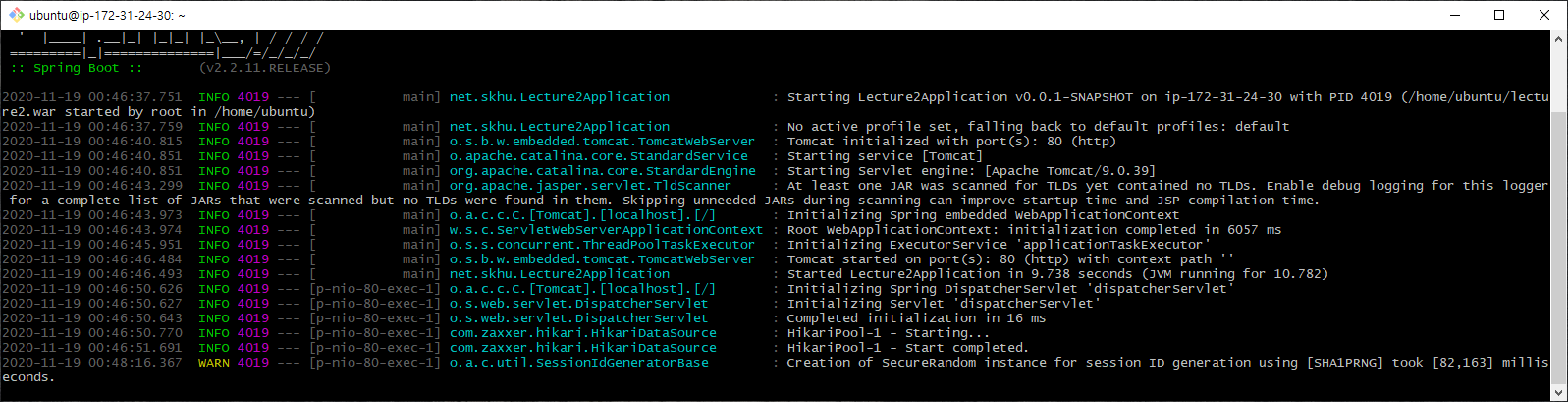
# **tomcat 버그**

## **증상**

ubuntu server 에서 spring boot 앱을 실행한 후, 웹브라우저에 URL을 입력하고 처음 접속하려면, 대기 상태로 빙빙 돌다가 3~6분 후에 접속이 가능해진다.



위 화면의 로그 메시지에서 WARN 로그 메시지 출력이 보인다.

|  |
| --- |
| 2020-11-19 00:48:16.367 WARN 4019 --- [p-nio-80-exec-1] o.a.c.util.SessionIdGeneratorBase :  Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [182,163] milliseconds. |

session ID를 생성하기 위한 SecureRandom 객체 생성이 182,163 밀리초가 걸렸다는 로그 메시지이다.

## **원인**

웹서버는 웹브라우저를 식별하기 위해 session ID를 사용한다.

이 session ID를 생성할 때, tomcat은 SecureRandom 객체를 생성하고, 이 객체는 유닉스에서 /dev/random 파일을 사용한다.

그런데 이 파일에 들어있는 데이터가 부족할 경우에, 몇 분 정도 정지(block) 상태에 빠지게 된다.

윈도우에서는 이런 문제가 없다. 랜덤 값을 생성하는 방식이 다르기 때문이다.

## **해결**

/dev/random 파일 대신, /dev/urandom 파일을 사용하라는 옵션을 java 가상 머신에 주면 된다.

|  |
| --- |
| java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war |

## **spring boot 앱 실행 방법 정리**

spring boot 앱 실행 방법은 다음과 같다.

|  |
| --- |
| java -jar lecture2.war |

그런데 앞의 버그 때문에 ubuntu 서버에서는 다음 명령으로 실행해야 한다.

|  |
| --- |
| java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war |

그런데 lecture2.war는 80 포트를 사용한다.

80 포트를 사용하려면 관리자 권한으로 실행해야한다.

|  |
| --- |
| sudo java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war |

터미널 창을 닫아도 실행이 계속되게 하려면

|  |
| --- |
| nohup sudo java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war |

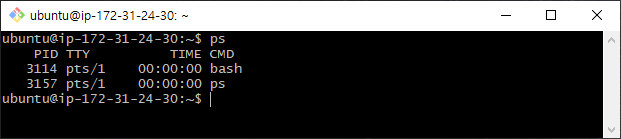
# **프로세스 관리**

## **ps 명령**

### **프로세스 목록**

|  |
| --- |
| ps |

아무 옵션 없이 ps 명령을 실행하면, 현재 터미널 창에서 시작된 프로세스들 정보가 출력된다.



bash 쉘 프로세스 정보와, ps 명령 프로세스 정보가 출력되었다.

위 화면의 출력

|  |  |
| --- | --- |
| PID | 프로세스 ID |
| TTY | 명령이 실행된 터미널창 ID |
| TIME | CPU가 프로세스를 실행한 누적 시간 |
| CMD | 실행 명령 |

bash 쉘을 실행한지 매우 오래되었지만 CPU 실행 누적 시간이 0초도 되지 않는다.

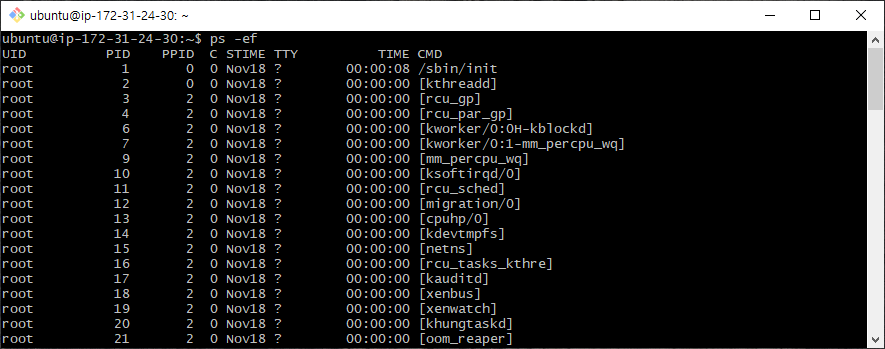
bash 쉘 프로세스 실행 시간의 99.99%가 입출력 대기 시간이라서 CPU가 할 일이 없었기 때문이다.

ps 명령을 실행 한 후에도 bash 프로세스는 여전히 실행중 이지만,

ps 명령이 종료되었으므로 ps 프로세스도 종료되었다.

### **모든 프로세스 목록**

|  |
| --- |
| ps -ef |



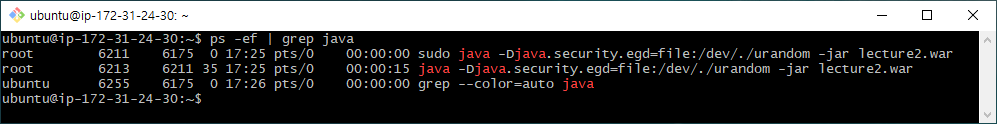
위 화면의 출력 설명

|  |  |
| --- | --- |
| UID | 명령을 실행한 사용자 계정 |
| PID | 프로세스 ID |
| PPID | 부모 프로세스 ID(q부모 프로세스에서 자식 프로세스를 생성한다) |
| C | 프로세스 전체 실행 시간에서, CPU 실행 시간의 % |
| STIME | 프로세스 시작 시각 |
| TTY | 명령이 실행된 터미널 창 ID |
| TIME | CPU가 프로세스를 실행한 누적 시간 |
| CMD | 실행 명령 |

### **특정 프로세스 정보 보기**

|  |
| --- |
| ps -ef | grep java |

ps -ef 명령의 출력 중에서 java 단어가 들어간 줄만 출력한다.



sudo java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war 명령의 프로세스 ID는 6211이다.

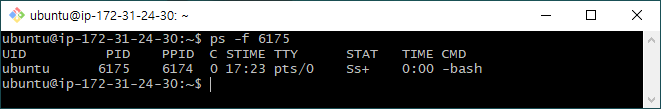
이 프로세스의 부모 프로세스ID는 6175이다.

6175 프로세스는 당연히 bash 일 것이다.

bash에서 이 명령을 실행 했으므로, bash 프로세스가 이 명령 프로세스의 부모 프로세스가 된다.

### **프로세스ID로 조회**

|  |
| --- |
| ps -f 프로세스ID |

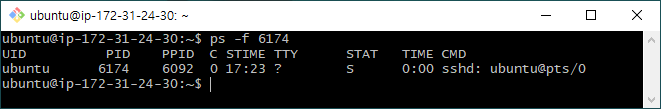


6175 프로세스는 역시 bash 이다.

이 프로세스의 부모 프로세스ID는 6174 이다.

내 PC에서 ssh로 서버에 연결한 다음에 서버의 6175 bash 프로세스가 시작되었다.

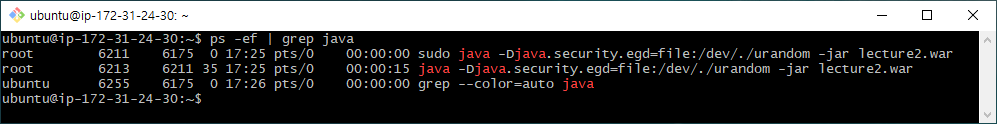
따라서 6175 bash 프로세스의 부모는 서버에서 실행중인 ssh 서버이다.



내 PC에서 실행된 ssh 클라이언트 이름은 ssh 이고,

EC2 서버에서 실행된 ssh 서버의 이름은 sshd 이다. (ssh daemon)

이 두 프로그램이 내 PC와 EC2 서버 사이에서 통신을 하고 있는 중이다.



sudo java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war 명령을 실행하는 프로그램은 sudo 이다.

sudo는 관리자 권한을 획득하고, 관리자 권한으로

java -Djava.security.egd=file:/dev/./urandom -jar lecture2.war 명령을 실행한다.

java -Djava.security.egd=file:/dev/./urandom -jar lecutre2.war 명령의 프로세스ID는 6213 이다.

이 프로세스의 부모 프로세스ID는 6211이다.

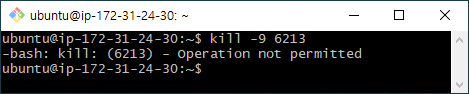
sudo가 이 명령을 실행했으므로, sudo 프로세스가 이 명령 프로세스의 부모 프로세스가 된다.

## **kill 명령**

kill 명령은 프로세스를 강제로 죽이는 명령이다.

nohup 으로 실행한 프로세스를 종료하려면 kill 명령으로 강제로 죽여야 한다.

|  |
| --- |
| kill -9 프로세스ID |

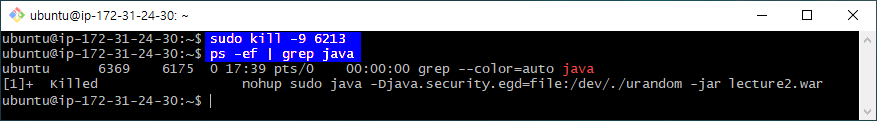


6213 프로세스를 죽이려고 했지만 권한이 없다는 에러가 발생했다.

보통 내가 만든 프로세스는 내 권한으로 죽일 수 있다.

하지만 6213 프로세스는 내가 만들기 했지만, 관리자 권한(sudo)으로 만들었기 때문에,

죽일 때도 관리자 권한이 필요하다.



관리자 권한으로 6213 프로세스를 죽이고, 다시 프로세스를 조회해서 종료를 확인했다.

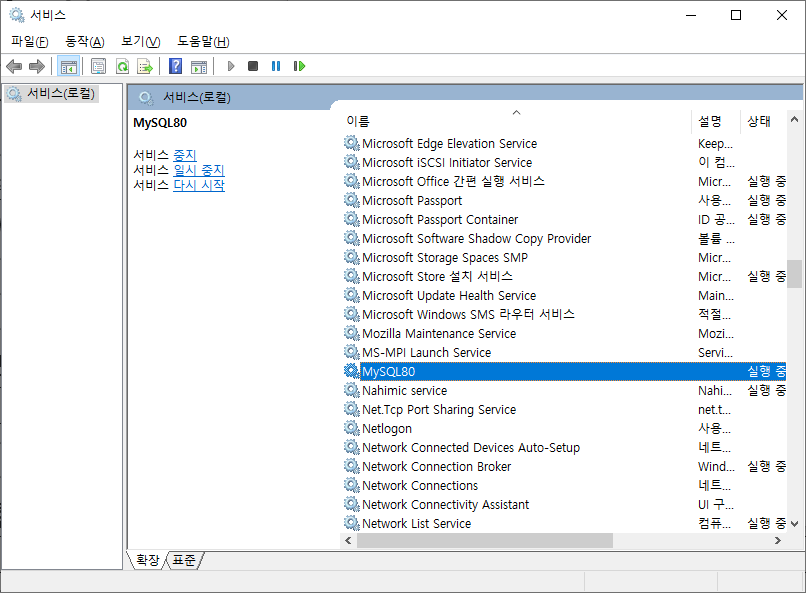
# **데몬(daemon)**

## **데몬이란**

운영체제가 부팅될 때 자동으로 같이 시작되어 백그라운드에서 계속 시행되는 서비스를 데몬(daemon)이라고 한다.

데몬 프로그램의 이름은 ‘d’ 문자로 끝내는 것이 관례이다. (예: sshd)

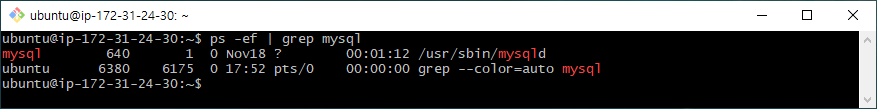
windows에서는 그냥 윈도우 서비스라고 부른다.



우리가 지난 시간에 설치한 mysql도 데몬이라서, 재부팅될 때 자동으로 시작되어 백그라운드에서 계속 실행되고있다.

백그라운드에서 실행중인 mysql 엔진의 이름도 mysqld 이다.

|  |
| --- |
| ps -ef | grep mysql |



## **데몬 관리 명령**

### **주요 데몬 목록**

|  |
| --- |
| service --status-all |

위 명령은 전체 데몬을 다 보여주지는 않는다.

## **현재 실행중인 데몬 전체 목록**

|  |
| --- |
| systemctl list-units --type=service --state=running |

현재 실행중인 데몬 전체를 보여주는 명령은 위와 같다.

### **데몬 전체 목록**

|  |
| --- |
| systemctl list-units --type=service |

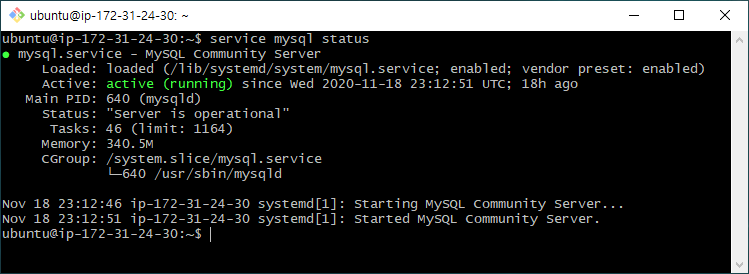
현재 상태와 무관하게 데몬 전체를 보여주는 명령은 위와 같다.

### **특정 데몬 상태 보기**

|  |
| --- |
| service 데몬이름 status |

명령 (실행 파일) 이름과 데몬의 이름은 다르게 지정할 수 있다. (같게 지정할 수도 있다)

mysql daemon 실행 파일의 이름은 mysqld, 데몬의 이름은 mysql 이다.



### **데몬 실행 시작**

|  |
| --- |
| sudo service 데몬이름 start |

### **데몬 종료**

|  |
| --- |
| sudo service 데몬이름 stop |

### **데몬 실행 재시작**

|  |
| --- |
| sudo service 데몬이름 restart |

### **데몬 자동 시작**

|  |
| --- |
| sudo systemctl enable 데몬이름 |

컴퓨터가 리부팅될 때 데몬이 자동 시작되도록 설정한다.

### **데몬 자동 시작 취소**

|  |
| --- |
| sudo systemctl disable 데몬이름 |

컴퓨터가 리부팅될 때 데몬이 자동 시작 등록이 취소된다.

### **데몬 설정 파일 다시 읽기**

|  |
| --- |
| sudo systemctl daemon-reload |

데몬 설정 파일을 수정한 다음에 위 명령을 실행해야 한다.

# **데몬 만들기**

## **데몬의 장점**

위에서 nohup 명령으로 spring boot 앱을 실행했는데 불편한 점이 있다.

* 컴퓨터가 재부팅될 때 마다 다시 실행해야 한다.
* 종료시킬 때, 프로세스를 찾아서 죽여야 한다.

spring boot 앱을 데몬으로 만들면 편하다.

* 컴퓨터가 재부팅될 때 자동 시작될 수 있다.
* 실행 상태를 보기 편하다.
* 실행 시작, 실행 중지, 재시작 하기 편하다.

## **war 파일 옮기기**

데몬으로 실행할 war 파일을 적당한 디렉토리에 옮긴다.

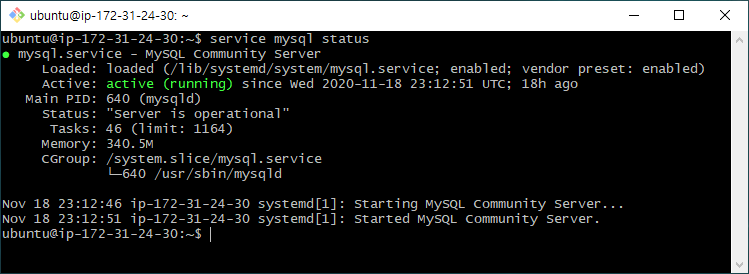
### **~/daemon 디렉토리 생성**

|  |
| --- |
| mkdir ~/daemon |

홈 디렉토리 아래에 daemon 디렉토리를 생성한다.

### **~/daemon 디렉토리로 이동**

|  |
| --- |
| mv ~/lecture2.war ~/daemon/ |



## **데몬 설정 파일 생성**

데몬 파일의 이름은 데몬이름.service 이어야 하고, /etc/systemd/system 디렉토리에 위치해야 한다.

사용자 홈 디렉토리 아래가 아닌 디렉토리에서 파일을 생성, 복사, 수정하려면 관리자 권한이 필요하다(sudo).

|  |
| --- |
| sudo vi /etc/systemd/system/lecture2.service |

|  |
| --- |
| [Unit]  Description=lecture2 spring boot apps  After=syslog.target  [Service]  User=ubuntu  ExecStart=sudo java -Djava.security.egd=file:/dev/./urandom -jar /home/ubuntu/daemon/lecture2.war  SuccessExitStatus=143  [Install]  WantedBy=multi-user.target |

녹색으로 칠한 부분 간단한 설명

노란색으로 칠한 부분 실행 명령

### **데몬 설정 파일 다시 읽기**

|  |
| --- |
| sudo systemctl daemon-reload |

데몬 설정 파일을 새로 만들었으므로 위 명령을 실행해야 한다.

### **데몬 실행 시작**

|  |
| --- |
| sudo service lecture2 start |

### **데몬 상태 보기**

|  |
| --- |
| sudo service lecture2 start |

### **데몬 자동 실행 등록**

|  |
| --- |
| sudo systemctl enable lecture2 |

### **데몬 종료**

|  |
| --- |
| sudo service lecture2 stop |

### **데몬 재시작**

|  |
| --- |
| sudo service lecture2 restart |

# **데몬 만들기2**

## **개요**

mybatis1.war spring boot 앱 프로젝트로 데몬을 만들어 실행

mybatis1.war는 8080 포트를 사용한다

서버에 여러 spring boot 앱 프로젝트를 실행하려면, 사용하는 포트 번호가 중복되지 않아야 한다.

## **8088 포트 열기**

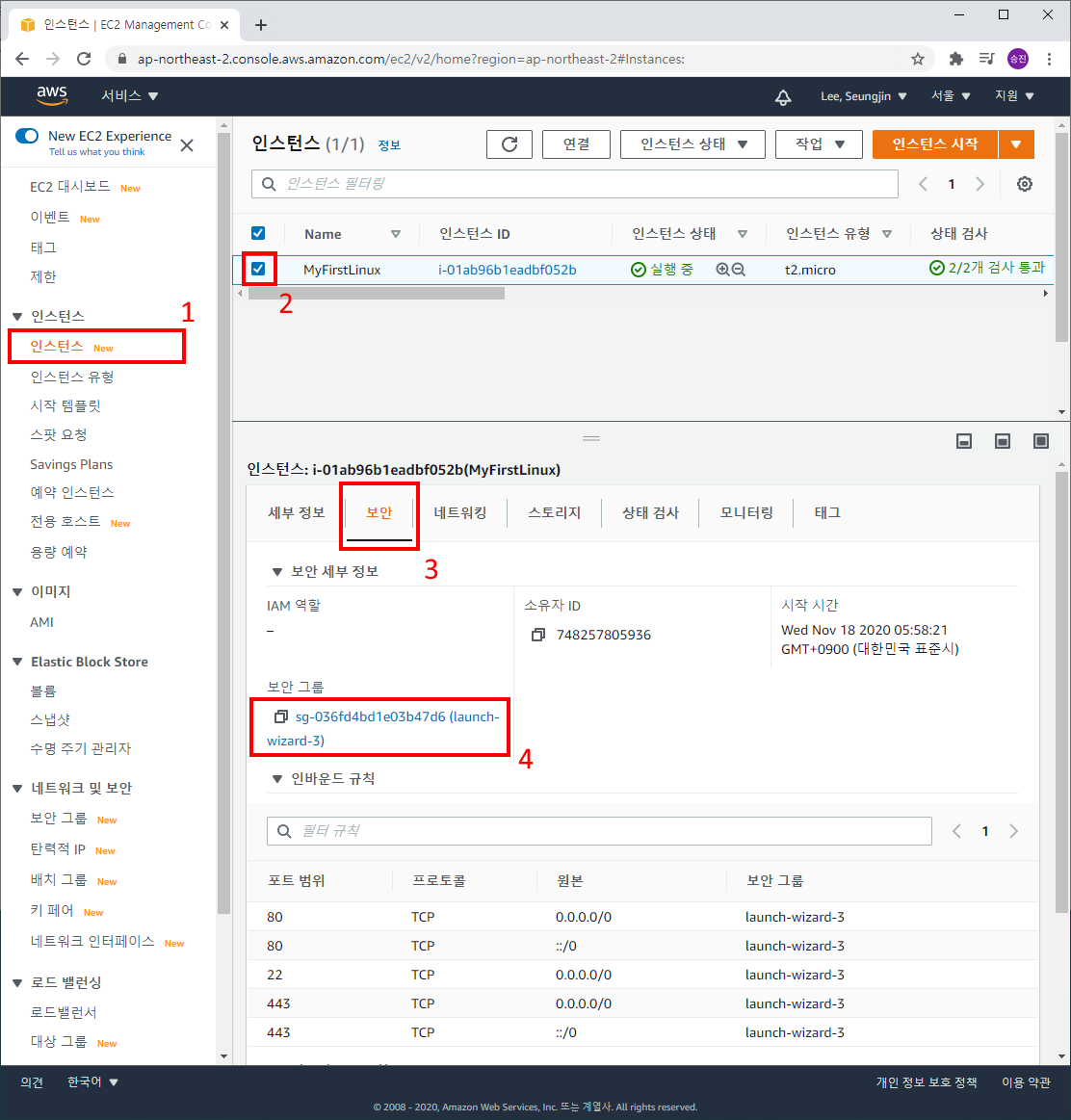
앞에서 EC2 인스턴스를 생성할 때 아래 포트만 열었다.

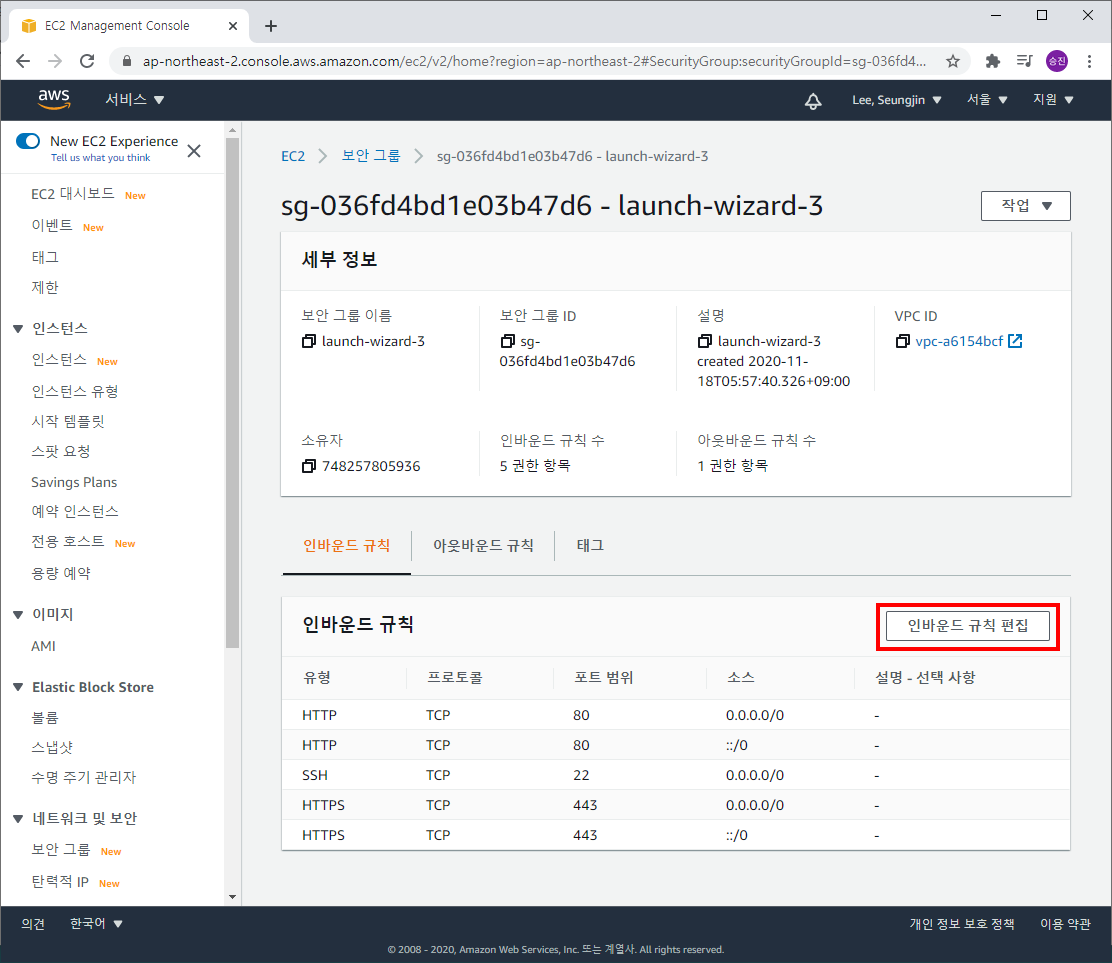
-22포트 SSH

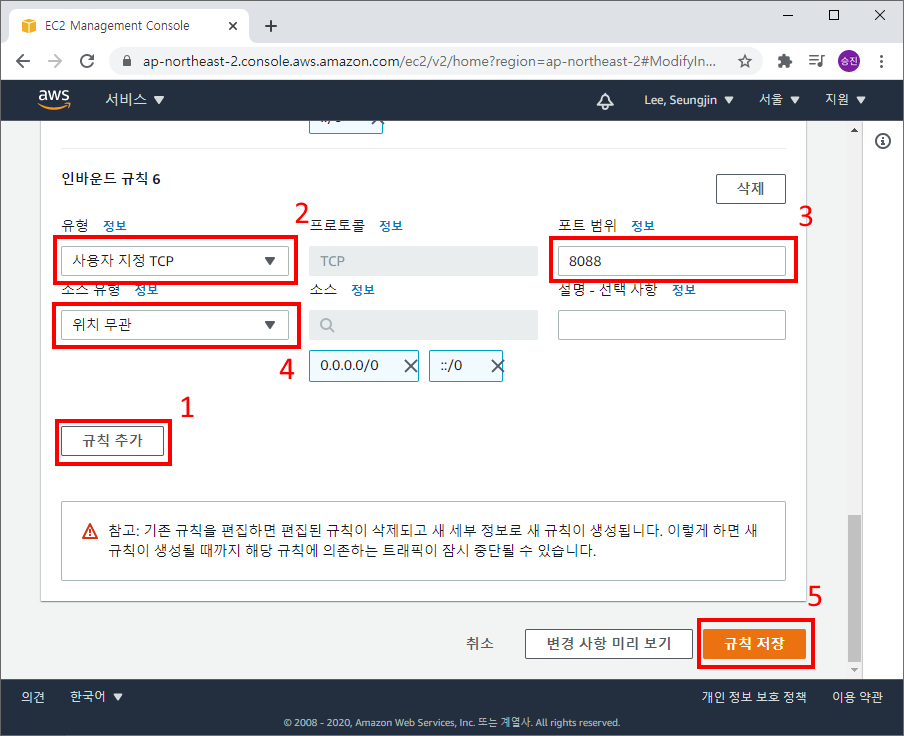
-80포트 HTTP

-443포트 HTTPS

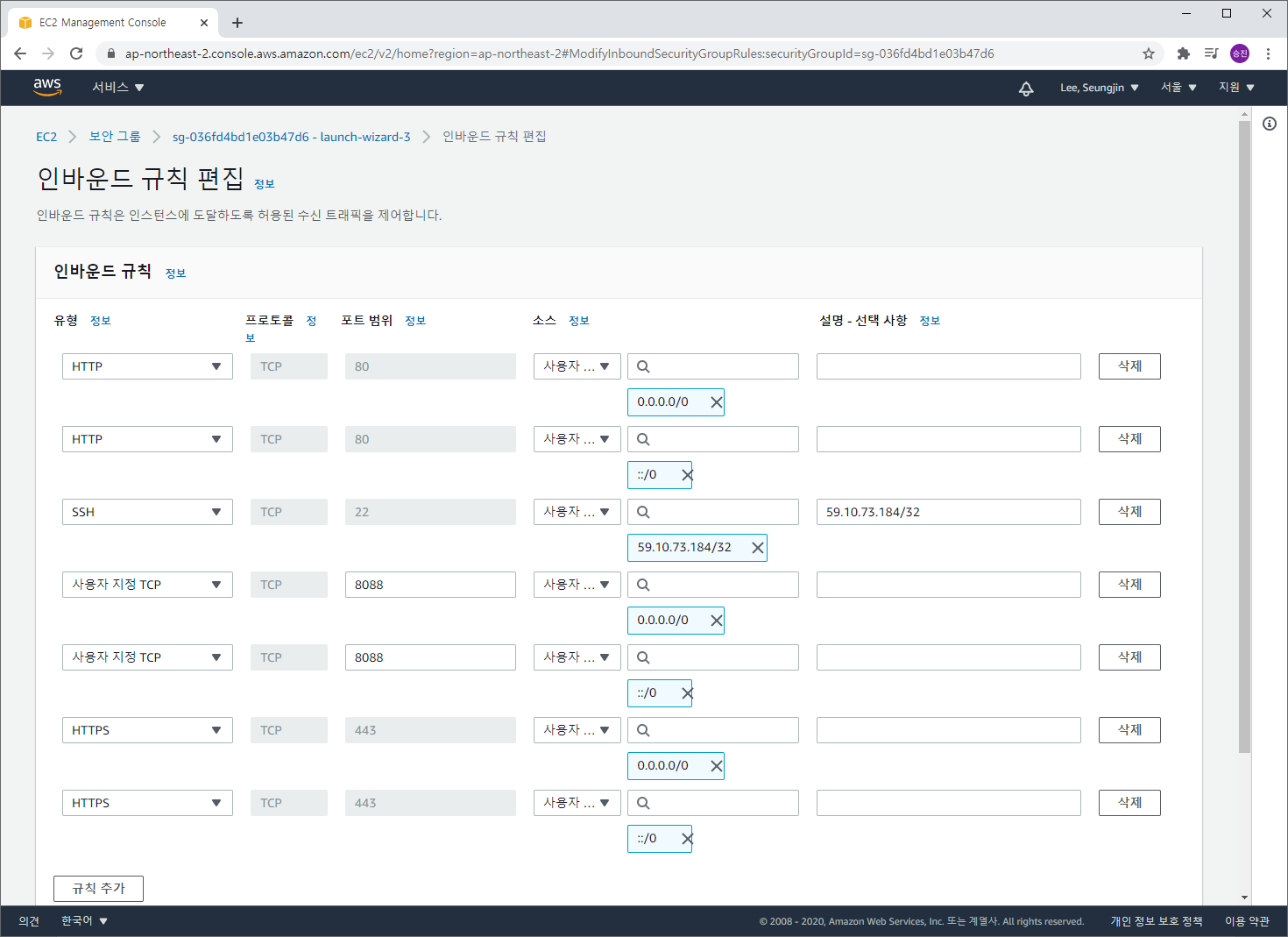
여기서 8088포트도 추가로 열 예정이다.







1. ‘규칙 추가’ 버튼 클릭
2. ‘사용자 지정 TCP’ 선택
3. ‘8088’ 포트 번호 입력
4. ‘위치 무관’ 항목 선택
5. ‘규칙 저장’ 클릭



수정 결과

### **참고(subnet 설정)**

0.0.0.0/0 => 모든 IPv4 주소 허용

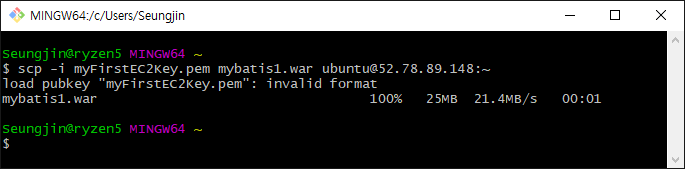
::/0 => 모든 IPv6 주소 허용

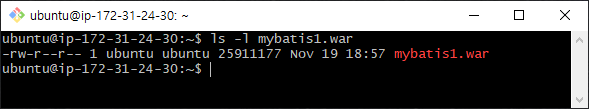
## **mybatis1.war**

강의자료 폴더에서 mybatis1.war 파일을 다운로드

Git bash 쉘에서 아래 명령을 실행하여, mybatis1.war 파일을 EC2 서버에 업로드

|  |
| --- |
| scp -i 키파일.pem mybatis1.war ubuntu@서버IP:~ |



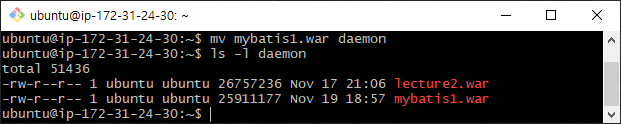


## **war 파일 옮기기**

데몬으로 실행할 war 파일을 적당한 디렉토리에 옮긴다

### **~/daemon 디렉토리로 옮기기**

|  |
| --- |
| mv mybatis1.war ~/daemon |



## **데몬 설정 파일 생성**

|  |
| --- |
| sudo vi /etc/systemd/system/mybatis1.service |

|  |
| --- |
| [Unit]  Description=mybatis1 spring boot apps  After=syslog.target  [Service]  User=ubuntu  ExecStart=java -Djava.security.egd=file:/dev/./urandom -jar /home/ubuntu/daemon/mybatis1.war  SuccessExitStatus=143  [Install]  WantedBy=multi-user.target |

mybatis1.war는 8088포트를 사용하기 때문에 실행할 때 sudo 명령이 필요 없다.

### **데몬 설정 파일 다시 읽기**

|  |
| --- |
| sudo systemctl daemon-reload |

데몬 설정 파일을 새로 만들었으니 위 명령을 실행해야 한다.

### **데몬 실행 시작**

|  |
| --- |
| sudo service mybatis1 start |

### **데몬 상태 보기**

|  |
| --- |
| sudo service mybatis1 status |

### **데몬 자동 실행 등록**

|  |
| --- |
| sudo systemctl enable mybatis1 |

재부팅 후에 자동 실행 시작되도록 등록

## **mybatis1 버그**

windows에서 실행되는 mysql 엔진은 테이블 이름의 대소문자와 무관하다.

그런데 linux에서 실행되는 mysql 엔진은 테이블 이름의 대소문자도 일치해야 한다.

mybatis1 강의예제의 SQL 문에서 student 테이블 이름들을 전부 소문자로 변경해야 한다.

# **성능 관리**

## **문제 발생**

EC2 인스턴스의 CPU 사용량이 100% 정도 가까이 오르면서 느려지는 경우

* 데몬으로 운영중인 Java 쪽에서 뭔가 문제가 또 있거나 (tomcat 버그 등)
* 메모리가 부족하거나 (t2-micro 인스턴스의 1gb 램이 부족)

## **CPU 사용량 많은 프로세스 보기**

CPU 사용량 많은 순으로 정렬하여 프로세스를 보는 명령은 다음과 같다.

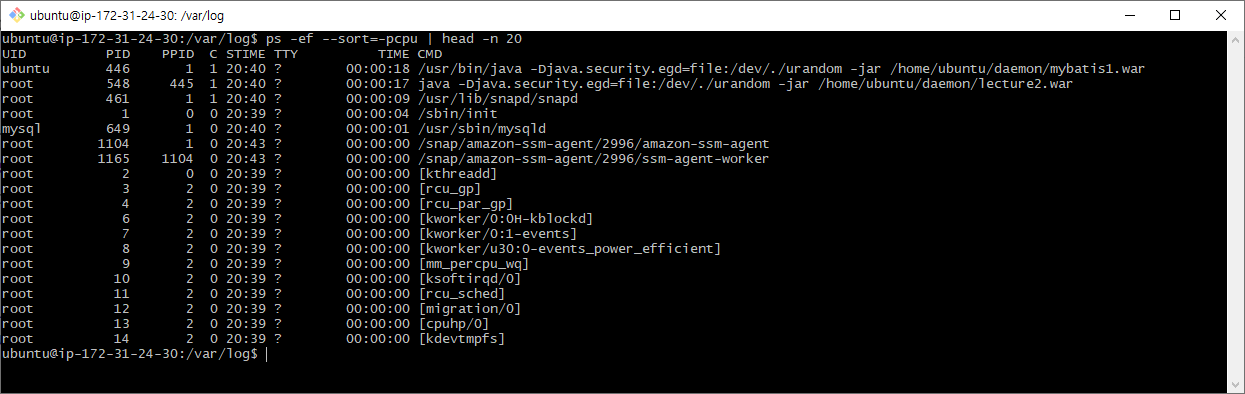
|  |
| --- |
| ps -ef --sort=-pcpu |

ps 명령에 --sort=-pcpu 옵션만 추가하면 된다.

CPU 사용량 많은 선두 20개 프로세스만 보는 명령

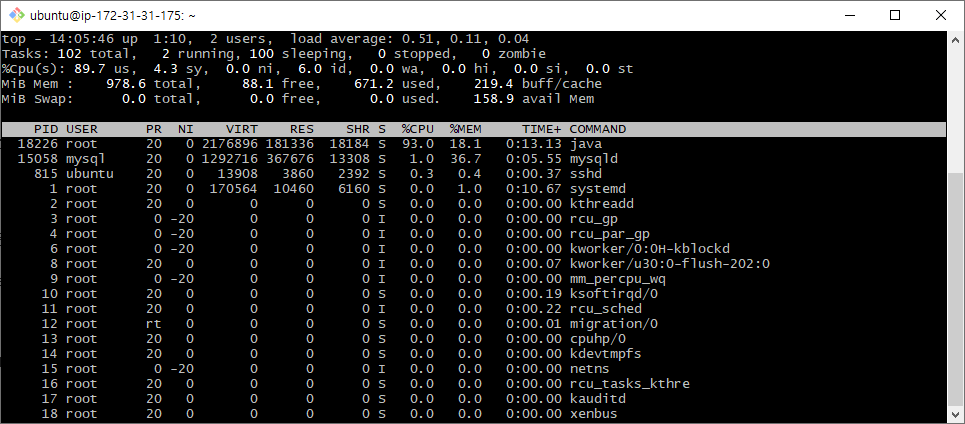
|  |
| --- |
| ps -ef --sort=-pcpu | head -n 20 |

head 필터는, 입력으로 받은 내용 중에서 선두 몇 라인만 출력한다.

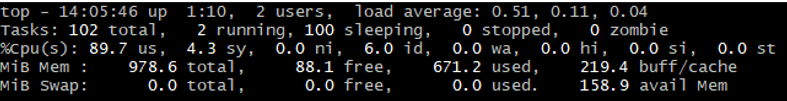


## **top 명령**

CPU 사용량, 메모리 사용량 등 전체적인 서버 성능을 실시간으로 지켜볼 수 있는 명령



### **화면 설명**



|  |  |
| --- | --- |
| 화면 내용 | 설명 |
| top - 14:05:46 | 현재 서버 시각이다. |
| up 1:10 | 컴퓨터가 시작된 후 1시간 10분 지났다 |
| 2 users | 현재 접속해 있는 사용자(터미널 창)의 수 |
| load average:  0.51,  0.11,  0.04 | CPU 평균 작업량 (쓰레드의 평균 숫자)  최근 1분 동안 평균 0.51 개의 쓰레드가 실행 중이었다  최근 5분 동안 평균 0.11 개의 쓰레드가 실행 중이었다.  최근 15분 동안 평균 0.04 개의 쓰레드가 실행 중이었다.  CPU 사용량 % = (쓰레드의 평균 수) / (CPU 코어 수) \* 100% |
| Tasks:  102 total  2 running  100 sleeping  0 stopped  0 zombie | 프로세스 숫자  현재 102 개의 프로세스가 실행 메모리에 있다  2 개의 프로세스가 실행 가능 상태 (running 상태, ready 상태)  100 개의 프로세스는 sleep 상태 (I/O 장치의 작업을 기다리는 중)  실행을 중지했고 곧 메모리에서 제거될 프로세스가 0 개.  실행을 중지했고 메모리에서 제거되었으나, 부모 프로세스에게 종료 상태 값을 전달하기 위해 프로세스 정보만 남아있는 프로세스가 0 개. |
| %CPU(s)  89.7 us  4.3 sy  0.0 ni  6.0 id  0.0 wa  0.0 hi  0.0 si  0.0 st | CPU 사용 %  애플리케이션 영역의 기계어 코드를 실행한 시간 (user space)  운영체제 커널의 기계어 코드를 실행한 시간 (system space)  낮은 우선순위의 프로세스를 실행한 시간 (nice)  할 일 없는 상태의 시간 (idle)  I/O 작업 완료를 기다리는 시간 (wait)  hardware interrupt를 처리한 시간  software interrupt를 처리한 시간  실제 CPU가 내 EC2 인스턴스를 실행하고 있지 않은 상태 (steal state) |
| MiB Mem:  978.6 total,  88.1 free,  671.2 used,  219.4 buff/cache | 메모리(RAM) 용량 (MiB = 1024 \* 1024 bytes)  총 978.6 램  남은 메모리 용량  사용된 메모리 용량  디스크 입출력 캐시로 사용된 용량 |
| MiB Swap:  0.0 total,  0.0 free,  0.0 used,  158.9 avail Mem | virtual memory로 할당된 디스크 용량 (swap memory)  총 용량  남은 용량  사용된 용량  새 프로세스가 실행될 경우 할당 가능한 메모리 용량 |



|  |  |
| --- | --- |
| PID | 프로세스 ID |
| USER | 프로세스를 만든 사용자 계정 |
| PR | 프로세스에 실제로 적용된 실행 우선순위 (priority)  가장 높은 우선순위: 0, 가장 낮은 우선 순위: 39  PR 항목의 값이 숫자가 아니고 rt 인 프로세스도 있다.  rt = realtime |
| NI | nice 값은 프로세스 실행 우선순위 설정 값이다.  nice 값으로부터 실제 우선순위가 계산된다.  가장 높은 우선순위: -20, 가장 낮은 우선 순위: 19 |
| VIRT | 가상 메모리(virtual memory) 사용량 |
| RES | 실제 메모리(real memory) 사용량. (resident size) |
| SHR | 공유 메모리 크기(shared memory)  다른 프로세스와 공유하는 메모리 용량 |
| S | 프로세스 상태 (state)  R : running  S : sleeping  T : stopped  Z : zombie  D : uninterruptible sleep  I : idle |
| %CPU | CPU 사용량 |
| %MEM | 메모리 사용량 |
| TIME+ | CPU 사용량 (0:13.13 = 0분 13.13초) |
| COMMAND | 명령 |

### **단축키**

|  |  |
| --- | --- |
| shift + t | 실행 시간 내림차순 정렬 |
| shift + m | 메모리 사용량 내림차순 정렬 |
| shift + p | CPU 사용량 내림차순 정렬 |
| k | 프로세스 죽이기.  k 입력 후 죽일 프로세스ID를 입력하고 엔터,  9를 입력하고 엔터 |
| u | 특정 사용자의 프로세스만 보기.  u 입력 후 사용자 계정명을 입력하고 엔터.  u 입력 후 그냥 엔터를 입력하면 모든 사용자의 프로세스를 보여준다. |
| q | top 명령 종료 |

## **가상 메모리 (virtual memory)와 스왑 영역(swap space)**