Study Introduction
&
Docker Basic 1

스터디 개요 및 도커 기본

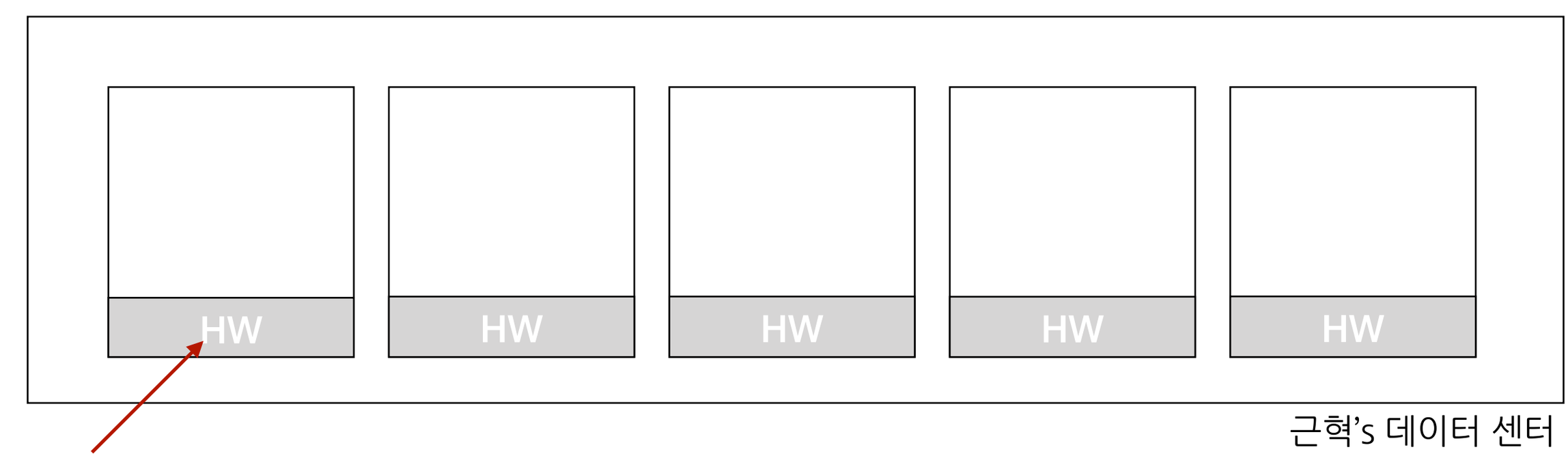
2021. 1. 29

- 도커 스터디 계획표 (약 7~8주)
    - Basic 1, 2
        - 컨테이너 기술 탄생 배경
        - 배경지식 및 기본 구조
        - Docker Registry (이미지 조회 및 푸시&풀)
        - 컨테이너 라이프 사이클
        - 활용 기초 (커맨드 활용)
    - Writing Docker Image
        - Dockerfile
        - 이미지 배포 (Private/Public Registry)
        - 단일 머신에서 여러 컨테이너 구성 (Docker Compose)
    - Storage and Volume
        - Persistent Data & Volume
        - Storage Class
        - (Optional) Database migration
    - Networking
        - 네트워크 기초
        - 도커 네트워크 (Bridge, None, Host, Custom)
        - 컨테이너간 통신과 DNS
    - Security
        - 네트워크 보안 기초
        - 도커 이미지 신뢰
        - 도커 레지스트리 통신 보안
    - Orchestration
        - 도커 스웜
        - 컨테이너 고가용성 구성과 자원 할당 및 제한

- 도커 스터디 계획표 (약 7~8주)
    - **Basic 1, 2**
        - **컨테이너 기술 탄생 배경**
        - **배경지식 및 기본 구조**
        - **Docker Registry (이미지 조회 및 푸시&풀)**
        - 컨테이너 라이프 사이클
        - 활용 기초 (커맨드 활용)
    - Writing Docker Image
        - Dockerfile
        - 이미지 배포 (Private/Public Registry)
        - 단일 머신에서 여러 컨테이너 구성 (Docker Compose)
    - Storage and Volume
        - Persistent Data & Volume
        - Storage Class
        - (Optional) Database migration
    - Networking
        - 네트워크 기초
        - 도커 네트워크 (Bridge, None, Host, Custom)
        - 컨테이너간 통신과 DNS
    - Security
        - 네트워크 보안 기초
        - 도커 이미지 신뢰
        - 도커 레지스트리 통신 보안
    - Orchestration
        - 도커 스웜
        - 컨테이너 고가용성 구성과 자원 할당 및 제한

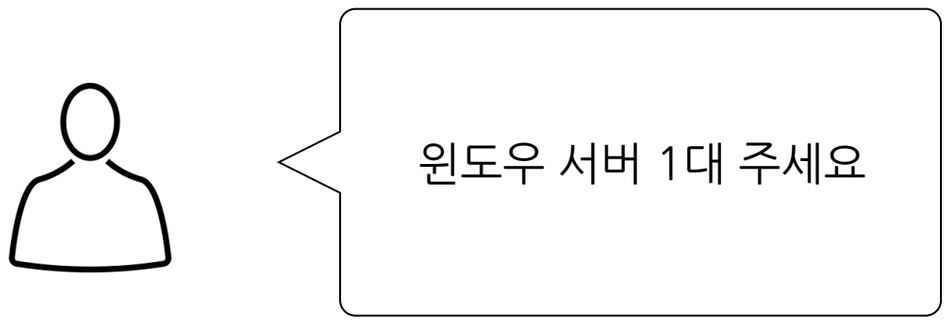
1. 컨테이너 기술 탄생 배경

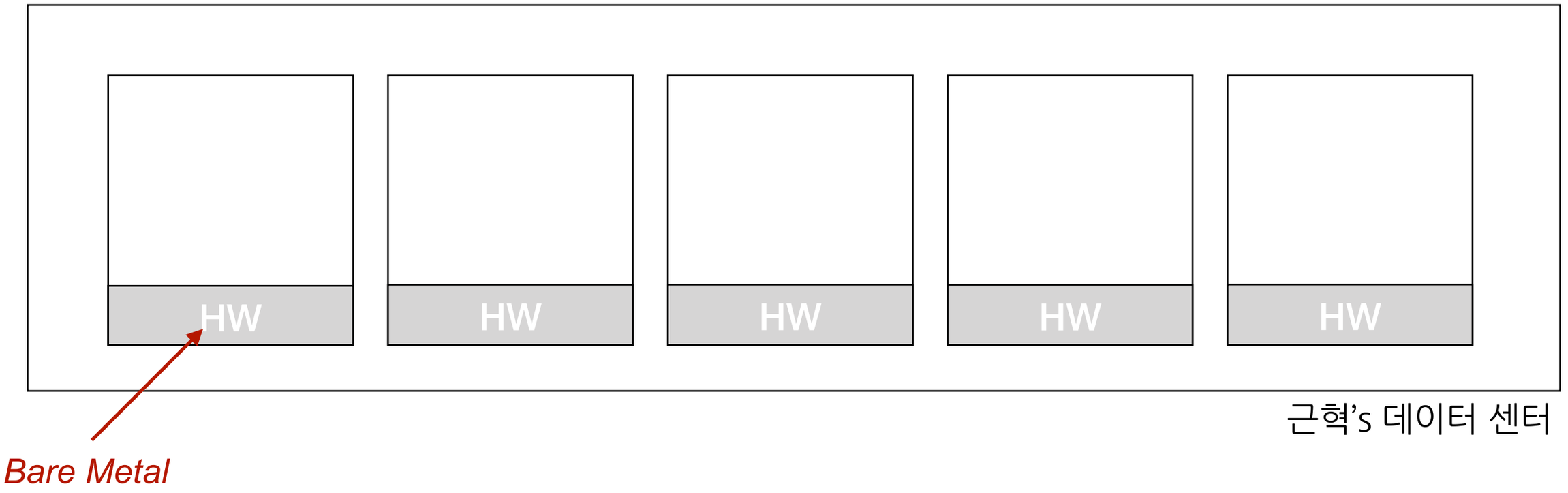
- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정



1. 컨테이너 기술 탄생 배경

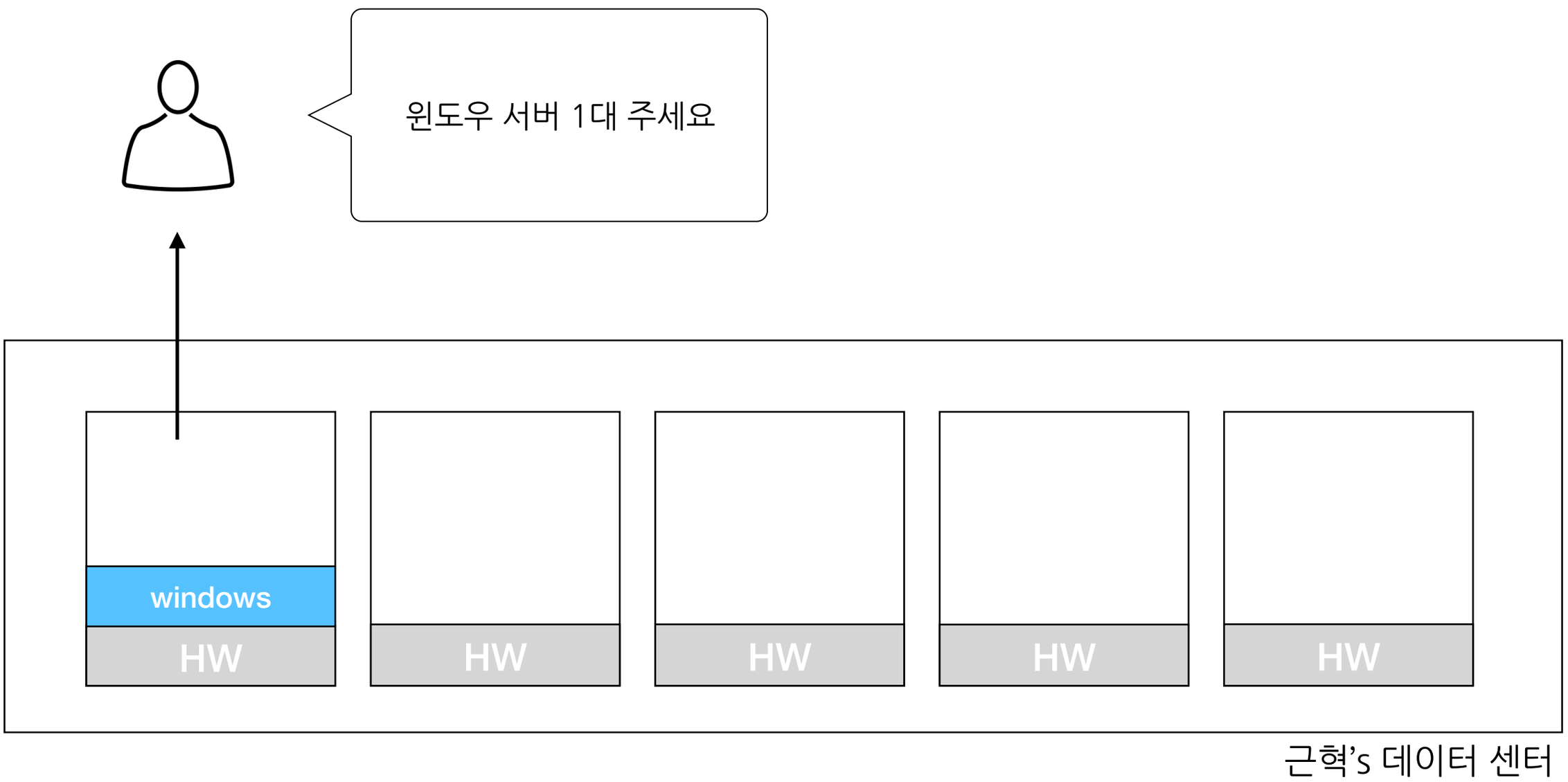
- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정





1. 컨테이너 기술 탄생 배경

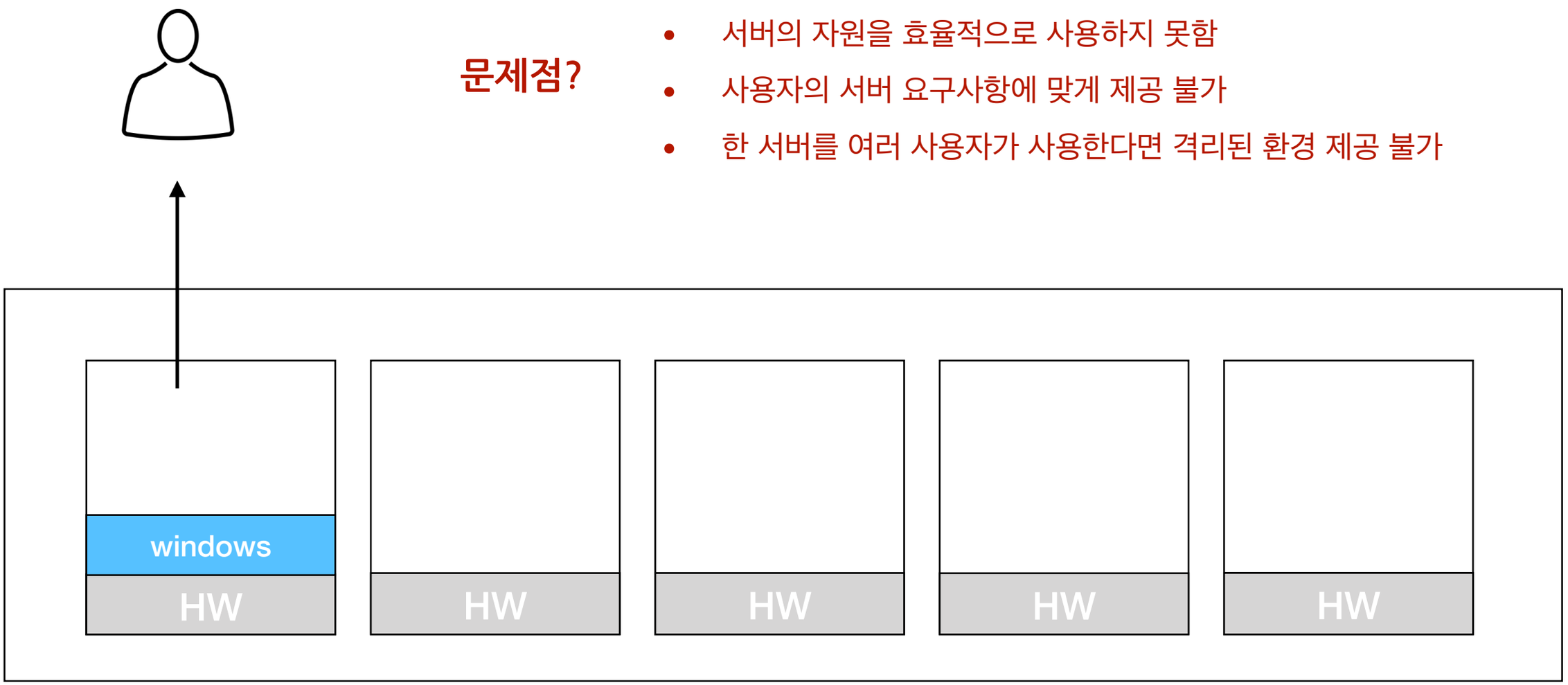
- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정



근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정



근혁's 데이터 센터

문제점?

- 제공할 수 있는 서버의 수가 물리 서버 수로 한정됨
- 서버의 자원을 효율적으로 사용하지 못함
- 사용자의 서버 요구사항에 맞게 제공 불가
- 한 서버를 여러 사용자가 사용한다면 격리된 환경 제공 불가

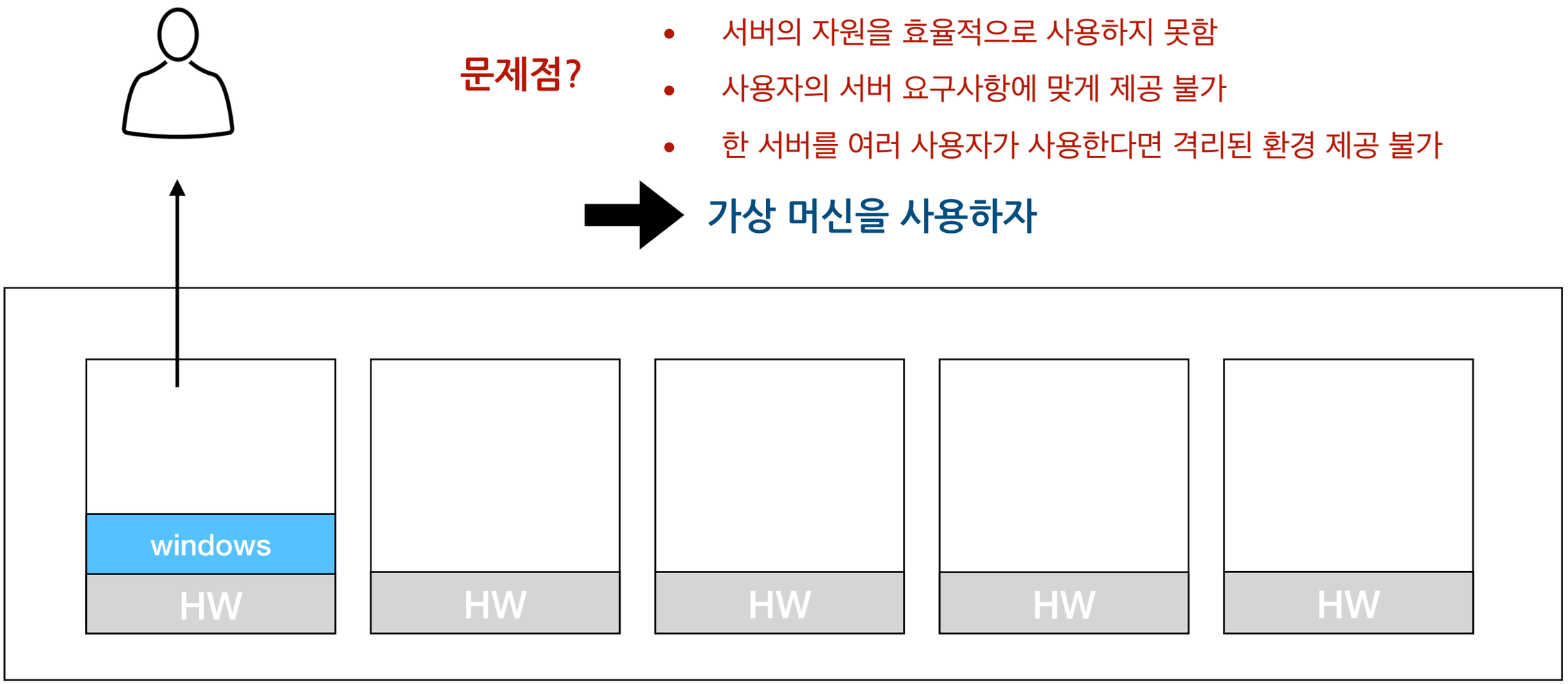
1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

문제점?

- 제공할 수 있는 서버의 수가 물리 서버 수로 한정됨
- 서버의 자원을 효율적으로 사용하지 못함
- 사용자의 서버 요구사항에 맞게 제공 불가
- 한 서버를 여러 사용자가 사용한다면 격리된 환경 제공 불가

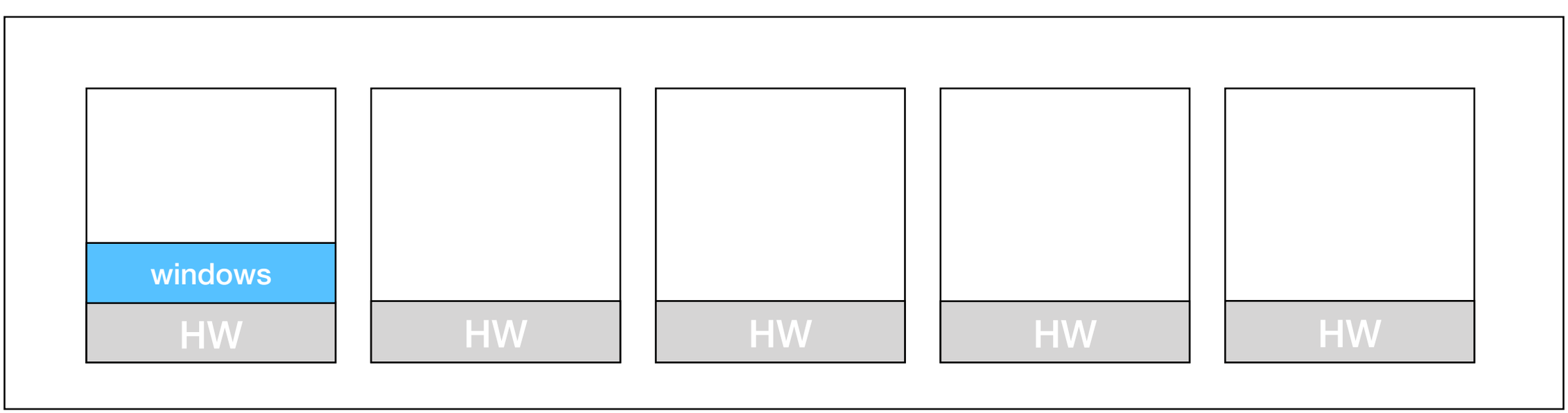
가상 머신을 사용하자



근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

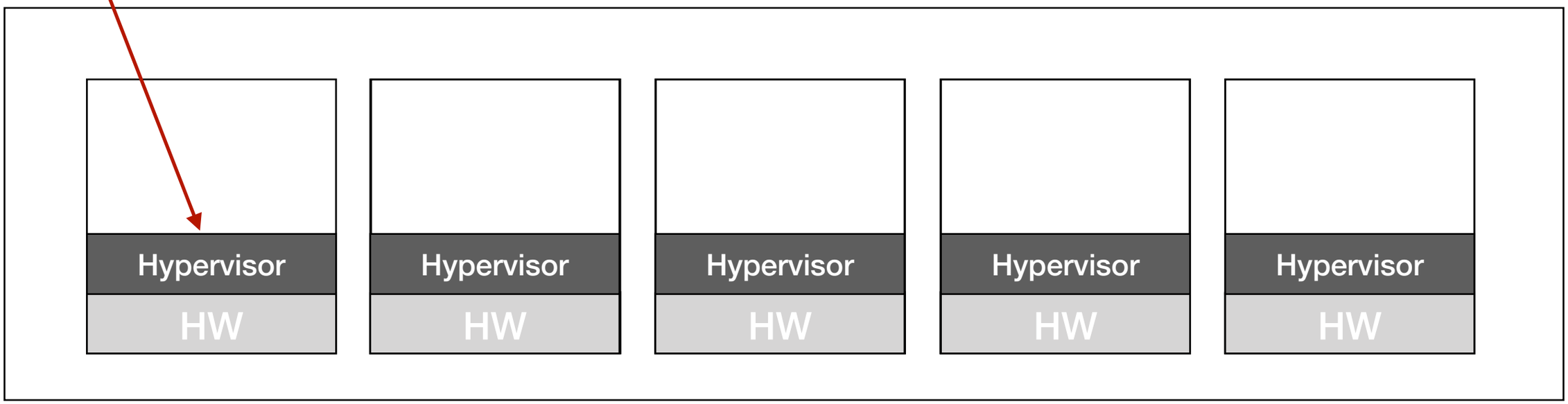


근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

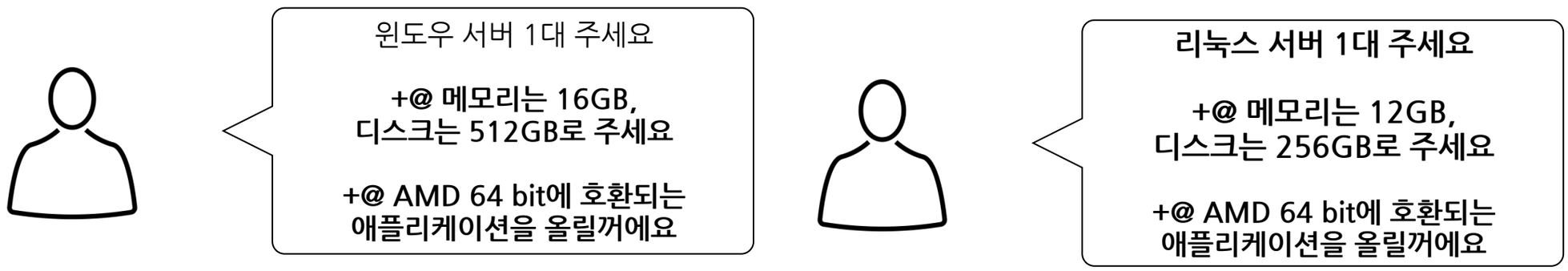
하드웨어 가상화하여
상위 레이어에 제공

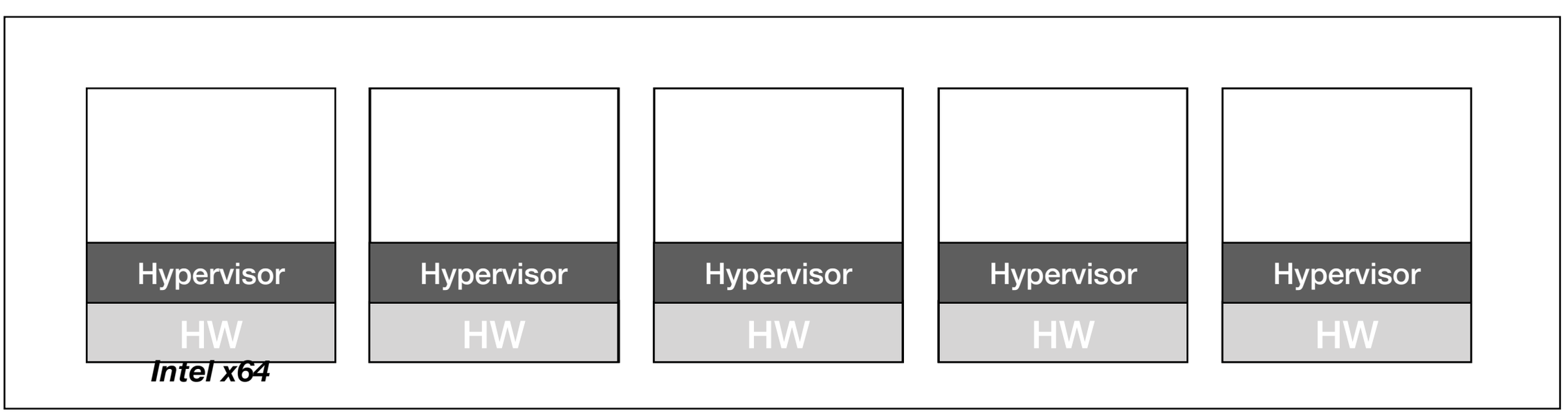


근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

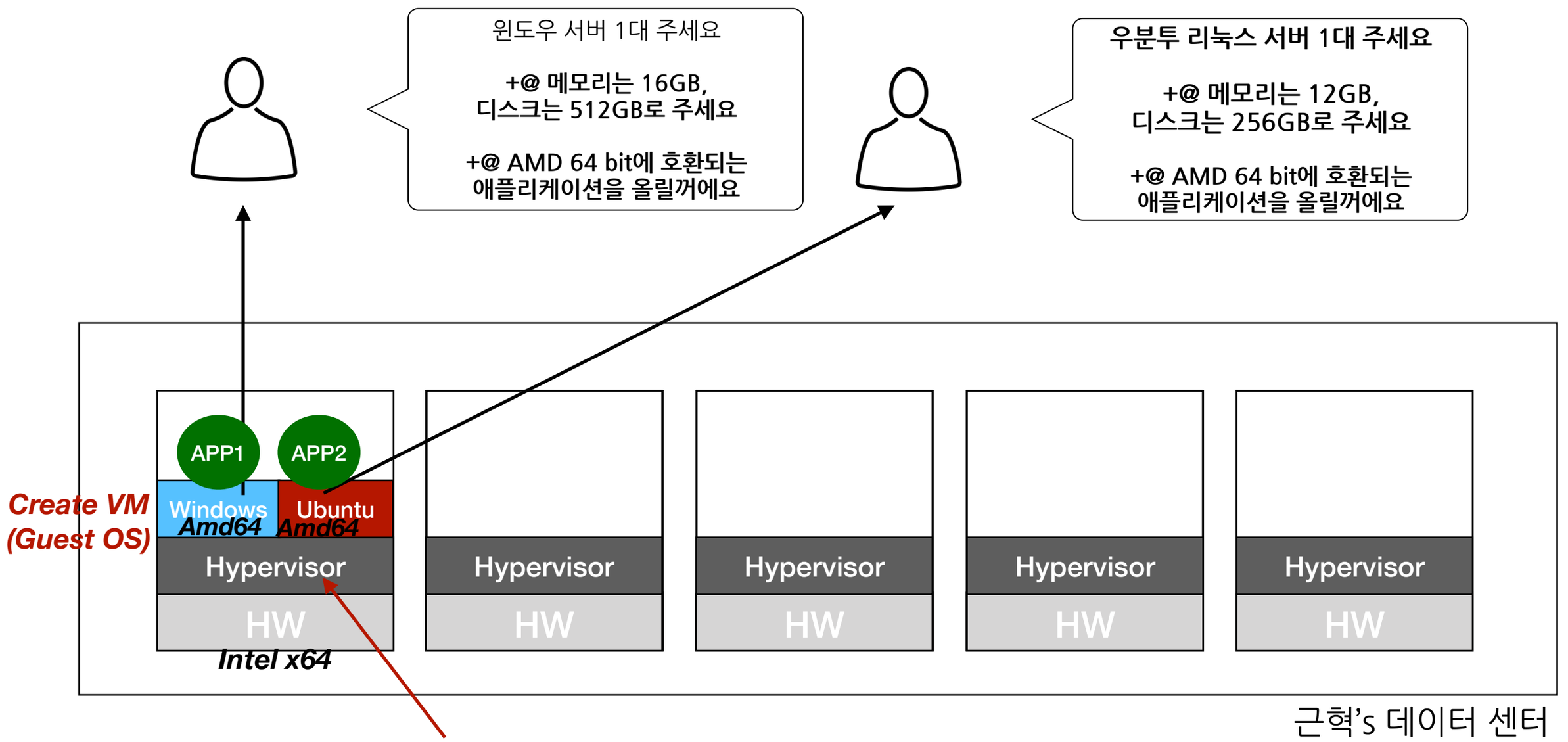




근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정



AMD 아키텍처 명령어를 *intel* 아키텍처 명령어로 변환
+@ Guest OS 간 격리된 환경 제공

c.f 마이크로아키텍처 명령어 변환 방식에 따라 전가상화, 반가상화로 나뉨

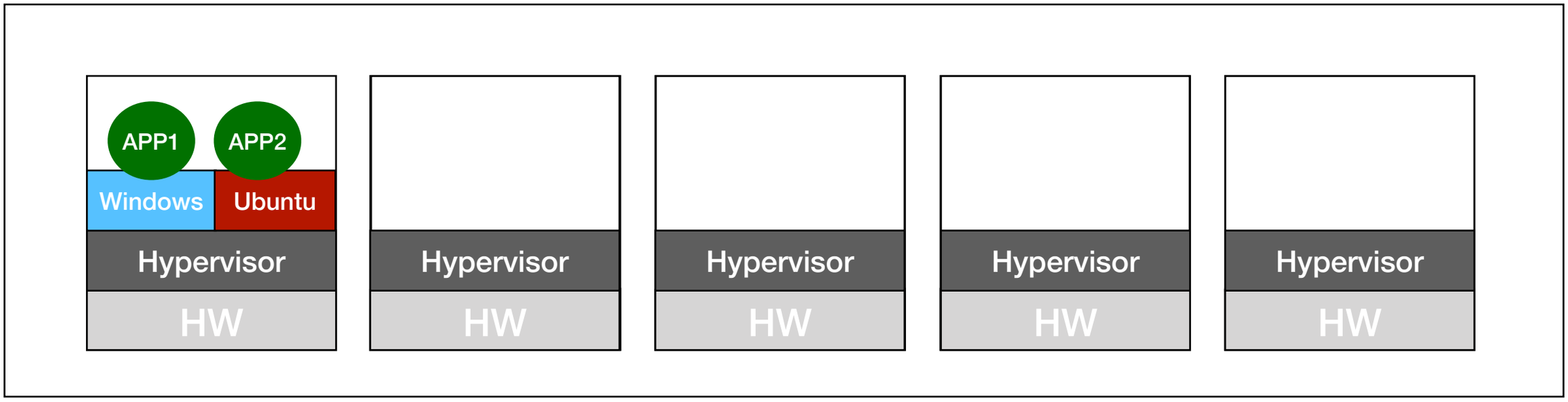
1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

기존 문제점?

- 제공할 수 있는 서버의 수가 물리 서버 수로 한정됨
- 서버의 자원을 효율적으로 사용하지 못함
- 사용자의 서버 요구사항에 맞게 제공 불가
- 한 서버를 여러 사용자가 사용한다면 격리된 환경 제공 불가

→ **해결!**



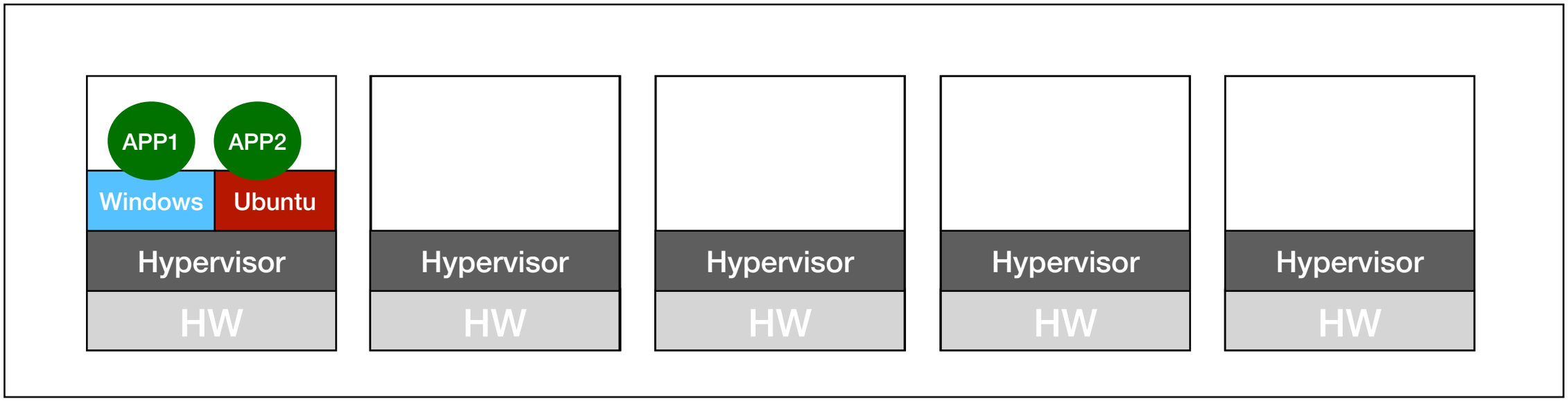
근혁's 데이터 센터

r.f. 내부적으로 사설 *IP* 가 *2* 단계로 구성되어 기존보다 네트워크 체계가 복잡해짐

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

시간이 흘러.. 다른 요구사항이 발생

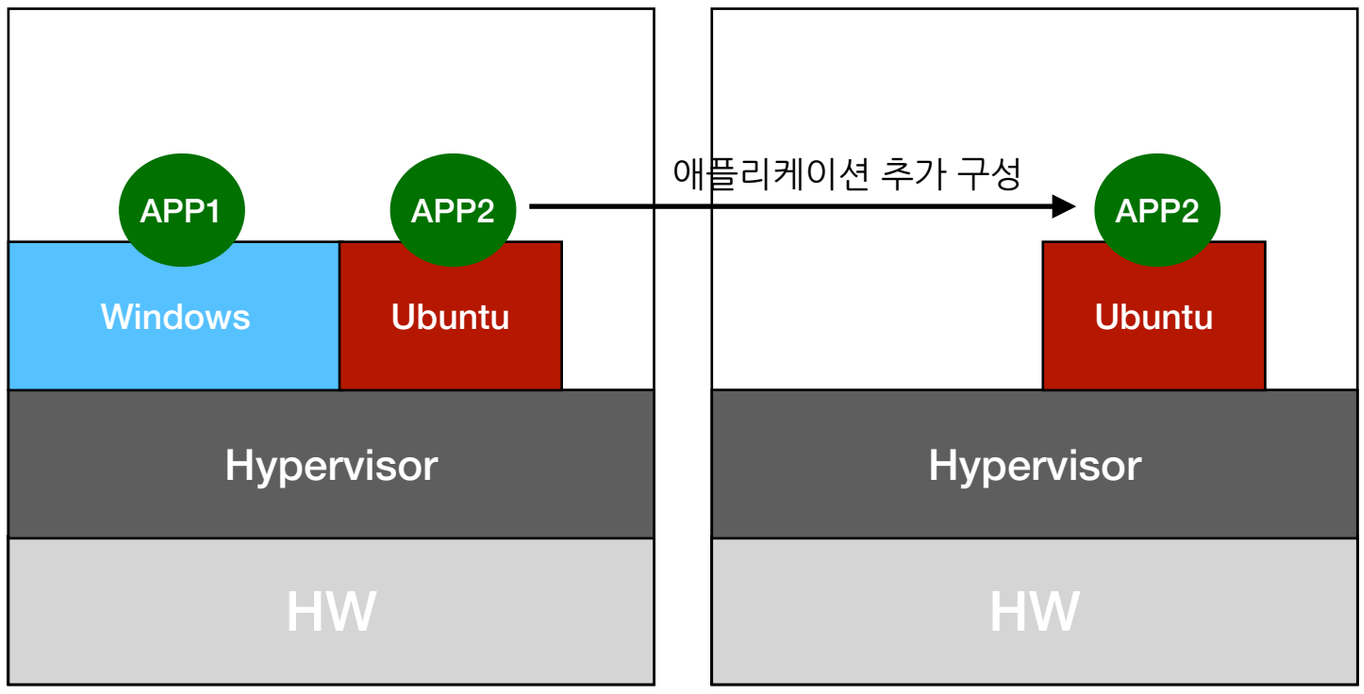


근혁's 데이터 센터

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자

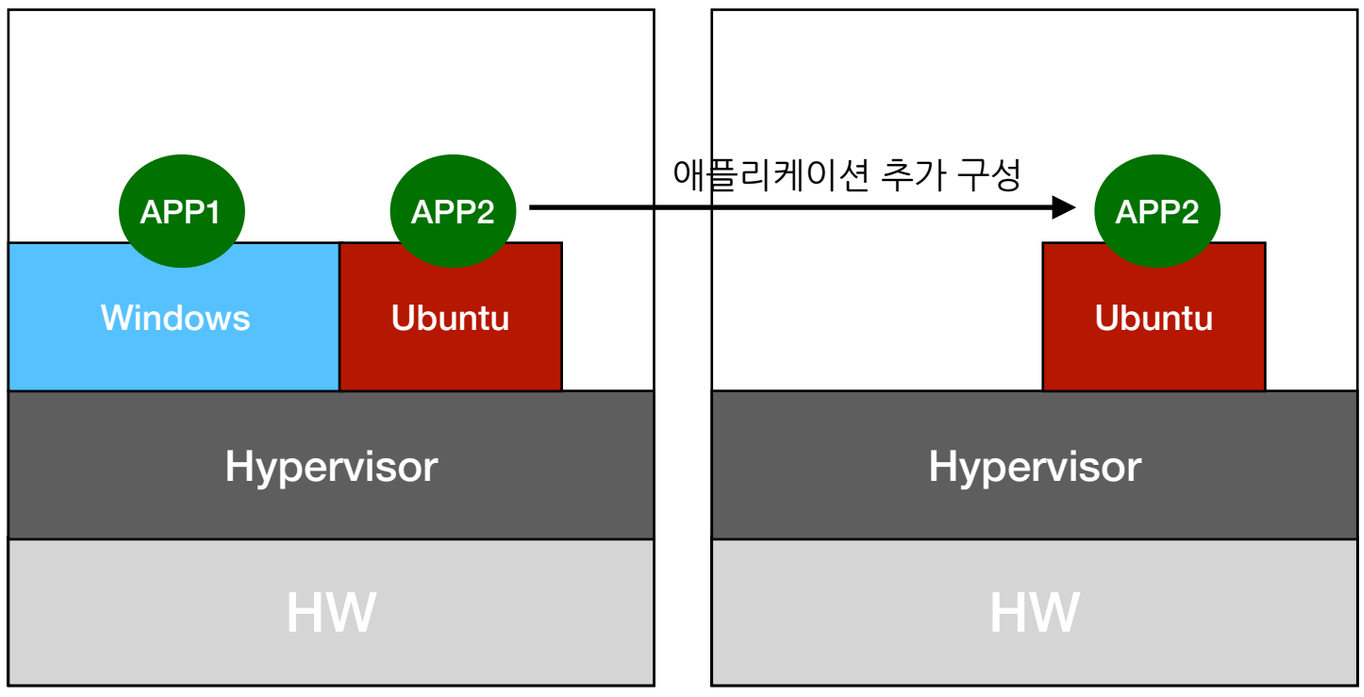


어떻게 해야 할까?

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



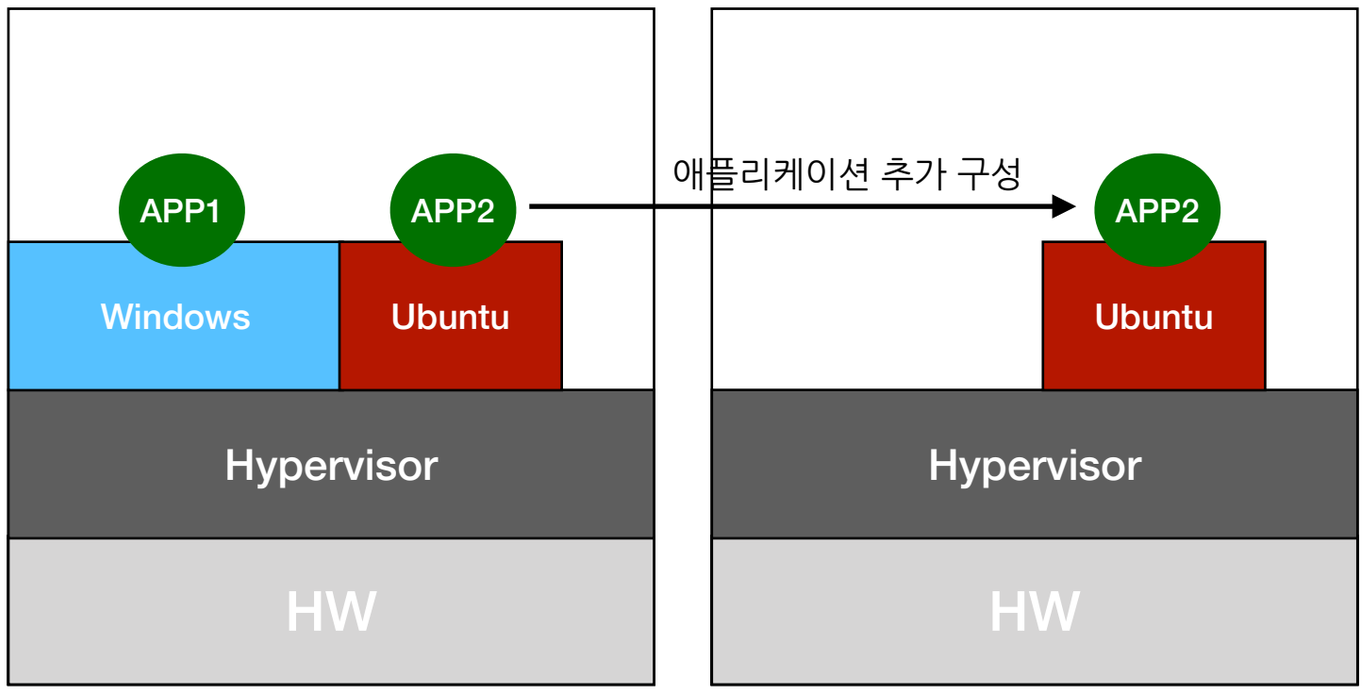
어떻게 해야 할까?

1. 새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.
   - 로그 폴더 생성
   - 시스템 의존성 다운로드
   - 외부 의존성 설치 (결제 모듈 등)
   - 기타 등등 ..
2. VM을 통째로 이미지화 하여 이전한다.

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



어떻게 해야 할까?

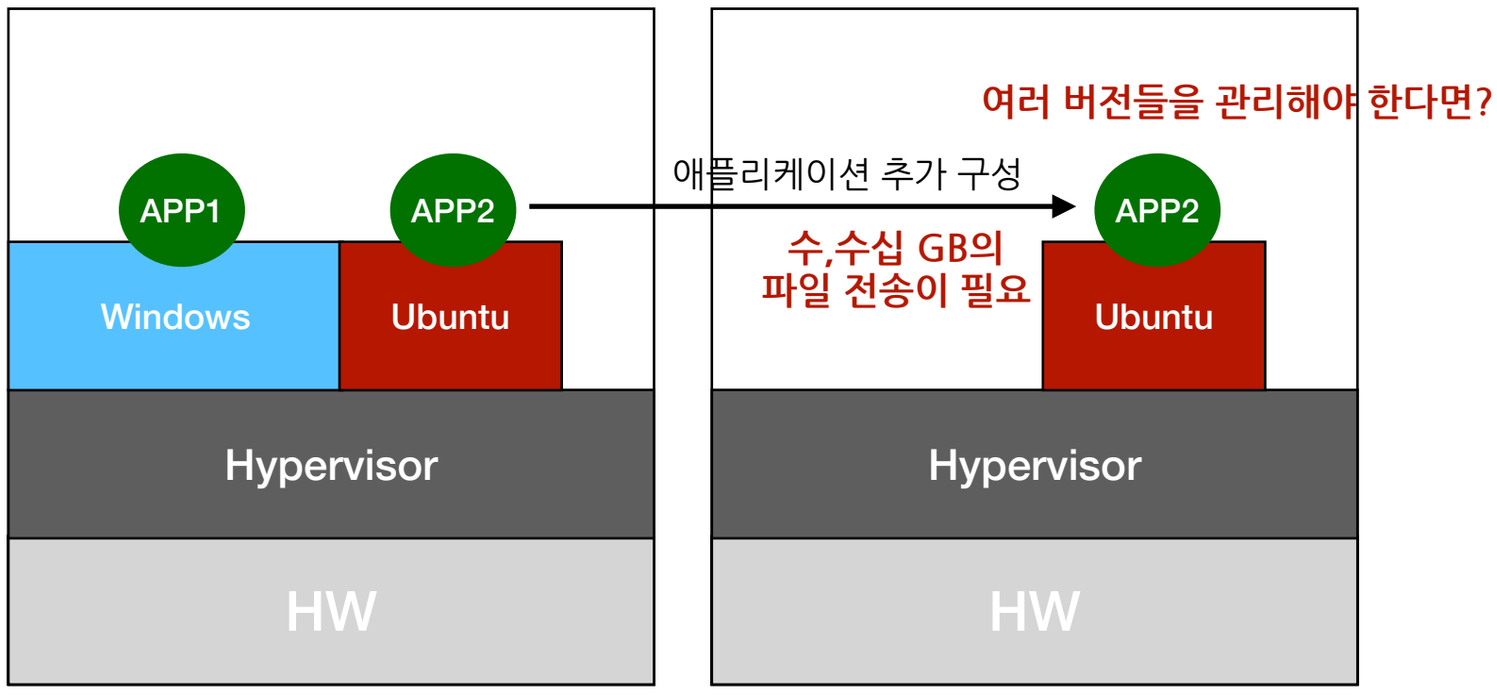
1. **새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.**
   - **로그 폴더 생성**
   - **시스템 의존성 다운로드**
   - **외부 의존성 설치 (결제 모듈 등)**
   - **기타 등등 ..**
2. VM을 통째로 이미지화 하여 이전한다.

경우에 따라 작업이 복잡&많은 시간이 소비되고
이전 작업 정상 여부를 확인하기 위한 테스트가 필요

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



여러 버전들을 관리해야 한다면?

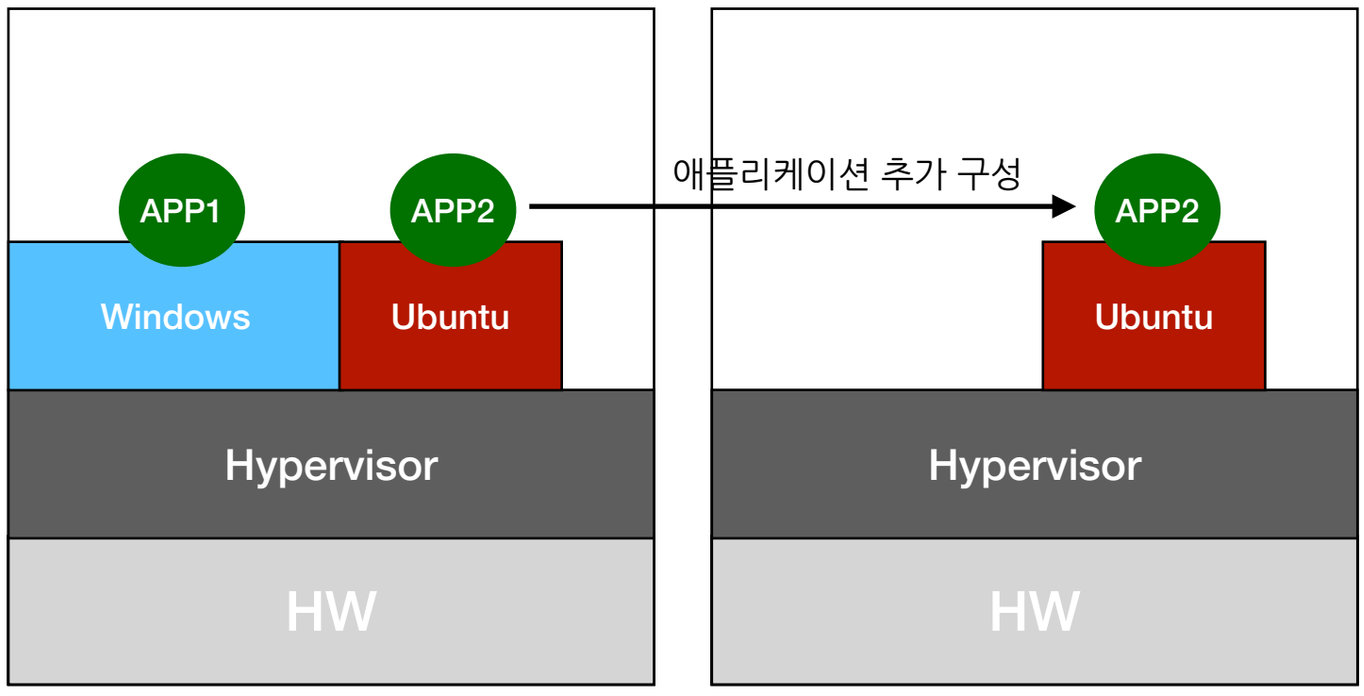
어떻게 해야 할까?

1. 새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.
   - 로그 폴더 생성
   - 시스템 의존성 다운로드
   - 외부 의존성 설치 (결제 모듈 등)
   - 기타 등등 ..
2. **VM을 통째로 이미지화 하여 이전한다.**

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



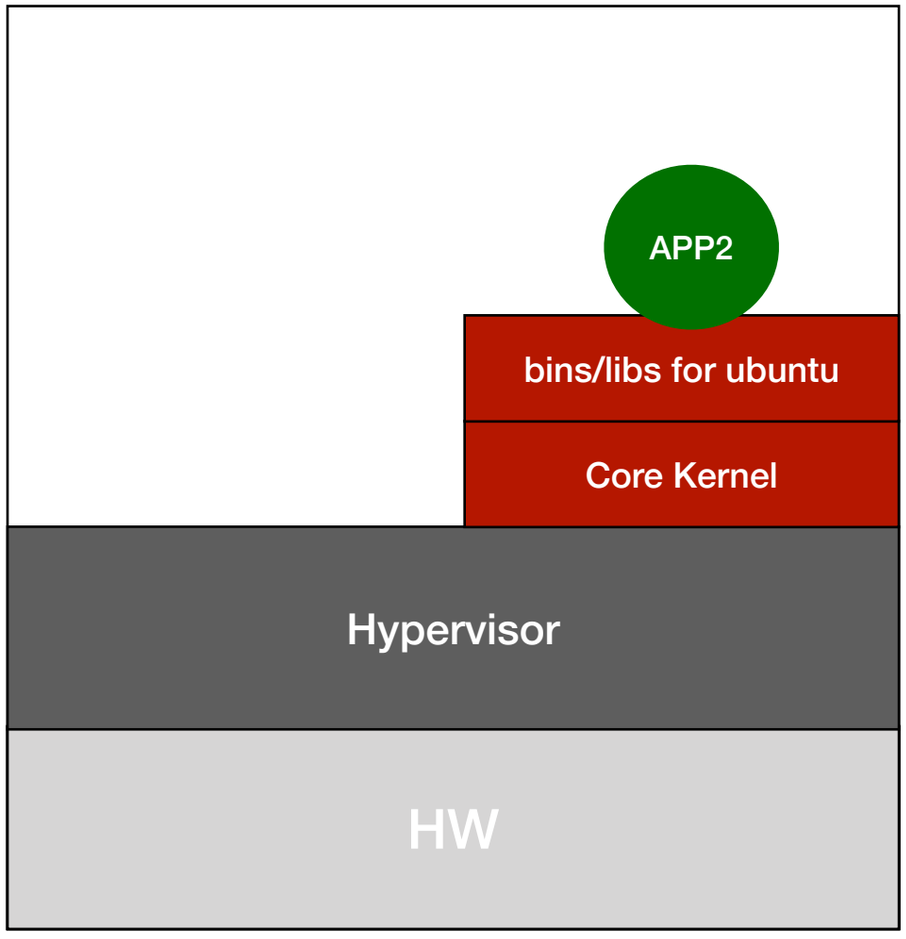
도커를 사용하자!

1. ~~새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.~~
   - ~~로그 폴더 생성~~
   - ~~시스템 의존성 다운로드~~
   - ~~외부 의존성 설치 (결제 모듈 등)~~
   - ~~기타 등등 ..~~
2. ~~VM을 통째로 이미지화 하여 이전한다.~~

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



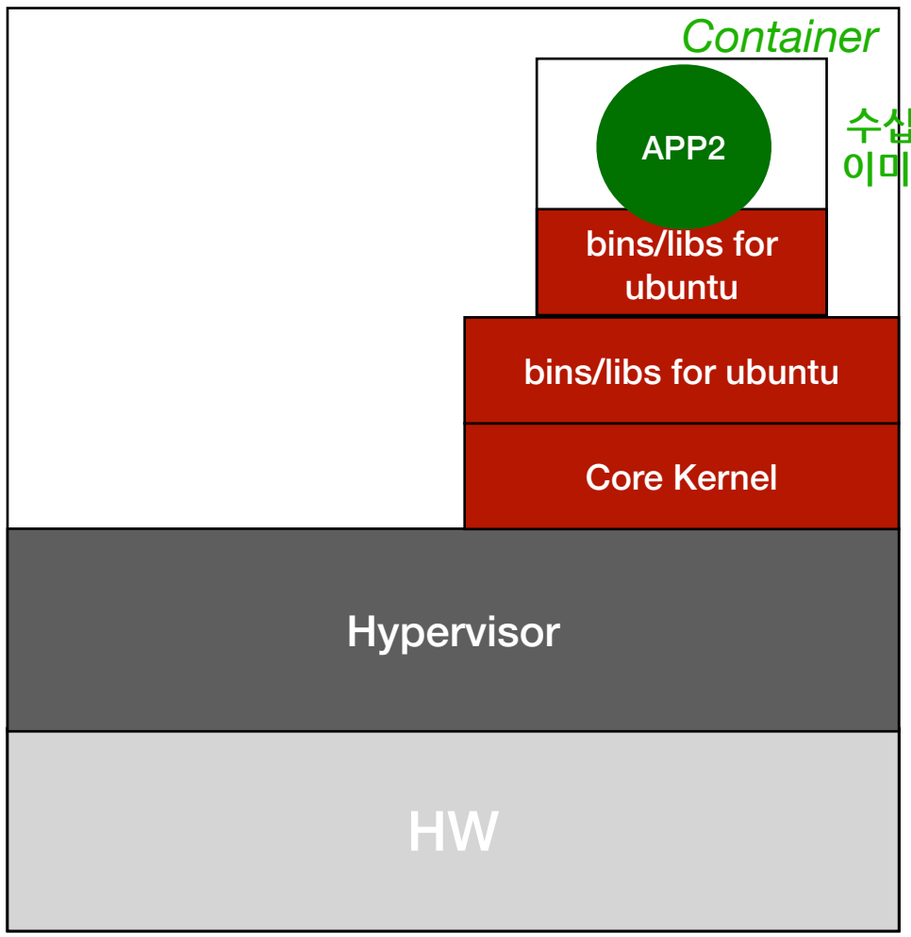
도커를 사용하자!

1. ~~새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.~~
   - ~~로그 폴더 생성~~
   - ~~시스템 의존성 다운로드~~
   - ~~외부 의존성 설치 (결제 모듈 등)~~
   - ~~기타 등등 ..~~
2. ~~VM을 통째로 이미지화 하여 이전한다.~~

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자



수십,수백 MB로 비교적 가벼움
이미지 구조가 버전 관리에 적합

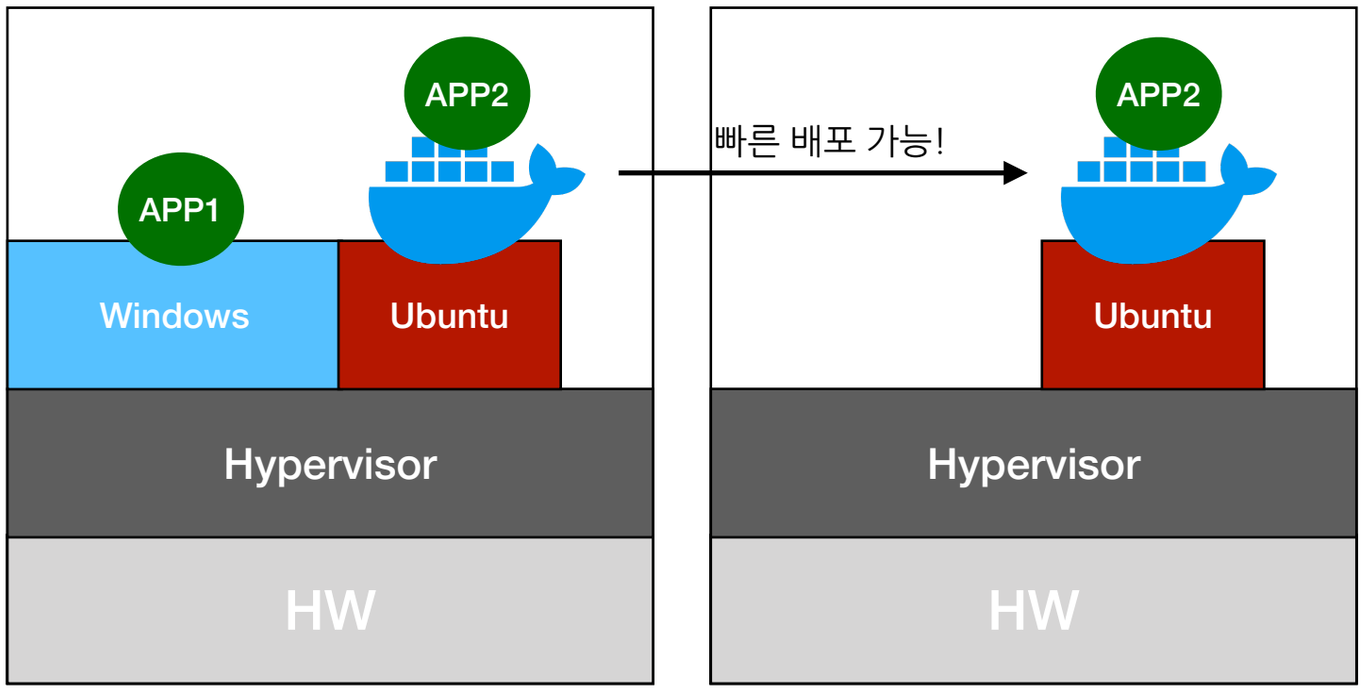
도커를 사용하자!

1. ~~새로운 리눅스 VM을 설치하고 APP2가 동작할 수 있는 환경을 만들어서 APP2를 실행한다.~~
   - ~~로그 폴더 생성~~
   - ~~시스템 의존성 다운로드~~
   - ~~외부 의존성 설치 (결제 모듈 등)~~
   - ~~기타 등등 ..~~
2. ~~VM을 통째로 이미지화 하여 이전한다.~~

1. 컨테이너 기술 탄생 배경

- 우리가 데이터 센터를 운영하며 서버를 대여해 준다고 가정

서비스의 사용자가 늘어 리눅스 애플리케이션 서버의 확장이 필요하다고 가정하자

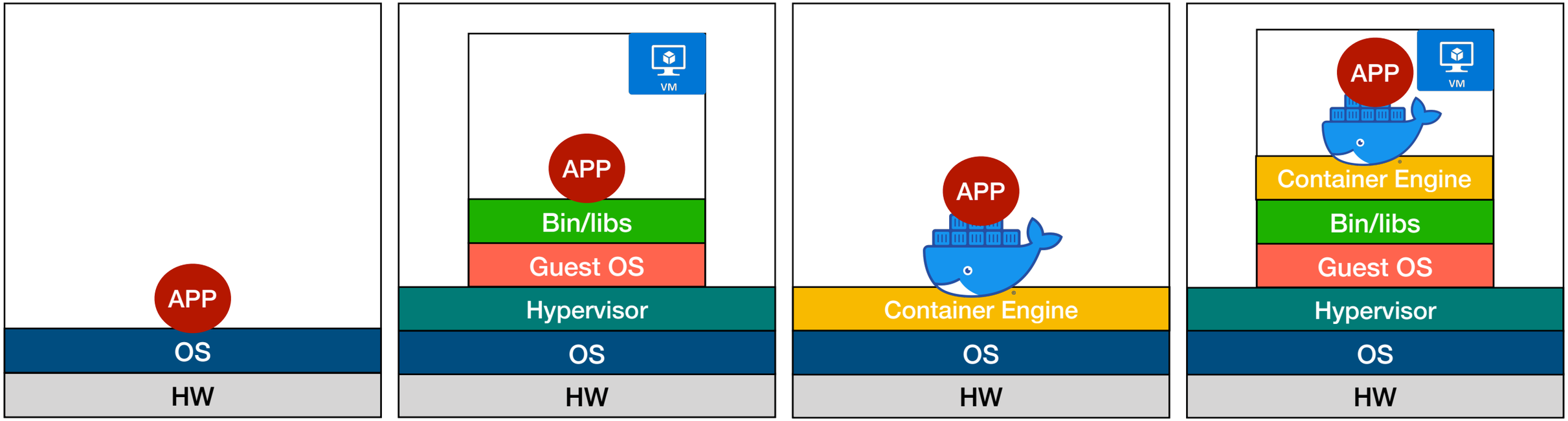


도커를 사용하자!

- 수,수십 MB로 가벼움
- 이미지 구조가 버전 관리에 적합
- APP 성능에 큰 영향을 끼치지 않음
- 빠르고 편리한 배포 가능

1. 컨테이너 기술 탄생 배경

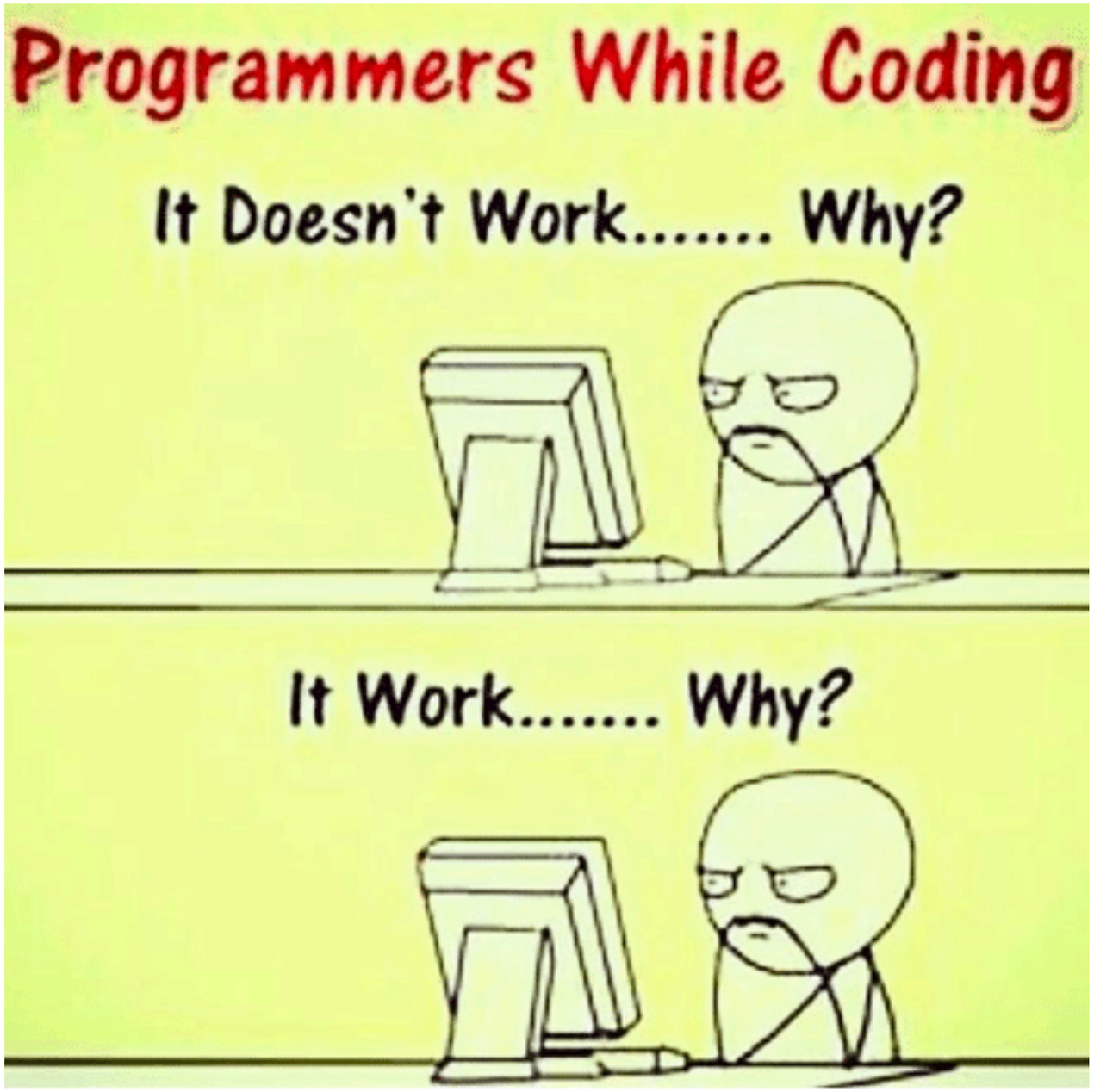
- 가상머신 & 컨테이너 탄생 배경 요약
  - 베어메탈(물리 서버)의 리소스를 효율적으로 사용하고 다수의 사용자에게 격리된 가상의 환경을 제공하고자 가상머신 기술 등장
  - 애플리케이션과 하부 미들웨어(OS 등)의 의존성을 끊고 격리된 환경에서 배포를 빠르고 효율적으로 하고자 컨테이너 기술 등장



다양한 형태의 애플리케이션 구동 환경
→ 목적에 맞게 활용

1. 컨테이너 기술 탄생 배경

- 컨테이너 기술을 사용하면 아래와 같은 상황도 방지할 수 있음



내 컴퓨터에서는 잘 돌아가는데?

2. 배경지식 및 기본 구조

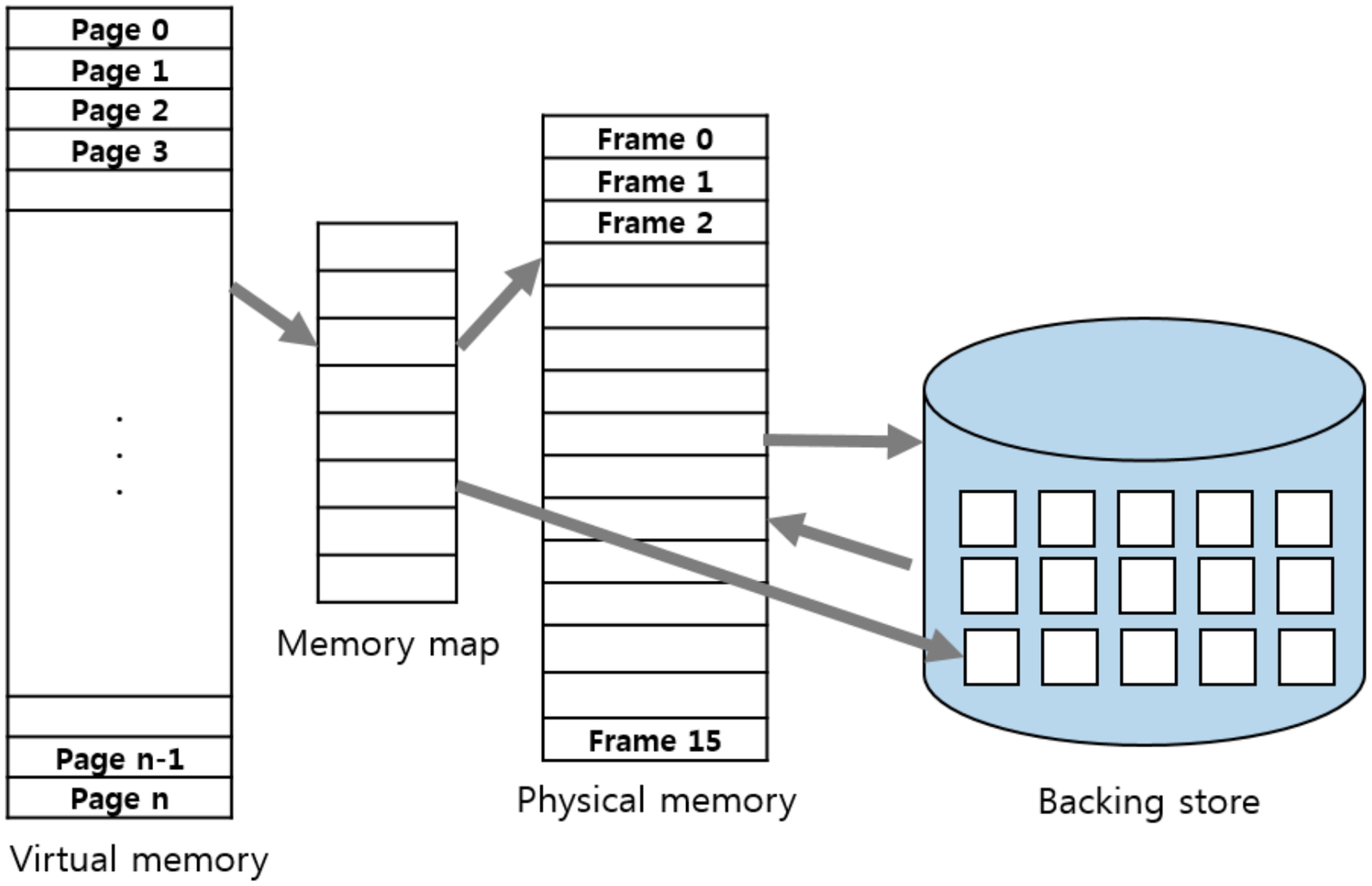
- 시스템 기반의 구성 요소
  - 기능적 요구사항: 비즈니스 로직 등
  - 비기능적 요구사항: 가용성, 확장성 등 (docker is here!)

- CI/CD (Continuous Integration Continuous Delivery, Deployments)
  - CI: 코드를 작성할 때마다 지속적으로 테스트를 실행하여 확실하게 작동되는 코드를 유지하는 방법
  - CD: 소프트웨어를 운영 환경에서 정상적으로 배포한다는 것을 보장하고 원하는 즉시 현재 버전의 소프트웨어를 운영환경에 배포할 수 있도록 함
  - 안쓴다면?

- 컨테이너
  - 격리된 환경에서 애플리케이션을 동작하게 하는 기술 (Solaris Container, LXC)
  - 완전한 가상 환경을 제공하는 것이 목표가 아님

2. 배경지식 및 기본 구조

- 자원 격리 & 가상 환경 활용 예시
  - 가상 메모리
    - 애플리케이션 별 독립적인 메모리 주소 공간을 사용



2. 배경지식 및 기본 구조

- 자원 격리 & 가상 환경 활용 예시
  - 자바 가상 머신
    - 애플리케이션과 하부 OS, 하드웨어 간의 의존성을 분리함

Java?

Write once, run everywhere!

Docker?

Build once, run everywhere!

2. 배경지식 및 기본 구조

- 도커
  - 하부 인프라와 애플리케이션의 의존성을 분리하여 빠르게 배포하고 구동할 수 있는 환경을 제공하는 오픈 플랫폼
  - 애플리케이션을 다루는 것과 동일한 방식으로 인프라를 다룰 수 있음
  - 호스트 시스템의 커널에서 직접 실행 되므로 가볍고 빠름

- 무엇을 할 수 있나
  - 빠르고 일관된 애플리케이션 배포 가능 (CI/CD)
  - 가볍고 이식성이 좋아 애플리케이션의 실시간 관리 및 확장/축소 가능

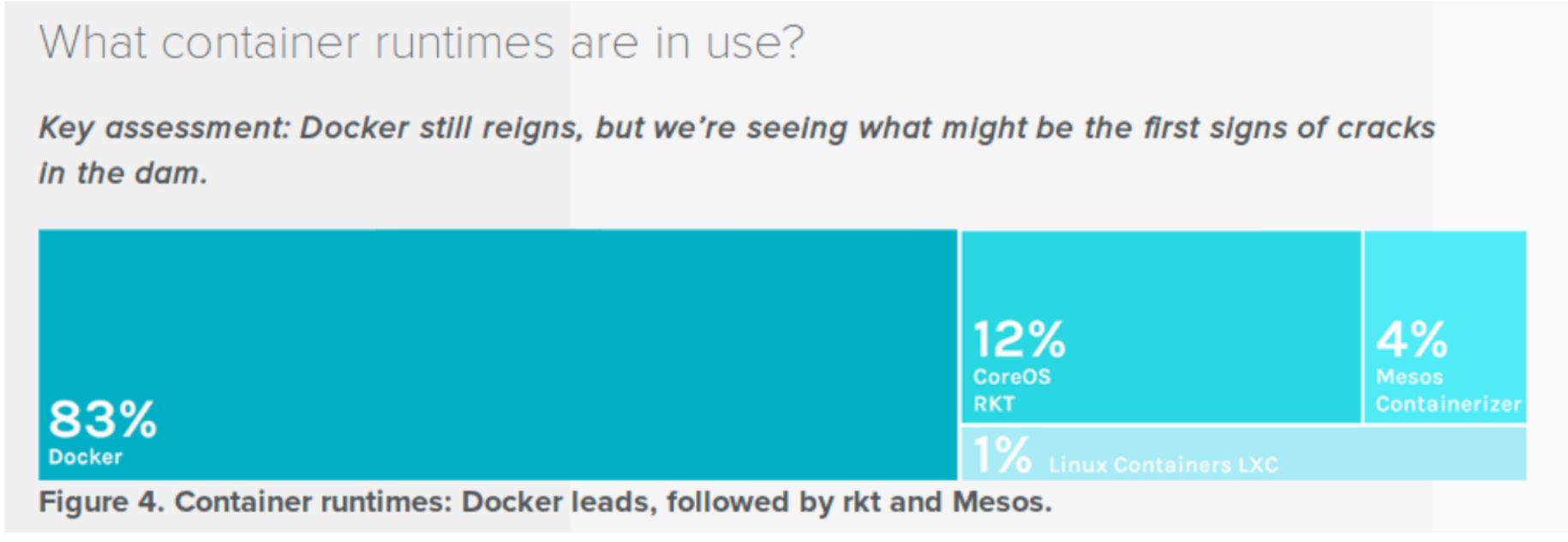


Figure 4. Container runtimes: Docker leads, followed by rkt and Mesos.

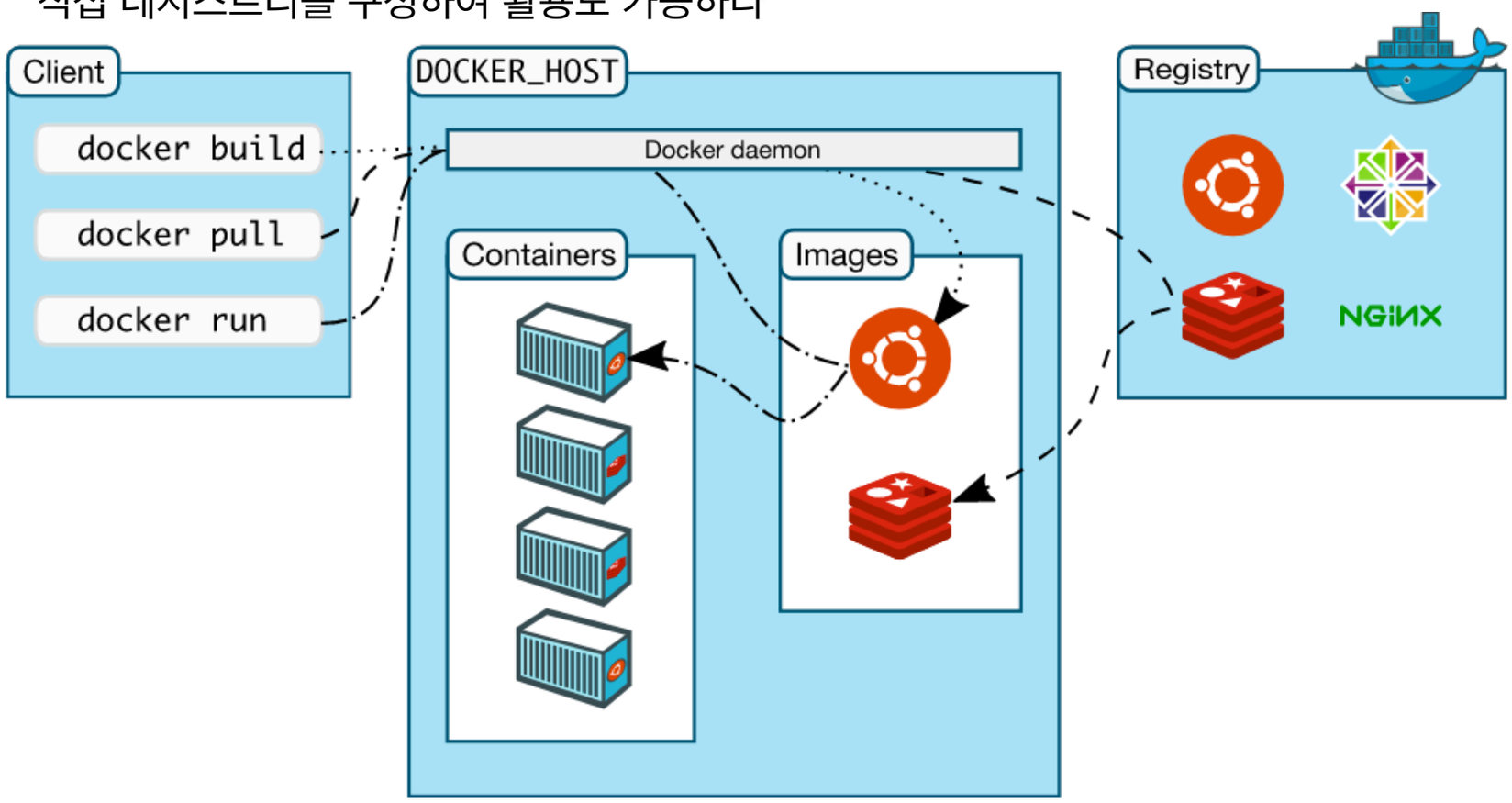
https://siliconangle.com/2018/05/29/report-software-container-adoption-grows-deployments-scale/

2. 배경지식 및 기본 구조

- 리눅스 기반 기술로 개발
  - namespaces
    - 프로세스, 네트워크, 파일시스템 등을 격리할 수 있는 기능
  - cgroups
    - 컨테이너의 하드웨어 리소스(CPU, 메모리 등) 을 제한할 수 있는 기능 제공
  - UnionFS
    - 도커 컨테이너와 이미지는 UnionFS 기반으로 동작
    - Read-Only 인 이미지 레이어와 Writable 인 컨테이너 레이어로 구성
    - 저장 공간을 효율적으로 사용 (COW, Copy-on-Write)

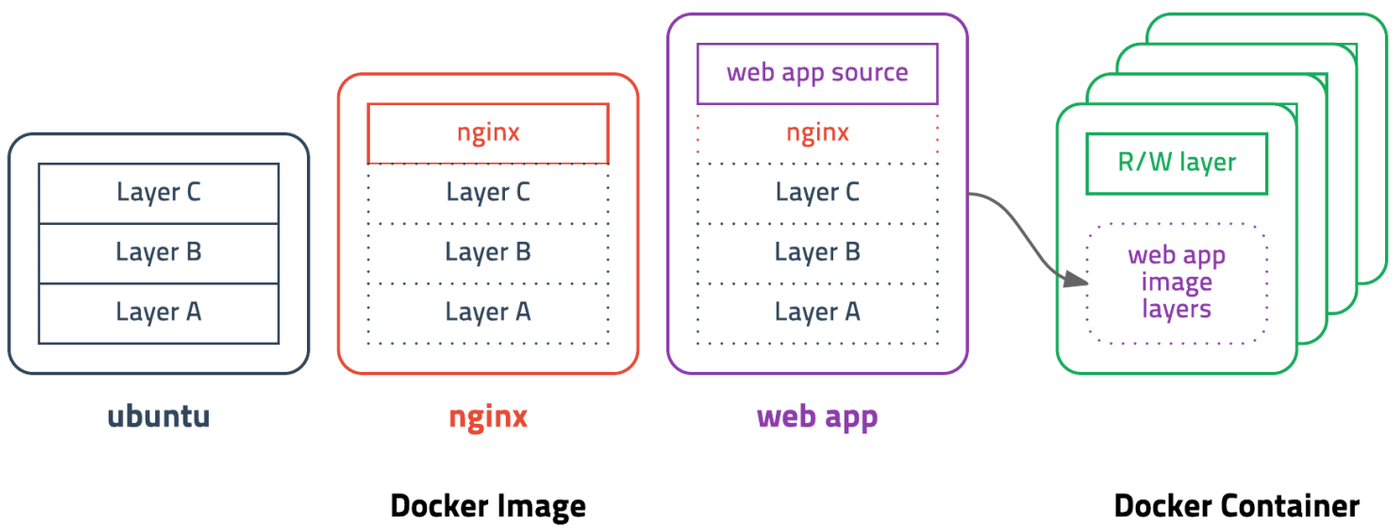
2. 배경지식 및 기본 구조

- 도커 아키텍쳐
  - 도커 클라이언트
    - 도커 사용자에게 도커 데몬과 통신할 수 있는 인터페이스를 제공한다.
      도커 데몬과 동일한 시스템에 설치되거나 원격에서 REST API를 통해 원격 호출도 가능하다.
  - 도커 데몬
    - 이미지 빌드, 컨테이너 실행과 같은 도커에서 무거운 작업 등을 네트워크 및 볼륨과 같은 Docker 객체도 관리한다.
  - 도커 레지스트리
    - 도커 이미지를 저장 및 관리한다. 기본적으로 Docker Hub 에서 이미지를 찾도록 구성되어 있으며 직접 레지스트리를 구성하여 활용도 가능하다

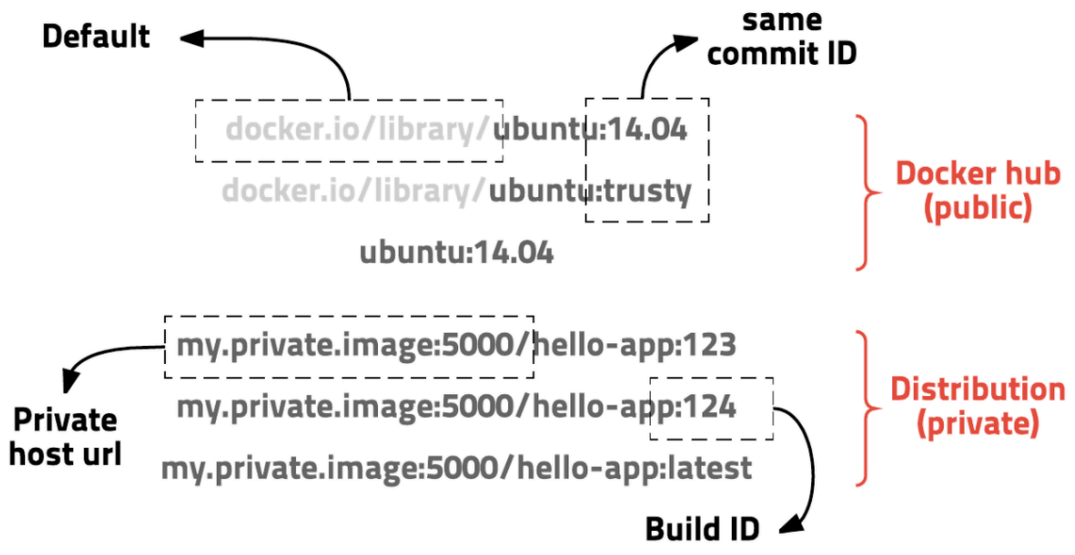


2. 배경지식 및 기본 구조

- 이미지와 컨테이너
  - 컨테이너 - 격리된 공간에서 프로세스가 동작하는 기술
  - 이미지 - 컨테이너 실행에 필요한 파일과 설정값들을 포함하고 있는 것으로 변하지 않음 (Immutable)
  - 이미지와 컨테이너는 여러 개의 레이어로 구성



- 이미지 네이밍
  - [REGISTRY[:PORT]/USER/]REPO[:TAG]]
  - Registry 기본 값은 docker.io
  - 태그를 입력하지 않을 경우, latest 로 조회
  - 이미지를 레지스트리에 업로드 시 네이밍 룰에 맞춰야 함



Docker image url

2. 배경지식 및 기본 구조

- 설치
  - 환경에 맞게 설치
    - Mac: https://docs.docker.com/docker-for-mac/install/
    - Windows: https://docs.docker.com/docker-for-windows/install/

  - 설치 후 테스트
    - 버전 확인

```
limkeunhak@limui-MacBookPro:~/Workspace:> docker --version
Docker version 19.03.8, build afacb8b
```

- Hello World

```
limkeunhak@limui-MacBookPro:~/Workspace:> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:8c5aeeb6a5f3ba4883347d3747a7249f491766ca1caa47e5da5dfcf6b9b717c0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

3. Docker Registry (이미지 조회 및 푸시&풀)

- 도커 레지스트리
  - 도커 이미지 저장소
    - 이미지 접근 권한에 따라 login 필요

  - 이미지 다운로드
    - $ docker pull <이미지>
  - 이미지 업로드
    - $ docker push <이미지>

  - 이미지 조회
    - $ docker search <이미지>

  - 이미지 만들기
    - 구동 중인 컨테이너에서 만들거나 Dockerfile 을 통해 생성 가능

3. Docker Registry (이미지 조회 및 푸시&풀)

- 도커 레지스트리