↑ 목록으로 ← 이전 단원 다음 단원 →

5단원. **Docker** 기초

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

Q1. 컨테이너와 가상머신(**VM**)의 차이를 설명하세요. ★★★

답안 **VM**은 하이퍼바이저 위에서 각각 독립된 게스트 **OS** 전체를 실행하기 때문에 격리 수준이 높지만 이미지 용량이 크고 부팅에 수십 초 이상이 걸립니다. 컨테이너는 호스트 OS의 커널을 공유하면서 프로세스, 파일시스템, 네트워크 등을 격리된 네임스페이스로 분리해 실행하기 때문에 이미지 용량이 작고 수 초 이내에 시작할 수 있습니다. 이러한 차이로 컨테이너는 하나의 호스트에서 훨씬 더 많은 인스턴스를 동시에 실행할 수 있어 자원 효율이 뛰어나며, 마이크로서비스 아키텍처와 궁합이 좋습니다. 다만 커널을 공유하기 때문에 완전히 다른 OS(예: 리눅스 컨테이너를 윈도우 커널에서)를 직접 실행할 수는 없고, VM 대비 격리 수준이 상대적으로 낮다는 한계가 있습니다.

관련 개념 네임스페이스, cgroups, 커널 공유, 부팅 속도, 마이크로서비스

Q2. Docker 이미지와 레이어(**Layer**)의 개념을 설명하세요. ★★★

답안 Docker 이미지는 컨테이너 실행에 필요한 애플리케이션 코드, 라이브러리, 환경설정을 모두 포함한 읽기 전용 템플릿입니다. 이미지는 Dockerfile의 각 명령어(RUN, COPY 등)마다 하나씩 생성되는 레이어들이 쌓여 만들어지며, 각 레이어는

이전 레이어와의 차이만 저장하는 방식으로 동작합니다. 이러한 레이어 구조 덕분에 여러 이미지가 동일한 레이어를 공유(캐싱)할 수 있어 저장 공간을 절약하고, 이미지를 빌드할 때 변경되지 않은 레이어는 캐시를 재사용해 빌드 속도를 크게 높일 수 있습니다. 컨테이너가 실행될 때는 이 읽기 전용 레이어들 위에 쓰기 가능한 컨테이너 레이어가 하나 추가되어, 컨테이너 내부에서 발생하는 변경사항은 이 최상위 레이어에만 기록됩니다.

관련 개념 Union File System, 빌드 캐시, Copy-on-Write, 읽기 전용 레이어

> **Q3. Dockerfile**의 주요 명령어(**FROM, RUN, COPY, CMD** 등)를 설명하세요. ★★★

답안 `FROM`은 이미지의 베이스 이미지를 지정하는 명령어로 Dockerfile의 첫 줄에 위치합니다. `RUN`은 이미지 빌드 시점에 실행할 명령어(패키지 설치 등)를 정의하며 실행 결과가 새로운 레이어로 저장됩니다. `COPY`는 호스트의 파일이나 디렉토리를 이미지 내부로 복사하고, `ADD`는 COPY와 유사하지만 압축 해제나 URL 다운로드 같은 추가 기능을 지원합니다. `CMD`는 컨테이너가 실행될 때 기본으로 수행할 명령어를 지정하며 `docker run` 시 인자로 덮어쓸 수 있는 반면, `ENTRYPOINT`는 컨테이너의 주 실행 파일을 고정하는 용도로 사용됩니다. 이 외에 `WORKDIR`(작업 디렉토리 지정), `EXPOSE`(사용 포트 명시), `ENV`(환경변수 설정)도 자주 사용되는 명령어입니다.

관련 개념 베이스 이미지, CMD vs ENTRYPOINT, WORKDIR, 빌드 컨텍스트

> **Q4. Docker** 볼륨(**Volume**)이 필요한 이유와 종류를 설명하세요. ★★★

답안 컨테이너는 삭제되면 컨테이너 레이어에 기록된 모든 데이터가 함께 사라지는 휘발성(**Stateless**) 특성을 갖기 때문에, 데이터베이스 파일처럼 영속적으로 유지해야 하는 데이터는 컨테이너 외부에 별도로 저장해야 합니다. 이를 해결하는 것이 볼륨으로, 대표적으로 Docker가 관리하는 전용 저장 영역을 사용하는 볼륨

(Volume) 방식, 호스트의 특정 디렉토리를 컨테이너에 직접 연결하는 바인드 마운트(**Bind Mount**) 방식, 그리고 메모리 기반의 임시 저장 영역인 **tmpfs** 마운트가 있습니다. 볼륨 방식은 Docker가 생명주기를 관리해 백업이나 마이그레이션이 용이하고 여러 컨테이너 간 데이터 공유에도 적합해 프로덕션 환경에서 권장되며, 바인드 마운트는 개발 중 소스코드를 실시간으로 반영할 때 유용합니다.

관련 개념 바인드 마운트, tmpfs, 데이터 영속성, 컨테이너 생명주기

> **Q5. Docker**의 네트워크 모드(**bridge, host, none**)를 설명하세요. ★★★

답안 **bridge**는 Docker의 기본 네트워크 모드로, 컨테이너마다 격리된 가상 네트워크 인터페이스를 부여하고 내부 브리지를 통해 호스트 및 다른 컨테이너와 통신하며, 외부와 통신하려면 포트 매핑(`-p`)이 필요합니다. **host** 모드는 컨테이너가 별도의 네트워크 네임스페이스를 갖지 않고 호스트의 네트워크를 그대로 공유하는 방식으로, 포트 매핑 없이 호스트의 포트를 바로 사용할 수 있어 성능은 좋지만 격리성이 낮아지고 포트 충돌 위험이 있습니다. **none** 모드는 컨테이너에 네트워크 인터페이스를 아예 부여하지 않아 외부와 통신이 완전히 차단되며, 네트워크가 필요 없는 배치 작업 등에 사용됩니다. 이 외에도 여러 컨테이너를 같은 네트워크 네임스페이스로 묶는 **container** 모드나 사용자 정의 브리지 네트워크도 실무에서 자주 사용됩니다.

관련 개념 포트 매핑, 사용자 정의 브리지, 네트워크 네임스페이스, 컨테이너 간 통신

> **Q6. Docker** 이미지 태그(**Tag**)와 **latest** 태그 사용 시 주의점을 설명하세요. ★★

답안 Docker 이미지의 태그는 동일한 이미지 이름 안에서 버전을 구분하는 식별자로, `이미지명:태그` 형식으로 표기됩니다. `latest`는 태그를 명시하지 않았을 때 기본으로 적용되는 태그일 뿐 "항상 최신 버전"을 보장하는 특별한 의미를 가진 것이 아니며, 빌드 시점에 따라 실제 내용이 계속 바뀔 수 있습니다. 프로덕션 환경에서

`latest`를 그대로 사용하면 어떤 버전이 배포되어 있는지 추적이 어렵고, 재배포 시 의도치 않게 다른 버전이 배포될 위험이 있습니다. 따라서 실무에서는 커밋 해시나 시맨틱 버전(예: `v1.2.3`)처럼 명시적이고 고유한 태그를 사용해 배포 이력을 추적 가능하게 관리하는 것이 권장됩니다.

관련 개념 시맨틱 버저닝, 이미지 다이제스트(digest), 태그 불변성, 배포 추적성

> **Q7. Docker** 컨테이너 격리에 사용되는 리눅스 커널 기술 **(namespace, cgroups)**을 설명하세요. ★★

답안 Docker 컨테이너의 격리는 리눅스 커널의 두 가지 핵심 기능으로 구현됩니다. 네임스페이스(**Namespace**)는 프로세스, 네트워크, 마운트, 사용자 등 시스템 자원을 논리적으로 분리해 각 컨테이너가 독립된 환경(자신만의 PID 목록, 자신만의 네트워크 인터페이스 등)을 가진 것처럼 보이게 합니다. **cgroups(Control Groups)**는 CPU, 메모리, 디스크 I/O 같은 물리적 자원의 사용량을 제한하고 격리하는 기능으로, 특정 컨테이너가 과도한 자원을 사용해 다른 컨테이너나 호스트에 영향을 주지 않도록 통제합니다. 즉 네임스페이스가 "무엇이 보이는가"를 격리한다면 cgroups는 "얼마나 사용할 수 있는가"를 제어하는 역할을 하며, 이 두 기술의 조합이 컨테이너 격리의 근간을 이룹니다.

관련 개념 PID 네임스페이스, 네트워크 네임스페이스, 자원 제한, chroot

> **Q8. Docker Hub**와 프라이빗 레지스트리의 역할을 설명하세요. ★★

답안 **Docker Hub**는 Docker에서 공식 제공하는 퍼블릭 이미지 저장소로, 공식 베이스 이미지(Ubuntu, Nginx, MySQL 등)나 커뮤니티가 배포한 이미지를 검색하고 내려받을 수 있는 공간입니다. 다만 기업 내부에서 개발한 애플리케이션 이미지를 외부에 공개된 저장소에 올리는 것은 보안상 바람직하지 않기 때문에, 조직 내부 전용으로 프라이빗 레지스트리를 구축하거나 클라우드 제공업체의 관리형 레지스트리(예: ECR)를 사용합니다. 프라이빗 레지스트리는 접근 권한을 세밀하게 통제할 수 있

고, CI/CD 파이프라인과 연동해 빌드된 이미지를 자동으로 푸시·배포하는 워크플로우의 핵심 구성요소로 사용됩니다.

관련 개념 이미지 push/pull, ECR, 이미지 스캐닝, CI/CD 연동

> **Q9. Dockerfile** 작성 시 이미지 크기를 최적화하는 방법을 설명하세요. ★★

답안 이미지 크기를 줄이기 위한 대표적인 방법은 먼저 알파인 리눅스(**Alpine**)처럼 경량화된 베이스 이미지를 선택하는 것입니다. 또한 여러 개의 `RUN` 명령어를 `&&`로 연결해 하나의 레이어로 묶으면 불필요한 중간 레이어 생성을 줄일 수 있고, 패키지 설치 후 캐시 파일이나 임시 파일을 같은 레이어 안에서 즉시 삭제해야 실제로 이미지 크기가 줄어듭니다. 빌드에만 필요한 도구와 실행에 필요한 파일을 분리하는 멀티스테이지 빌드를 사용하면 최종 이미지에는 실행에 필요한 산출물만 남길 수 있어 크기를 크게 줄일 수 있습니다. 또한 `.dockerignore` 파일을 작성해 불필요한 파일이 빌드 컨텍스트에 포함되지 않도록 하는 것도 빌드 속도와 이미지 크기 관리에 도움이 됩니다.

관련 개념 Alpine 이미지, 멀티스테이지 빌드, .dockerignore, 레이어 병합

> **Q10.** 컨테이너의 상태(**state**)를 저장하지 않는 **Stateless** 설계가 중요한 이유는 무엇인가요? ★

답안 컨테이너는 언제든 재시작되거나 다른 노드로 재배치될 수 있기 때문에, 애플리케이션의 상태(세션 정보, 업로드 파일 등)를 컨테이너 내부에만 저장하면 컨테이너가 사라질 때 데이터도 함께 유실됩니다. **Stateless** 설계는 애플리케이션 자체는 상태를 갖지 않도록 하고, 세션 정보는 Redis 같은 외부 저장소에, 파일은 S3 같은 외부 스토리지에, 영속 데이터는 별도의 데이터베이스에 저장하는 방식입니다. 이렇게 설계하면 컨테이너를 자유롭게 확장(스케일 아웃)하거나 장애 시 새 컨테이너로 즉시 교체할 수 있어 쿠버네티스 같은 오케스트레이션 환경에서 무중단 배포와

자동 복구가 가능해집니다. 결과적으로 Stateless 설계는 컨테이너 기반 아키텍처의 확장성과 복원력을 뒷받침하는 핵심 원칙입니다.

관련 개념 외부 세션 저장소, 12 Factor App, 무중단 배포, 오케스트레이션