분리해 운영하거나, 팀별로 네임스페이스를 나눠 자원 사용량과 접근 권한을 독립적으로 관리할 수 있습니다. 네임스페이스 단위로 리소스 쿼터 **(ResourceQuota)**를 설정해 특정 팀이 사용할 수 있는 CPU·메모리 총량을 제한할 수 있고, RBAC(역할 기반 접근제어)와 결합해 네임스페이스별로 접근 권한을 세밀하게 통제할 수 있습니다. 다만 노드나 퍼시스턴트 볼륨처럼 클러스터 전체에 걸친 일부 리소스는 네임스페이스로 격리되지 않는다는 점도 알아둘 필요가 있습니다.

관련 개념 ResourceQuota, RBAC, 멀티 테넌시, 클러스터 전역 리소스

> **Q10.** 쿠버네티스의 선언적(**Declarative**) 관리 방식이 명령형 **(Imperative)** 방식과 다른 점을 설명하세요. ★★

답안 명령형 방식은 "무엇을 어떻게 할지" 단계별 명령을 직접 실행하는 방식으로, 예를 들어 `kubectl run`, `kubectl scale` 같은 명령어로 즉시 특정 동작을 수행합니다. 선언적 방식은 YAML 매니페스트에 "최종적으로 어떤 상태가 되어야 하는지(Desired State)"만 기술하고, `kubectl apply`로 이를 클러스터에 적용하면 쿠버네티스의 컨트롤러들이 현재 상태를 그 목표 상태와 지속적으로 비교하며 스스로 필요한 조치를 취합니다. 선언적 방식의 장점은 매니페스트 파일 자체가 시스템의 상태를 코드로 기록하는 **IaC(Infrastructure as Code)** 역할을 해 버전 관리와 재현이 쉽고, 컨트롤러가 지속적으로 상태를 감시하며 자동으로 복구해주기 때문에 운

영 안정성이 높다는 점입니다. 실무에서는 대부분 선언적 방식을 기본으로 사용하며 GitOps 같은 워크플로우와도 잘 결합됩니다.

관련 개념 Desired State, kubectl apply, IaC, GitOps, 컨트롤 루프
