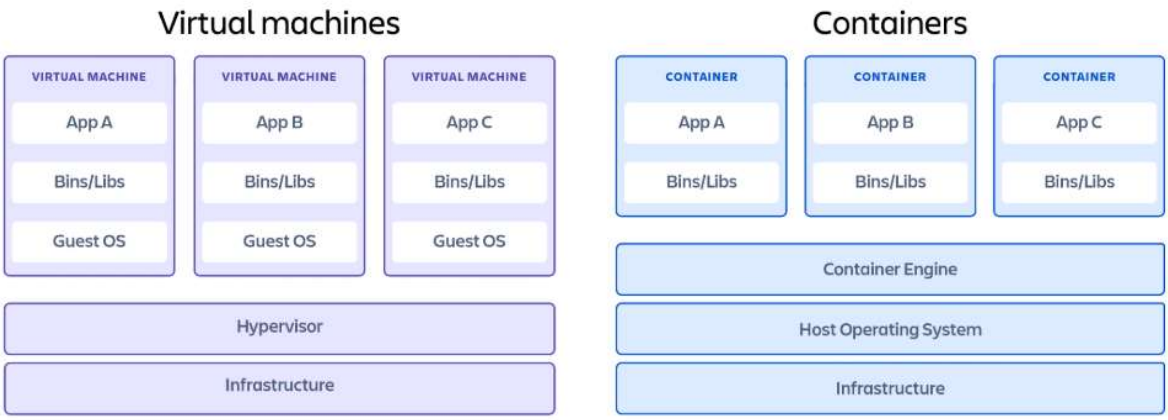
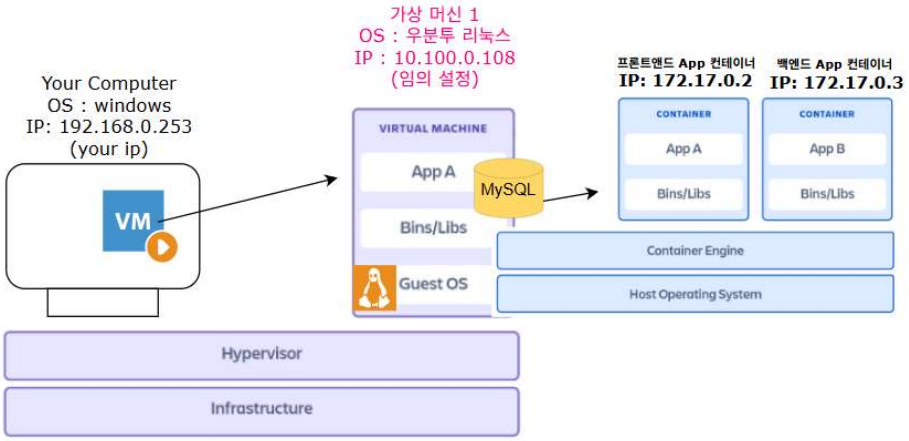
가상화



최근 사용하고 있는 가상화는 하이퍼바이저를 이용한 가상머신과 컨테이너를 이용한 도커 방식이다.

- **하이퍼바이저 Hypervisor** :
  - 하이퍼바이저는 가상머신 VM 을 생성하고 실행하는 역할, 가상화된 하드웨어와 각각의 가상머신을 모니터링하는 중간 관리자이다.가상화 프로그램은 VMware, VirtualBox 등이다.
  - 하이퍼 바이저는 가상화 프로그램을 이용하여 게스트 OS 가 사용할 수 있는 물리적 공간을 격리하는 기술이다.
- 컨테이너(Container) 기술
  - 호스트 OS 커널을 공유하면서 애플리케이션을 격리된 사용자 공간(User Space) 안에서 실행하는 방식이다. 하이퍼바이저 방식과 달리, 게스트 OS를 필요로 하지 않기 때문에 경량화되어 있으며, 부팅 시간도 매우 빠르다.
  - 대표적인 컨테이너 플랫폼은 Docker이며, 이를 통해 애플리케이션을 이미지 형태로 패키징하고, 어디서든 일관된 환경에서 실행할 수 있도록 해준다.

가상화 연습 시스템 아키텍처



(위의 MySQL 은 예시 입니다. - Oracle 등 다른 DBMS 사용도 가능합니다.)

- 컨테이너의 주요 개념

1. 애플리케이션 패키징: 애플리케이션의 코드, 런타임, 라이브러리, 시스템 도구 등 실행에 필요한 모든 것을 하나의 표준화된 방식으로 패키징 합니다.
2. 격리된 환경: 컨테이너는 다른 컨테이너나 호스트 시스템과 독립적으로 분리되어 있어, 서로 영향을 주지 않고 안전하게 작동합니다.
3. 표준화 및 일관성: 개발, 테스트, 운영 등 어떤 환경에서나 동일하게 애플리케이션을 실행할 수 있도록 보장하여 환경 차이로 인한 문제를 줄입니다.
4. 경량성 및 빠른 실행: VM과 달리 운영체제 전체를 포함하지 않고 호스트 OS의 커널을 공유하기 때문에, 훨씬 적은 자원을 사용하고 빠르게 시작됩니다.

- 컨테이너를 사용하는 이유

- 환경 일관성: 개발 환경과 운영 환경의 차이로 발생하는 "내 PC에서는 잘 되는데..."와 같은 문제를 해결합니다.
- 배포 용이성: 컨테이너는 어디서든 실행 가능하므로, 클라우드, 온프레미스 등 다양한 인프라에 애플리케이션을 쉽게 배포할 수 있습니다.
- 개발 생산성 향상: 개발자는 복잡한 환경 설정 대신 컨테이너를 활용하여 애플리케이션 개발에 집중할 수 있습니다.

가상화 네트워크

NAT(Network Address Translation)

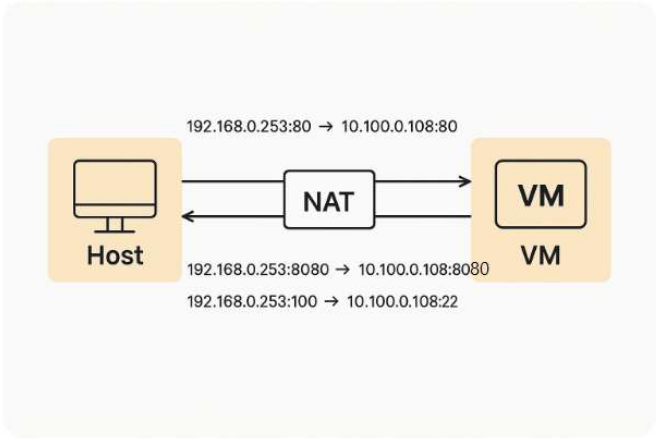
가상 머신이 호스트 머신을 통해 외부 네트워크(예: 인터넷)에 접속할 수 있도록 해주는 네트워크 모드 중 하나입니다. 가상머신은 Virtual Box 에 추가한 리눅스 , 호스트 머신은 실제 컴퓨터의 운영체제(윈도우즈) 입니다. NAT는 다음과 같이 동작합니다.

가상 머신 → NAT → 호스트 → 인터넷으로 요청이 전달됨
인터넷 → 호스트 → NAT → 가상 머신으로 응답이 전달됨(포트 포워딩 필요)

Port Forwarding

호스트 머신의 특정 포트에 들어온 요청을 가상 머신의 특정 포트로 전달하는 설정입니다. 가상 머신이 NAT 모드일 때 외부에서 직접 접근이 안 되지만, 포트 포워딩을 통해 이를 우회할 수 있습니다.

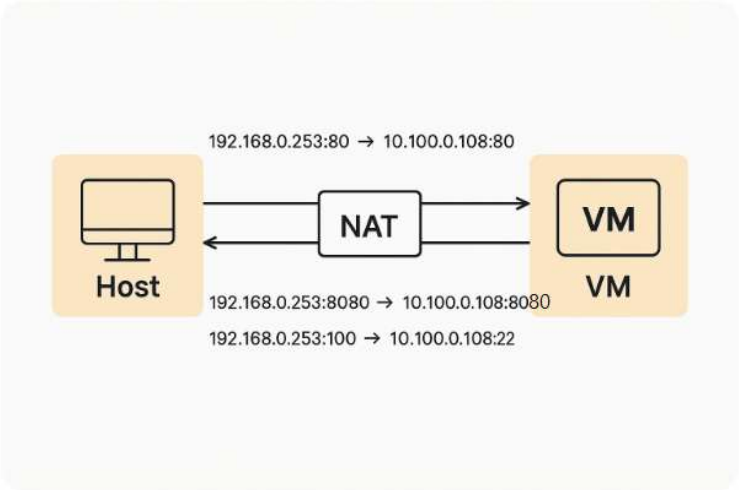
- 프론트엔드 react 는 80 포트 포워딩
- 백엔드 spring boot 는 8080 포트 포워딩
- 원격접속 ssh 는 호스트 100포트가 가상머신 22포트 포워딩



😍 강사용 컴퓨터 192.168.0.253 이므로 여러분들은 앞으로 이 ip 가 나오면 자신이 사용하는 컴퓨터의 IP 로 바꿔서 쓰세요. IP 확인은 cmd 명령창에서 **ipconfig** 명령을 실행하세요. 그 외에 호스트머신 IP 는 설명 예시와 똑같이 하시면 됩니다.

→ **포트 포워딩** 탭

→ 완료 후 적용



SSH(Secure Shell)

네트워크를 통해 **원격 시스템에 안전하게 접속**할 수 있게 해주는 **암호화된 통신 프로토콜** 입니다.
원격접속 프로그램을 사용합니다.

IP

가상 머신은 사설 IP 사용합니다.
호스트 머신도 대부분 사설 IP를 사용하여 라우터를 통해 외부 인터넷과 연결되므로하고 이것도 NAT 입니다.
가정이나 기업에서 사용하는 공유기는 NAT(Network Address Translation)를 수행하여 내부 네트워크와 외부 인터넷 간의 IP 변환을 처리합니다.

사설 IP 주소 범위

클래스	범위	서브넷 마스크	주소 수
A	`10.0.0.0` ~ `10.255.255.255`	`255.0.0.0` (또는 `/8`)	약 1,600만 개
B	`172.16.0.0` ~ `172.31.255.255`	`255.240.0.0` (또는 `/12`)	약 100만 개
C	`192.168.0.0` ~ `192.168.255.255`	`255.255.0.0` (또는 `/16`)	약 6만 5천 개

사설 IP 주소란?

- **인터넷에 직접 노출되지 않는 주소로, 내부 네트워크에서만 사용됨**
- **라우터나 방화벽을 통해 NAT(Network Address Translation) 방식으로 외부와 통신**
- **보안성과 주소 절약을 위해 널리 사용됨**

브리지 모드

→ NAT 와 다르게 호스트 머신과 가상 머신이 같은 네트워크(같은 서브넷 주소)인 경우로 포트포워딩이 필요없지만 가상 머신의 IP 를 제약적으로 사용해야 한다. 그래서 , NAT 를 사용하여 실습함.

도커

도커(Docker)는 애플리케이션을 **컨테이너(Container)**라는 가상화된 환경에서 실행할 수 있도록 해주는 오픈소스 플랫폼입니다. 컨테이너를 사용하면 애플리케이션과 그 실행 환경을 **독립적으로 패키징**하여 어디서든 동일한 환경에서 실행할 수 있습니다.

도커의 핵심 개념

1. **이미지(Image)**
   - 컨테이너를 생성하기 위한 **템플릿** 역할을 합니다.(일종의 애플리케이션 설치파일)
   - 애플리케이션 실행에 필요한 코드, 라이브러리, 설정 파일 등을 포함합니다.
   - 읽기 전용으로 저장되며, Docker Hub에서 다운로드할 수 있습니다.
2. **컨테이너(Container)**
   - 이미지를 실행하여 생성된 **격리된 실행 환경**입니다.
   - 호스트 운영체제의 커널을 공유하면서도 독립적으로 실행됩니다.
   - 가상 머신보다 가볍고 빠르게 실행됩니다.
3. **도커 파일(Dockerfile)**
   - 이미지를 생성하기 위한 **설정 파일**입니다.
   - 애플리케이션을 자동으로 빌드하고 배포할 수 있도록 도와줍니다.
4. **도커 허브(Docker Hub)**
   - 도커 이미지를 저장하고 공유할 수 있는 **공식 레지스트리 서비스**입니다.
   - 다양한 오픈소스 및 기업용 이미지를 다운로드할 수 있습니다.

도커의 장점

- **환경 일관성**: 개발, 테스트, 운영 환경에서 동일한 설정을 유지할 수 있습니다.
- **빠른 배포**: 컨테이너를 빠르게 생성하고 실행할 수 있습니다.
- **효율적인 자원 사용**: 가상 머신보다 가볍고, 시스템 자원을 효율적으로 활용합니다.
- **이식성**: 컨테이너를 어디서든 실행할 수 있어 클라우드 및 온프레미스 환경에서 유용합니다.

원활한 수업을 위해 가상 머신에 이미 docker 를 설치했습니다. 아래와 같이 확인하세요. 새로운 시스템에 도커 설치는 https://docs.docker.com/engine/install/ubuntu/ 를 참고하여 설치합니다.(아래 내용은 자주 업데이트 됩니다. 공식문서를 항상 참고하여 설치합니다.)

도커 설치

공식 문서 https://docs.docker.com/engine/install/ubuntu/

Install using the `apt` repository

여기서 부터 👆

```
# Add Docker's official GPG key:
# apt: advanced pakaging tool - 패키지 관리도구. 최신패키지로 업데이트
# 도커 설치를 위해 의존성이 있는 패키지 설치(디지털 서명, 웹에서 다운로드 실행)
# 도커의 공식 GPG(GNU Privacy Guard) key 다운로드 하여 저장
# apt가 패키지 인증을 위해 사용
# 권한 변경
# Add the repository to Apt sources: 패키지 설치 리포지토리 추가
# 패키지 업데이트!! (리포지토리 추가했으므로 꼭 해야함.)
# docker 관련 패키지 설치(포트 사용중으로 나오면 포트 kill -9)
# 설치 확인하기
# sudo docker version
# 사용자 계정으로 docker 명령 실행 권한 설정
# 나는 누구
# docker 그룹에 현재 사용자 추가 실행 후 재접속
```

또는 간단히

낮은 버전 설치 가능성 있음. 설치 후 버전 확인하기
(AWS 에서)

```
sudo apt update
# ubuntu 기본 패키지 저장소(apt)에서 제공
sudo apt install docker.io -y

# docker 데몬 실행
sudo systemctl start docker
# 시스템 종료 후 시작할 때 자동 실행
sudo systemctl enable docker
# 사용자 계정을 docker 그룹에 추가.
sudo usermod -aG docker ubuntu


#   docker compose 설치
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-compose-$
(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

도커 명령어

🚀 컨테이너 상태 확인

`docker ps`	실행 중인 컨테이너 목록 확인
`docker ps -a`	**모든 컨테이너**(중지 포함) 목록 확인
`docker inspect <컨테이너>`	컨테이너 상세 정보 확인
`docker stats`	실시간 리소스 사용량 확인 (CPU, 메모리 등)

📜 컨테이너 로그 확인

`docker logs <컨테이너>`	로그 출력
`docker logs -f <컨테이너>`	로그 실시간 스트리밍 (tail -f처럼)
`docker logs --tail 100 <컨테이너>`	최근 100줄만 출력

🧑‍💻 컨테이너 쉘 접속

`docker exec -it <컨테이너> sh`	`sh` 쉘 접속 (Alpine 등 경량 이미지)
`docker exec -it <컨테이너> bash`	`bash` 쉘 접속 (Ubuntu 등 일반 이미지)

🧹 컨테이너 및 이미지 정리

`docker stop <컨테이너>`	컨테이너 중지
`docker rm <컨테이너>`	컨테이너 삭제
`docker rmi <이미지>`	이미지 삭제
`docker system prune`	사용하지 않는 리소스 전체 정리 (주의: 삭제됨)
`docker rm $(docker ps -a -f status=exited -q)`	정지된 컨테이너만 안전하게 삭제

📦 이미지 및 빌드

`docker build -t my-image .`	Dockerfile 기반 이미지 빌드

`docker images`	이미지 목록 확인
`docker pull <이미지>`	이미지 다운로드
`docker run -d -p 8080:80 --name my-container my-image`	백그라운드 실행 + 포트 매핑

🧩 자주 사용하는 옵션

- `-d`: 백그라운드 실행
- `-p`: 포트 매핑 (예: `-p 8080:80`)
- `--name`: 컨테이너 이름 지정
- `-v`: 볼륨 마운트 (예: `-v /host:/container`)
- `-it`: 터미널 입력 가능하게 실행