↑ 목록으로 ← 이전 단원

8단원. 쿠버네티스 실무 마스터

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

Q1. **ConfigMap**과 **Secret**의 차이와 사용 목적을 설명하세요.

★★★

답안 **ConfigMap**은 애플리케이션의 설정값(환경 변수, 설정 파일 등)처럼 민감하지 않은 구성 정보를 코드와 분리해 관리하기 위한 오브젝트로, Pod에 환경변수나 볼륨 파일 형태로 주입할 수 있습니다. **Secret**은 비밀번호, API 키, 인증서처럼 민감한 정보를 저장하기 위한 오브젝트로, 기본적으로 base64로 인코딩되어 저장되지만 이는 암호화가 아니라 인코딩일 뿐이므로 etcd 암호화나 외부 시크릿 관리 도구와 함께 사용해야 실질적인 보안이 확보됩니다. 두 오브젝트 모두 설정 정보를 애플리케이션 이미지와 분리함으로써, 동일한 이미지를 서로 다른 환경(개발/운영)에 재사용할 수 있게 해주는 것이 핵심 목적입니다. 실무에서는 Secret을 etcd 암호화, RBAC 접근 제어와 함께 사용하고, 더 나아가 Vault 등의 외부 시크릿 관리 서비스와 연동하는 것이 권장됩니다.

관련 개념 base64 인코딩, etcd 암호화, 환경변수 주입, 볼륨 마운트, 외부 시크릿 연동

> **Q2. HPA(Horizontal Pod Autoscaler)**의 동작 원리를 설명하세요. ★★★

답안 **HPA**는 CPU 사용률이나 메모리 사용률, 또는 사용자 정의 메트릭을 기준으로 **Pod**의 개수를 자동으로 늘리거나 줄이는 오토스케일링 기능입니다. HPA는 주기적으로 메트릭 서버(Metrics Server)로부터 대상 Pod들의 현재 리소스 사용률을 수집하고, 이를 사용자가 설정한 목표 값과 비교해 필요한 Pod 개수를 계산한 뒤 Deployment나 ReplicaSet의 replica 수를 조정합니다. 예를 들어 목표 CPU 사용률을 50%로 설정했는데 실제 평균 사용률이 80%라면, HPA는 replica 수를 늘려 평균 사용률을 목표치에 가깝게 낮추려고 시도합니다. 다만 HPA가 효과적으로 동작하려면 Pod에 `requests`(자원 요청량)가 반드시 설정되어 있어야 사용률 계산이 가능하며, 급격한 스케일링을 방지하기 위한 안정화 기간(Stabilization Window) 설정도 함께 고려해야 합니다.

관련 개념 Metrics Server, requests 설정, Stabilization Window, VPA(수직 오토스케일러)

> **Q3.** 롤링 업데이트(**Rolling Update**)의 동작 방식과 무중단 배포 원리를 설명하세요. ★★★

답안 롤링 업데이트는 Deployment의 기본 배포 전략으로, 기존 버전의 Pod를 한꺼번에 종료하지 않고 점진적으로 새 버전 **Pod**를 추가하면서 기존 **Pod**를 하나씩 제거하는 방식으로 진행됩니다. 이 과정에서 `maxSurge`(원래 개수보다 추가로 생성할 수 있는 최대 Pod 수)와 `maxUnavailable`(동시에 사용 불가능해도 되는 최대 Pod 수) 값을 조절해 배포 속도와 안정성의 균형을 맞출 수 있습니다. 새로 생성된 Pod는 **readiness probe**를 통과해야 Service의 트래픽 대상에 포함되기 때문에, 아직 준비되지 않은 새 버전 Pod로 트래픽이 유입되는 것을 방지할 수 있습니다. 배포 중 문제가 발생하면 `kubectl rollout undo` 명령으로 이전 ReplicaSet으로 즉시 롤백할 수 있어, 이러한 메커니즘 전체가 결합되어 서비스 중단 없는 배포를 가능하게 합니다.

관련 개념 maxSurge/maxUnavailable, readiness probe, rollout undo, 카나리 배포

Q4. **Liveness Probe**와 **Readiness Probe**의 차이를 설명하세요. ★★★

답안 **Liveness Probe**는 컨테이너가 여전히 살아서 정상 동작하고 있는지를 주기적으로 확인하는 헬스체크로, 이 검사에 실패하면 kubelet이 해당 컨테이너가 비정상 상태(예: 데드락에 빠짐)라고 판단해 컨테이너를 재시작합니다. **Readiness Probe**는 컨테이너가 트래픽을 받을 준비가 되었는지를 확인하는 헬스체크로, 이 검사에 실패하면 컨테이너를 재시작하지는 않지만 해당 Pod를 **Service**의 엔드포인트에서 일시적으로 제외해 트래픽이 전달되지 않도록 합니다. 예를 들어 애플리케이션이 시작 후 초기 데이터 로딩에 시간이 걸리는 경우, 그 기간 동안 readiness probe를 실패시켜 트래픽을 받지 않게 하다가 준비가 완료되면 다시 트래픽을 받도록 할 수 있습니다. 두 프로브를 적절히 함께 사용하는 것이 안정적인 서비스 운영과 무중단 배포의 핵심입니다.

관련 개념 헬스체크, 컨테이너 재시작, 엔드포인트 제외, startupProbe

Q5. **NetworkPolicy**란 무엇이며 어떤 문제를 해결하나요? ★★★

답안 기본적으로 쿠버네티스 클러스터 내의 모든 Pod는 네임스페이스와 무관하게 서로 자유롭게 통신할 수 있는데, 이는 편리하지만 특정 Pod가 침해당했을 때 공격이 클러스터 내부 전체로 확산될 위험을 키웁니다. **NetworkPolicy**는 Label Selector를 이용해 특정 Pod 그룹에 대해 허용할 인바운드(Ingress)와 아웃바운드(Egress) 트래픽 규칙을 정의함으로써, 필요한 통신만 허용하고 나머지는 차단하는 마이크로 세그멘테이션을 구현하는 오브젝트입니다. 예를 들어 데이터베이스 Pod는 애플리케이션 계층 Pod로부터의 특정 포트 접근만 허용하고 그 외 모든 트래픽은 차단하도록 설정할 수 있습니다. 다만 NetworkPolicy는 이를 지원하는

CNI(Container Network Interface) 플러그인(예: Calico, Cilium)이 클러스터에 설치되어 있어야 실제로 동작한다는 점에 유의해야 합니다.

관련 개념 마이크로 세그멘테이션, CNI 플러그인, Ingress/Egress 규칙, 최소 권한 네트워크

Q6. PersistentVolume(PV)과 PersistentVolumeClaim(PVC)의 관계를 설명하세요. ★★

답안 **PersistentVolume(PV)**은 클러스터 관리자가 미리 준비해두거나 스토리지 클래스에 의해 동적으로 프로비저닝되는 실제 스토리지 자원을 나타내는 오브젝트입니다. **PersistentVolumeClaim(PVC)**은 사용자(개발자)가 "이만큼의 용량과 특정 접근 모드를 가진 스토리지가 필요하다"고 요청하는 오브젝트로, 쿠버네티스는 이 요청에 맞는 PV를 찾아 자동으로 연결(바인딩)합니다. 이러한 분리 구조 덕분에 애플리케이션 개발자는 실제 스토리지가 NFS인지, 클라우드 블록 스토리지인지 등 구체적인 구현을 알 필요 없이 PVC만 선언하면 되고, 인프라 관리자는 스토리지 프로비저닝을 별도로 관리할 수 있습니다. Pod가 삭제되어도 PV의 데이터를 유지할지 여부는 `Reclaim Policy`(Retain, Delete 등) 설정에 따라 결정됩니다.

관련 개념 StorageClass, 동적 프로비저닝, Reclaim Policy, 접근 모드 (ReadWriteOnce 등)

Q7. StatefulSet과 Deployment의 차이를 설명하세요. ★★

답안 **Deployment**는 관리하는 Pod들이 서로 동일하고 상호 교체 가능하다고 가정하기 때문에, Pod 이름이 임의로 생성되고 삭제 시 순서를 보장하지 않으며 저장소도 공유되거나 임시적입니다. **StatefulSet**은 데이터베이스나 분산 시스템처럼 각 **Pod**가 고유한 정체성(고정된 이름, 고정된 네트워크 식별자)과 독립적인 영속 스토리지를 가져야 하는 애플리케이션을 위한 오브젝트입니다. StatefulSet이 관리하는 Pod는 `앱이름-0`, `앱이름-1`처럼 순서가 있는 이름을 부여받고, 생성과 삭제도 순차적으로 이루어지며, 각 Pod는 자신만의 PVC를 가져 재시작되어도 동일한 스토리

지에 다시 연결됩니다. 이러한 특성 때문에 MySQL 클러스터, Kafka, Elasticsearch처럼 노드 간 순서와 정체성이 중요한 분산 스테이트풀 애플리케이션 배포에 StatefulSet이 사용됩니다.

관련 개념 순서 보장, 고정 네트워크 식별자, Headless Service, 분산 스테이트풀 애플리케이션

> **Q8. DaemonSet**의 용도를 설명하세요. ★★

답안 **DaemonSet**은 클러스터 내의 모든(또는 특정 조건에 맞는) 노드마다 정확히 하나씩 특정 Pod를 실행하도록 보장하는 오브젝트입니다. 새로운 노드가 클러스터에 추가되면 자동으로 해당 노드에도 DaemonSet Pod가 생성되고, 노드가 제거되면 해당 Pod도 함께 정리됩니다. 대표적인 사용 사례로는 각 노드의 로그를 수집하는 로그 에이전트(Fluentd 등), 각 노드의 자원 사용률을 모니터링하는 메트릭 수집 에이전트(Node Exporter 등), 그리고 각 노드에 네트워크 기능을 제공하는 CNI 플러그인이나 kube-proxy 등이 있습니다. 이는 "노드 단위로 반드시 실행되어야 하는 인프라성 워크로드"를 배포할 때 표준적으로 사용되는 패턴입니다.

관련 개념 노드당 하나씩 배포, 로그/모니터링 에이전트, kube-proxy, 인프라성 워크로드

> **Q9.** 쿠버네티스에서 리소스 **requests**와 **limits**의 차이와 중요성을 설명하세요. ★★★

답안 **requests**는 컨테이너가 정상 동작을 위해 최소한으로 필요로 하는 자원량을 나타내며, 스케줄러가 Pod를 어느 노드에 배치할지 결정할 때 이 값을 기준으로 해당 노드에 충분한 여유 자원이 있는지 판단합니다. **limits**는 컨테이너가 사용할 수 있는 최대 자원량으로, 이를 초과해 메모리를 사용하면 OOM Killer에 의해 컨테이너가 강제 종료되고, CPU의 경우 초과 사용 시 스로틀링(제한)이 걸립니다. 이 두 값을 설정하지 않으면 특정 Pod가 노드의 자원을 과도하게 점유해 같은 노드의 다른 Pod에 영향을 줄 수 있는 **Noisy Neighbor** 문제가 발생할 수 있으므로, 프로덕션

환경에서는 반드시 적절한 requests/limits를 설정하는 것이 권장됩니다. 또한 앞서 다룬 HPA도 requests 값을 기준으로 사용률을 계산하기 때문에, 이 설정은 오토스케일링의 정확도에도 직접적인 영향을 미칩니다.

관련 개념 QoS 클래스(Guaranteed/Burstable/BestEffort), OOM Killer, CPU 스로틀링, Noisy Neighbor

---

> **Q10.** 쿠버네티스 클러스터의 노드 장애 시 어떤 방식으로 복구가 이루어지나요? ★★

답안 각 워커 노드는 주기적으로 컨트롤 플레인에 상태(Heartbeat)를 보고하는데, 특정 노드로부터 일정 시간 이상 응답이 없으면 컨트롤 플레인이 해당 노드를 **NotReady** 상태로 표시합니다. 이후 일정 유예 시간이 지나도 노드가 복구되지 않으면, 해당 노드에서 실행 중이던 Pod들은 자동으로 축출(**Eviction**)되고, 그 Pod들을 관리하던 ReplicaSet이나 Deployment 컨트롤러가 이를 감지해 다른 정상 노드에 동일한 개수의 새 **Pod**를 다시 생성합니다. 이 과정에서 Service는 Label Selector를 통해 새로 생성된 Pod를 자동으로 엔드포인트에 포함시키므로 클라이언트는 특별한 조치 없이도 계속 서비스를 이용할 수 있습니다. 다만 StatefulSet처럼 특정 스토리지에 종속된 Pod의 경우 복구 시간이 더 걸릴 수 있으며, PV의 종류(예: 특정 AZ에 종속된 볼륨)에 따라 다른 AZ의 노드로는 즉시 이동하지 못할 수도 있다는 점을 함께 언급할 수 있습니다.

관련 개념 Node Heartbeat, NotReady, Pod Eviction, 컨트롤러의 자동 복구, AZ 종속 볼륨
