WebSocket

made by sangwon

WebSocket 설명 전에
HTTP의 개념을 잠깐 집고 넘어가자!

HTTP(HyperText Transfer Protocol)

HTTP는 클라이언트에서 서버로의
단방향 통신을 위해 만들어진 프로토콜!

즉,HTTP로 양방향 데이터를 주고 받는 다는건
말~도~ 안되는 소리! 매우 멍청한 프로토콜(통신)이다.

웹소켓 이전에 썼던 방법

AJAX로 트릭을 사용했는데..
전문용어로 COMET 이라 하고,
COMET에는 2가지 방법이 있는데.. 아래와 같다!

1. Polling - 일정 간격으로 request를 날림

2. Long polling - 우선 request를 날림 response를 받으면 연결종료 그리고 다시 request ... 무한반복

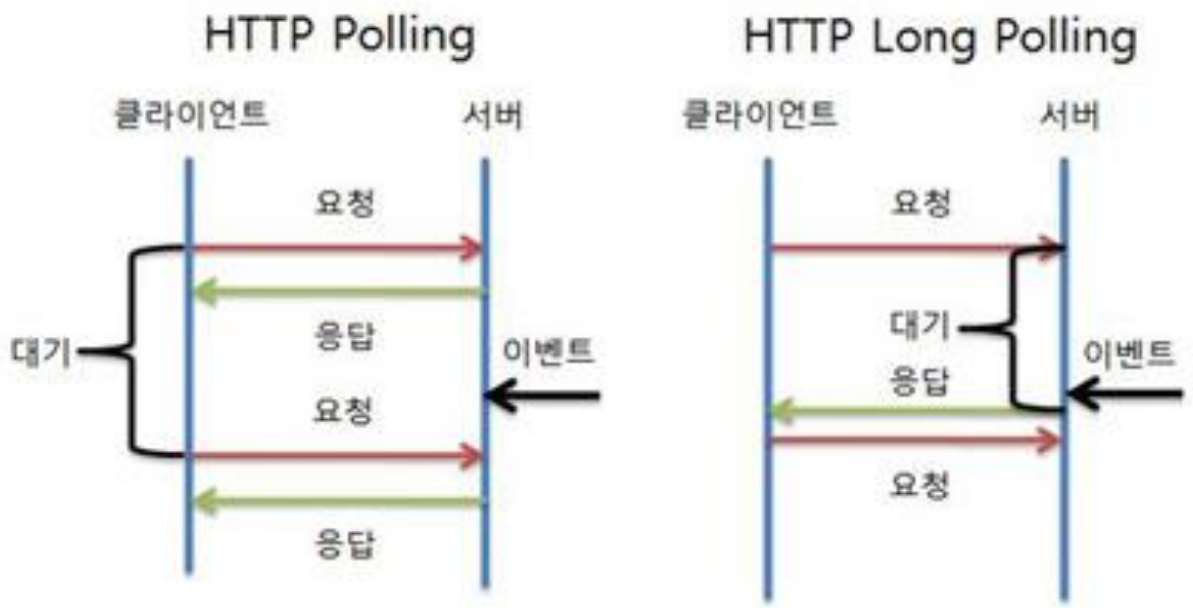
폴링?

- 클라이언트 코드:

```
1.  function ajax() {
2.
3.      Ajax.request({
4.          url: 'http://m.fpscamp.com/m.aspx',
5.          type: 'text',
6.          method: 'post',
7.          headers: {
8.              'content-type': 'application/x-www-form-urlencoded'
9.          },
10.         data: {
11.         },
12.         onprogress: function (e, total, loaded, per, computable) {
13.         },
14.         onerror: function () {
15.             alert('onerror');
16.         },
17.         callback: function (data, status) {
18.             document.body.innerHTML = data;
19.         }
20.     });
21. };
22.
23. var timer = window.setInterval(function () { ajax(); }, 1000);
```

window.setInterval() 메소드를 사용하여 일정한 주기로 HTTP요청을 보낸다.

폴링 구조도



이것은 클라이언트가 많아지면 서버에 부담이 된다.
왜? 계속 HTTP요청을 일정 간격으로 함으로써

COMET의 한계

COMET 방식은 불필요한 많은 양의
네트워크 *오버 헤드를 발생! 또한 HTTP 대기
시간에 따른 성능 저하도 덤으로 준다.. 1+1

*오버헤드 : 어떤 처리를 하기 위해 들어가는
간접적인 처리 시간 ,메모리 등을 말한다.

WebSocket의 등장

HTTP의 단점을 보완하기 위하여 등장!

1. HTTP는 Client-Server간 **접속을 유지하지 않음**
2. Client-Server간 한번에 **한 방향으로만** 통신(half-duplex)

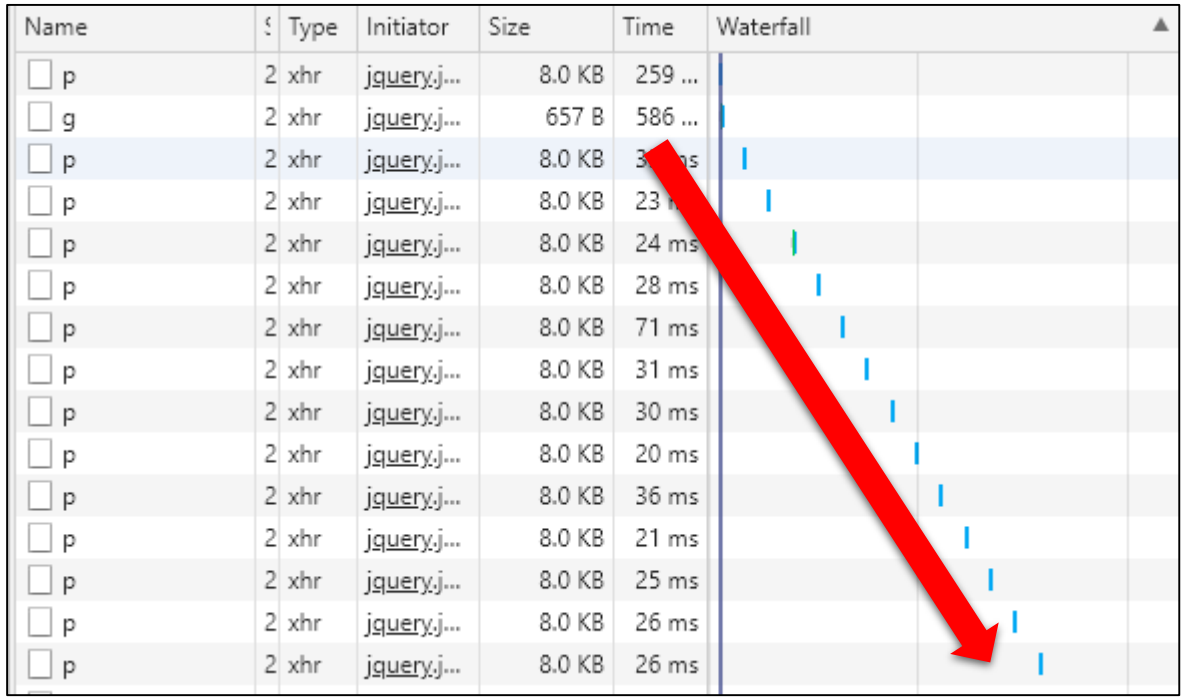
COMET과 WebSocket 비교 1

COMET :
서버 응답후 요청에 대한 대기시간이 추가로 발생

WebSocket :
최초 서버와 연결 후, 연결이 그대로 유지
클라이언트가 재요청을 보낼 필요가 없다.
즉, 추가적인 대기시간 발생하지 않는다!

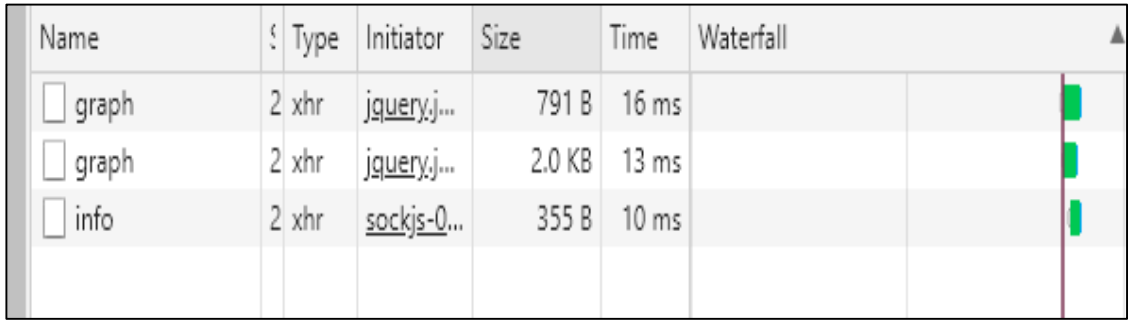
COMET과 WebSocket 비교 2

COMMET



Name	S	Type	Initiator	Size	Time	Waterfall
p	2	xhr	jquery.j...	8.0 KB	259 ...	
g	2	xhr	jquery.j...	657 B	586 ...	
p	2	xhr	jquery.j...	8.0 KB	3 ms	
p	2	xhr	jquery.j...	8.0 KB	23 ms	
p	2	xhr	jquery.j...	8.0 KB	24 ms	
p	2	xhr	jquery.j...	8.0 KB	28 ms	
p	2	xhr	jquery.j...	8.0 KB	71 ms	
p	2	xhr	jquery.j...	8.0 KB	31 ms	
p	2	xhr	jquery.j...	8.0 KB	30 ms	
p	2	xhr	jquery.j...	8.0 KB	20 ms	
p	2	xhr	jquery.j...	8.0 KB	36 ms	
p	2	xhr	jquery.j...	8.0 KB	21 ms	
p	2	xhr	jquery.j...	8.0 KB	25 ms	
p	2	xhr	jquery.j...	8.0 KB	26 ms	
p	2	xhr	jquery.j...	8.0 KB	26 ms	

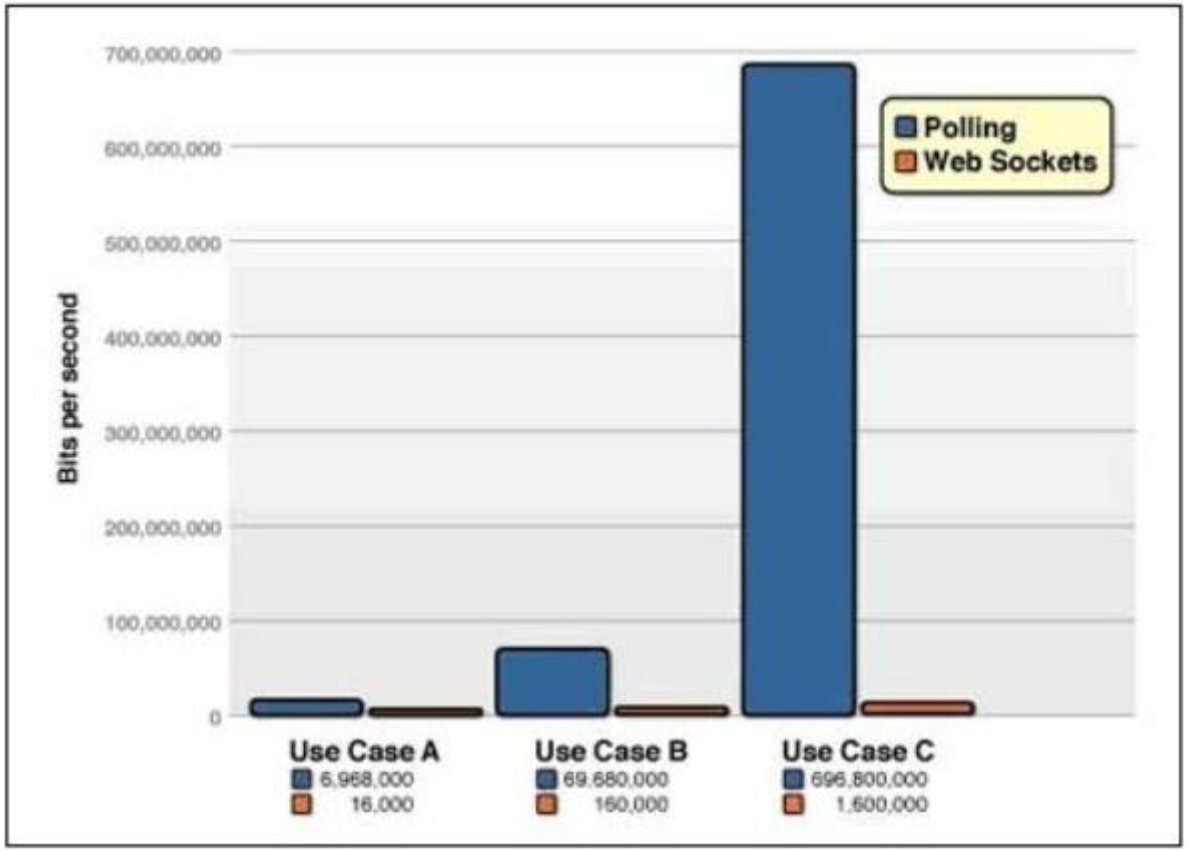
WebSocket



Name	S	Type	Initiator	Size	Time	Waterfall
graph	2	xhr	jquery.j...	791 B	16 ms	
graph	2	xhr	jquery.j...	2.0 KB	13 ms	
info	2	xhr	sockjs-0...	355 B	10 ms	

왼쪽 COMMET을 보게 되면, Waterfall이 점점 깊어진다.
계속 요청한는것을 볼 수 있다. 하지만 반대로 오른쪽
웹소켓은 한번 호출후 지속적인 연결을 하고 있다.

COMET과 WebSocket 비교 3



네트워크 오버헤드 비교

Comet과 WebSocket을 성능적으로 비교해도
압도적인 WebSocket의 승리

WebSocket 장점

1. Stateful protocol(연결 유지) 때문에 클라이언트와 한번 연결되면
같은 라인을 사용 즉, HTTP와 *TCP 연결 트래픽을 피할 수 있다.

2. WebSocket도 80포트를 사용하기에 추가적인 병화벽 설정 필요 없다.
(웹소켓 프로토콜은 wss://로 시작)

*TCP : IP로 컴퓨터의 위치를 찾았다면, 그
위치로 데이터를 신뢰성있게 전달하는 통신방법

WebSocket 단점

1. 서버와 클라이언트 간의 연결을 항상 유지해야 하며 만약
비정상적으로 연결이 끊어졌을 때 적절하게 대응

2. 트래픽양이 많은 서버 같은 경우에는 CPU에 부담이
크기에 *Scale out 기술 필요

3. 아파치에 설정을 따로 해줘야함 많은 삽질을 함(개인적인 생각)

*Scale out : 접속된 서버의 대수를 늘려 처리 능력을
향상시키는 기술. 수평 스케일 이라고도 한다.

WebSocket 대표적인 사용 예

1. 페이스북, 인스타그램 SNS
2. 배틀그라운드,LOL 같은 멀티플레이어 게임
3. 클릭 동향 분석 데이터
4. 증권 거래 정보(코인 사이트)
5. 스포츠 업데이트
6. 화상 채팅
7. 위치 기반 프로그램
8. 온라인 교육

Spring + Websocket

Spring WebSocket은 4버전 이후로 가능하다.
버전 수정후 메이븐을 등록해주자!

```
<org.springframework-version>4.1.4.RELEASE</org.springframework-version>
```

Spring Websocket

WebSocket
- IE10+

SockJS
- Node.js의 socket.io 와 같은 매커니즘
- IE 8+

STOMP (*토픽 구독방식)
- Spring만 사용가능
- 패킷을 다양하게 설정할 수 있다.

*토픽 : 서버에서 A라는 토픽에 무언가를 내보내면 그쪽을 보고 있는 클라이언트들은 모두 메세지를 받는방법.

SockJS와 STOMP

Websocket은 IE10+ 지원하기에 SockJS 라이브러리 IE8+
이랑 같이 써야한다.

STOMP는 텍스트 기반의 메세징 프로토콜이다. TCP나 Websocket과
같은 양방향 통신에 사용될 수 있다.
HTTP와 같은 *Frame 기반의 프로토콜이다.

*Frame : 주소와 명령어,명령 수행을 위한
데이터가 포함 프레임은 헤더와 바디로 구성

STOMP을 Spring에 적용할때 알아야 할 개념

Broker : STOMP의 통신 방식을 최종결정
1:N 통신이냐? (topic)
1:1 통신이냐? (queue)

Endpoint : 클라이언트와 서버랑 연결할 주소!

```
@Configuration
@EnableWebSocketMessageBroker
public class SocketConfiguration extends AbstractWebSocketMessageBrokerConfigurer {

	@Override
	  public void configureMessageBroker(MessageBrokerRegistry config) { //STOMP 방식과 경로를 지정해주는 메소드
	      config.enableSimpleBroker("/topic/"); //토픽방식이 있고 큐방식이 있다. 통신방법 수정
	      config.setApplicationDestinationPrefixes("/dap"); //path
	  }
	@Override
	  public void registerStompEndpoints(StompEndpointRegistry registry) {//STOMP 기능 추가해주는 메소드
	       registry
	       .addEndpoint("/block") // 클라이언트랑 연결할 주소
	       .setAllowedOrigins("*") // http,https 통신 가능
	       .withSockJS() // SockJS를 사용하겠다
	       .setStreamBytesLimit(512*1024) //보낼 데이터 바이트 제한
	       .setHttpMessageCacheSize(1000); //캐시 사이즈 제한
	  }

}
```

Client <-> Server 웹소켓

Client

```
function socket_init(){
    var sock = new SockJS("/dap/block");
    var client = Stomp.over(sock);

     client.connect({}, function(frame){
            client.subscribe('/topic/blockInfo', function(response) {
            add_block(response.body);
             });
            client.subscribe('/topic/tranInfo', function(response) {
            add_tranInfo(response.body);
            });
            client.subscribe('/topic/supply', function(response) {
            show_now_dap_statistics(response.body);
            });
    });
 }
```

Server (스케줄러)

```
simpMessagingTemplate.setMessageConverter(new StringMessageConverter());
simpMessagingTemplate.convertAndSend("/topic/blockInfo",blockJson);
simpMessagingTemplate.convertAndSend("/topic/tranInfo",tranJson);
simpMessagingTemplate.convertAndSend("/topic/supply",tranAndSupplyJson);
```

STOMP 방법으로 서버와 클라이언트가 실시간으로 통신을 하게 된다.

Client <-> Server 웹소켓

Client

```
function socket_init(){
    var sock = new SockJS("/dap/block");
    var client = Stomp.over(sock);

     client.connect({}, function(frame){
            client.subscribe('/topic/blockInfo', function(response) {
            add_block(response.body);
             });
            client.subscribe('/topic/tranInfo', function(response) {
            add_tranInfo(response.body);
            });
            client.subscribe('/topic/supply', function(response) {
            show_now_dap_statistics(response.body);
            });
    });
 }
```

Server (스케줄러)

```
@RestController
public class MessageHandler {
    @MessageMapping("/hello") //설정한 prefix를 포함하면 /app/hello이다.//
    @SendTo("/topic/roomId") //전달할려는 곳의 subscribe//
    public MessageVO broadcasting(MessageVO message) throws Exception{
        System.out.println("message: " + message.getSendMessage());
        return message;
    }

    //각각의 메세지 유형에 따라 Mapping을 추가해줄 수 있다.//
    @MessageMapping("/out") //설정한 prefix를 포함하면 /app/hello이다.//
    @SendTo("/topic/out") //전달할려는 곳의 subscribe//
    public String outroom(String message) throws Exception{
        System.out.println("out message: " + message);
        return message;
    }

    @MessageMapping("/in") //설정한 prefix를 포함하면 /app/hello이다.//
    @SendTo("/topic/in") //전달할려는 곳의 subscribe//
    public String inroom(String message) throws Exception{
        System.out.println("in message: " + message);
        return message;
    }
}
```

@MessageMapping() , @SendTo() 어노테이션을 사용해서 컨트롤러와 통신

WebSocket 아파치 적용시 설정 값

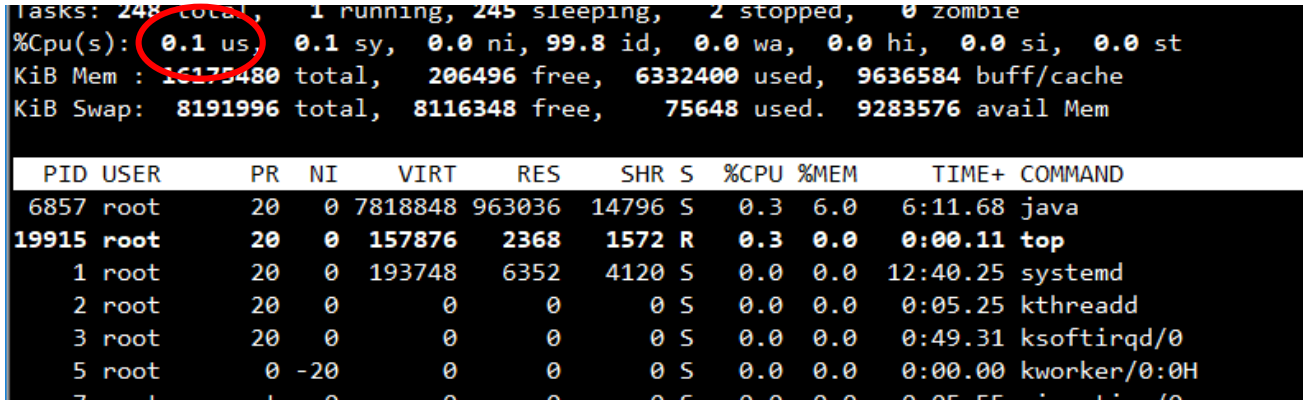
```
RewriteEngine On

RewriteCond %{HTTP:Upgrade} =websocket [NC]
RewriteRule /(.*\/websocket) ws://59.26.68.149:8080/$1 [P,L]
RewriteCond %{HTTP:Upgrade} !=websocket [NC]
RewriteRule /(.*) http://59.26.68.149:8080/$1 [P,L]
```

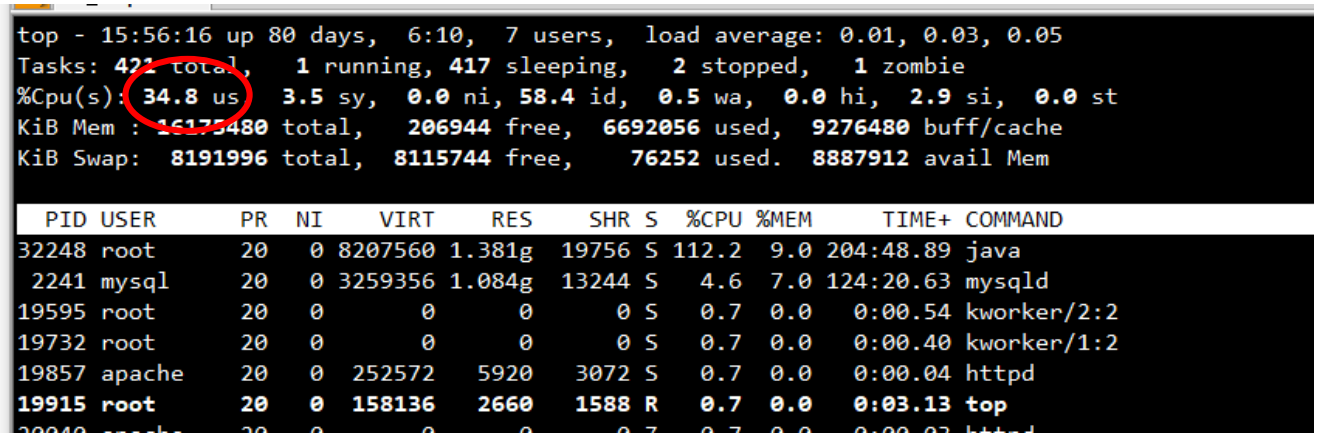
디폴트로 RewriteEngin은 Off이다. On으로 변경
RewriteEngin으로 URL을 유동적으로 관리해준다.
http -> https 로 전환도 여기서 해준다.

RewriteCond는 if문 이다.
RewriteRule 은 조건에 맞는 실행문 이다.

성능 테스트 (Jmeter)

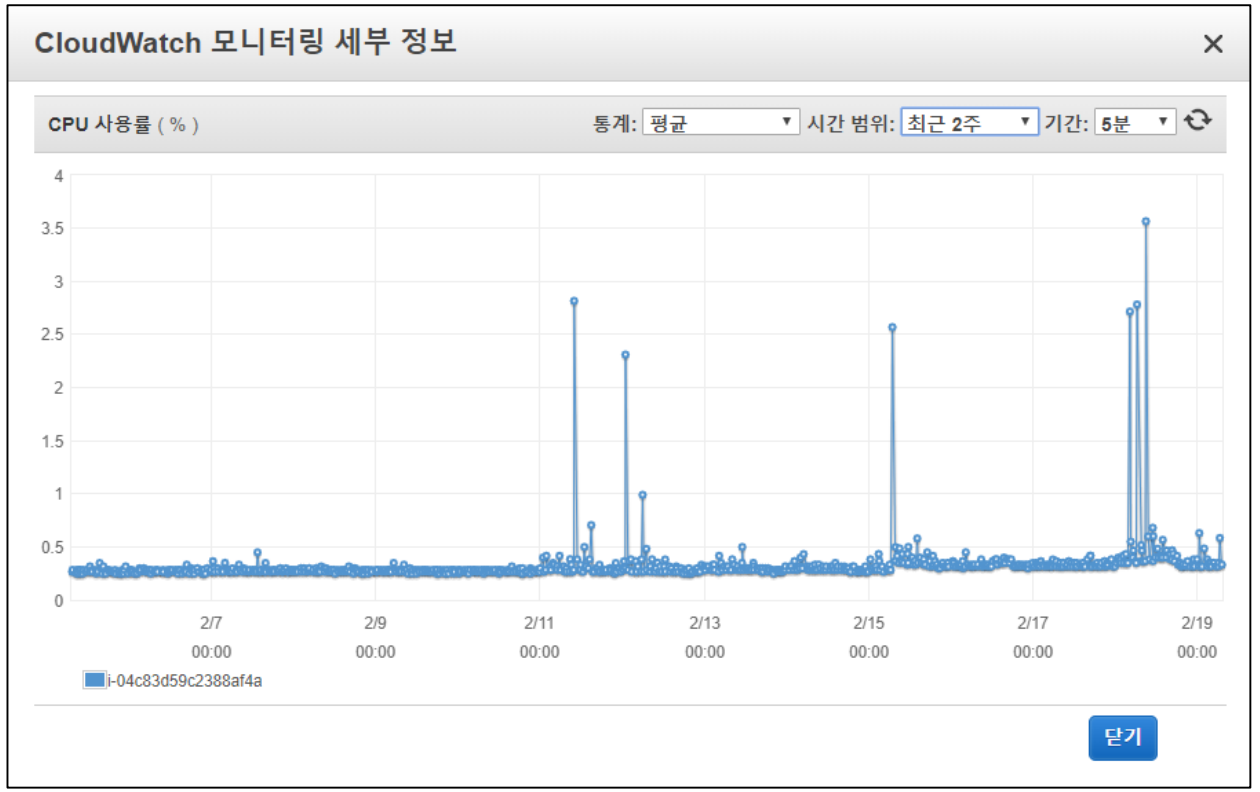


부하 전



부하 후 (500명 1초간격 접속시)

AWS 2주간 CPU 사용률



크게 부하가 되거나 널뛰현상은 일어나지 않는다.

WebSocket 결론

웹소켓 꽃은 채팅프로그램이 아닐까라는 생각이 든다.
간단한 실시간 데이터 처리였지만, 서버와 클라이언트간
지속적인 통신이 필요할때는 웹소켓을 써야하는게 맞는것 같다.