내가 만든 WebFlux가 느렸던 이유

NHN Global, FG AD Development Team
김병부

NHN FORWARD

소개

- 미국 법인
- 패션고(www.fashiongo.net)
- 광고 개발팀
- 응답속도:100ms 이하
- 처리량: Fashiongo 전체 트레픽 * 3

소개

Spring WebFlux 선택

- 응답 속도 :100 ms 이하
- 처리량: Fashiongo 전체 트래픽 * 3

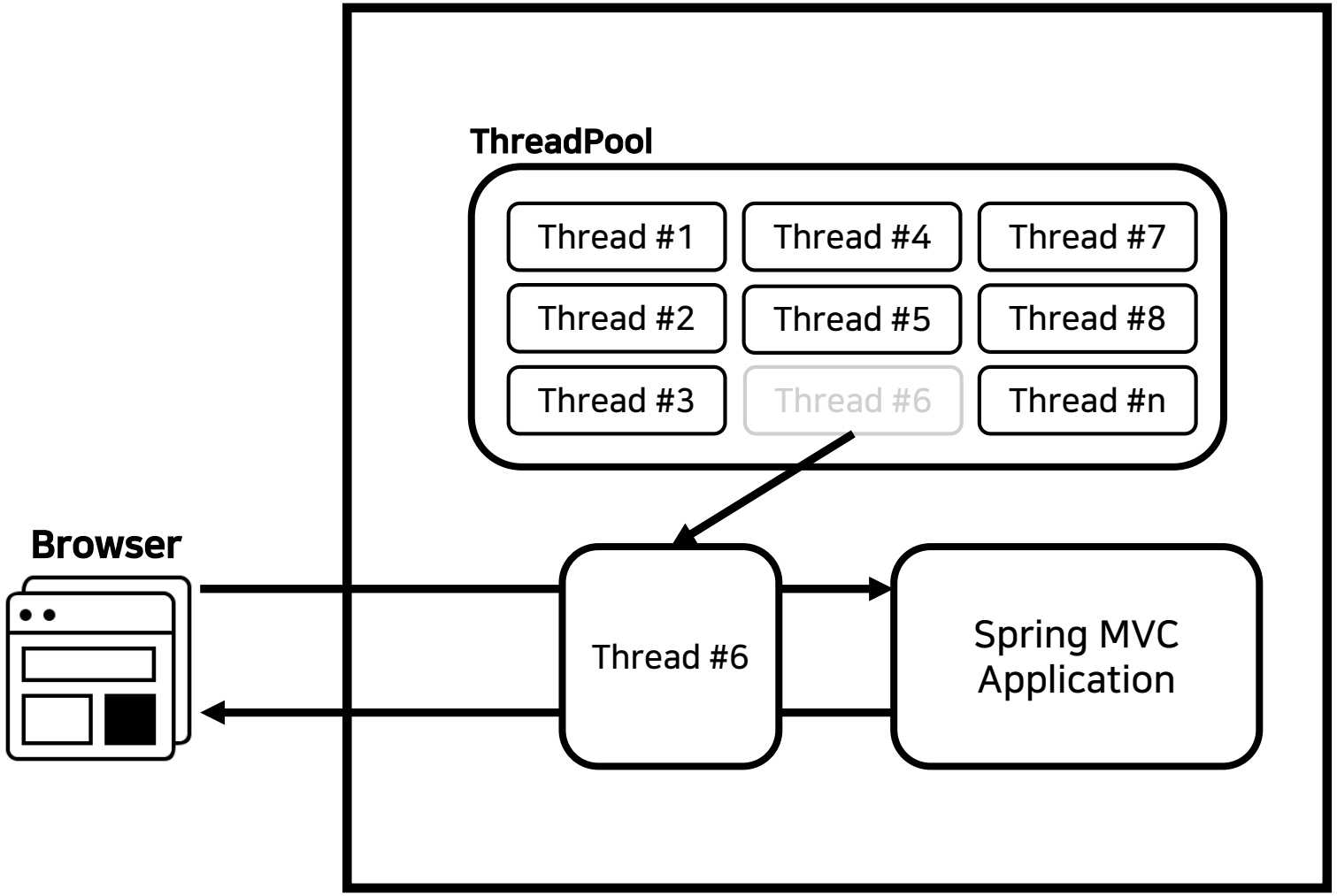
다룰 내용

1. WebFlux를 선택한 이유
2. 나의 WebFlux가 느린 이유
   - 성능 측정 결과
   - 성능 개선 사항

WebFlux를 선택한 이유

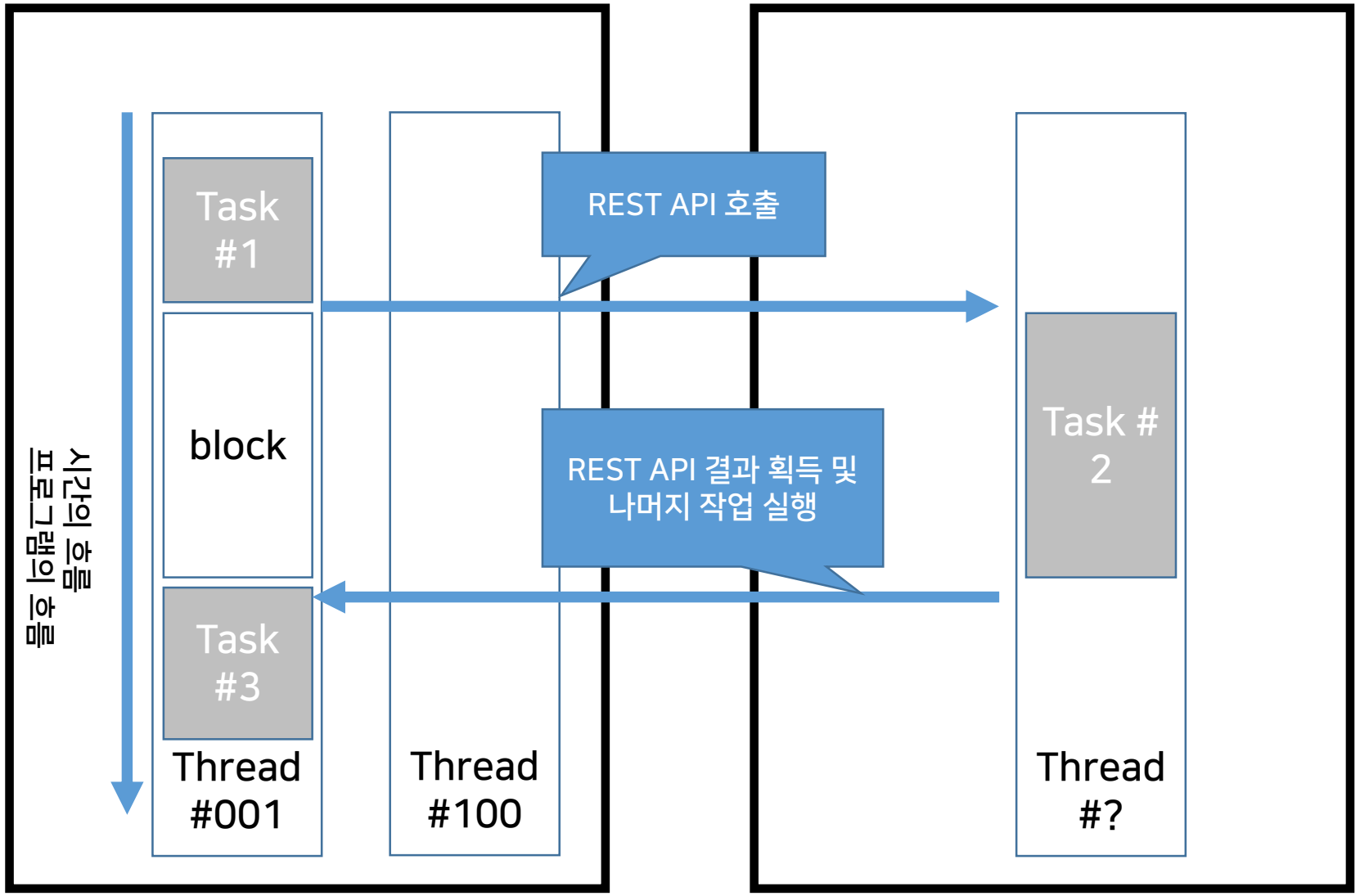
Spring MVC

- Thread Per Request Model
- Thread 개수 == 200



WebFlux를 선택한 이유

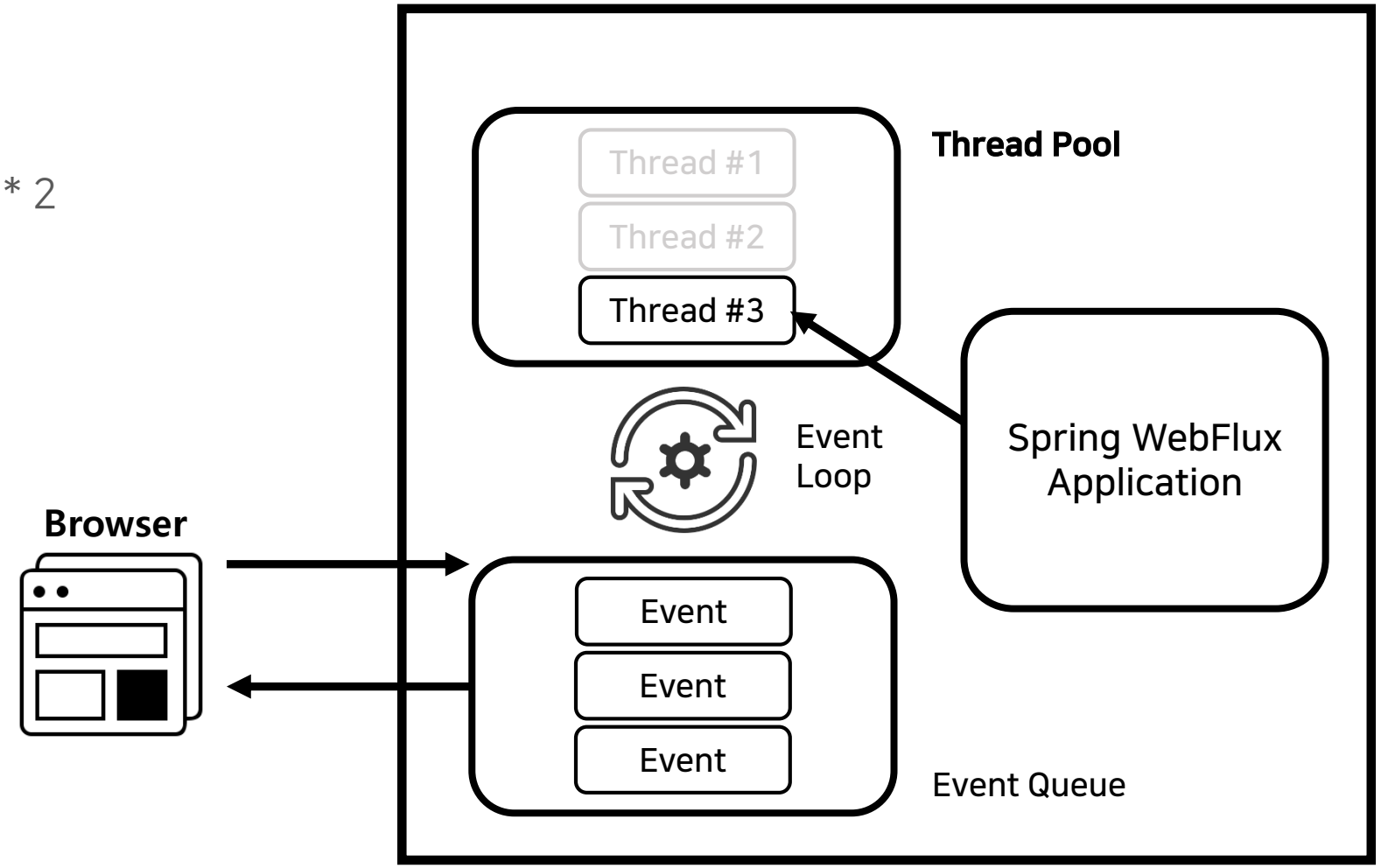
Spring MVC



WebFlux를 선택한 이유

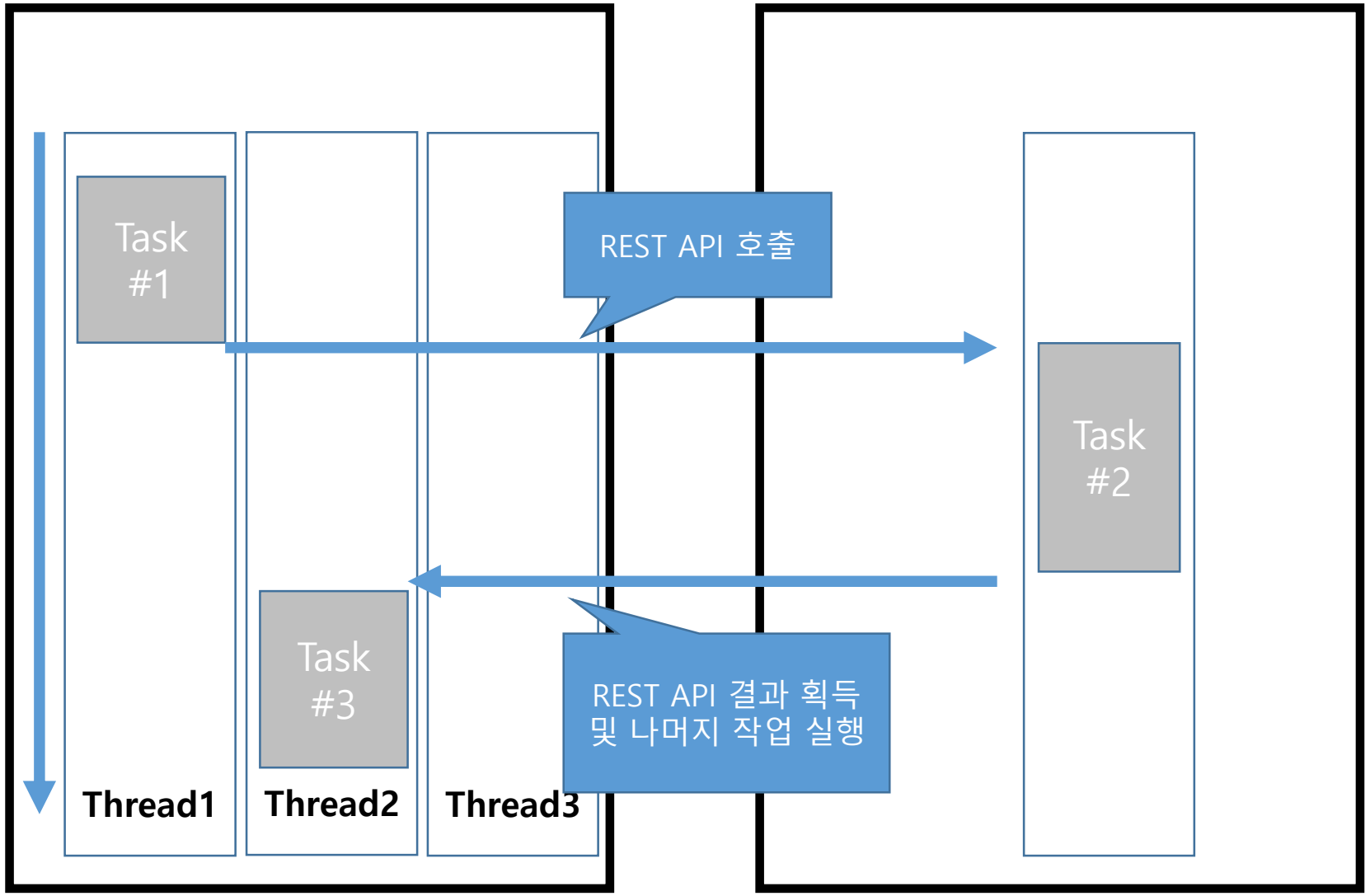
Spring WebFlux

- EventLoop Model
- Thread 개수 == Core 의 개수 * 2



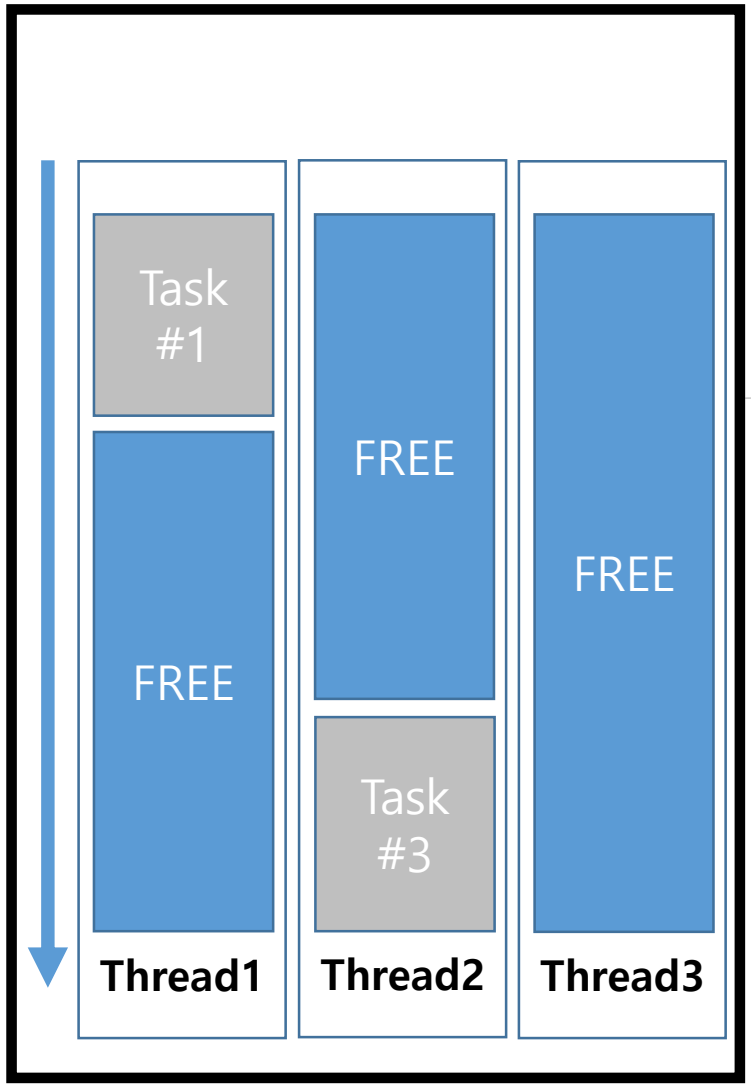
WebFlux를 선택한 이유

Spring WebFlux



WebFlux를 선택한 이유

Spring WebFlux



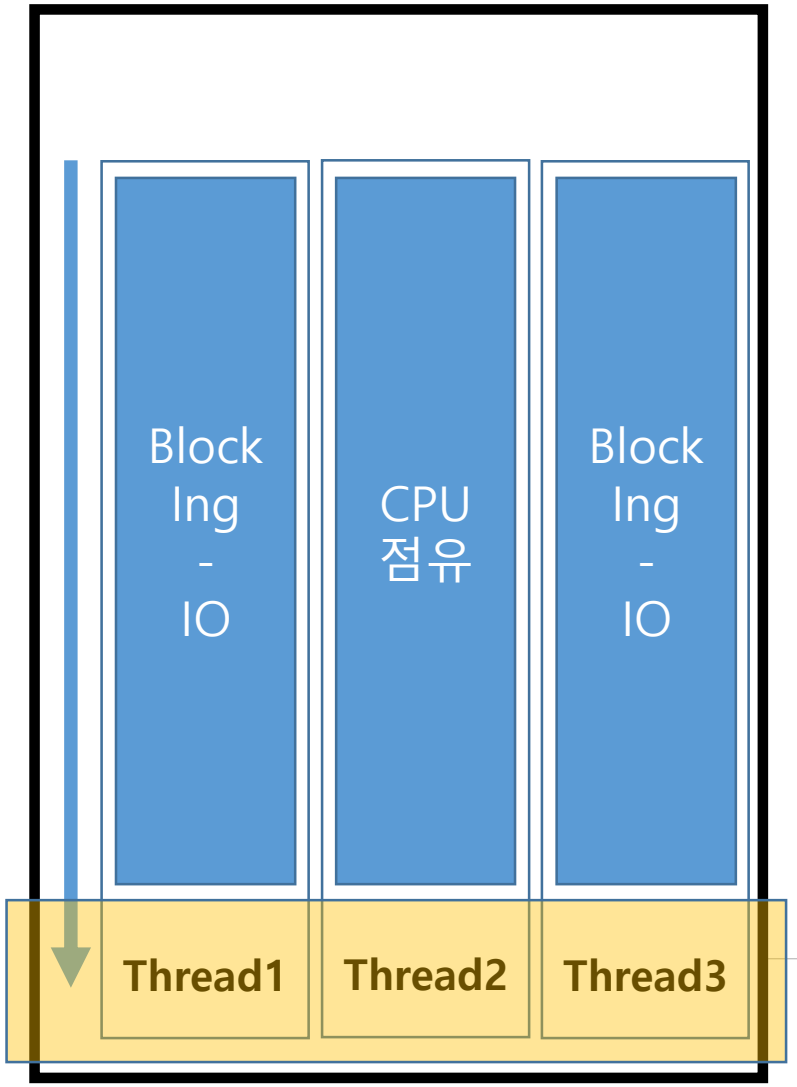
높은 처리량의 이유

EventLoop 의한 고효율

Non-Blocking IO

WebFlux를 선택한 이유

Spring WebFlux



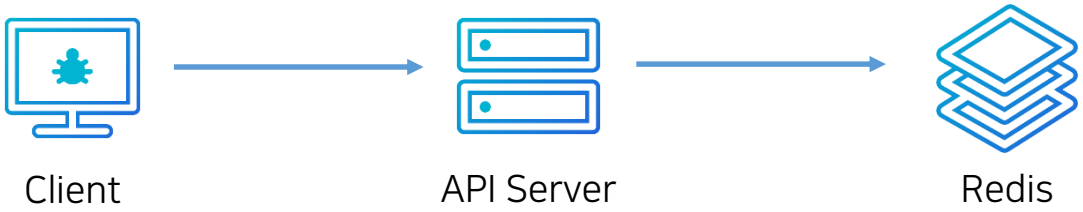
성능 저하의 원인

CPU 사용이 많은 작업

Blocking IO

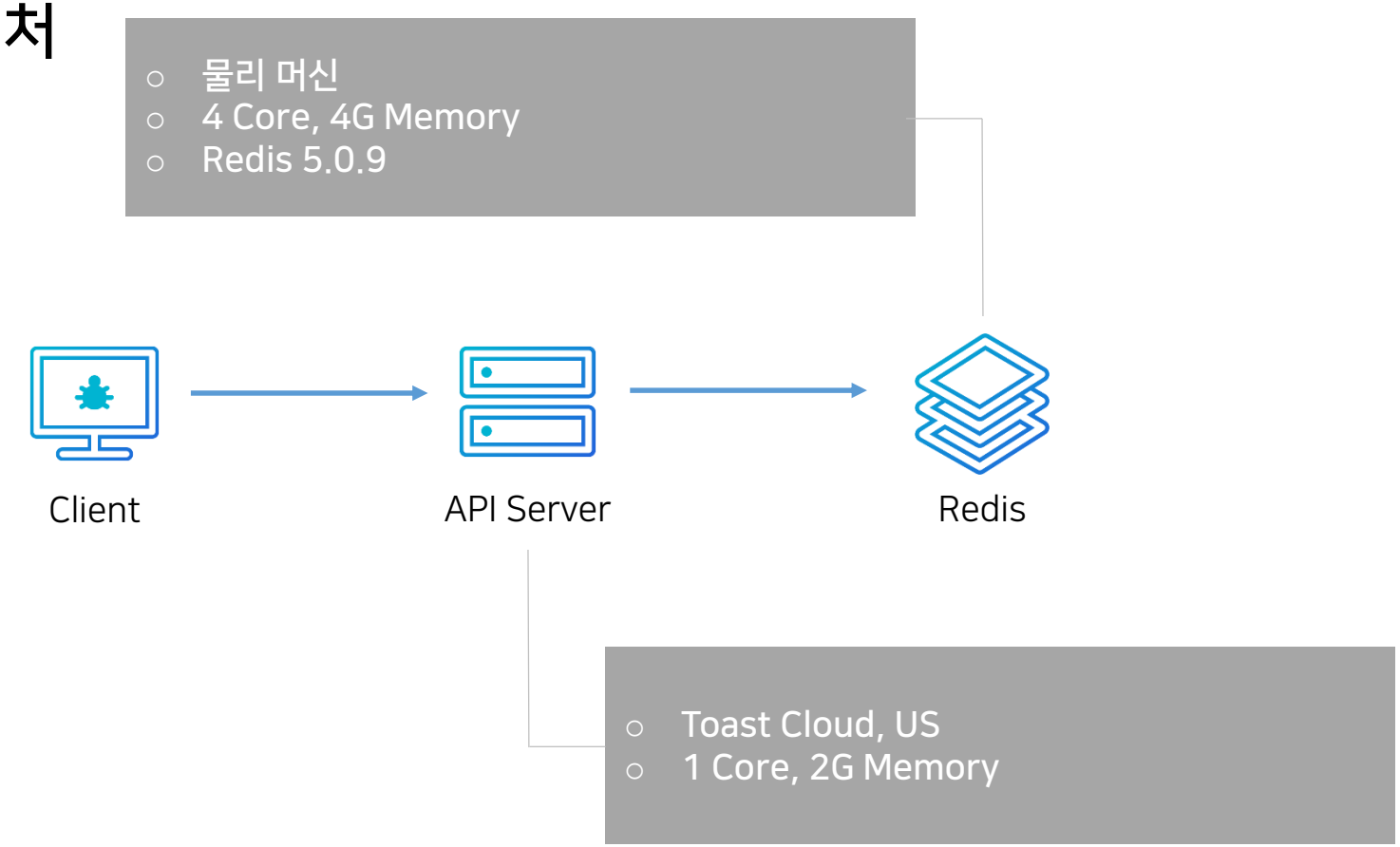
나의 WebFlux가 느린 이유

테스트 시나리오 아키텍처



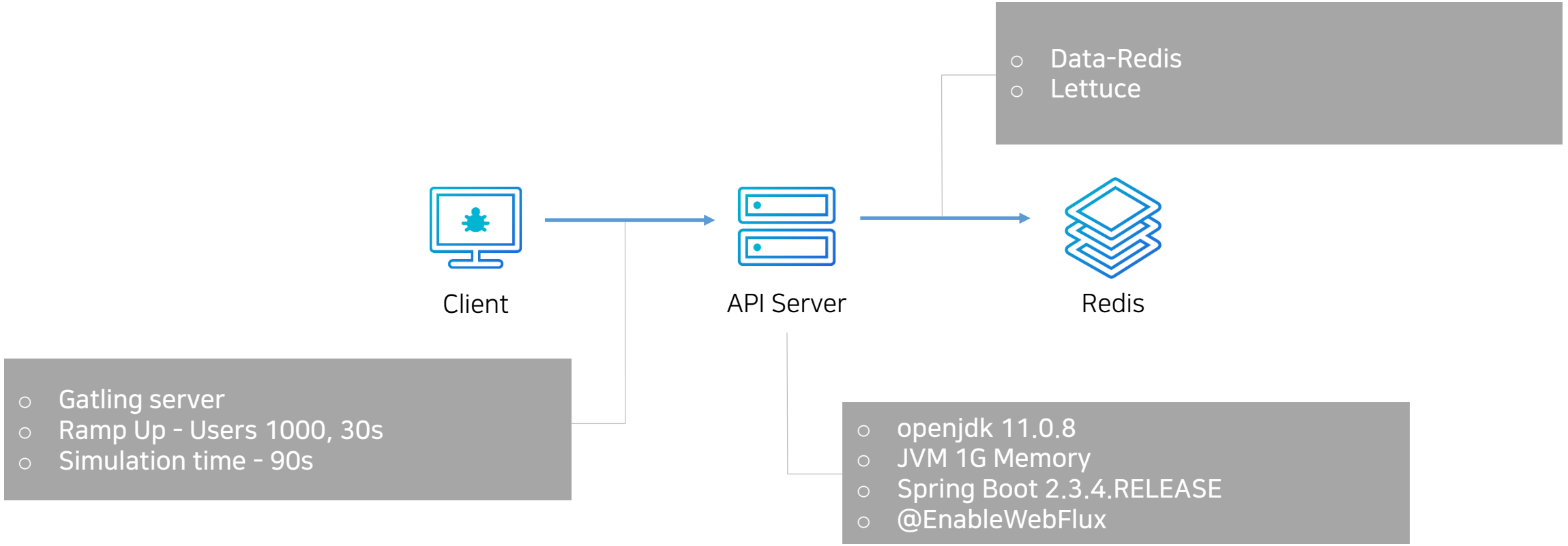
나의 WebFlux가 느린 이유

테스트 시나리오 아키텍처



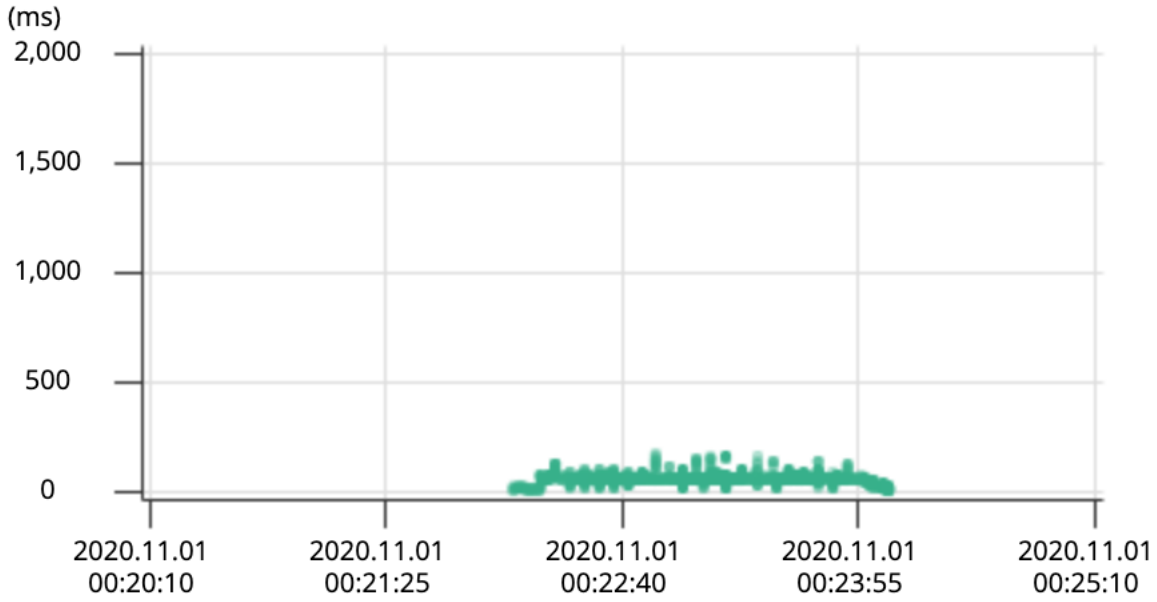
나의 WebFlux가 느린 이유

Spring WebMVC + Lettuce 구성



나의 WebFlux가 느린 이유

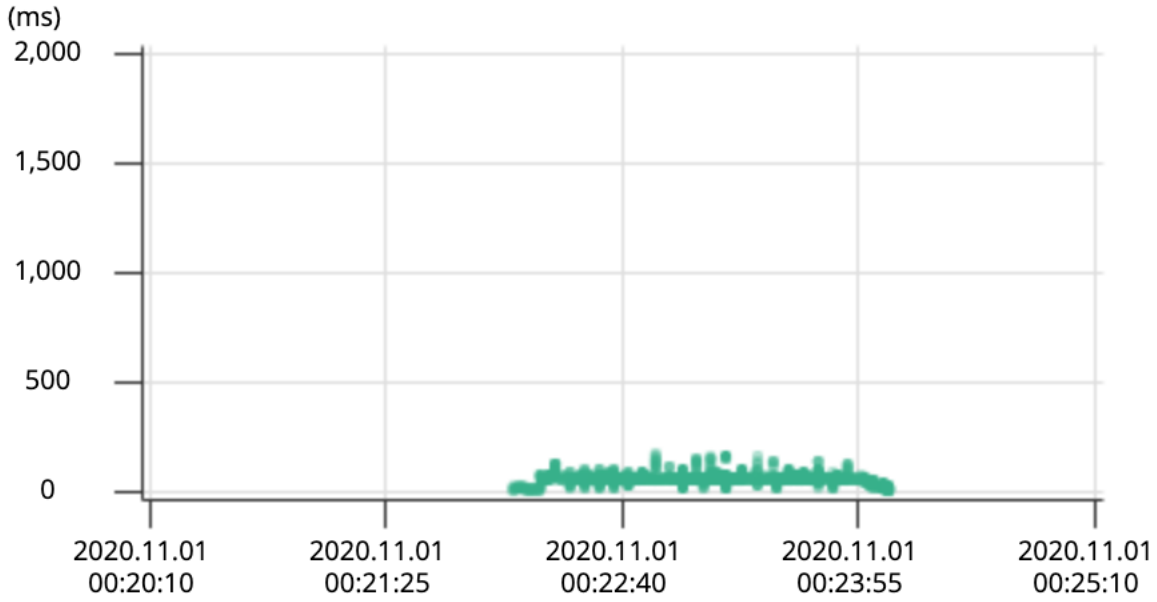
Spring WebMVC + Lettuce 구성




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     452999 (OK=452999 KO=0     )
> min response time                                      1 (OK=1      KO=-     )
> max response time                                    442 (OK=442    KO=-     )
> mean response time                                   199 (OK=199    KO=-     )
> std deviation                                         85 (OK=85     KO=-     )
> response time 50th percentile                        236 (OK=236    KO=-     )
> response time 75th percentile                        263 (OK=263    KO=-     )
> response time 95th percentile                        288 (OK=288    KO=-     )
> response time 99th percentile                        316 (OK=316    KO=-     )
> mean requests/sec                                3774.992 (OK=3774.992 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        452999 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

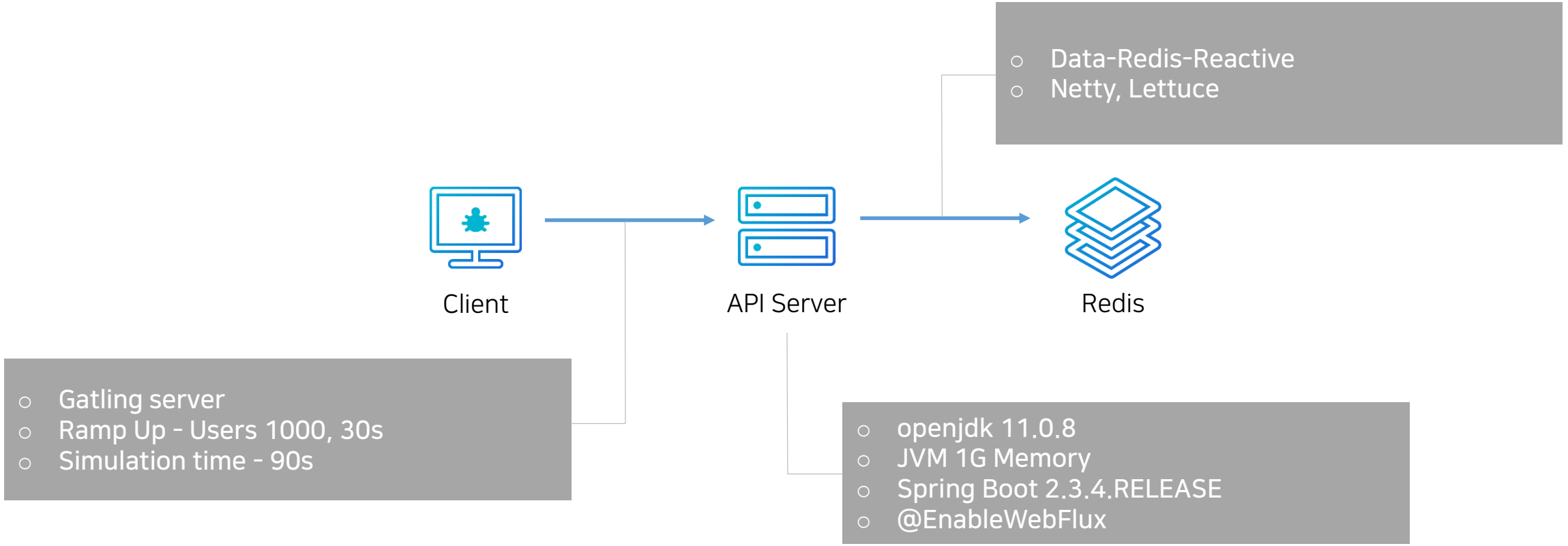
Spring WebMVC + Lettuce 구성




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     452999 (OK=452999 KO=0     )
> min response time                                      1 (OK=1      KO=-     )
> max response time                                    442 (OK=442    KO=-     )
> mean response time                                   199 (OK=199    KO=-     )
> std deviation                                         85 (OK=85     KO=-     )
> response time 50th percentile                        236 (OK=236    KO=-     )
> response time 75th percentile                        263 (OK=263    KO=-     )
> response time 95th percentile                        288 (OK=288    KO=-     )
> response time 99th percentile                        316 (OK=316    KO=-     )
> mean requests/sec                                3774.992 (OK=3774.992 KO=-    )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        452999 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

Spring Web Flux + Reactive Redis 구성



나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

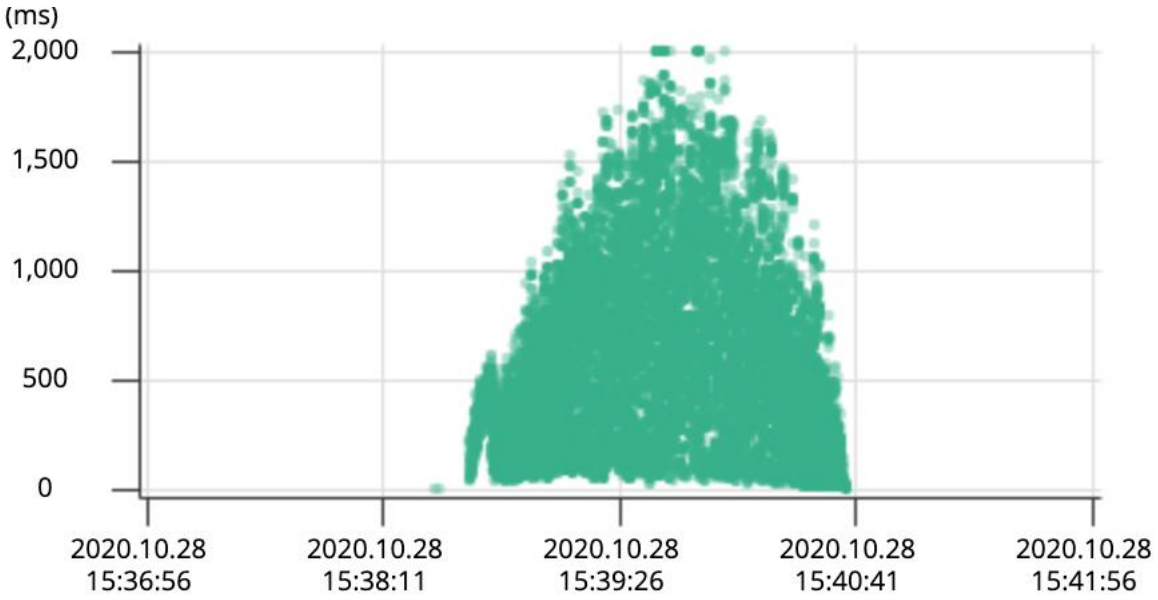
```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                .log()
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

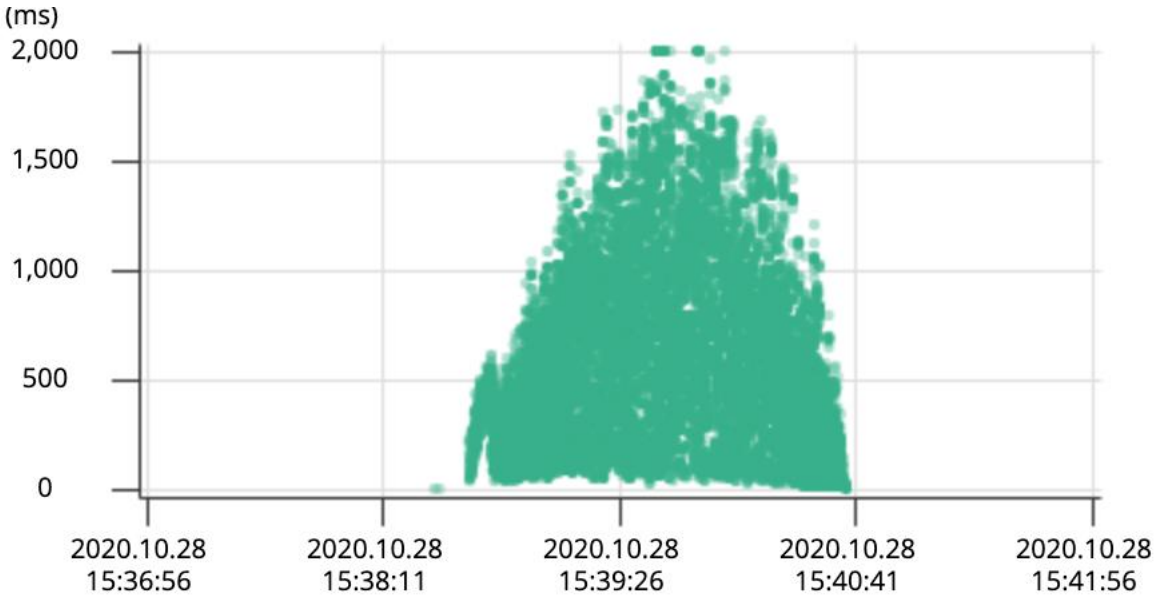
WebFlux 성능 측정 (1/4)




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                      84752 (OK=84752  KO=0     )
> min response time                                      2 (OK=2      KO=-     )
> max response time                                   4738 (OK=4738   KO=-     )
> mean response time                                  1066 (OK=1066   KO=-     )
> std deviation                                        541 (OK=541    KO=-     )
> response time 50th percentile                       1070 (OK=1070   KO=-     )
> response time 75th percentile                       1411 (OK=1411   KO=-     )
> response time 95th percentile                       1893 (OK=1893   KO=-     )
> response time 99th percentile                       2460 (OK=2460   KO=-     )
> mean requests/sec                                706.267 (OK=706.267 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                         27023 ( 32%)
> 800 ms < t < 1200 ms                               23707 ( 28%)
> t > 1200 ms                                        34022 ( 40%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

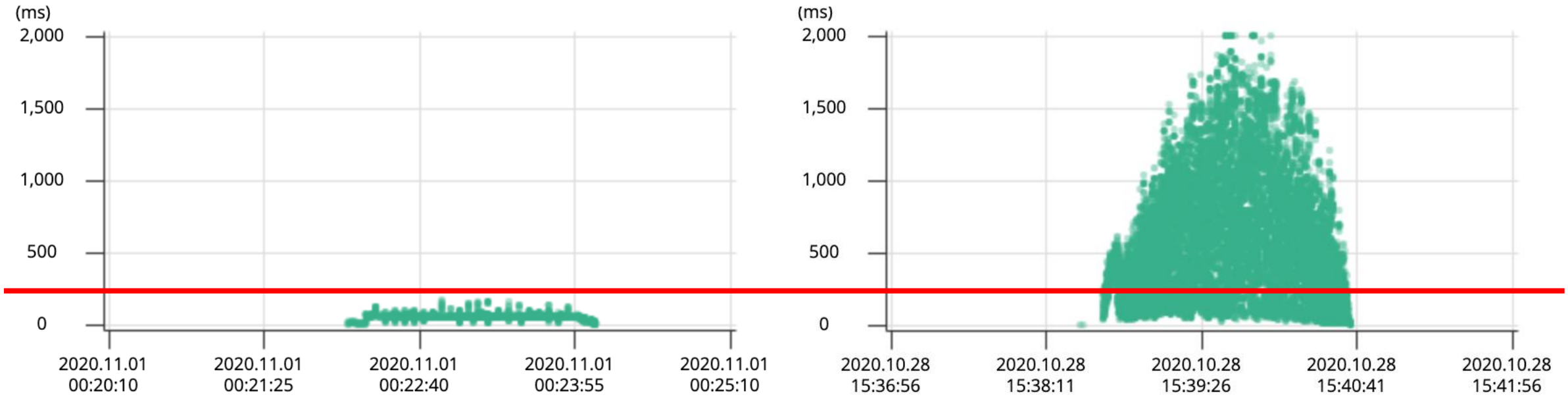
WebFlux 성능 측정 (1/4)




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                      84752 (OK=84752  KO=0     )
> min response time                                      2 (OK=2      KO=-     )
> max response time                                   4738 (OK=4738   KO=-     )
> mean response time                                  1066 (OK=1066   KO=-     )
> std deviation                                        541 (OK=541    KO=-     )
> response time 50th percentile                       1070 (OK=1070   KO=-     )
> response time 75th percentile                       1411 (OK=1411   KO=-     )
> response time 95th percentile                       1893 (OK=1893   KO=-     )
> response time 99th percentile                       2460 (OK=2460   KO=-     )
> mean requests/sec                                706.267 (OK=706.267 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                         27023 ( 32%)
> 800 ms < t < 1200 ms                               23707 ( 28%)
> t > 1200 ms                                        34022 ( 40%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

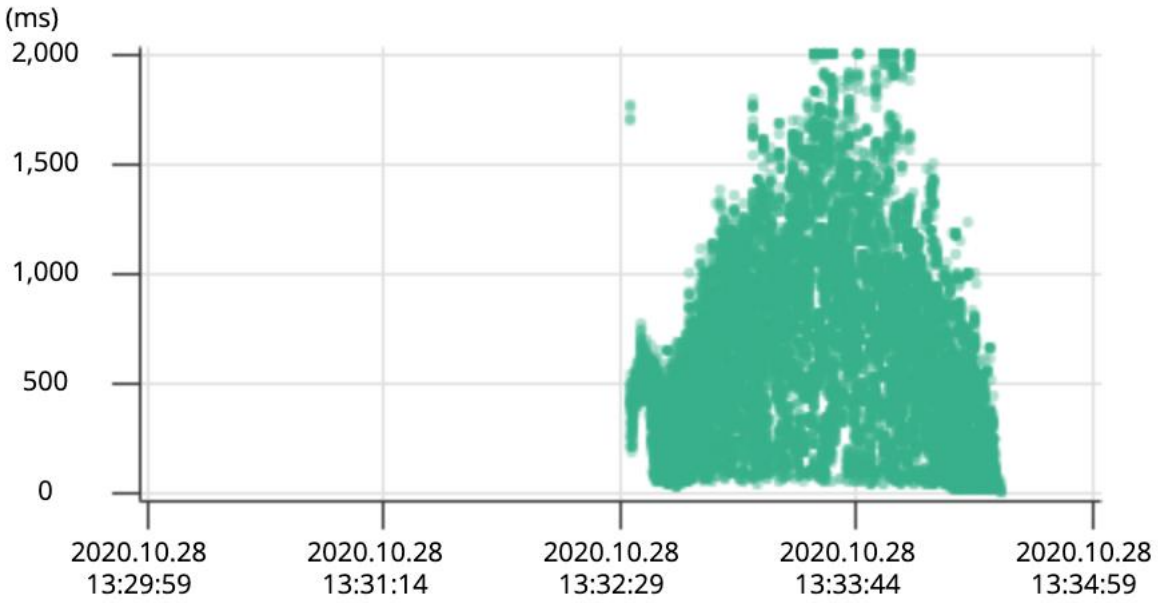
WebMVC vs WebFlux 비교



나의 WebFlux가 느린 이유

성능 측정 2차

- LogBack의 AsyncAppender 적용
- 가끔씩 JVM Hang

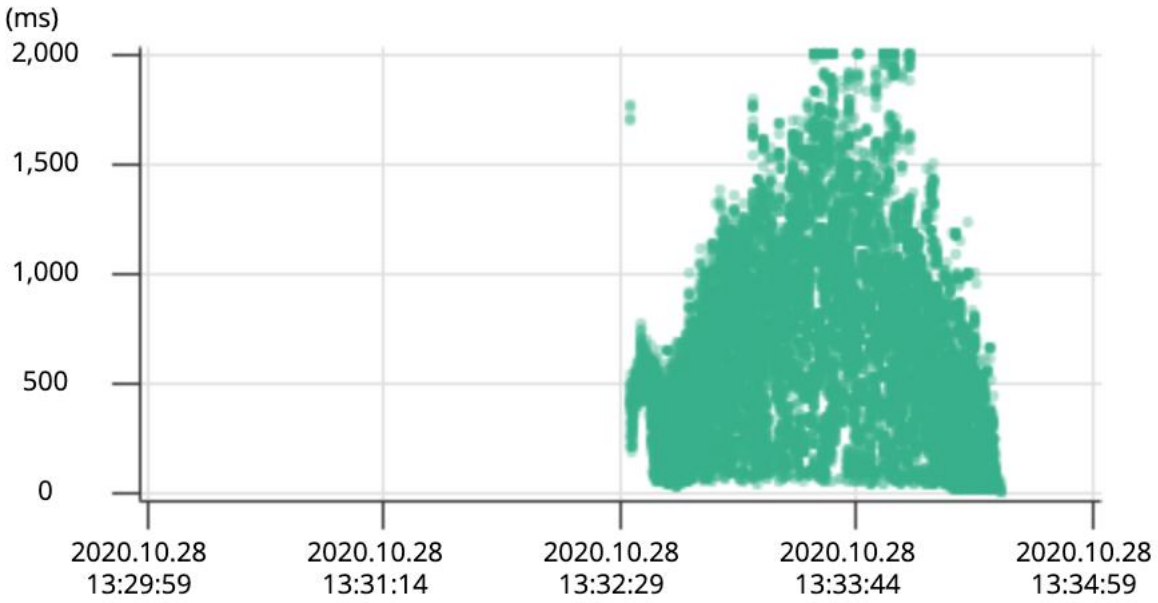



```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                      83172 (OK=83172  KO=0     )
> min response time                                      2 (OK=2      KO=-     )
> max response time                                   5194 (OK=5194   KO=-     )
> mean response time                                  1087 (OK=1087   KO=-     )
> std deviation                                        567 (OK=567    KO=-     )
> response time 50th percentile                       1107 (OK=1108   KO=-     )
> response time 75th percentile                       1409 (OK=1409   KO=-     )
> response time 95th percentile                       2014 (OK=2013   KO=-     )
> response time 99th percentile                       2722 (OK=2722   KO=-     )
> mean requests/sec                                693.1 (OK=693.1  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                         26982 ( 32%)
> 800 ms < t < 1200 ms                               21830 ( 26%)
> t > 1200 ms                                        34360 ( 41%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

성능 측정 2차

- LogBack의 AsyncAppender 적용
- 가끔씩 JVM Hang




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                      83172 (OK=83172  KO=0     )
> min response time                                      2 (OK=2      KO=-     )
> max response time                                   5194 (OK=5194   KO=-     )
> mean response time                                  1087 (OK=1087   KO=-     )
> std deviation                                        567 (OK=567    KO=-     )
> response time 50th percentile                       1107 (OK=1108   KO=-     )
> response time 95th percentile                       2014 (OK=2013   KO=-     )
> response time 99th percentile                       2722 (OK=2722   KO=-     )
> mean requests/sec                                693.1 (OK=693.1  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                         26982 ( 32%)
> 800 ms < t < 1200 ms                               21830 ( 26%)
> t > 1200 ms                                        34360 ( 41%)
> failed                                                 0 (  0%)
================================================================================
```

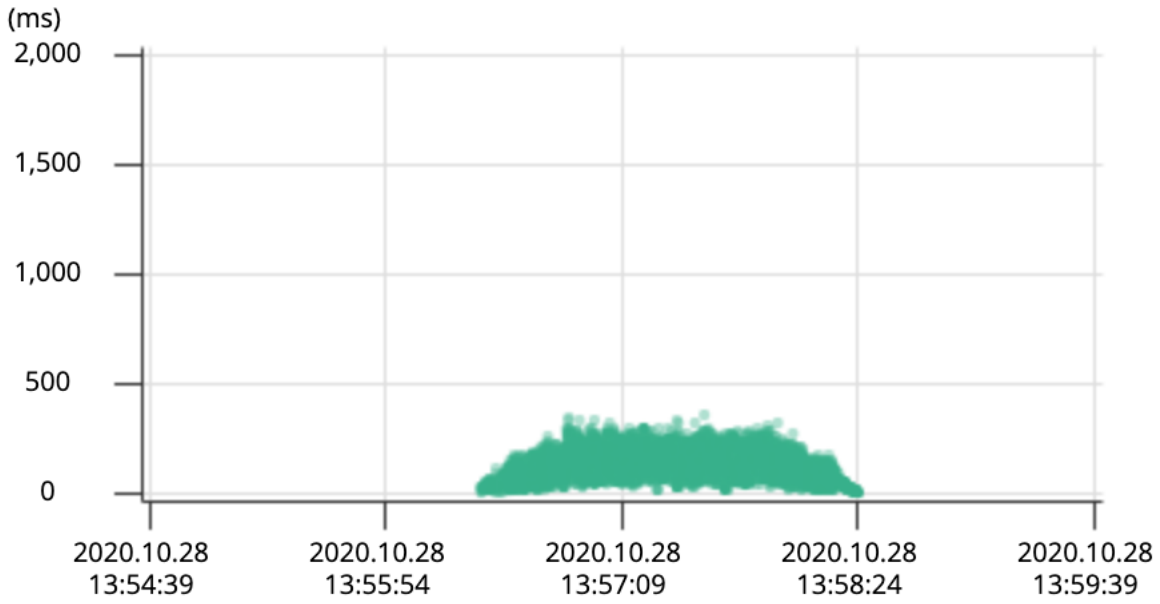
나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {
        return serverRequest.bodyToMono(AdRequest.class)
                //.log() <- 제거
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

성능 측정 3차

- log() 제거 및 RollingFileAppender로 변경

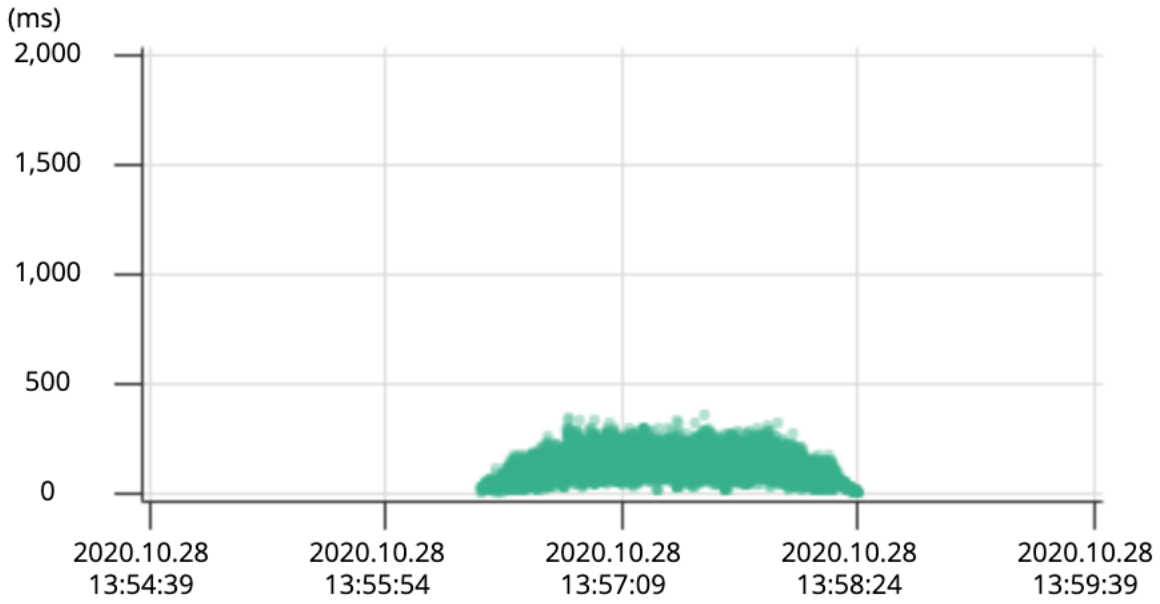



```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     534428 (OK=534428 KO=0     )
> min response time                                      0 (OK=0      KO=-     )
> max response time                                    673 (OK=673    KO=-     )
> mean response time                                   168 (OK=168    KO=-     )
> std deviation                                         87 (OK=87     KO=-     )
> response time 50th percentile                        176 (OK=176    KO=-     )
> response time 75th percentile                        213 (OK=213    KO=-     )
> response time 95th percentile                        345 (OK=345    KO=-     )
> response time 99th percentile                        397 (OK=397    KO=-     )
> mean requests/sec                               4453.567 (OK=4453.567 KO=-    )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        534428 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

성능 측정 3차

- log() 제거 및 RollingFileAppender로 변경




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     534428 (OK=534428 KO=0     )
> min response time                                      0 (OK=0      KO=-     )
> max response time                                    673 (OK=673    KO=-     )
> mean response time                                   168 (OK=168    KO=-     )
> std deviation                                         87 (OK=87     KO=-     )
> response time 50th percentile                        176 (OK=176    KO=-     )
> response time 75th percentile                        213 (OK=213    KO=-     )
> response time 95th percentile                        345 (OK=345    KO=-     )
> response time 99th percentile                        397 (OK=397    KO=-     )
> mean requests/sec                                4453.567 (OK=4453.567 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        534428 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

성능 측정 3차

- map()과 flatMap() 의 차이점
  - map () - Transform the item emitted by this Mono by applying a synchronous function to it.
  - flatMap() - Transform the item emitted by this Mono asynchronously, returning the value emitted by another Mono (possibly changing the value type).
- 너무 많은 map() 메서드 조합
  - map() 연산마다 객체를 생성한다.

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {

        return serverRequest.bodyToMono(AdRequest.class)
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {

        return serverRequest.bodyToMono(AdRequest.class)
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {

        return serverRequest.bodyToMono(AdRequest.class)
                .map(AdRequest::getCode)
                .map(AdCodeId::of)
                .map(adCodeId -> {
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .map(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId))
                .flatMap(adValue ->
                        ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                                .body(adValue, adValue.class)
                );
    }
}
```

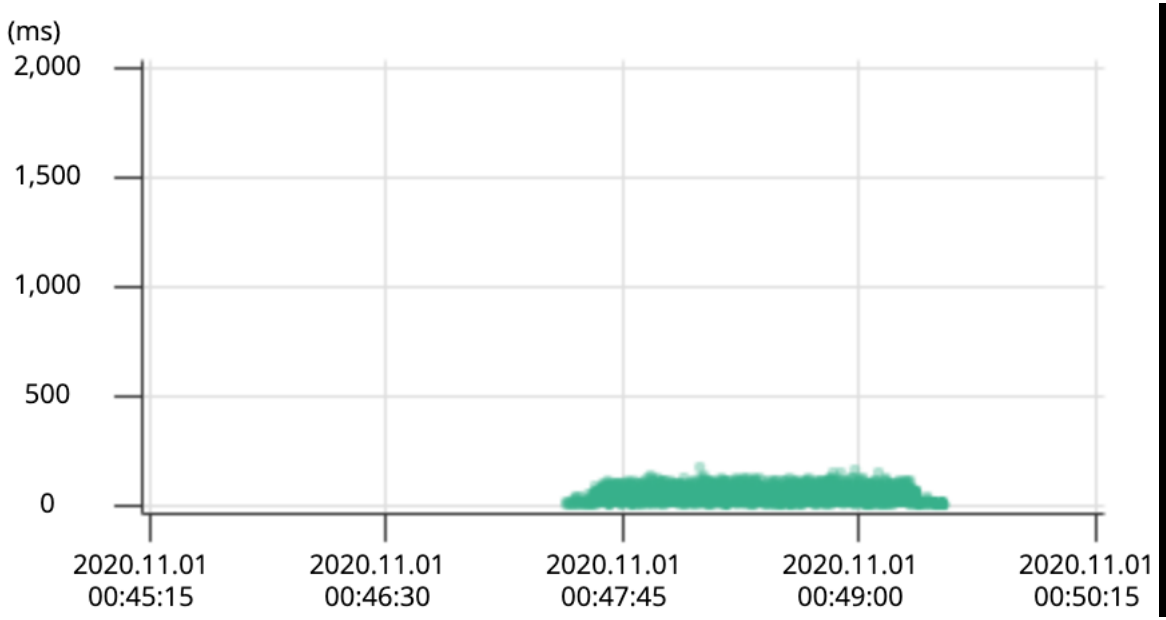
나의 WebFlux가 느린 이유

```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {

        Mono<AdValue> adValueMono = serverRequest.bodyToMono(AdRequest.class)
                .map(adRequest -> {
                    AdCodeId adCodeId = AdCodeId.of(AdRequest.getCode());
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .flatMap(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId));

        return ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                .body(adValueMono, AdValue.class);
    }
}
```

나의 WebFlux가 느린 이유

성능측정 4차

- map(), flatMap() 수정

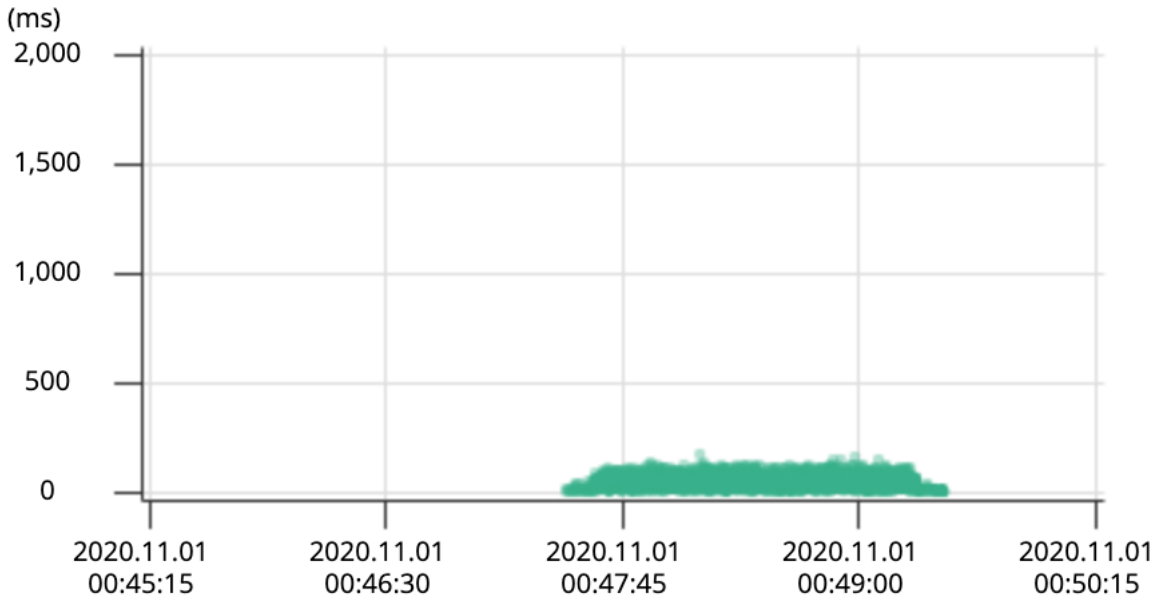



```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     717349 (OK=717349 KO=0     )
> min response time                                      0 (OK=0      KO=-     )
> max response time                                    384 (OK=384    KO=-     )
> mean response time                                   125 (OK=125    KO=-     )
> std deviation                                         64 (OK=64     KO=-     )
> response time 50th percentile                        124 (OK=125    KO=-     )
> response time 75th percentile                        169 (OK=169    KO=-     )
> response time 95th percentile                        227 (OK=227    KO=-     )
> response time 99th percentile                        267 (OK=267    KO=-     )
> mean requests/sec                               5977.908 (OK=5977.908 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        717349 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

나의 WebFlux가 느린 이유

성능측정 4차

- map(), flatMap() 수정




```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                     717349 (OK=717349 KO=0     )
> min response time                                      0 (OK=0      KO=-     )
> max response time                                    384 (OK=384    KO=-     )
> mean response time                                   125 (OK=125    KO=-     )
> std deviation                                         64 (OK=64     KO=-     )
> response time 50th percentile                        124 (OK=125    KO=-     )
> response time 75th percentile                        169 (OK=169    KO=-     )
> response time 95th percentile                        227 (OK=227    KO=-     )
> response time 99th percentile                        267 (OK=267    KO=-     )
> mean requests/sec                                5977.908 (OK=5977.908 KO=-    )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                        717349 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

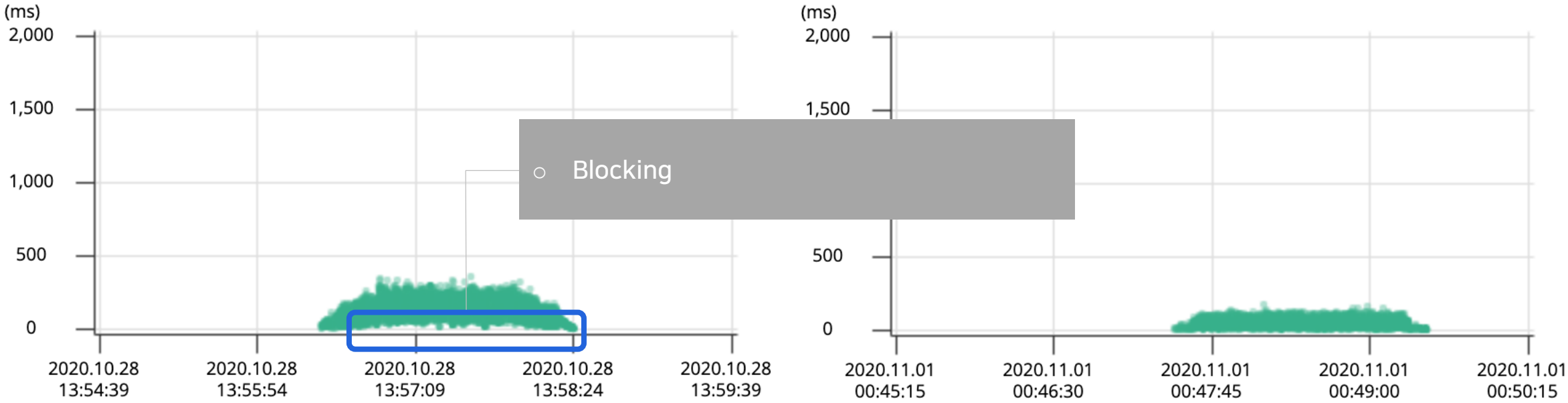
나의 WebFlux가 느린 이유

정리

	TPS	Response Time 95%
WebFlux 1차 – log() 메서드	706.3 tps	1,893 ms
WebFlux 2차 – AsyncAppender	693.1 tps	2,014 ms
Spring WebMVC	**3744.9 tps**	**288 ms**
WebFlux 3차 – log() 메서드 제거	4453.5 tps	345 ms
WebFlux 4차 – flatmap, map 리펙토링	5977.9 tps	227 ms

나의 WebFlux가 느린 이유

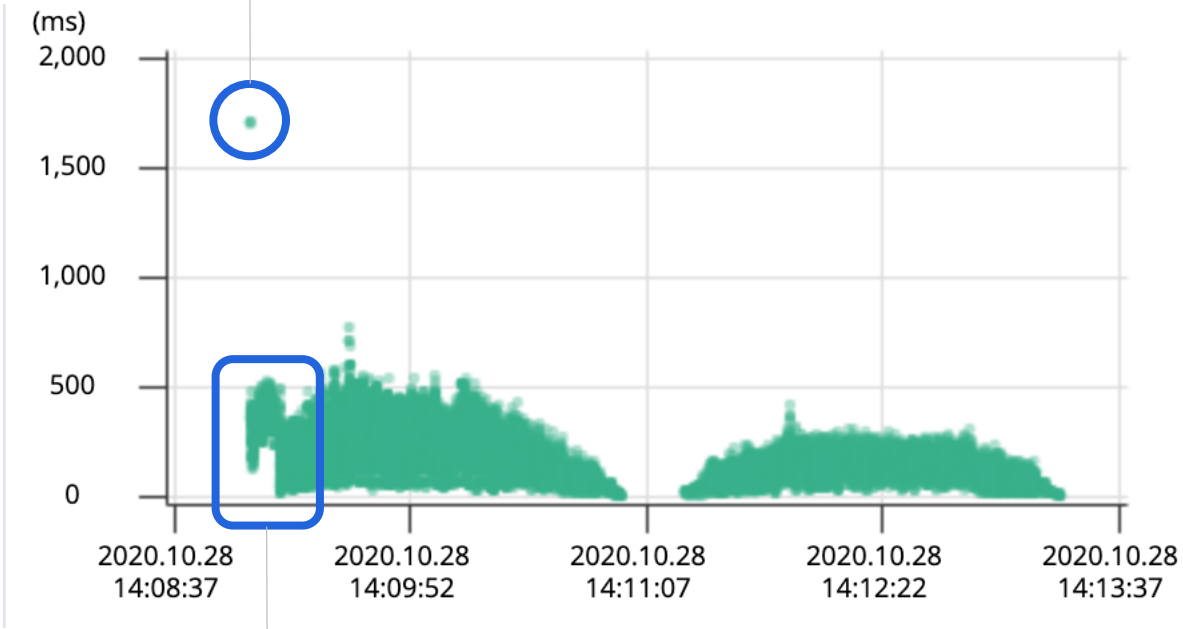
성능측정 4차 고찰



나의 WebFlux가 느린 이유

성능측정 4차 고찰

- Lettuce Lazy Connection



- JIT Compiler

나의 WebFlux가 느린 이유

다른 성능 개선 사항

- BlockHound
- Lettuce 설정
- Avoiding Reactor Meltdown (from InfoQ)

개선 사항

BlockHound

- Blocking 코드를 찾아주는 라이브러리
- reactor-core 3.3.0 부터 내장
- block(), blockFirst(), blockLast() 같은 메서드를 사용하면 예외 발생

```
java.lang.IllegalStateException Create breakpoint : block()/blockFirst()/blockLast() are blocking, which is not supported in thread reactor-http-nio-2
    at reactor.core.publisher.BlockingSingleSubscriber.blockingGet(BlockingSingleSubscriber.java:83)
    Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException:
Error has been observed at the following site(s):
    |_ checkpoint ⇢ HTTP POST "/v1/hotels/fetch-by-code" [ExceptionHandlingWebHandler]
Stack trace:
        at reactor.core.publisher.BlockingSingleSubscriber.blockingGet(BlockingSingleSubscriber.java:83)
        at reactor.core.publisher.Mono.block(Mono.java:1680)
        at com.springtour.demo.hotel.HotelHandler.fetchByHotelRequest(HotelHandler.java:25)
        at org.springframework.web.reactive.function.server.support.HandlerFunctionAdapter.handle(HandlerFunctionAdapter.java:61)
        at org.springframework.web.reactive.DispatcherHandler.invokeHandler(DispatcherHandler.java:161)
```

개선 사항

Lettuce 설정

- Connection validation 시, synchronous 로 동작.
- Command 실행 마다 ping command 를 synchronous 로 실행.
- 성능 하락의 원인

개선 사항

```
@Bean(name = "redisConnectionFactory")
public ReactiveRedisConnectionFactory connectionFactory() {
  // 중략

  RedisStandaloneConfiguration redisConfig
                  = new RedisStandaloneConfiguration(redisHost,
Integer.valueOf(redisPort));

  LettuceConnectionFactory factory
                  = new LettuceConnectionFactory(redisConfig, clientConfig);

  factory.setValidateConnection(true);
  factory.setShareNativeConnection(true);

  return factory;
}
```

개선 사항

```
public class LettuceConnectionFactory {

  void validateConnection() {
    // 중략
    synchronized(this.connectionMonitor) {
    boolean valid = false;
    if (this.connection != null && this.connection.isOpen()) {
      try {
        if (this.connection instanceof StatefulRedisConnection) {
          ((StatefulRedisConnection)this.connection).sync().ping();
        }
        // 중략
      }
  }
}
```

개선 사항

```
public class StatefulRedisConnectionImpl<K, V> extends RedisChannelHandler<K, V> implements
StatefulRedisConnection<K, V> {

    protected final RedisCodec<K, V> codec;
    protected final RedisCommands<K, V> sync;
    protected final RedisAsyncCommandsImpl<K, V> async;
    protected final RedisReactiveCommandsImpl<K, V> reactive;

    // 중략
}
```

개선 사항

Avoiding Reactor Meltdown

- Reactor Meltdown
  - Event Loop 의 Thread들이 Blocking API 때문에 Reactor 시스템이 Hang 걸리는 현상
  - Blocking API 를 위한 별도의 Thread Pool로 격리시키는 방법
  - subcribeOn(), publishOn()

나의 WebFlux가 느린 이유


```
public class AdHandler {
    public Mono<ServerResponse> fetchByAdRequest(ServerRequest serverRequest) {

        Mono<AdValue> adValueMono = serverRequest.bodyToMono(AdRequest.class)
                .publishOn(Schedulers.boundedElastic())
                .map(adRequest -> {
                    AdCodeId adCodeId = AdCodeId.of(AdRequest.getCode());
                    log.warn("Requested AdCodeId = {}", adCodeId.toKeyString());
                    return adCodeId;
                })
                .flatMap(adCodeId -> cacheStorageAdapter.getAdValue(adCodeId));

        return ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
                .body(adValueMono, AdValue.class);
    }
}
```

고맙습니다.

NHN FORWARD