

클라우드 환경의 이해

개발자가 되기 전에 알았으면 좋았을 것들



강사 소개



구름의 SRE 스쿼드 리더 **김성주 (Rex)**입니다.

구름에서 풀스택 개발 인턴으로 시작하여, 현재는 SRE 조직의 리더로서 구름 전체 서비스 운영을 맡고 있습니다.

안정적인 서비스 운영을 위한 기술 도입을 위한 로드맵 설정과 R&D 업무를 주로 담당하고 있습니다.

강사 소개

- 학력
 - 한국디지털미디어고등학교 해킹방어과 졸업
 - 경희대학교 컴퓨터공학과 졸업
- 경력
 - (현) 구름 SRE 스쿼드 리더
 - (전) 구름 인프라 엔지니어
 - (전) 구름 풀스택 개발자
- 기타
 - 소프트웨어 마에스트로 인증

개요

- 왜 클라우드인가?
- CPU 아키텍처와 운영체제의 이해
 - CPU 아키텍처에 대해 꼭 알아야 할까요?
 - 개발자에게 운영체제란?
 - 애플리케이션이 실행되는 가장 기본적인 환경에 대해
- 컨테이너의 이해 / 컨테이너 기반의 서비스 운영
 - 꼭 컨테이너를 사용해야 할까요?
 - 컨테이너 구성의 기본
 - 컨테이너 기반의 서비스 운영에 대하여

개요

- 클라우드 환경에서의 DevOps, CI/CD

- 클라우드에서는 어떻게 다를까요?
 - 클라우드 컴퓨팅과 클라우드 네이티브
 - DevOps? SRE? Platform Engineering?

- Kubernetes 그리고 MSA로의 길

- MSA를 위한 Kubernetes
 - 아키텍처에 정답은 없다

- 새로운 기술을 맞이하는 우리의 자세

왜 클라우드인가?

- **클라우드 컴퓨팅**

- AWS에서 정의하는 클라우드 컴퓨팅

클라우드 컴퓨팅은 IT 리소스를 인터넷을 통해 온디맨드로 제공하고 사용한 만큼만 비용을 지불하는 것을 말합니다.

물리적 데이터 센터와 서버를 구입, 소유 및 유지 관리하는 대신, Amazon Web Services(AWS)와 같은 클라우드 공급자로부터 필요에 따라 컴퓨팅 파워, 스토리지, 데이터베이스와 같은 기술 서비스에 액세스할 수 있습니다.

- **클라우드 네이티브**

- CNCF에서 정의하는 클라우드 네이티브

클라우드 네이티브 기술은 조직이 퍼블릭, 프라이빗, 그리고 하이브리드 클라우드와 같은 현대적이고 동적인 환경에서 확장 가능한 애플리케이션을 개발하고 실행할 수 있게 해준다.

컨테이너, 서비스 메쉬, 마이크로서비스, 불변(Immutable) 인프라, 그리고 선언형(Declarative) API가 이러한 접근 방식의 예시들이다. 이 기술은 회복성, 관리 편의성, 가시성을 갖춘 느슨하게 결합된 시스템을 가능하게 한다. 견고한 자동화 기능을 함께 사용하면, 엔지니어는 영향이 큰 변경을 최소한의 노력으로 자주, 예측 가능하게 수행할 수 있다.

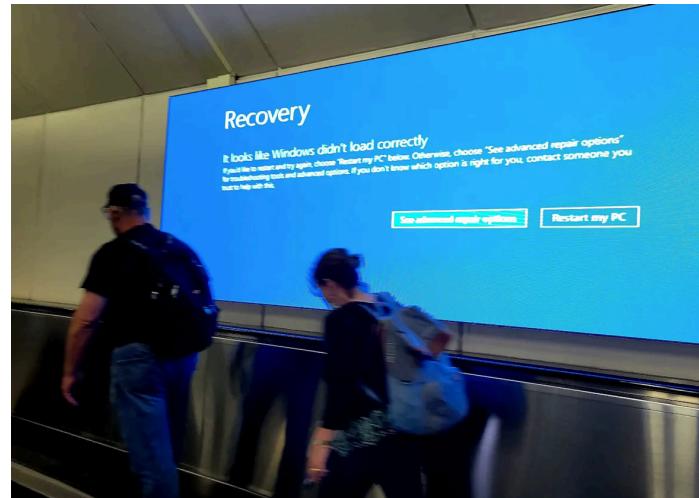
왜 클라우드인가?

- **클라우드를 사용하는 이유**

- 유연한 확장
 - 리소스 사용량, 네트워크 트래픽을 기준으로 사용량에 알맞게 확장 및 축소를 할 수 있습니다.
- 비용 절감
 - 하드웨어를 직접 구매하여 인프라를 구성하는 온프레미스 환경과 달리, 사용한만큼 비용을 지불하기 때문에 비용 절감효과가 굉장히 크다.
- 효율적인 관리
 - 인프라의 유지 보수를 직접하는 것이 아니라 클라우드 벤더사에서 담당하기 때문에 제품 개발에 집중할 수 있습니다.

왜 클라우드인가?

- 위험성
 - 데이터 보안
 - 데이터가 클라우드 환경에서 저장되기 때문에, 온프레미스 환경에 비해 훨씬 데이터 유출, 해킹의 위험에 노출되기 쉽습니다.
 - 서비스 가용성
 - 서비스 가용성을 오롯이 클라우드 벤더사에 의존하게 되기 때문에 클라우드 벤더사에 문제가 생기면, 그대로 서비스 문제로 연결된다.
 - 비용 관리의 어려움
 - 확장이 손쉽게 이뤄지기 때문에, 적절한 운영은 비용 절감으로 이어지지만 이를 효과적으로 관리하지 못하면 오히려 비용을 과하게 지불하게 되는 경우가 발생할 수 있다.



클라우드스트라이크발 Azure 전산 마비 사태

왜 클라우드인가?

- 온프레미스 환경은 어떨 때 사용할까?
 - 보안에 민감한 서비스
 - 금융권, 의료, 정부 사업과 같이 보안에 굉장히 민간한 사업의 경우에는 인프라, 데이터가 물리적으로 분리되어 있어야 할 필요가 있는 경우가 있다.
 - 낮은 지연 시간이 필요한 서비스
 - 게임 산업, 금융 서비스, 군사 산업 등 지연 시간을 최소로 하는 것이 중요한 산업들에서 온프레미스 환경을 사용하는 경우가 있다.
- 단점들을 보완하기 위해, 하이브리드 클라우드, 멀티 클라우드 등 다양한 형태의 클라우드 사용 사례들이 나타나고 있다.

CPU 아키텍처에 대해 꼭 알아야 할까요?

- CPU 아키텍처란?
 - 마이크로아키텍쳐(microarchitecture)로도 부르며, 하드웨어(CPU)가 작동하는 방식을 나타낸다.
 - 상용화된 제품을 기준으로는 ARM(RISC)과 x86(CISC)이 대표적인 예이다.



CPU 아키텍처에 대해 꼭 알아야 할까요?



신입 개발자로 일하게 된 김구름.

M3 칩을 탑재한 최신형 MacBook을 지급받아
기쁜 마음으로 개발을 시작합니다.

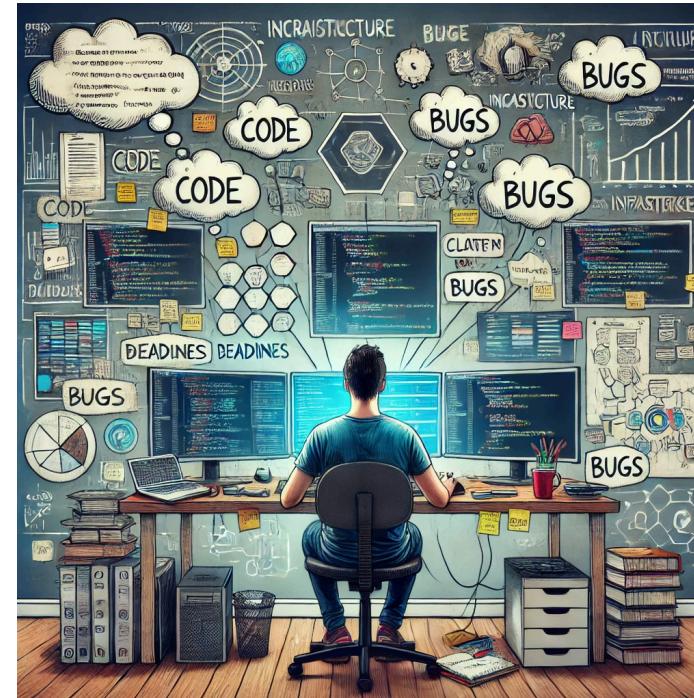
그런데, 혼자서 테스트할 때는 문제가 없던 코드가 DEV 환경에서는
실행이 되지 않게 됩니다!

CPU 아키텍처에 대해 꼭 알아야 할까요?



CPU 아키텍처에 대해 꼭 알아야 할까요?

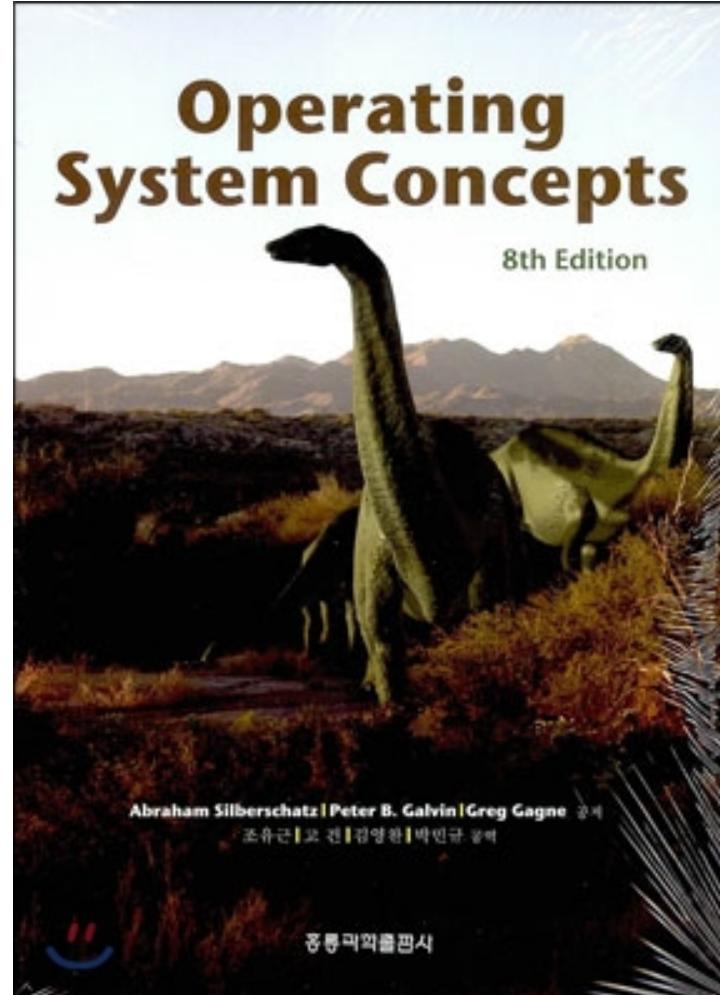
- 서비스, 애플리케이션이 운영되는 환경은 하드웨어에서부터 다양하다.
 - arm, x86과 같은 CPU부터 GPU, TPU, NPU에 이르기까지 고려해야 할 환경이 많다.
 - 사용 가능한 리소스의 제한에 따라 개발 방향이 변경될 수 있다.
 - CPU, 메모리, Storage 성능, 네트워크 성능 등 다양한 변수가 존재



개발자에게 운영체제란?

- 운영체제라 함은?
 - 하드웨어와 소프트웨어 간의 인터페이스를 제공
 - 하드웨어 리소스 관리
 - 파일 시스템 제공
 - 네트워크 통신
 - 시스템 콜
 - 뮤텍스, 세마포어
 - 페이징, 세그멘테이션
 - 페이징 교체 알고리즘
 - 보안
 - ...
 - ...

개발자에게 운영체제란?



개발자에게 운영체제란?

- 어디까지 알아야 할까요?
 - 물론 많이 알수록 좋습니다.
 - 리눅스와 컨테이너 기술의 배경
 - 개발자이던 시절 알았다면 좋았을 것들
 - 프로세스 관리
 - 메모리 관리
 - 파일 시스템
 - 스케줄링
 - 네트워크
 - ...

리눅스와 컨테이너 기술의 배경

- 리눅스의 멀티 유저 특성
 - 리눅스는 본질적으로 여러 사용자가 하나의 시스템에서 동시에 작업할 수 있는 환경을 지원한다.
 - 각 사용자는 자신의 환경과 파일, 프로세스 등을 독립적으로 관리할 수 있다.
- 컨테이너 기술
 - 컨테이너 기술은 이러한 리눅스의 멀티 유저를 지원하는 특성을 확장하여, 프로세스, 리소스 등을 격리된 환경에서 사용할 수 있도록 제공한다.
 - 이는 나아가, 클라우드 컴퓨팅과 마이크로서비스 아키텍처를 구성하는데 있어 중요한 역할을 하게 된다.

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- Too many open files / 최대 생성 프로세스 제한
 - 보안상의 이유로 Linux 상에서 생성할 수 있는 프로세스, 프로세스가 열 수 있는 파일의 수 등을 제한하고 있다.
 - **때문에 개발 과정에서 한번쯤은 마주하게 되는 오류 중 하나**
 - 실제로 프로세스 생성이나 파일 접근을 동시에 다량으로 하거나, Socket을 다루는 과정에서 발생할 수 있다. (Socket이 파일로 관리되기 때문)

```
container: OCI runtime exec failed: exec failed: unable to start container process: apparmor failed to apply profile:  
open /proc/self/attr/apparmor/exec: too many open files in system: unknown
```

```
[PROBLEM] Average - Nginx Apps Unhealthy - during 1m (timeout 4s)  
NGINX unhealthy apps: sudo: unable to open /etc/sudoers: Too many open files in system
```

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- `/etc/security/limits.conf` 혹은 `ulimit` 명령어를 통해 제한 값을 조정할 수 있다.

```
/* soft core 0
#root hard core 100000
/* hard rss 10000
#@student hard nproc 20
#@faculty soft nproc 20
#@faculty hard nproc 50
#ftp hard nproc 0
#ftp - chroot /ftp
#@student - maxlogins 4
* hard nofile 1000000
* soft nofile 1000000
root hard nofile 1000000
root soft nofile 1000000

# End of file
```

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- `/etc/security/limits.conf` 혹은 `ulimit` 명령어를 통해 제한 값을 조정할 수 있다.

```
root@rex-chime-test:/home/ubuntu# ulimit -a
core file size      (blocks, -c) 0
data seg size       (kbytes, -d) unlimited
scheduling priority (-e) 0
file size           (blocks, -f) unlimited
pending signals     (-i) 30689
max locked memory   (kbytes, -l) 64
max memory size     (kbytes, -m) unlimited
open files          (-n) 1000000
pipe size           (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority  (-r) 0
stack size          (kbytes, -s) 8192
cpu time            (seconds, -t) unlimited
max user processes  (-u) 30689
virtual memory       (kbytes, -v) unlimited
file locks          (-x) unlimited
```

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- 부모 프로세스 - 자식 프로세스의 관리
 - 필요에 따라서는 자식 프로세스를 생성할 때, 부모 프로세스에 종속되지 않게 독립된 프로세스로 생성할 수 있다.
 - 고아 프로세스/좀비 프로세스 문제
 - 시스템 자원이 낭비되고 있는 상태
 - 프로세스 관리의 어려움
 - PID(Process ID)의 낭비
 - 시스템에서 사용할 수 있는 PID 수가 제한되어 있으므로, 프로세스 생성이 불가능해질 수 있다.

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- OOM (Out Of Memory)
 - 크게 두가지로 분류할 수 있습니다.
 - 메모리 누수(Memory Leak)
 - 계획된 메모리를 초과하는 요청이 오는 경우
 - (하드웨어적 제한) 32bit CPU의 경우 최대 4GB, 물리적인 메모리의 한계에 달하는 경우 등
 - (소프트웨어적 제한) JVM이나 V8 Engine 등에서 제한하는 경우 등

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- 메모리 누수(Memory Leak) 해결
 - APM(Application Performance Monitoring) 혹은 프로파일러를 활용하여 메모리 누수 지점을 찾고 분석
 - GC(Garbage Collector) 최적화
 - 미사용 자원 해제



Aw, Snap!

Something went wrong while displaying this webpage.

Error code: Out of Memory

[Learn more](#)

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- 계획된 메모리를 초과하는 요청이 오는 경우
 - 정답은 없다 => 설득의 영역
 - 상황에 가장 알맞은 사양을 결정해야한다. 어떨 때는 스펙 변경으로, 또 다를 상황에서 코드의 최적화 등으로 해결해야할 수 있다.
 - 트레이드 오프를 고려하여 적절한 선택을 할 수 있어야 한다.



개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- [실습 1] ulimit 수정 및 테스트

```
# open files 변경  
# ulimit -n [변경할 수]  
  
# max user processes 변경  
# ulimit -u [변경할 수]
```

개발자이던 시절 알았다면 좋았을 것들 - 운영체제편

- [실습 2] OOM 테스트 및 소프트웨어적 제한 조정

```
node --max-old-space-size=[heap 메모리 사이즈(MB)] index.js
```