가천대학교 2024년도 1학기

시스템 아키텍처

Epic.2

이기훈 교수
(litkhai@gachon.ac.kr)

Chapter 1.
3tier 아키텍처

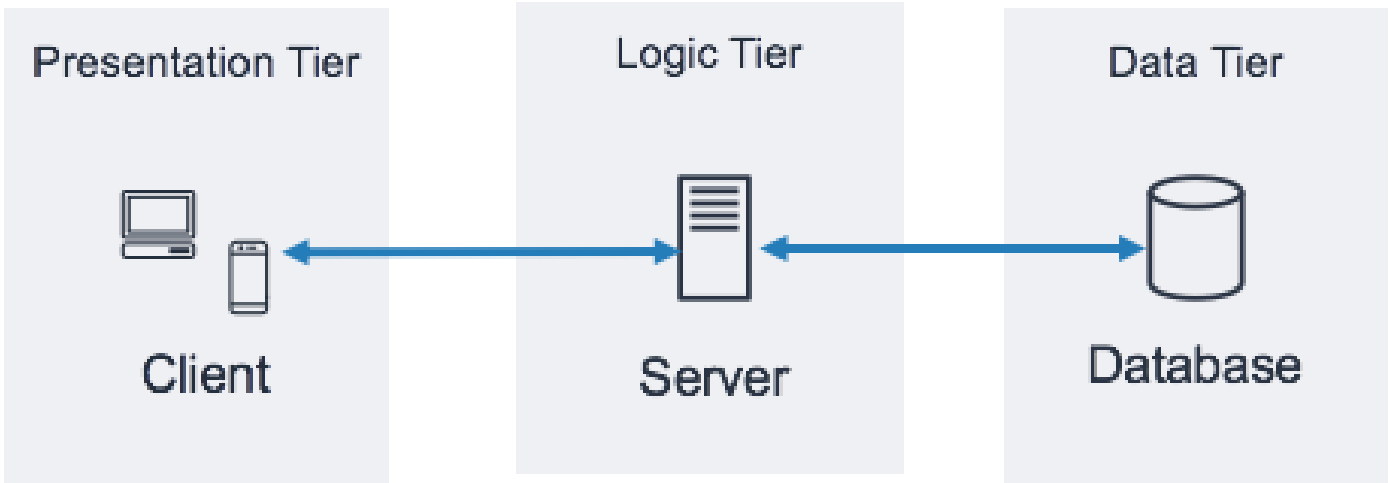
학습목표

3tier 아키텍처

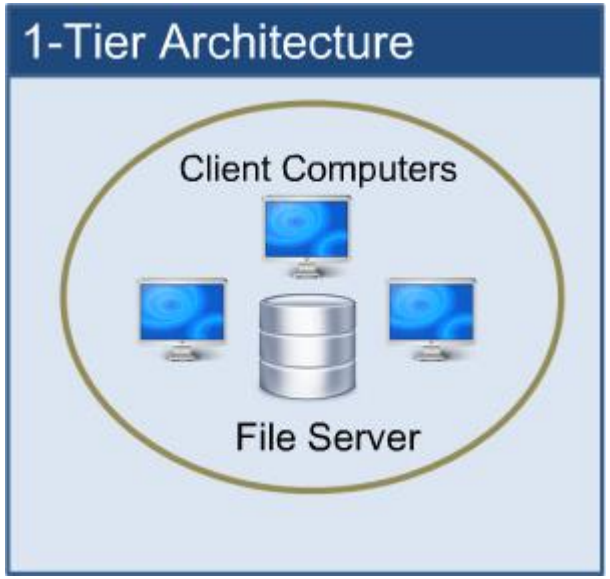
- 3tier 아키텍처가 등장하기까지의 배경을 이해한다
- 3tier 구조의 특징을 설명할 수 있다
- 3tier 구조와 클라우드를 연결지을 수 있다

3-tier architecture

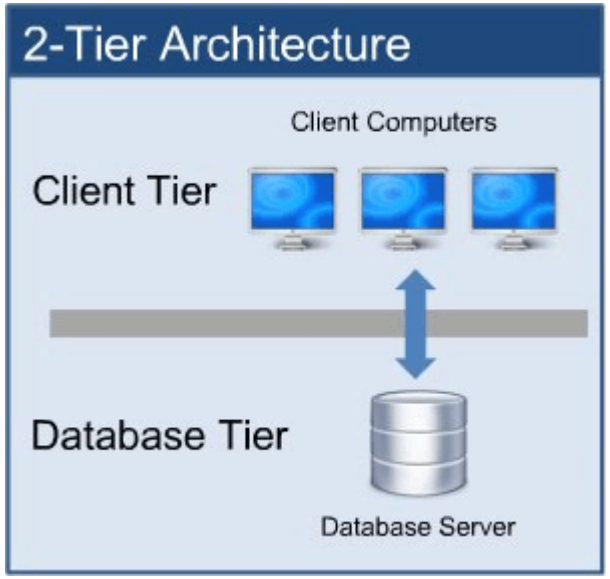
3-tier architecture는 모든 계층이 3개의 논리 계층으로 분할되는 아키텍처 패턴이다. 다중 계층 아키텍처에서 가장 널리 사용되며 단일 프레젠테이션 계층, 로직 계층 및 데이터 계층으로 구성된다.



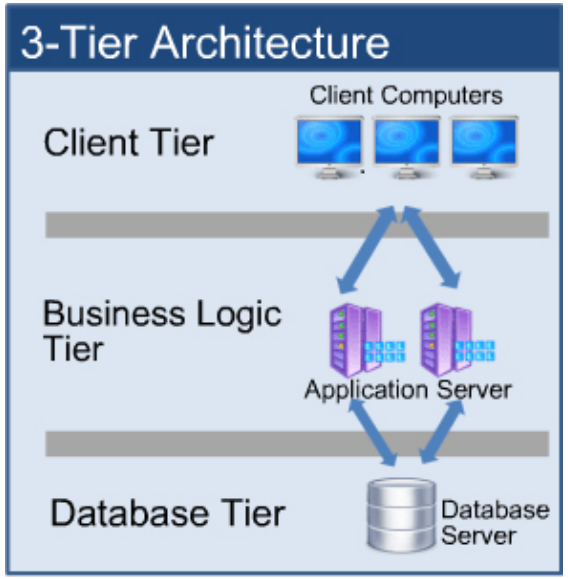
N-tier의 기원



단일 계층 아키텍처는 동일한 tier, 즉 클라이언트 tier에 프레젠테이션 layer, 비즈니스 layer 및 데이터 layer가 있다. 이름에서 알 수 있듯이 모든 계층과 구성요소를 동일한 시스템에서 사용할 수 있다.



클라이언트 tier에서 프레젠테이션 layer와 어플리케이션 layer를 모두 처리하고 서버는 데이터베이스 layer를 처리한다.
2계층 아키텍처는 '클라이언트-서버 어플리케이션'이라고도 한다.
2계층 아키텍처에서 통신은 클라이언트와 서버 간에 이루어지는데 클라이언트 시스템은 요청을 서버 시스템으로 전송하고 서버 시스템은 요청을 처리한 후 응답을 클라이언트 시스템으로 다시 보낸다.



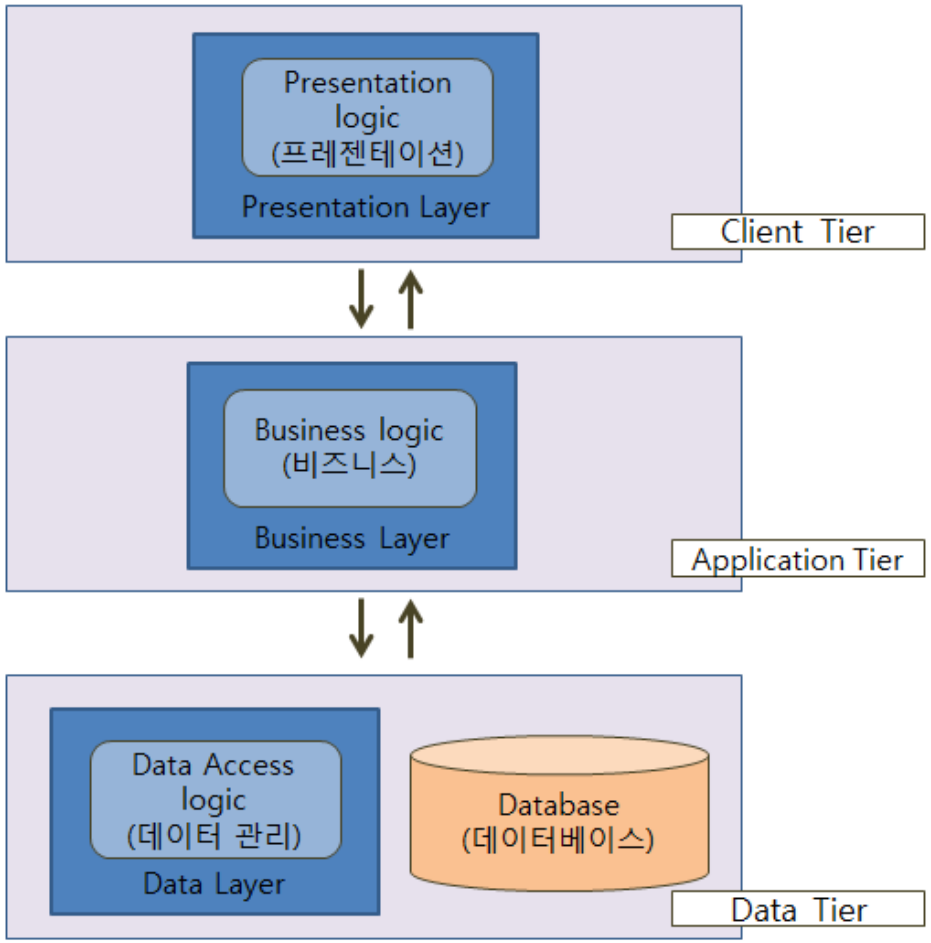
세 개의 주요 층이 모두 분리되어 있다. 프레젠테이션layer는 클라이언트tier에 있고 어플리케이션 layer는 미들웨어 역할을 하며 비즈니스 Tier에 있다. 그리고 데이터layer는 데이터tier에서 사용 할 수 있다. 매우 일반적인 아키텍처이다

3tier 구조 살펴보기

프레젠테이션 계층 (Presentation Tier)
사용자가 직접 마주하게 되는 계층이다. 따라서 주로 사용자 인터페이스(인터넷 브라우저 등)를 지원하며 이 계층은 GUI 또는 프론트엔드(front-end) 라고도 부른다. 그러므로 이 계층에서는 사용자 인터페이스와 관계없는 데이터를 처리하는 로직은 포함하지 않는다. 주로 웹 서버를 예시로 들 수 있고, HTML, Javascript, CSS 등이 이 계층에 해당 된다.

어플리케이션 계층 (Application Tier)
이 계층에서는 (프레젠테이션 계층) 요청되는 정보를 어떠한 규칙을 바탕으로 처리하고 가공하는 것들을 담당한다. (동적인 데이터 제공!) 비즈니스 로직 계층 또는 트랜잭션 계층 이라고도 한다. 첫 번째 계층에서 이 계층을 바라볼 때에는 서버처럼 동작하고(응답), 세 번째 계층의 프로그램에 대해서는 마치 클라이언트처럼 행동한다.(요청)
따라서 이 계층은 미들웨어(Middleware) 또는 백엔드(back-end)라고도 불린다. 이 계층에서는 프레젠테이션코드 (예를 들면 HTML, CSS)나 데이터 관리를 위한 코드는 포함하지 않는다. 주로 PHP, Java 등이 이 계층에 해당한다.

데이터 계층 (Data Tier)
데이터 계층은 데이터베이스와 데이터베이스에 접근하여 데이터를 읽거나 쓰는 것을 관리하는 것을 포함한다.
주로 DBMS (Database Management System)이 이 계층에 해당된다. 데이터 계층 또한 백엔드(back-end)라고도 부른다. 주로 MySQL, MongoDB 등이 이 계층에 해당된다.



3tier 의 장단점

•**빠른 개발**: 각 계층이 서로 다른 팀에서 동시에 개발될 수 있으므로, 기업은 애플리케이션을 보다 빠르게 시장에 출시할 수 있으며 프로그래머는 각 계층에 최신 및 최상의 언어와 툴을 사용할 수 있다.

•**개선된 확장성**: 필요에 따라 어느 계층이든 다른 계층과 독립적으로 확장할 수 있다.

•**개선된 신뢰성**: 한 계층의 가동 중단은 다른 계층의 가용성 또는 성능에 별로 영향을 미치지 않는다.

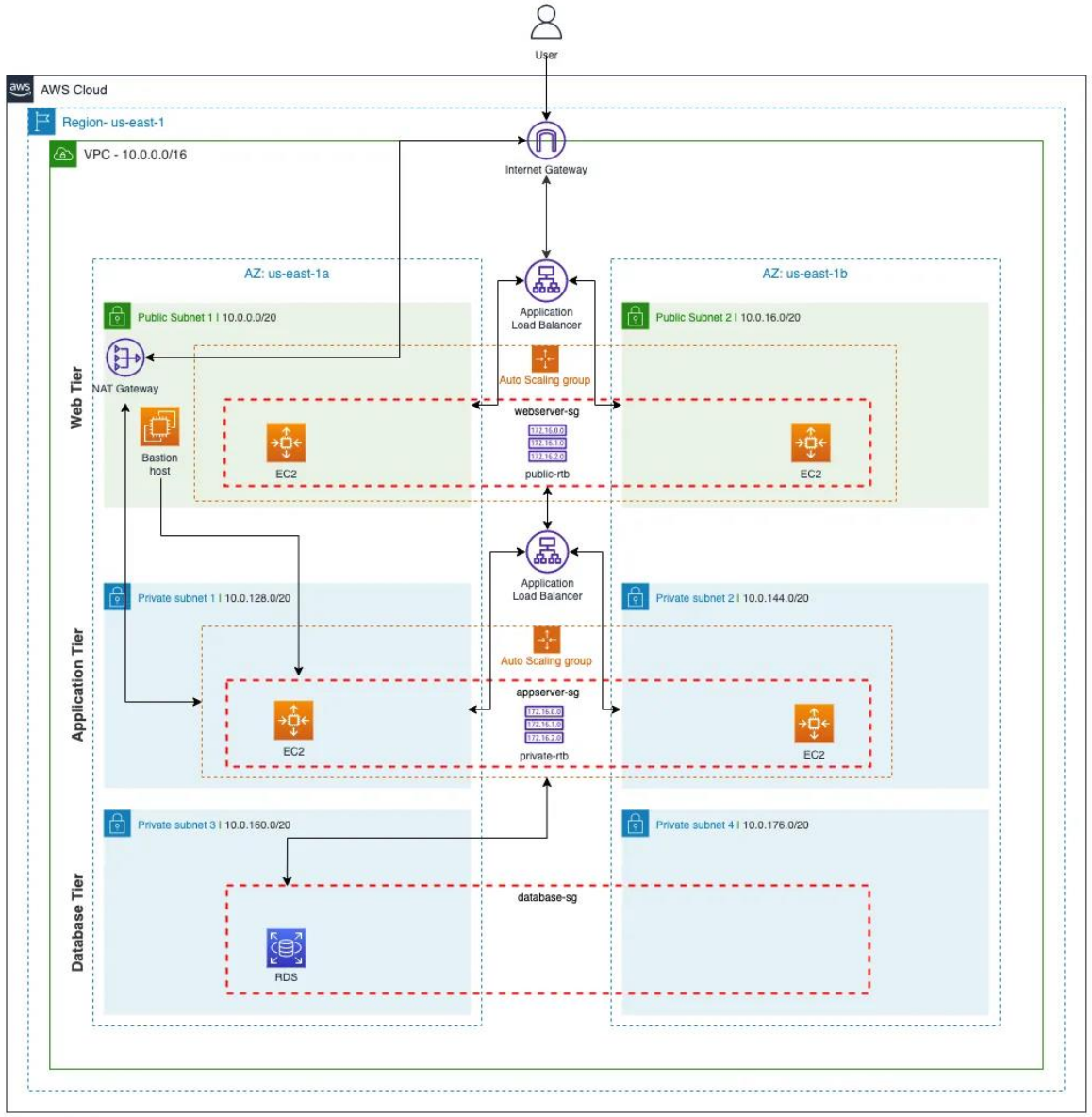
•**개선된 보안**: 프레젠테이션 계층과 데이터 계층이 직접 통신할 수 없으므로, 잘 설계된 애플리케이션 계층은 내부 방화벽의 일종으로 작동하여 SQL 인젝션 및 기타 악의적 공격을 방지할 수 있다.

Web의 3-tier

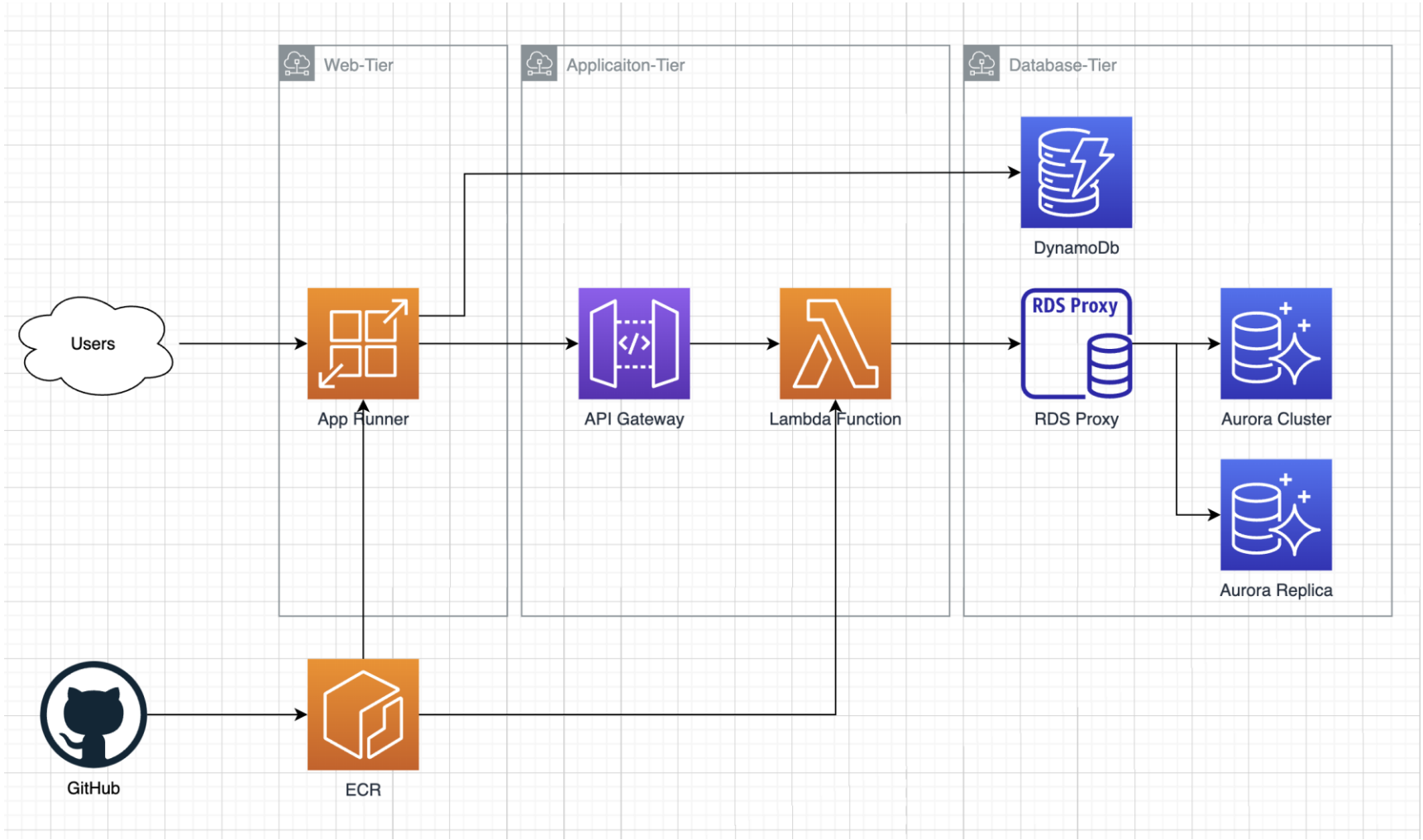
•**웹 서버는** 프레젠테이션 계층이며 사용자 인터페이스를 제공한다. 이는 일반적으로 사용자가 장바구니에 상품을 추가하거나 지불 정보를 추가하거나 계정을 작성하는 전자상거래 사이트와 같은 웹 페이지 또는 웹 사이트이다. 콘텐츠는 정적 또는 동적일 수 있으며, 이는 일반적으로 HTML, CSS 및 Javascript를 사용하여 개발된다.

•**애플리케이션 서버** 는 사용자 입력을 처리하는 데 사용되는 비즈니스 논리를 수용하는 중간 계층에 해당한다. 전자상거래 사례를 계속하자면, 이는 인벤토리 데이터베이스를 조회하여 제품 가용성을 리턴하거나 고객 프로파일에 세부사항을 추가하는 계층다. 이 레이어는 종종 Python, Ruby 또는 PHP를 사용하여 개발되며, 예를 들어 e Django, Rails, Symphony 또는 ASP.NET 등의 프레임워크를 실행한다.

•**데이터베이스 서버** 는 웹 애플리케이션의 데이터 또는 백엔드 계층이다. 이 서버는 데이터베이스 관리 소프트웨어(예: MySQL, Oracle, DB2 또는 PostgreSQL)에서 실행된다.



클라우드 트렌드에 따른 3-tier 응용



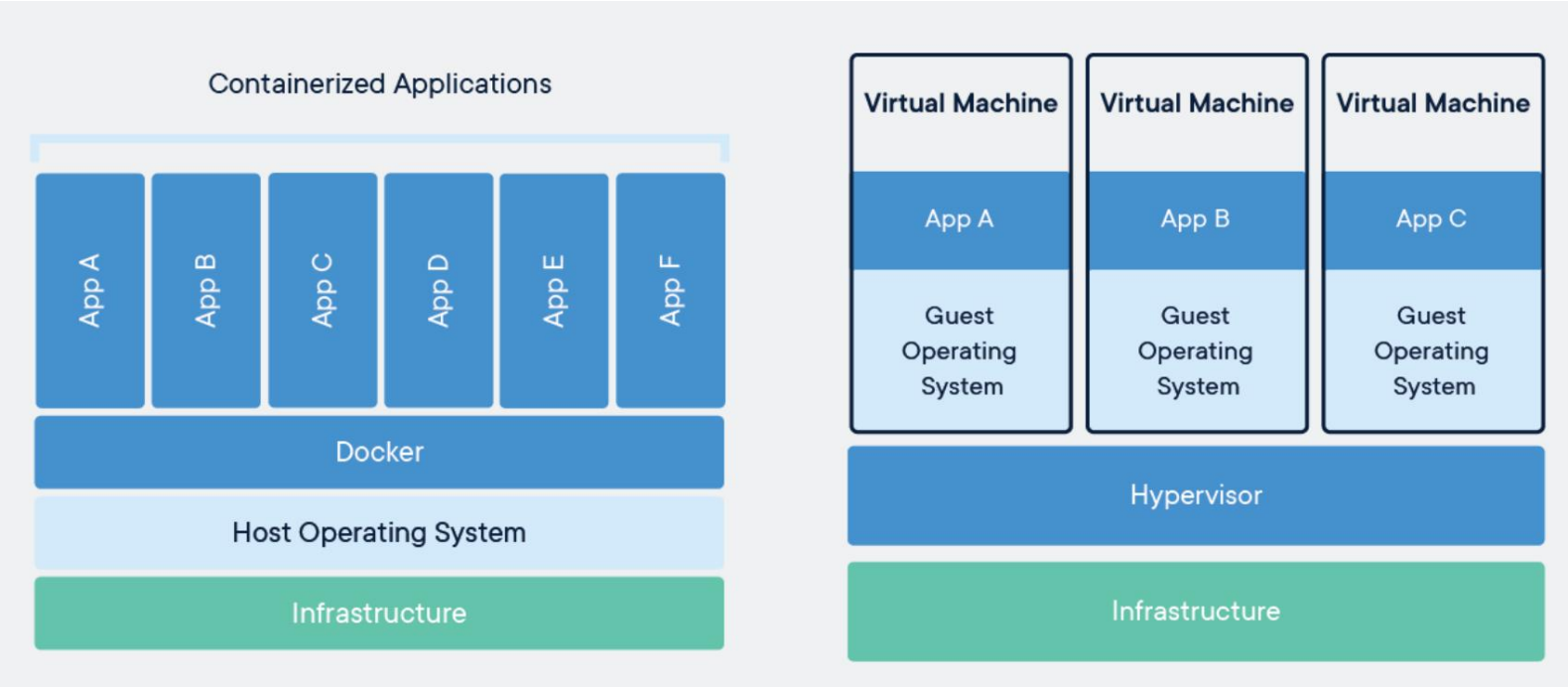
Chapter 2.
컨테이너의 이해

학습목표

컨테이너의 이해

- 컨테이너의 특징에 대해서 이해한다
- 기본적인 가상화의 발전 방향을 이해한다
- 컨테이너의 필요성이 증대된 이유를 설명할 수 있다

컨테이너는 왜 필요해졌을까?



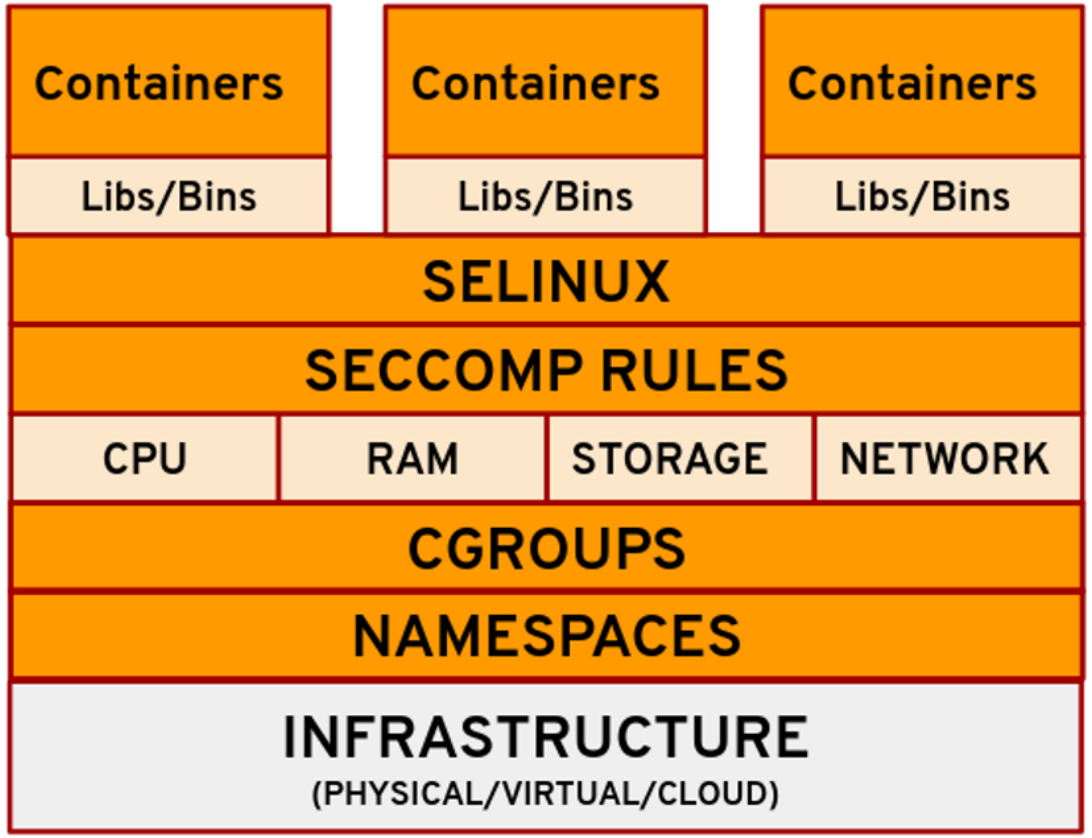
소프트웨어는 OS와 라이브러리에 의존성을 띈다. 그러므로 하나의 컴퓨터에서 성격이 다른(OS,라이브러리 버전이 다른) 소프트웨어를 한번에 실행할 때 어려움을 가질 수 있고 관련된 구성을 관리하기가 어렵다.

컨테이너(Container)는 개별 Software의 실행에 필요한 **실행환경을 독립적으로 운용할 수 있도록** 기반환경 또는 다른 실행환경과의 간섭을 막고 **실행의 독립성을 확보해주는 운영체계 수준의 격리 기술**을 말합니다.
컨테이너는 애플리케이션을 실제 구동 환경으로부터 추상화할 수 있는 논리 패키징 메커니즘을 제공한다.

역사

컨테이너의 간략한 역사
우리가 컨테이너 기술이라고 부르는 개념은 2000년에 처음 등장한 FreeBSD Jail이며, FreeBSD 시스템을 여러 하위 시스템 또는 Jail로 분할할 수 있도록 하는 기술이다. Jail은 시스템 관리자가 조직 내외부의 여러 사용자와 공유할 수 있는 안전한 환경으로 개발되었음

2001년, Jacques Gélinas의 Vserver 프로젝트를 통해 Linux에 격리된 환경이 구현되었다. Linux에서 다수의 통제된 사용자 공간에 이러한 기반을 마련한 후에 오늘날의 Linux 컨테이너가 형태를 갖추기 시작했다.
더 많은 기술이 매우 빠르게 결합되면서 이러한 격리된 접근 방식이 실제화 되었는데, 제어 그룹(cgroups)은 프로세스 또는 프로세스 그룹의 리소스 사용을 제어하고 제한하는 커널 기능 등의 개발이 여기에 해당한다. Cgroups는 사용자 공간을 설정하고 해당 프로세스를 관리하는 초기화 시스템인 systemd를 사용하여 이처럼 격리된 프로세스를 더 효과적으로 제어한다. 이러한 기술은 모두 Linux에 대한 전반적인 제어 기능을 추가하면서 환경을 분리된 상태로 유지할 방법을 제시하는 프레임워크가 되었다.

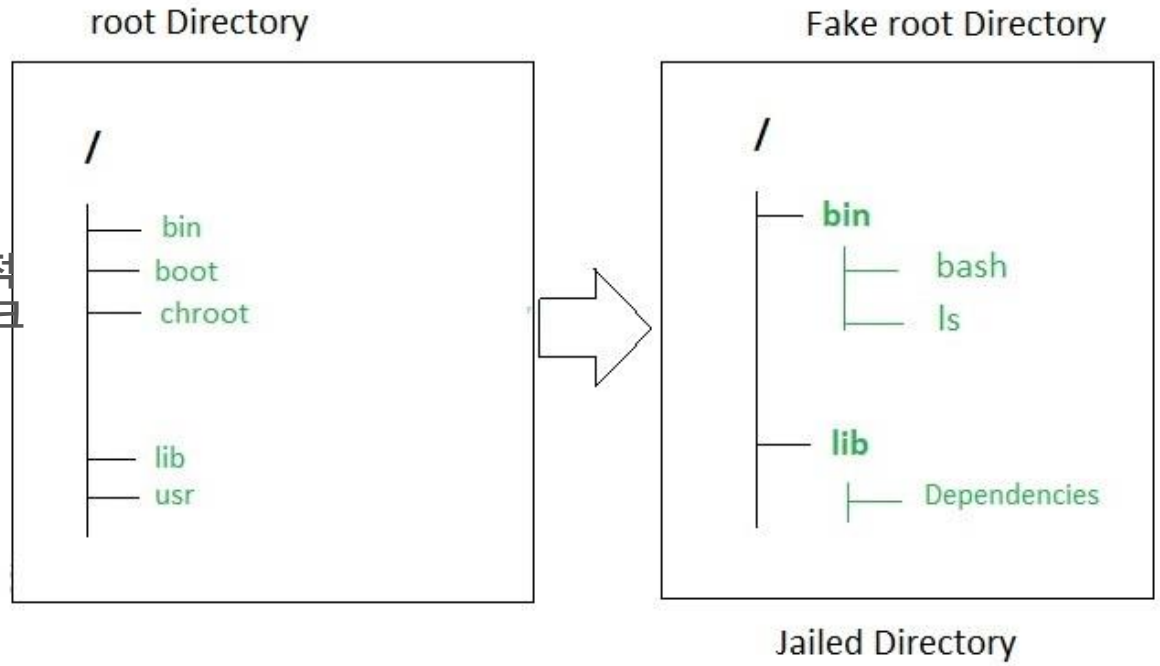


역사 – FreeBDS 기반 Chroot Jail

Chroot and Chroot Jail
최신 컨테이너의 계보는 1979년, 루트 시스템 호출 및 명령줄 도구가 유닉스 버전 7의 일부로 제공되기 시작한 때가 시작이다.
Chroot(Change Root)를 사용하면 프로세스의 Root 디렉터리를 변경하여 파일 시스템에서 특정 파일 및 디렉토리로 줄일 수 있다.

BSD 기반 Jail의 완성
컴퓨터 네트워킹과 인터넷이 확장되면서 1980년대 말과 1990년대 초, **잠재적 공격자를 시스템의 나머지 부분으로부터 "jail(감옥)"에 가두기 위해 네트워크 서비스를 분리하는 데 점점 더 많이 사용**되었다. 여기서 ' Chroot Jail ' 또는 단순히 ' Jail ' 이라는 용어가 만들어졌다. 하지만 **chroot는 파일 시스템 사용을 제한하는 데만 사용**할 수 있고, root user로 실행되는 프로세스를 제한하는 데는 사용할 수 없고 잘못 구성하기 쉬워 "Jail"에서 chrooted 된 프로세스가 쉽게 탈출할 수 있었다. "JailBreak(탈옥)'이라는 용어가 등장한 것도 이때다.

OS 수준의 가상화를 제공하기 위해 FreeBSD 4.0은 크게 확장된 jail 개념을 도입했습니다.
각 jail에는 고유한 IP 주소, 루트 디렉터리 및 CPU, 메모리 등의 리소스 제한이 있습니다. FreeBSD 커널은 또한 서로 다른 jail에서 동일한 UID와 GID를 구별하기 시작하여 , **각 jail에는 고유한 root user(UID가 0)가 있지만 root jail의 root user만 제한 없는 기능을 사용할 수 있었다.**

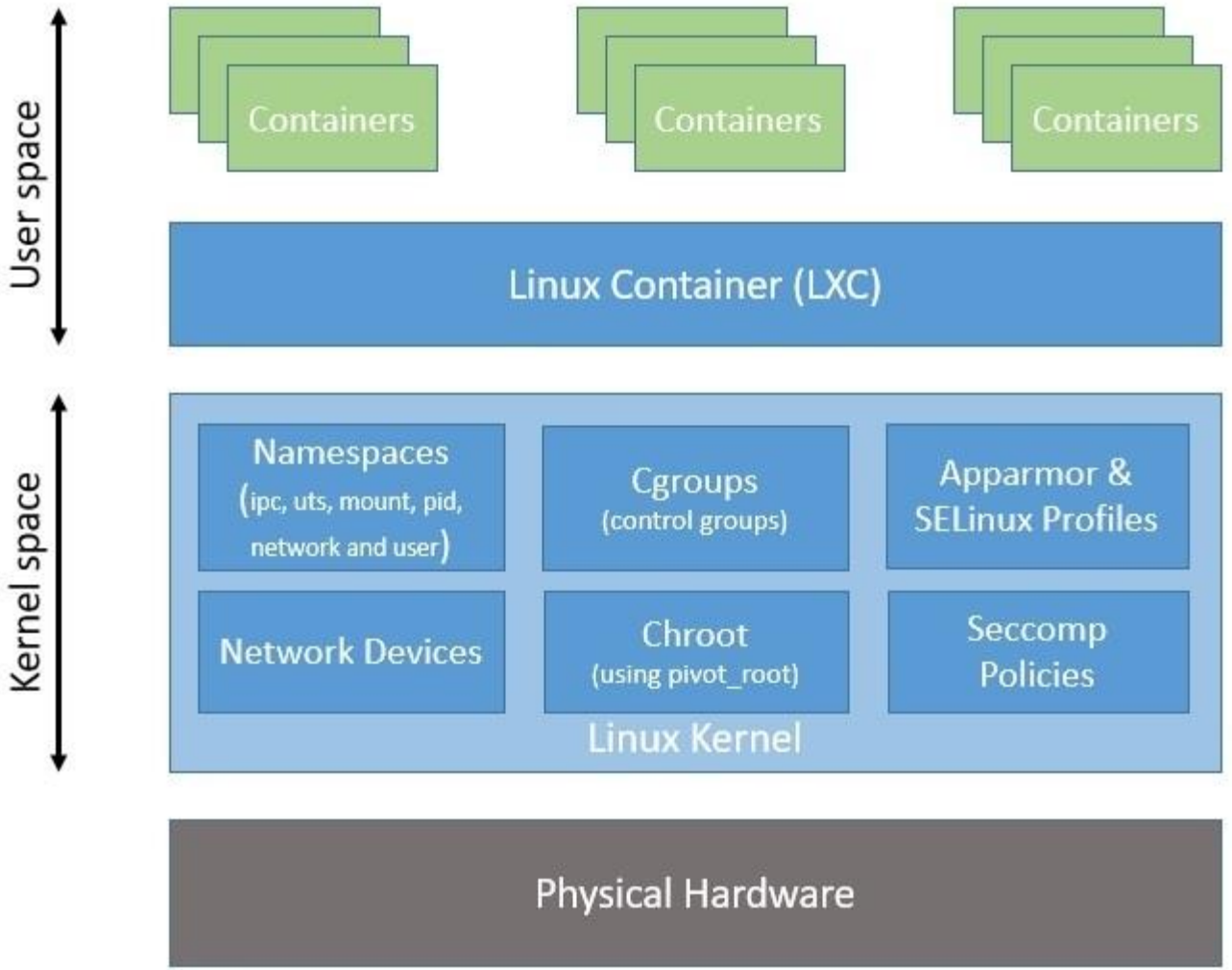


LXC의 보급

LXC는 chroot를 사용하여 각 컨테이너에 대한 파일 시스템 경계를 만들어 준다.

- 새 컨테이너가 생성되면 LXC는 호스트 시스템의 root 파일 시스템과 격리된 새 root 파일 시스템을 생성
- 이러한 격리를 통해 **각 컨테이너는 고유한 파일 시스템 계층 구조를 가질 수 있으며, 한 컨테이너 내의 프로세스가 다른 컨테이너의 파일에 액세스 하거나 수정할 수 없다.**

LXC는 가상 머신과 유사하지만 오버헤드가 훨씬 낮은 격리된 컨테이너 환경을 만들 수 있다.
LXC 컨테이너는 호스트 시스템과 동일한 커널을 공유하므로 오버헤드를 최소화하면서 빠르게 시작하고 중지할 수 있다.
따라서 LXC는 각각 고립된 환경과 리소스 제한이 있는 단일 호스트 시스템에서 여러 애플리케이션을 실행할 수 있게 된다.



조직적 문화의 변화

인프라의 구조가 바뀌면 게임 개발자도 이에 적응해야 할텐데, 문제는 없나요?

강정식 팀장 : 저희가 돌아보면 쿠버네티스나 컨테이너 기술이 중요한 게 아니고 이걸 사용하는 개발조직의 문화가 중요합니다. 개발조직의 경우 기존의 게임이 잘 돌아가고 있는 상황에서 굳이 컨테이너를 만들고 기존과 다른 방식으로 해야하는지 의문을 가질 수도 있습니다.

https://byline.network/2019/08/8-51/

읽을 거리:
https://www.samsungsds.com/kr/techreport/what-is-devops-container.html

Chapter 3.
도커의 이해

학습목표

도커의 이해

- 도커의 등장에 대해서 이해한다
- 도커의 아키텍처에 대해서 이해한다
- 도커의 한계점에 대해서 이해하고 쿠버네티스 등의 등장 배경을 이해한다

Docker의 등장

2013년 3월 산타클라라에서 열린 PyCon에서 발표

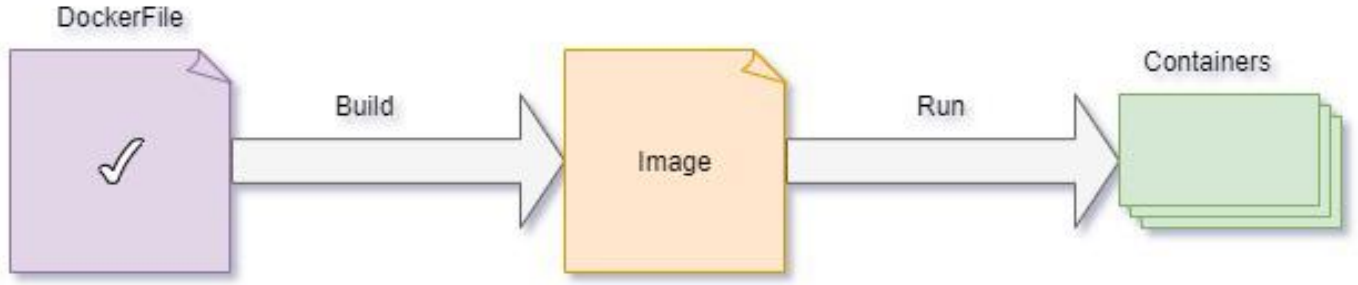
기본 규격만 맞으면 뭐든 실을 수 있는 화물
컨테이너에서 착안하여서 부두 노동자를 뜻하는
Docker라는 네이밍을 가져옴

Docker는 처음에는 LXC를 기반으로 구현되었지만
0.9버전부터는 LXC를 libcontainer로 대체하게
되는등 Docker 중심으로 오픈소스 생태계가 변경

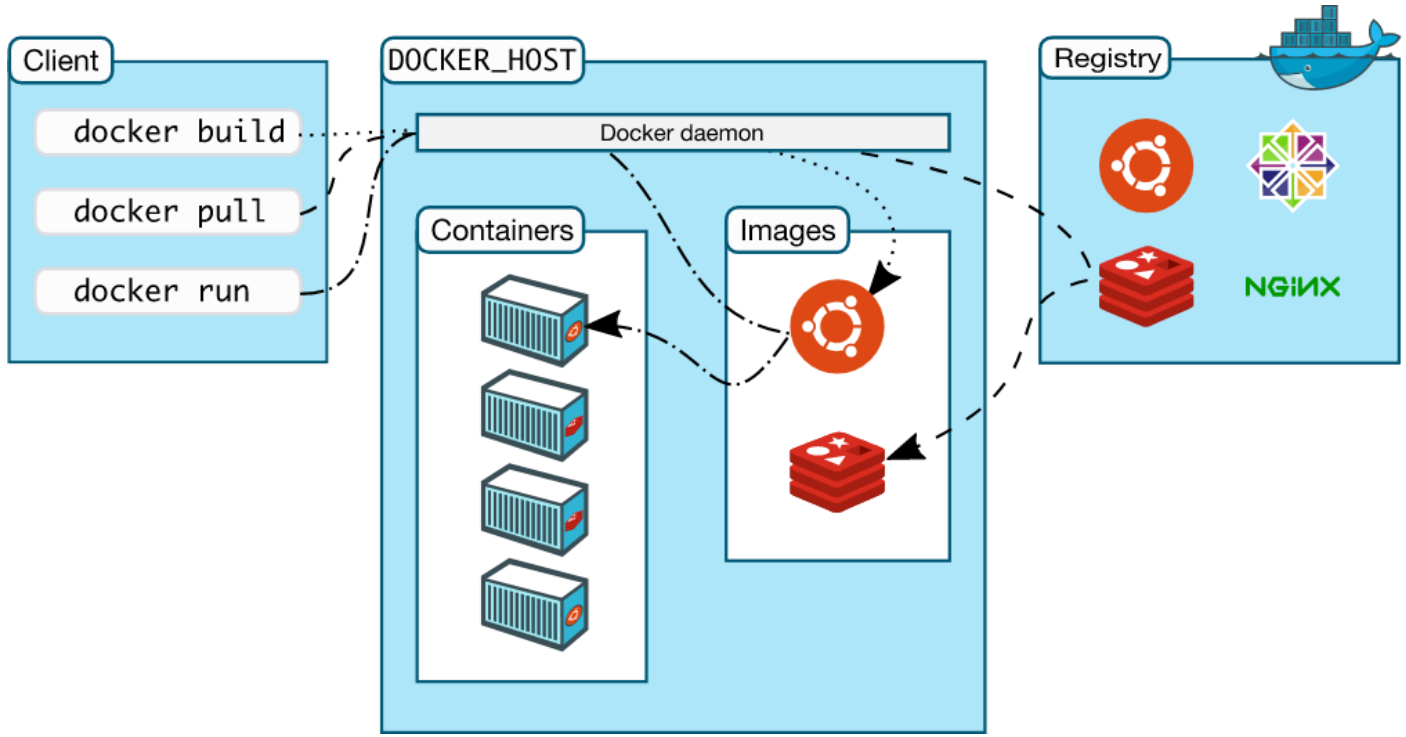
https://www.youtube.com/watch?v=wW9CAH9nSLs

Docker Image

이미지는 컨테이너를 실행하기 위한 일종의 Template임

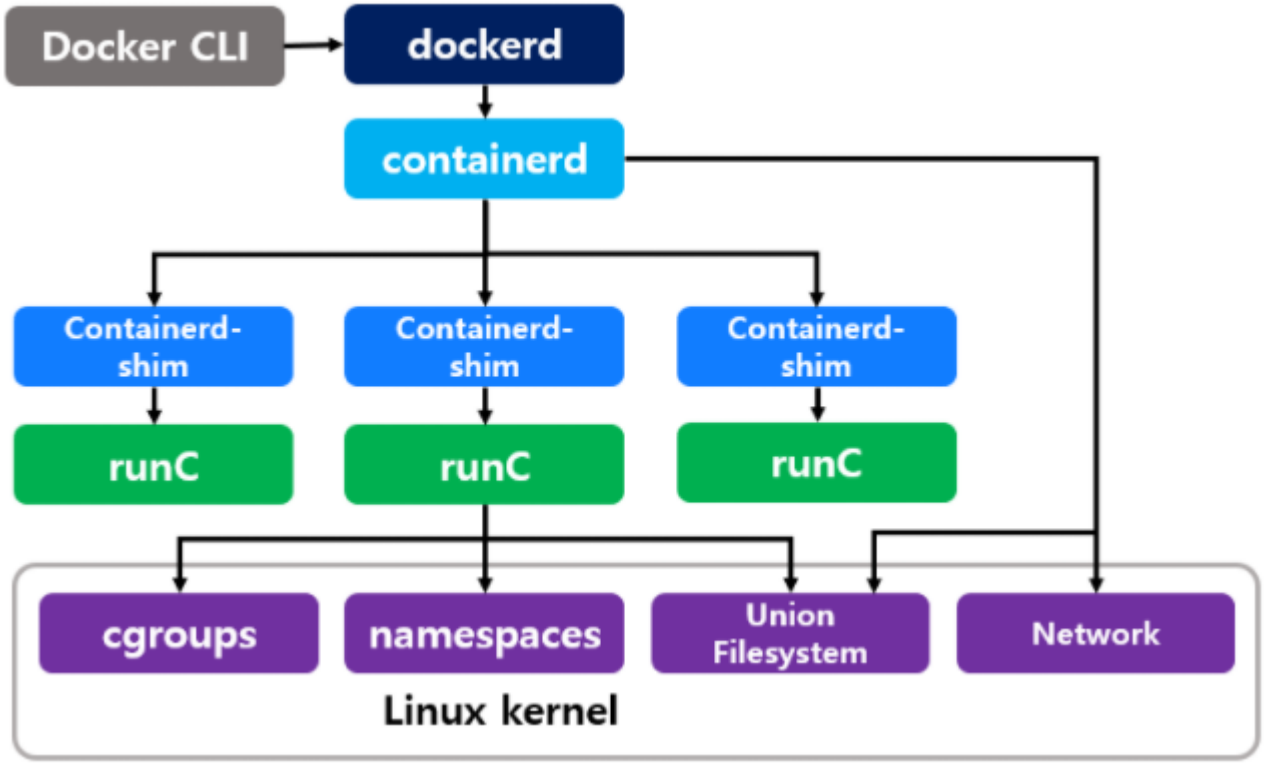


Docker 환경의 구조



Docker 기반 컨테이너 아키텍처

- 현재는 runC를 사용
- LXC, libcontainer, runC 등은 cgroups와 namespace를 표준으로 정의한 OCI(Open Container Initative) 런타임 스펙을 구현한 컨테이너 기술의 구현체임



dockerd
dockerd는 container build, security, volume, networking, secrets 등과 같은 docker의 기능을 RESTful API로 제공하여 docker-cli와 클라이언트 사용을 가능하게 함. docker engine이라고도 부름

containerd
containerd는 network, volume과 같은 것들은 dockerd에게 맡기고 오로지 컨테이너에 대한 기능들(컨테이너 생성, 이미지 pull, 컨테이너 라이프 사이클 관리 등)만을 수행하는 데몬 프로세스임

containerd-shim
containerd가 컨테이너 생성에 관여하기는 하지만 실제로 컨테이너를 생성하는 것은 containerd-shim과 runC로, containerd가 containerd-shim을 통해 runC를 호출하여 컨테이너를 생성하는데, runC는 컨테이너를 생성까지 담당하고 컨테이너의 stdin/out/err와 같은 관리는 containerd-shim이 담당함

runC
runC는 OCI 런타임 스펙을 구현하고 있는 저수준 컨테이너 런타임으로, namespace와 cgroups를 통해 실제로 컨테이너를 생성하는 역할을 수행함
cgroups는 cpu, memory, network bandwidth 등의 자원 할당에 대한 기능을, namespace는 시스템의 리소스들(user, 파일, 네트워크, 프로세스 등)을 가상화(격리)하는 기능을 제공함

도커의 주변 구성 요소

- Docker Hub
  - 전 세계 도커 사용자들과 함께 도커 컨테이너 이미지를 공유하는 클라우드 서비스
- Docker-compose
  - 의존성 있는 독립된 컨테이너에 대한 구성 정보를 YAML 코드로 작성하여 일원환된 애플리케이션 관리를 가능하게 하는 도구
- Docker Kitematic
  - 컨테이너를 이용한 작업을 수행할 수 있는 GUI 제공
- Docker Registry
  - 도커 이미지 저장소
- Docker Machine
  - 가상머신 프로그램 및 AWS EC2, MS Azure 환경에 도커 실행 환경을 생성하기 위한 도구
- Docker Swarm
  - 여러 도커 호스트를 클러스터로 구축하여 관리할 수 있는 도커 오케스트레이션 도구

오케스트레이션 도구의 필요성

1.배포 관리 : 어떤 컨테이너를 어느 호스트에 배치하여 구동시킬 것인가? 각 호스트가 가진 한정된 리소스에 맞춰 어떻게 최적의 스케줄링을 구현할 것인가? 어떻게 하면 이러한 배포 상태를 최소한의 노력으로 유지 관리할 수 있을 것인가?

2.제어 및 모니터링 : 구동 중인 각 컨테이너들의 상태를 어떻게 추적하고 관리할 것인가?

3.스케일링 : 수시로 변화하는 운영 상황과 사용량 규모에 어떻게 대응할 것인가?

4.네트워킹 : 이렇게 운영되는 인스턴스 및 컨테이너들을 어떻게 상호 연결할 것인가?

생각해볼 문제

1. 데이터 관련 기술과 컨테이너의 효율성

2. 인프라 자동화 측면에서의 고려 사항

3. 보안/인증 측면에서의 고려 사항

THANK YOU!