쉬운 도커

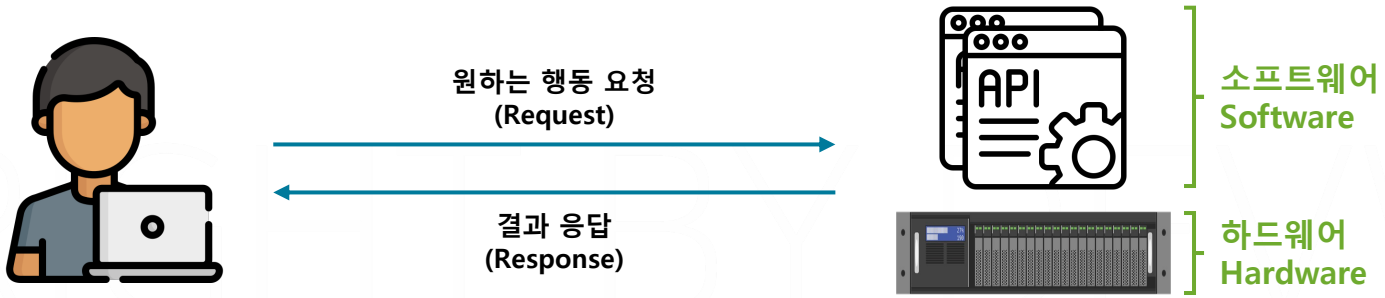
PART1. 가상화 기술

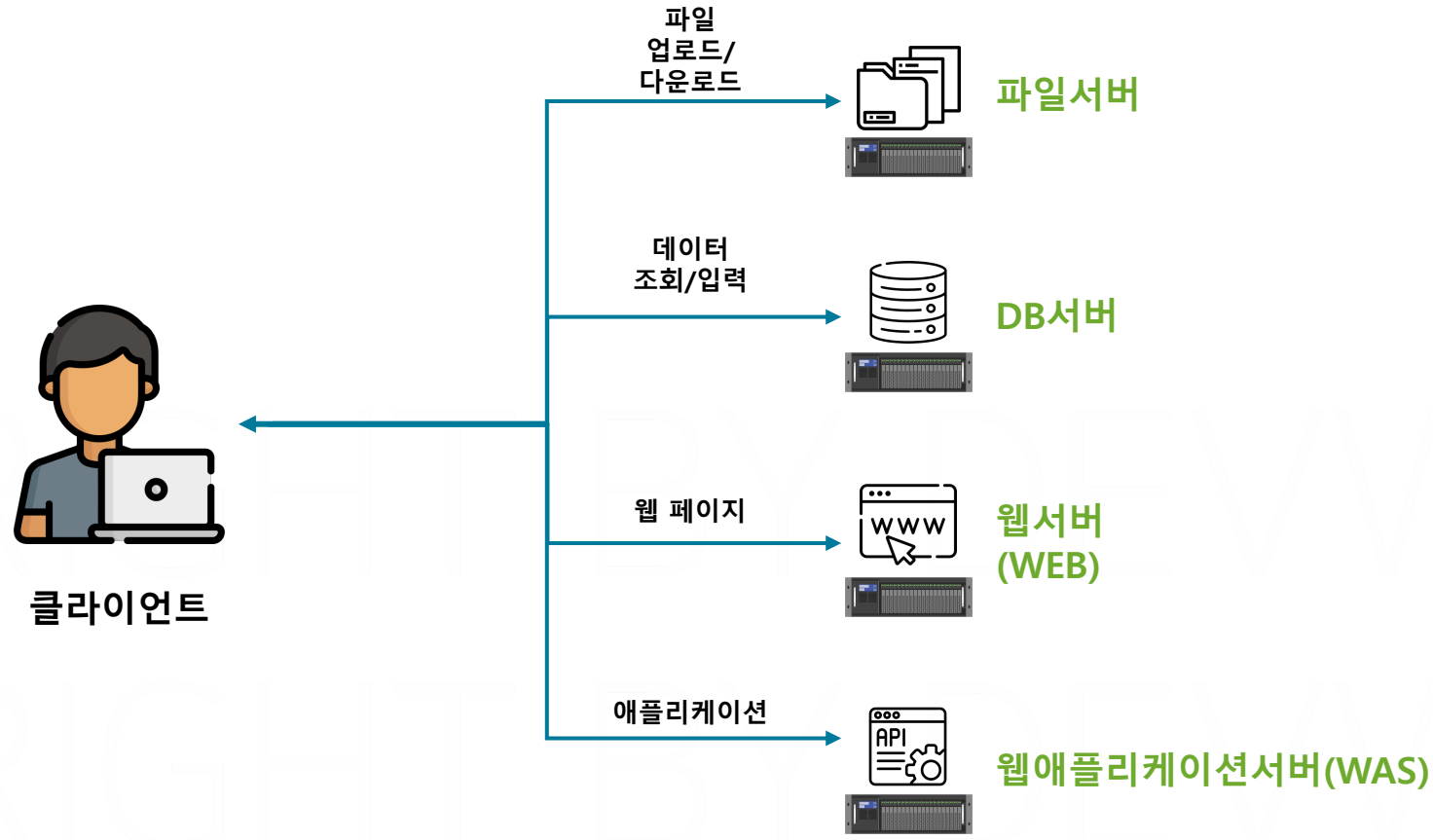
쉬운 도커

PART1. 가상화 기술

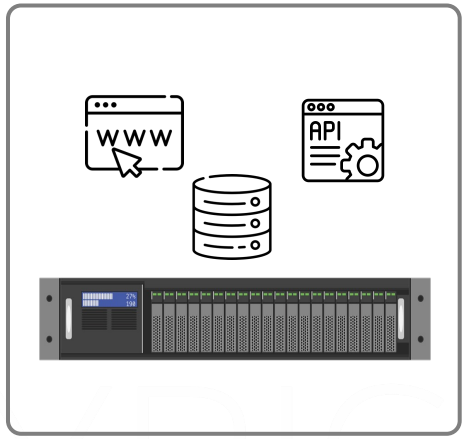
애플리케이션 서버

COPYRIGHT BY DEVMK

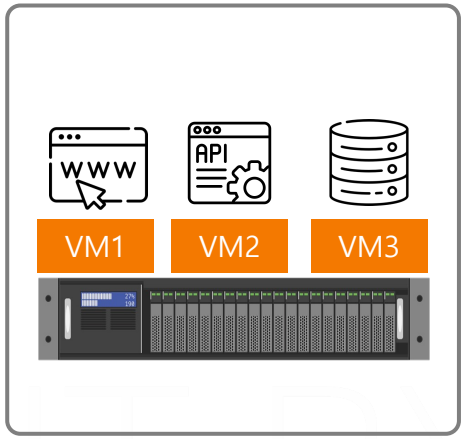



파일
업로드/
다운로드
파일서버
데이터
조회/입력
DB서버
웹 페이지
웹서버
(WEB)
애플리케이션
API
웹애플리케이션서버(WAS)
클라이언트

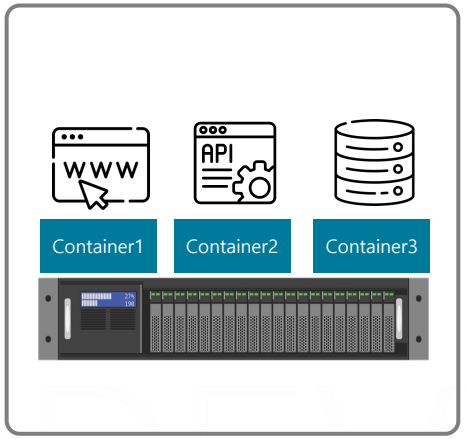
베어메탈
(Baremetal)



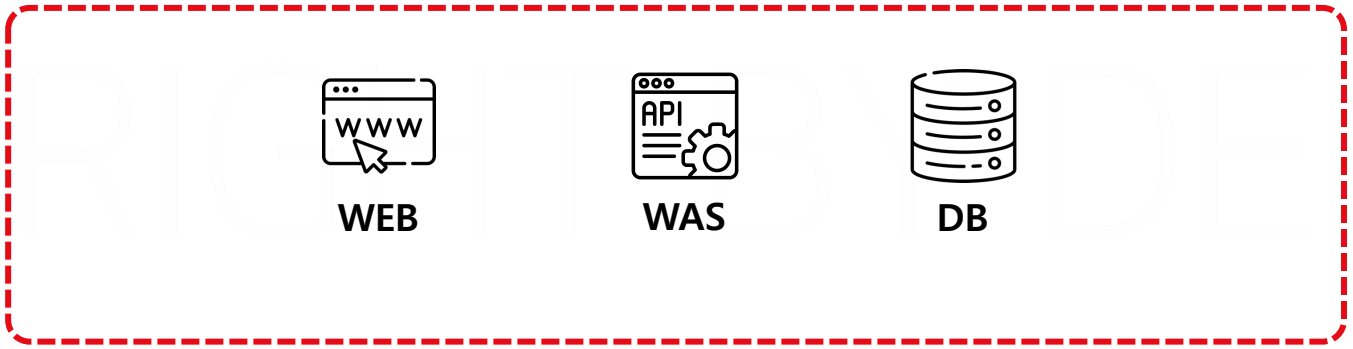
하이퍼바이저
(Hypervisor)



컨테이너
(Container)



엔터프라이즈 운영 환경

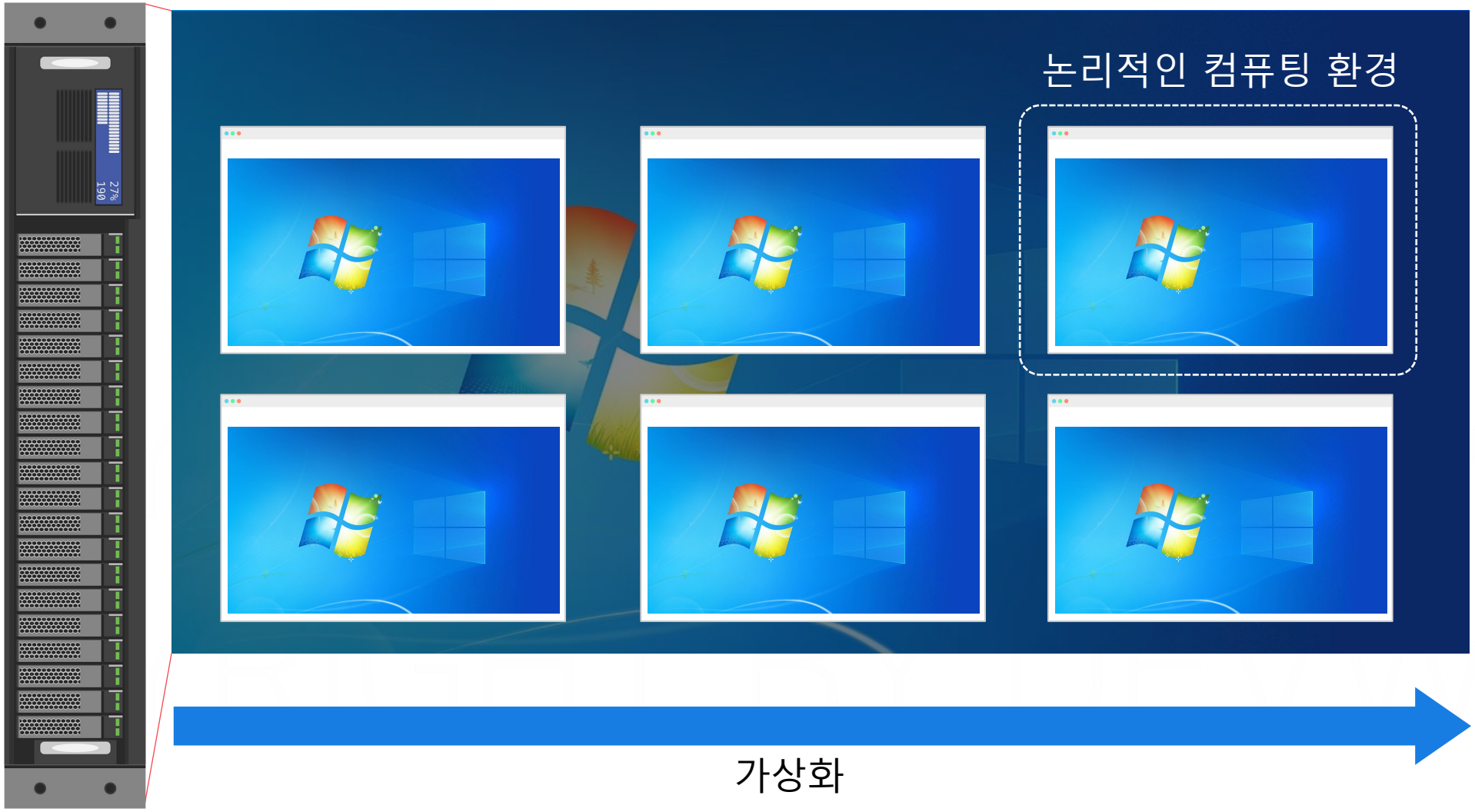


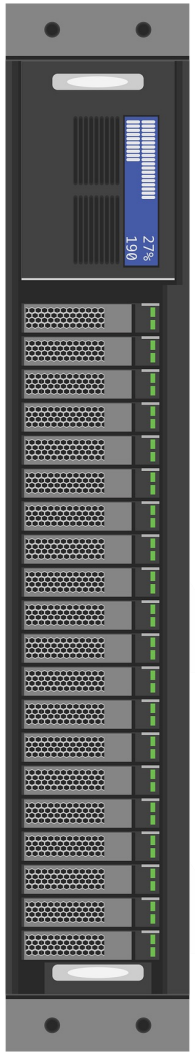
쉬운 도커

PART1. 가상화 기술

가상화 기술과
하이퍼바이저 가상화

COPYRIGHT BY DEVWIKI


논리적인 컴퓨팅 환경
가상화
서버

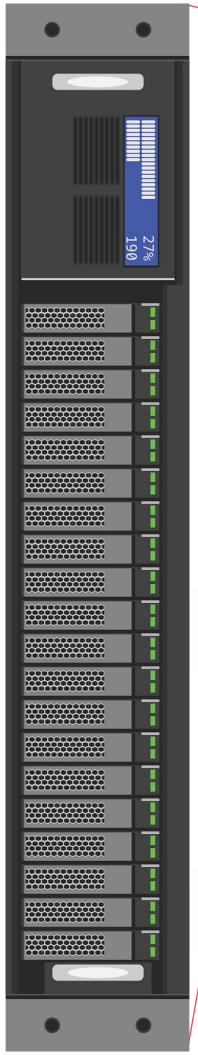


8Core /
64 GB RAM

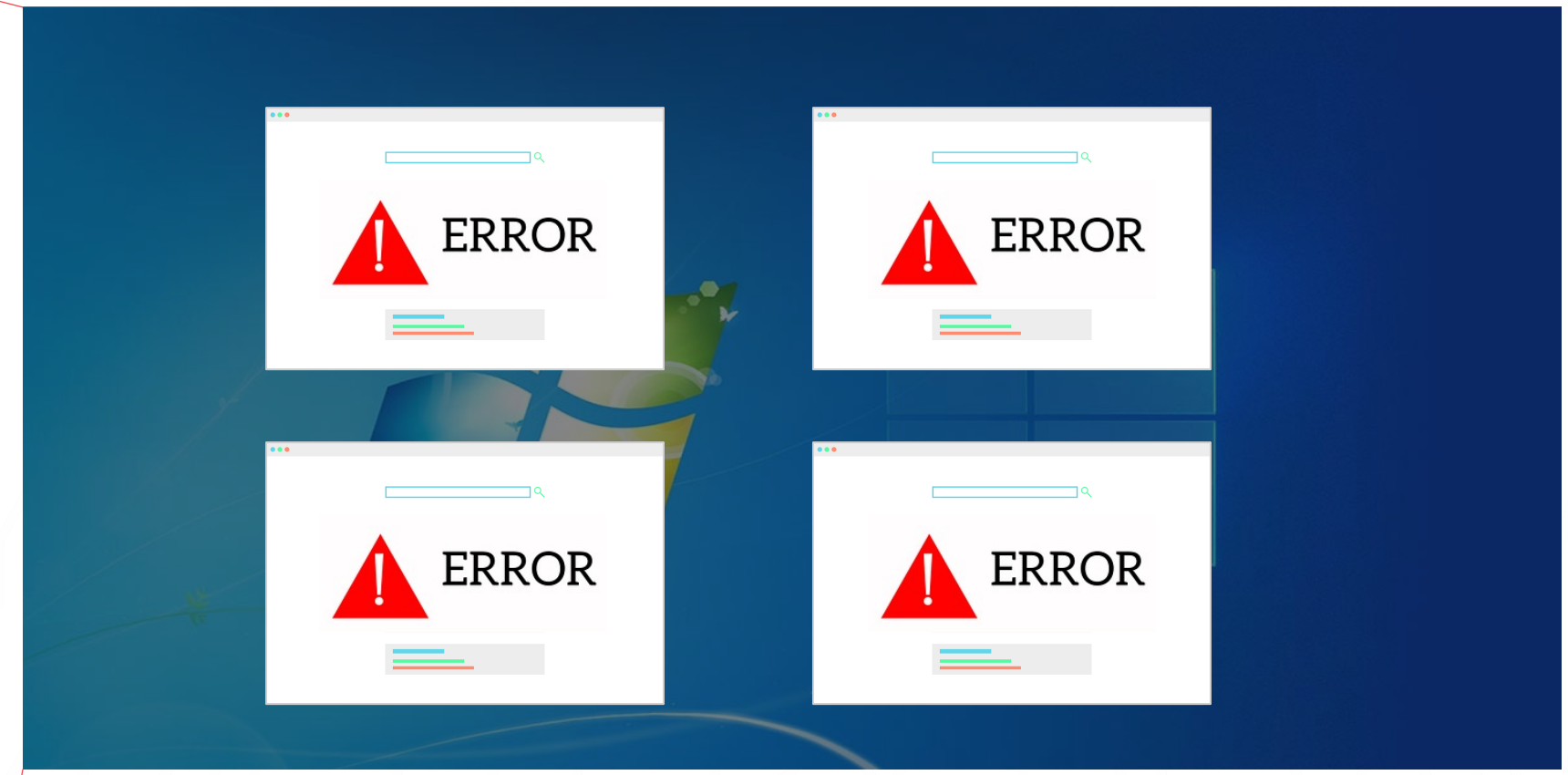
서버

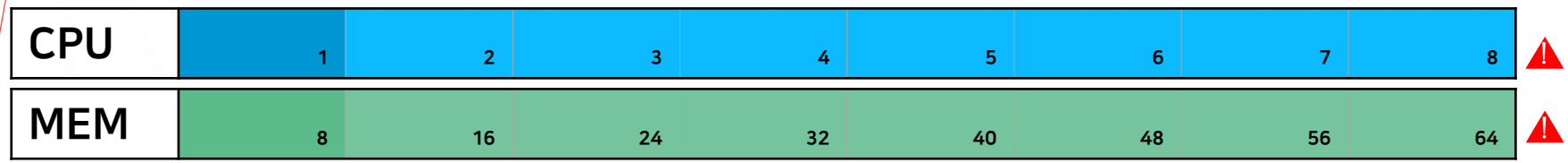
1 Core /
8 GB RAM 사용

4개 = 총
4 Core /
32 GB RAM 사용

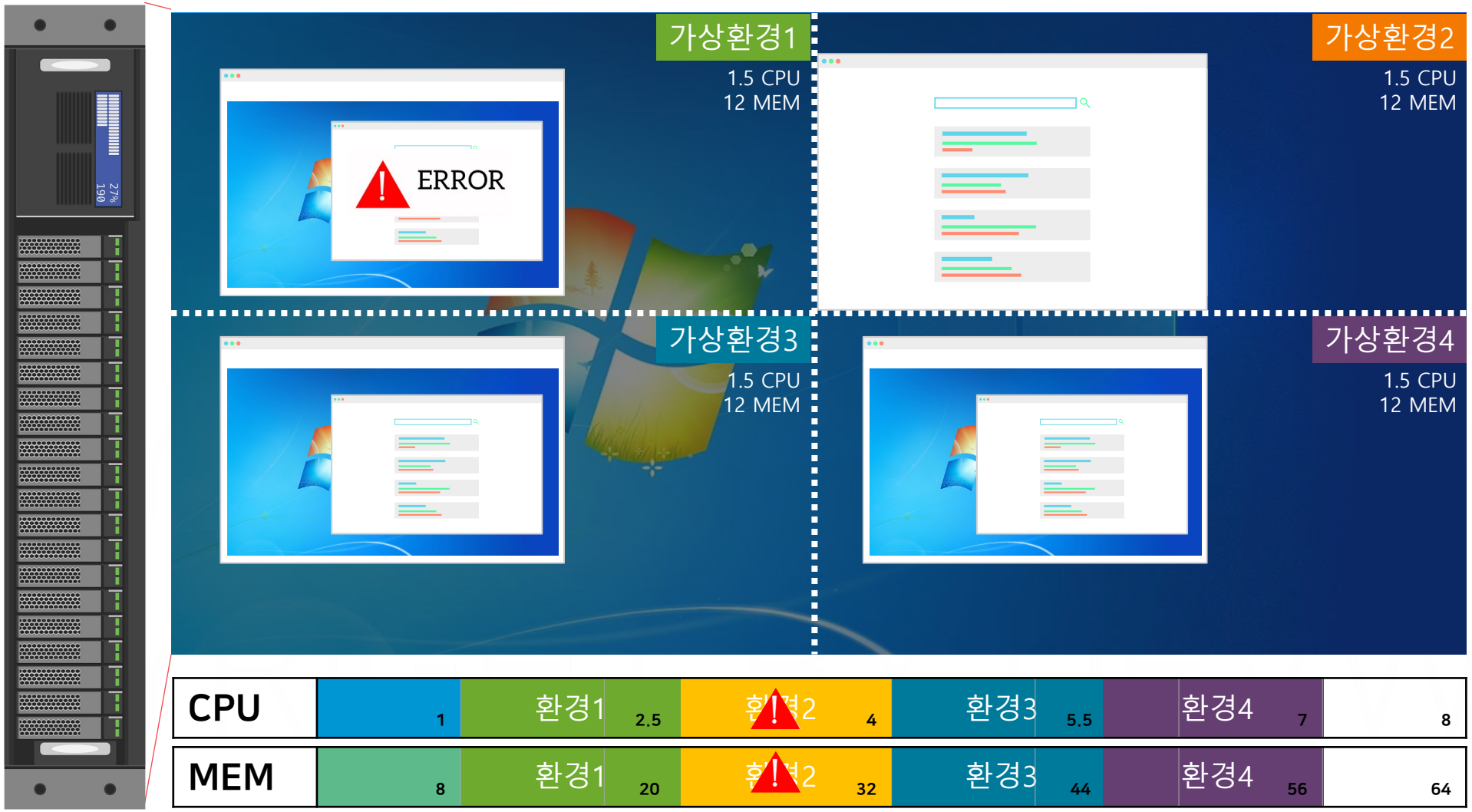


서버



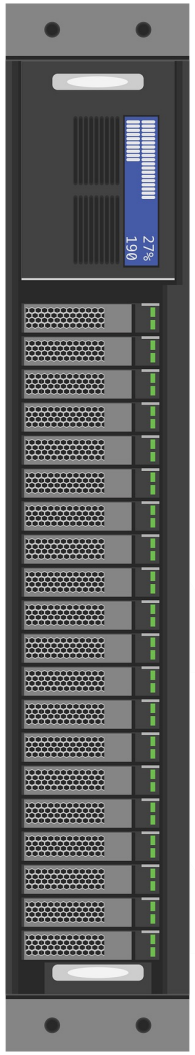


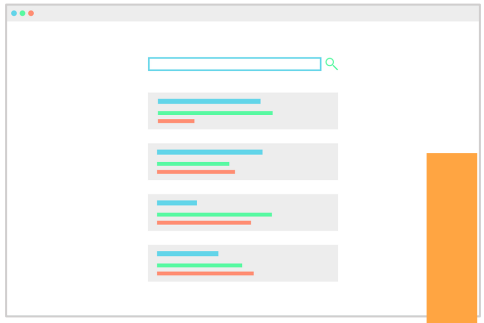
소프트웨어 간의 간섭 문제 발생



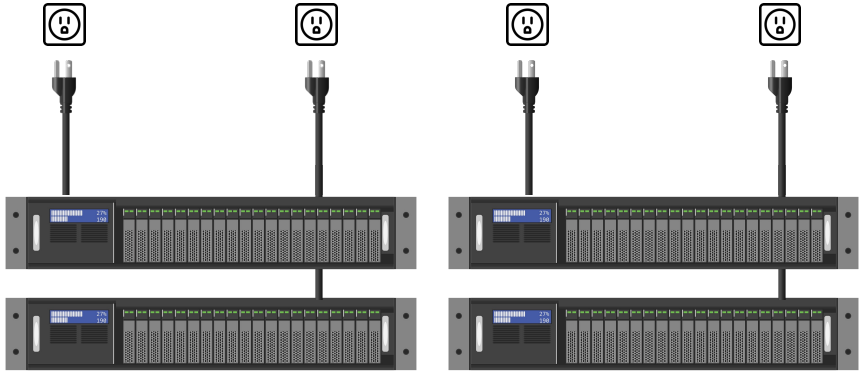
서버

가상화 기술을 사용해 격리된 환경에서 프로세스 실행



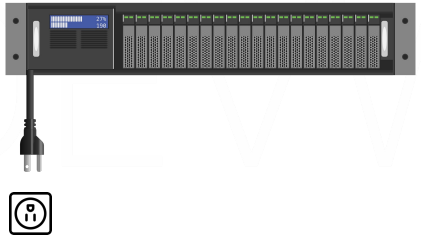


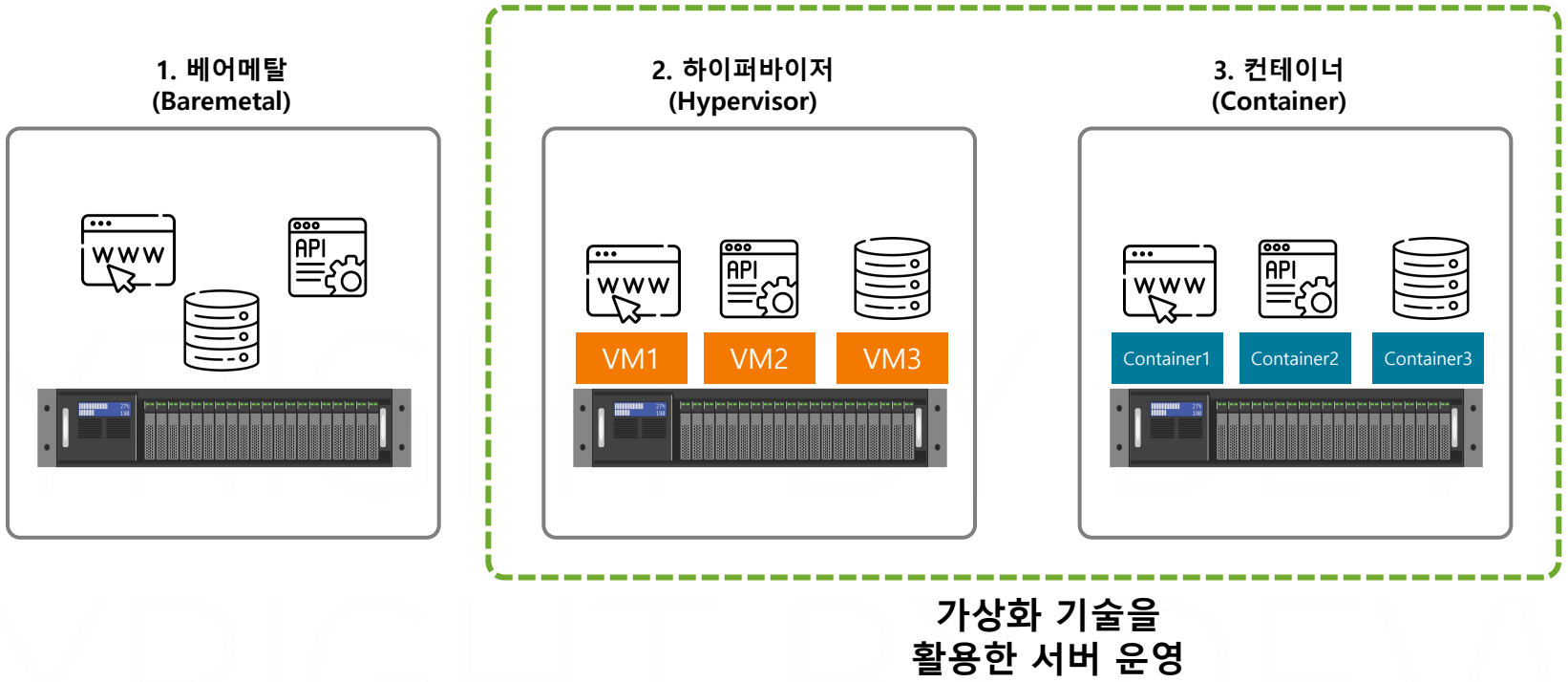
하드웨어 성능 증가 &
소프트웨어의 요구사항
감소



낮은 성능
여러대

높은 성능 한대



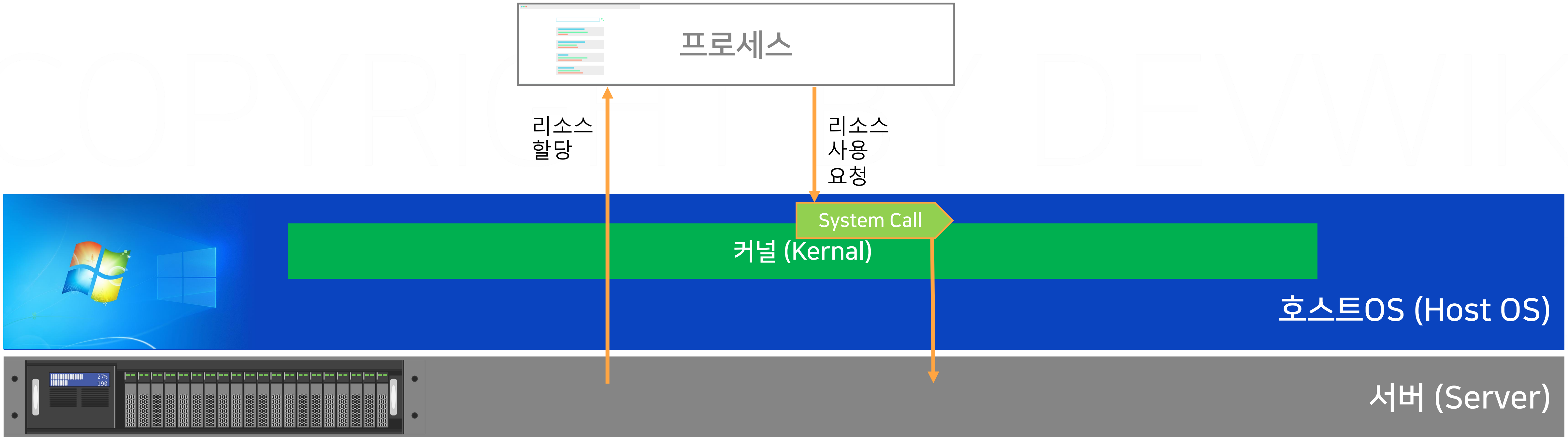


가상화 기술을
활용한 서버 운영

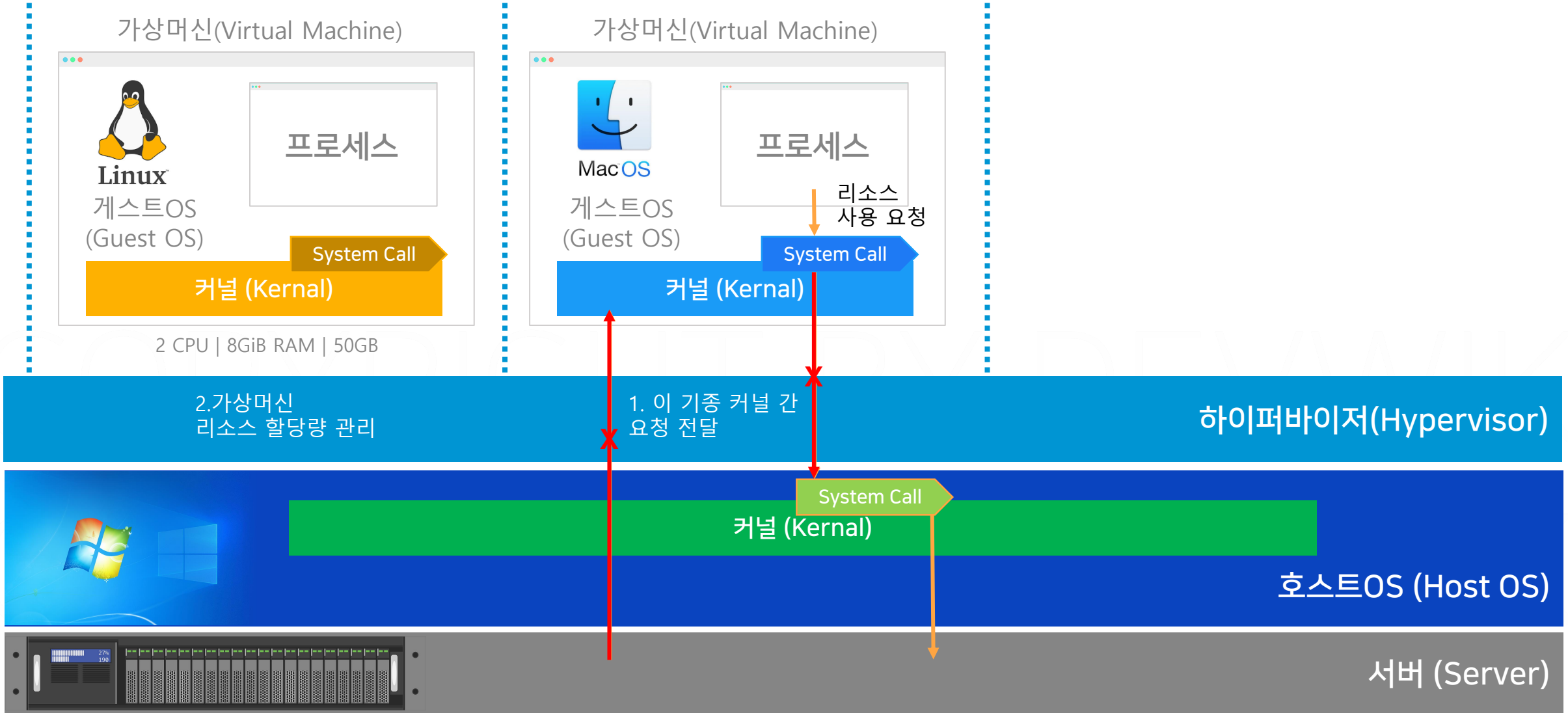
하이퍼바이저 가상화


CPU
OS
머신1
머신2
머신3
MEM
OS
머신1
머신2
머신3
Storage
OS
머신1
머신2
머신3
가상머신1
가상머신2
가상머신3
게스트 OS
프로세스
게스트 OS
게스트 OS
가상환경 관리
하이퍼바이저
가상머신1
CPU 2 / MEM 8
Storage 100GB
실행
가상머신2
CPU 2 / MEM 4
Storage 50GB
실행
가상머신3
CPU 2 / MEM 8
Storage 100GB
실행
서버
호스트OS
(Host OS)

가상화 기술을 사용해 격리된 환경에서 프로세스 실행



하이퍼바이저 가상화


가상머신(Virtual Machine)
Linux
게스트OS
(Guest OS)
프로세스
System Call
커널 (Kernal)
2 CPU | 8GiB RAM | 50GB
가상머신(Virtual Machine)
MacOS
게스트OS
(Guest OS)
프로세스
리소스
사용 요청
System Call
커널 (Kernal)
2.가상머신
리소스 할당량 관리
1. 이 기종 커널 간
요청 전달
하이퍼바이저(Hypervisor)
System Call
커널 (Kernal)
호스트OS (Host OS)
서버 (Server)

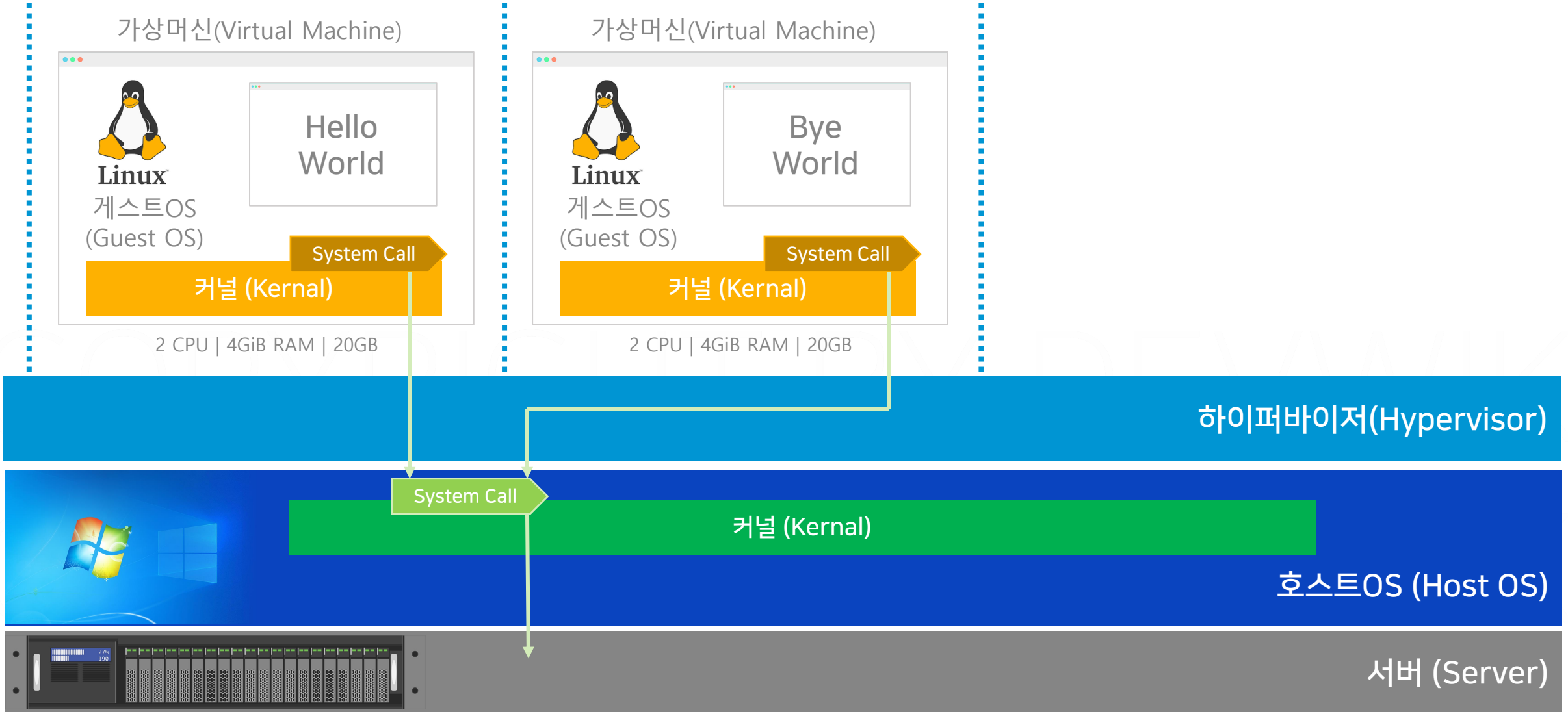
Workstation / ESXi

VirtualBox

Red Hat

RHEV (KVM)

가상화 기술을 사용하기 위해
다양한 회사의 하이퍼바이저 제품 활용


가상머신(Virtual Machine)
Linux
게스트OS
(Guest OS)
Hello World
System Call
커널 (Kernal)
2 CPU | 4GiB RAM | 20GB
가상머신(Virtual Machine)
Linux
게스트OS
(Guest OS)
Bye World
System Call
커널 (Kernal)
2 CPU | 4GiB RAM | 20GB
하이퍼바이저(Hypervisor)
System Call
커널 (Kernal)
호스트OS (Host OS)
서버 (Server)

쉬운 도커

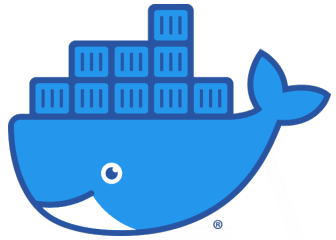
PART1. 가상화 기술

컨테이너 가상화

COPYRIGHT BY DEVWIKI

가볍다

빠르다

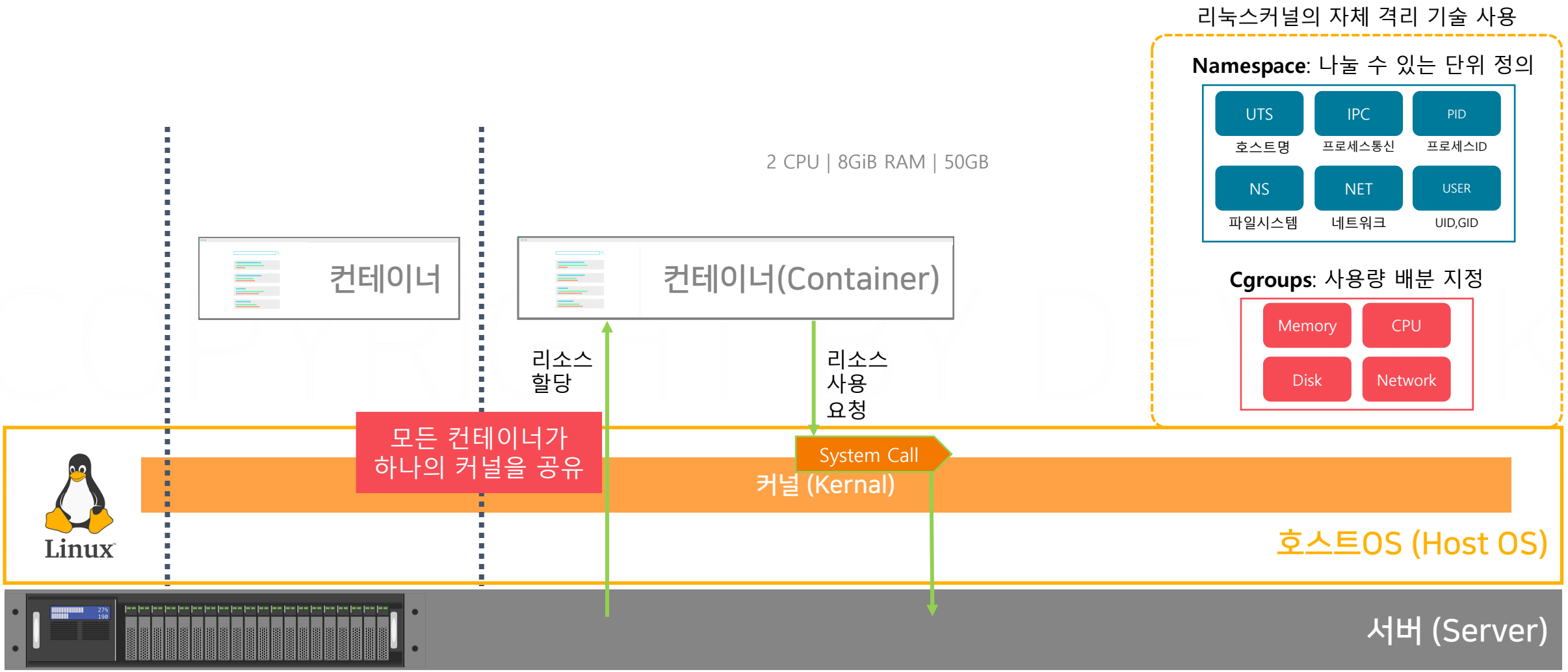


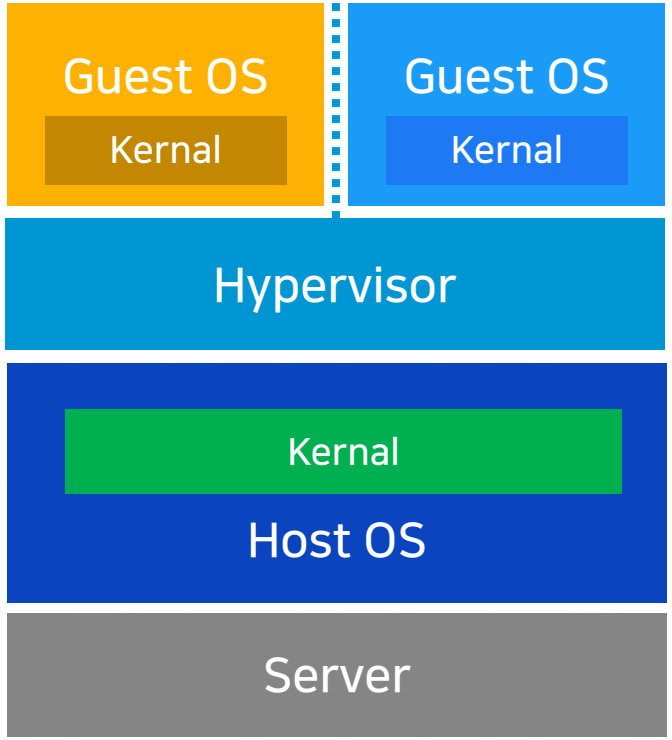
DOCKER

하이퍼바이저
가상화

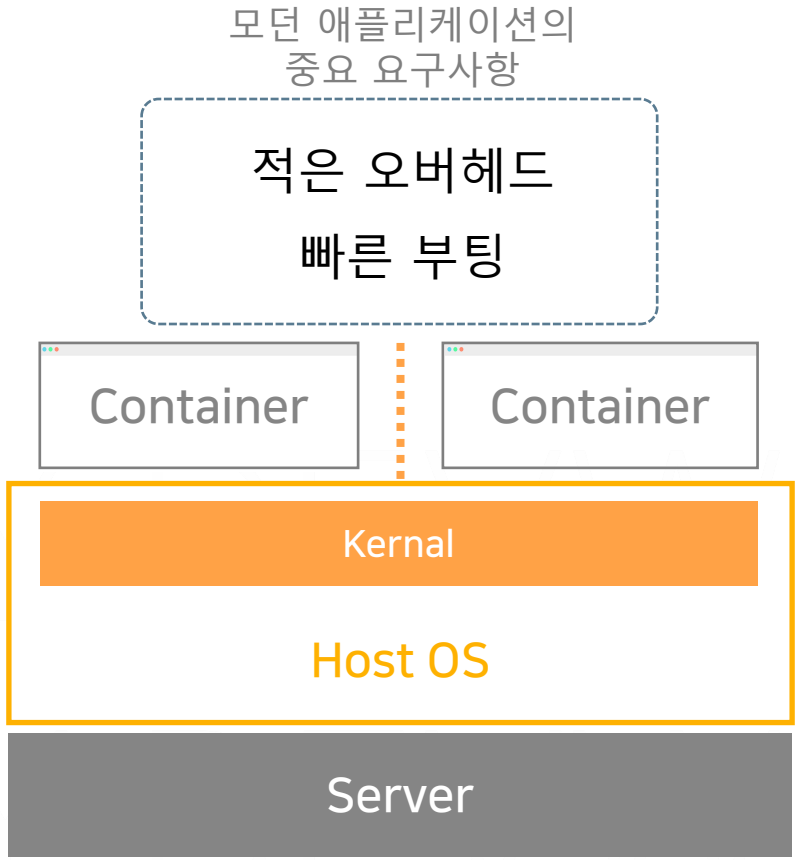
컨테이너 가상화

컨테이너 가상화


리눅스커널의 자체 격리 기술 사용
Namespace: 나눌 수 있는 단위 정의
UTS
호스트명
IPC
프로세스통신
PID
프로세스ID
NS
파일시스템
NET
네트워크
USER
UID,GID
Cgroups: 사용량 배분 지정
Memory
CPU
Disk
Network
2 CPU | 8GiB RAM | 50GB
컨테이너
컨테이너(Container)
리소스 할당
리소스 사용 요청
모든 컨테이너가 하나의 커널을 공유
System Call
커널 (Kernal)
Linux
호스트OS (Host OS)
서버 (Server)

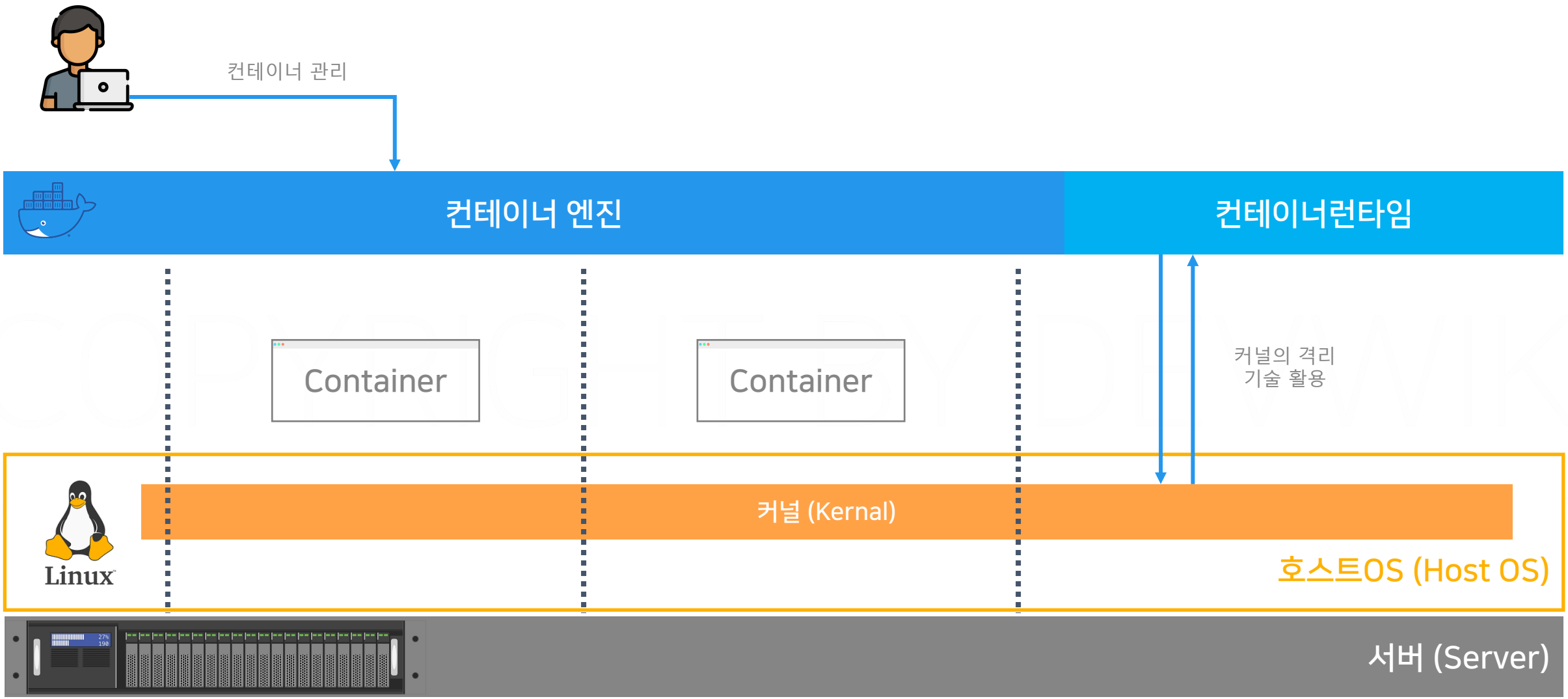


하이퍼바이저
가상화



컨테이너 가상화

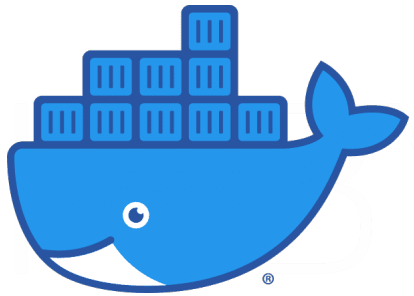
컨테이너 가상화


컨테이너 관리
컨테이너 엔진
컨테이너런타임
Container
Container
커널의 격리
기술 활용
커널 (Kernal)
Linux
호스트OS (Host OS)
서버 (Server)

쉬운
도커

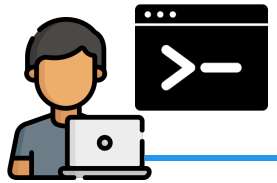
PART1. 가상화 기술

도커(Docker)



DOCKER

도커 아키텍처



컨테이너 관리

RUNC

컨테이너 엔진

컨테이너런타임

OCI 표준 구현

Container

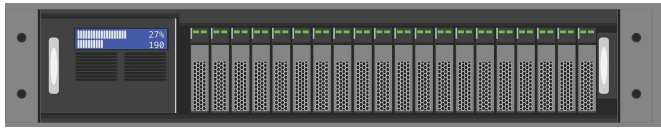
Container

Container Runtime Interface

Linux

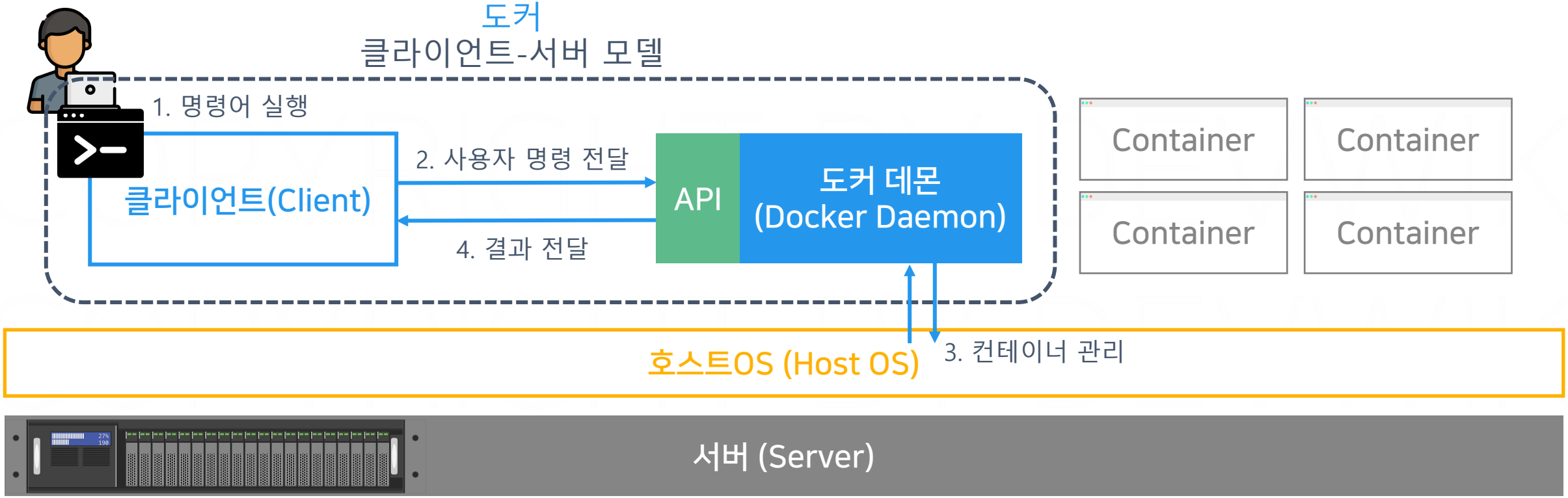
커널 (Kernal)

호스트OS (Host OS)



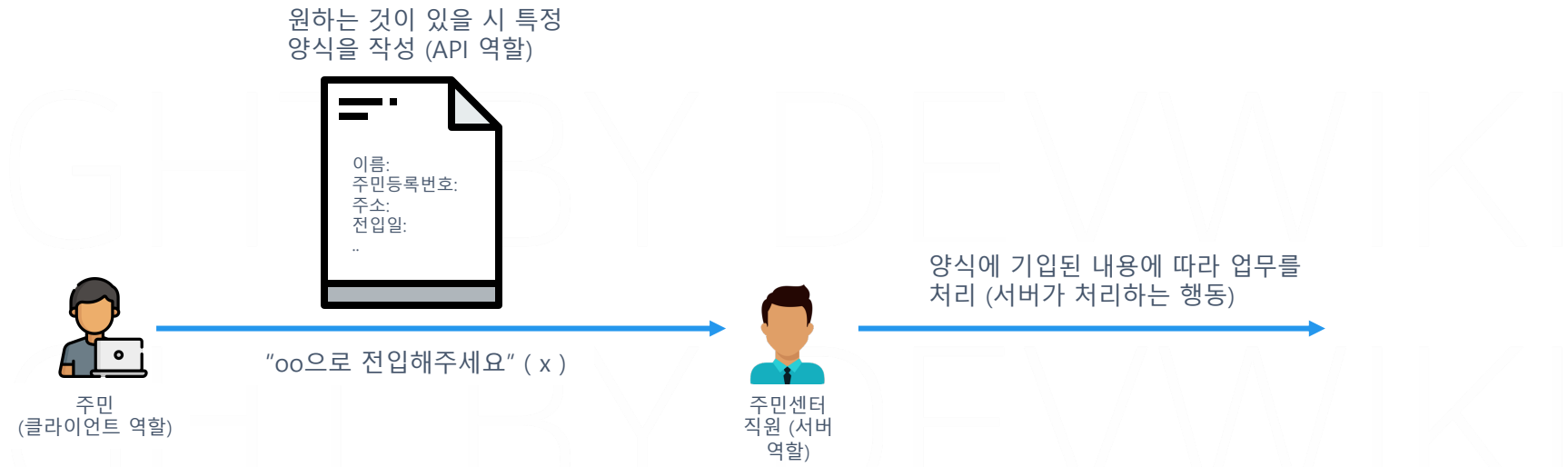
서버 (Server)

도커 아키텍처


도커
클라이언트-서버 모델
1. 명령어 실행
클라이언트(Client)
2. 사용자 명령 전달
4. 결과 전달
API
도커 데몬
(Docker Daemon)
Container
Container
Container
Container
호스트OS (Host OS)
3. 컨테이너 관리
서버 (Server)

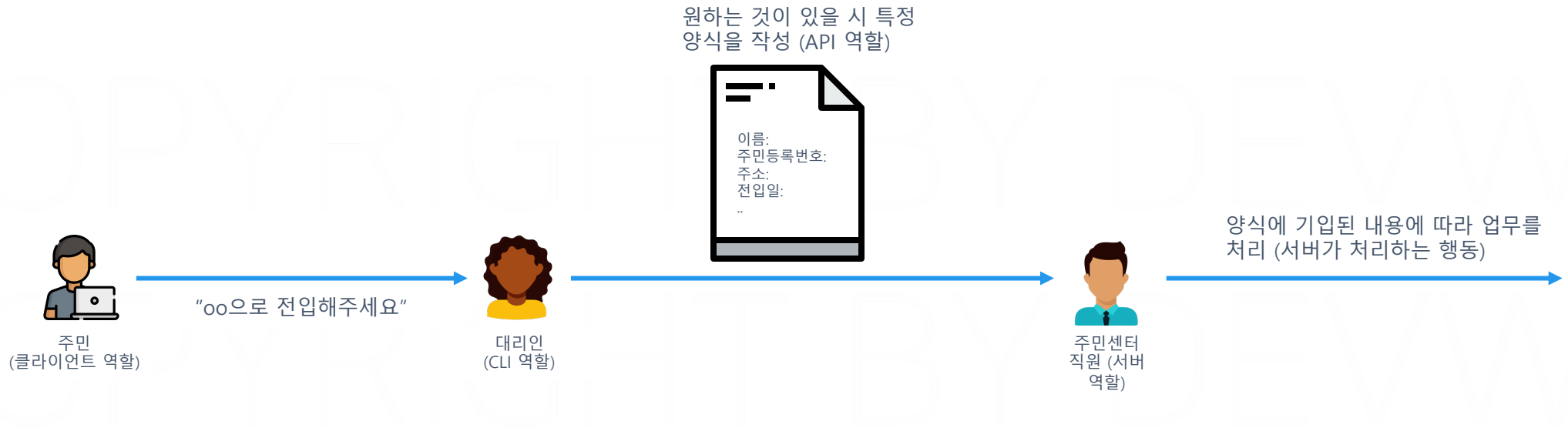
도커 아키텍처

https://docs.docker.com/engine/api/v1.41/



주민센터

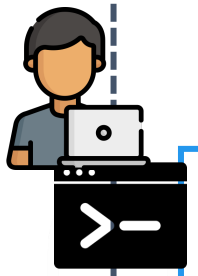
도커 아키텍처


원하는 것이 있을 시 특정 양식을 작성 (API 역할)
이름:
주민등록번호:
주소:
전입일:
..
양식에 기입된 내용에 따라 업무를 처리 (서버가 처리하는 행동)
"oo으로 전입해주세요"
주민
(클라이언트 역할)
대리인
(CLI 역할)
주민센터
직원 (서버 역할)

주민센터

도커 아키텍처

도커
클라이언트-서버 모델



1. 명령어 실행
$ docker ps

클라이언트(Client)

CLI(Command Line Interface)

2. 사용자 명령을 API에 맞추어 변환

3. 데몬에 API 요청

API

- 컨테이너 실행
- 컨테이너 조회
- 컨테이너 삭제

4. 결과 전달
(컨테이너 리스트)

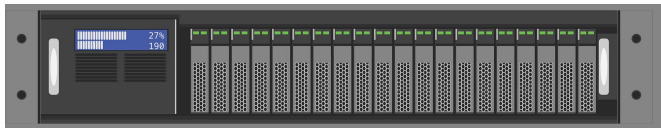
5. 결과 표시

도커 데몬
dockerd

Container Runtime Interface

3. 컨테이너 관리

호스트OS (Host OS)



서버 (Server)

쉬운 도커

PART1. 가상화 기술

컨테이너 실행

docker version

Client, Server의 버전 및 상태 확인

docker info

플러그인, 시스템 상세 정보 확인

Docker 설치 상태 및 버전 확인

1. docker 버전 확인

```
docker version
```

2. docker 시스템 정보 확인

```
docker info
```

docker --help

메뉴얼 확인

docker (Management Command) Command

대분류 소분류

docker container run .. -> docker run ..

Docker 설치 상태 및 버전 확인

1. docker 가이드 확인

```
docker --help
docker container --help
docker container run --help
```

docker run (실행옵션) 이미지명

컨테이너 실행

docker rm 컨테이너명/ID

컨테이너 삭제

Docker 설치 상태 및 버전 확인

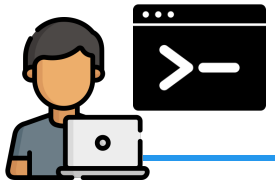
1. nginx 컨테이너 실행, http://localhost 접속 및 화면 확인

```
docker run -p 80:80 --name hellonginx nginx
```

2. 컨테이너 삭제

```
docker rm hellonginx
```

컨테이너 실행



1. docker run .. nginx

컨테이너 엔진

컨테이너런타임

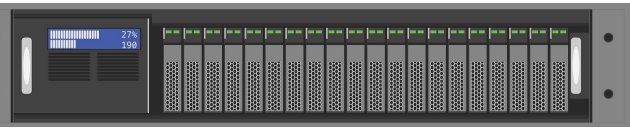
Name: hellonginx
ID: 000xxx

Nginx Process

Memory | CPU | Disk | Network

커널 (Kernal)

호스트OS (Host OS)



서버 (Server)