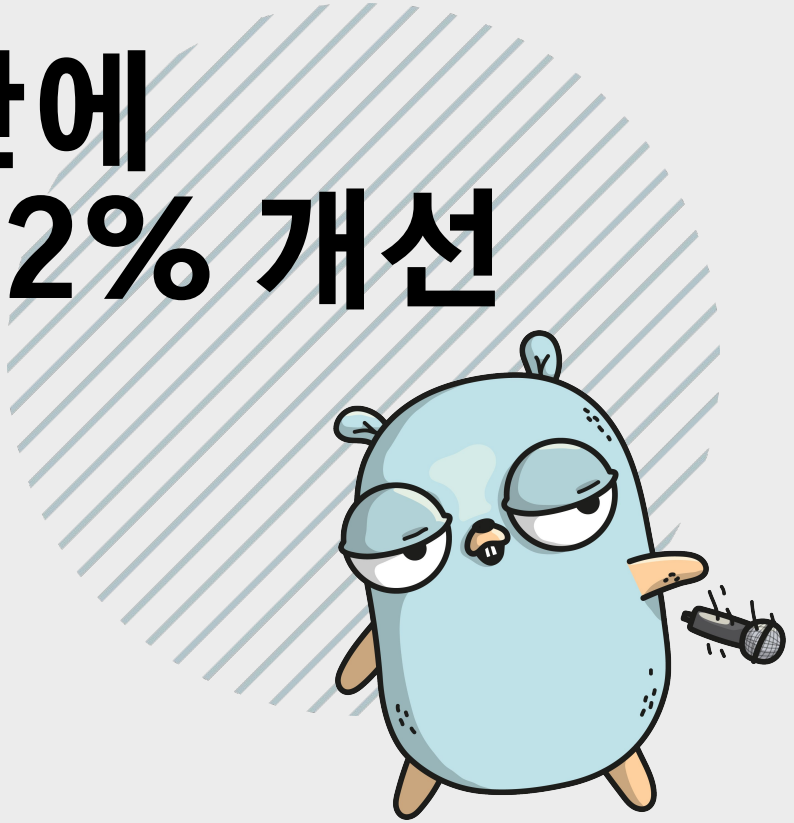
Golang으로 4일 만에 Image 서버 성능 72% 개선



백경준

백패커(아이디어스, 텀블벅)

Speaker

백경준

백패커 DevOps Engineer

백엔드 개발자로 커리어를 시작하였고,
현재 **백패커**에서 **DevOps, Data Engineer**로
일하고 있습니다.

평소 오픈소스에 관심이 많으며
Golang Korea 운영진으로 활동하면서
개발 커뮤니티의 성장을 도모합니다.

paikend

Backpackr

Global No.1
Creator Ecosystem

이런 분들에게 도움이 될 거예요

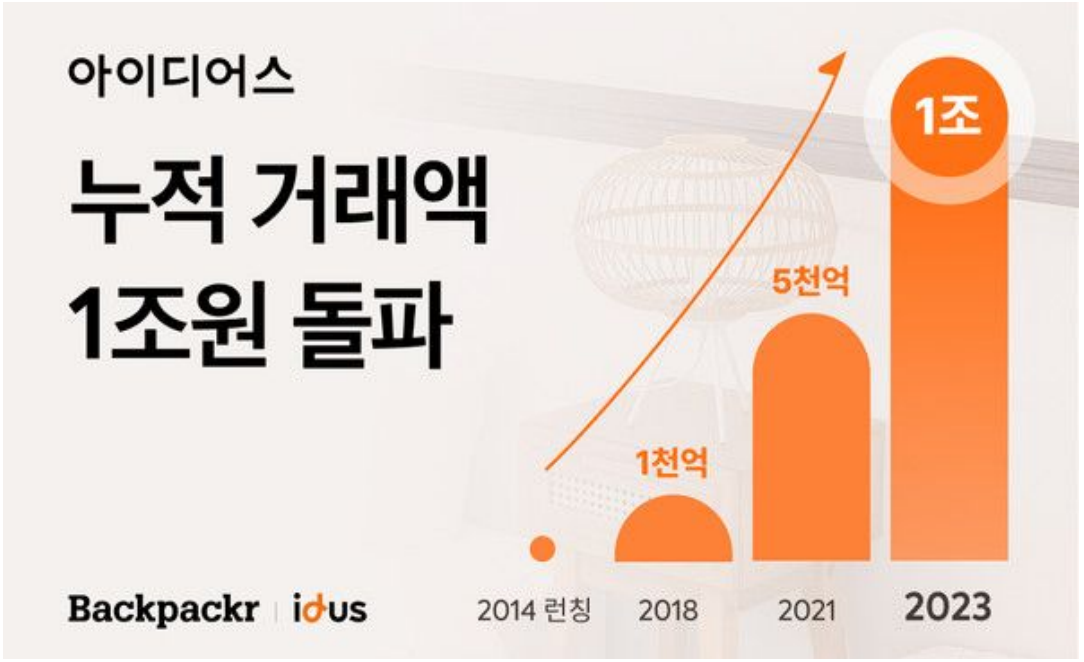
- 레거시 서비스를 빠르게 **Golang으로 마이그레이션**하고 싶으신 분
- Golang의 장점은 알지만 **실제 적용**에 어려움을 겪고 계신 분
- Go 애플리케이션의 **이미지 처리와 컨테이너 환경 운영**에 관심 있는 분

목차

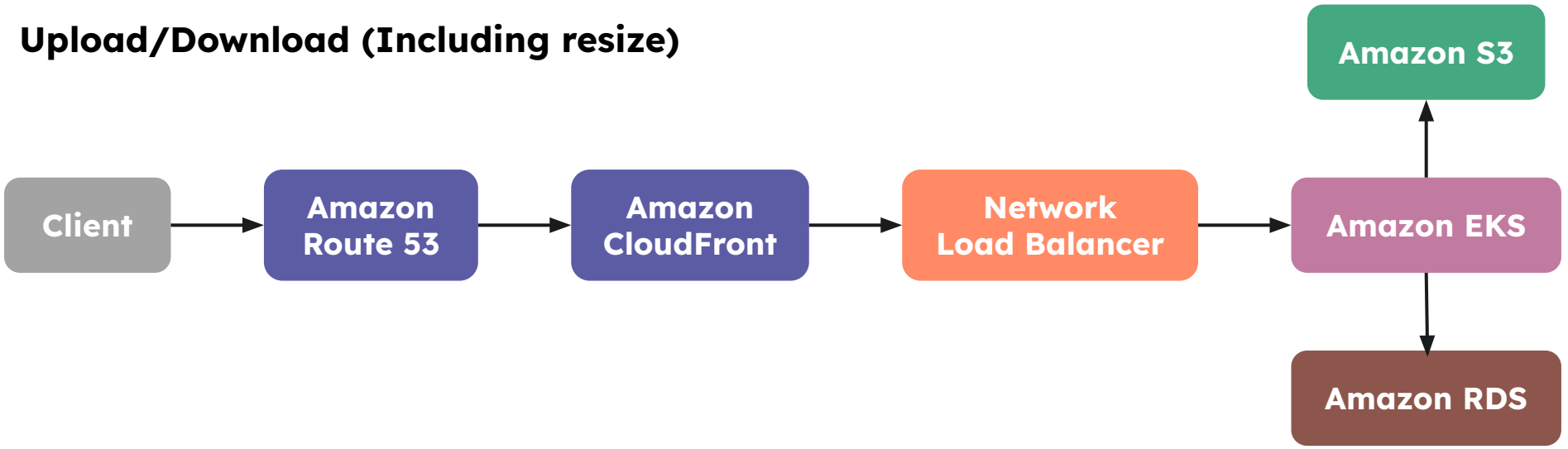
1. **Image 서버를 개선하게 된 배경**
2. **Image 서버 개선 작업**
   - 문제 정의 및 목표
   - 기술스택 정하기
   - 테스트하기
   - 배포하기
   - 성능 개선 결과
   - Golang 컨테이너 환경 최적화 Tip
3. **Image 서버 개선 이후의 이야기**
   - 앞으로의 과제
   - Image 서버 아키텍처 재설계
4. **마무리**

Image 서버를 개선하게 된 배경

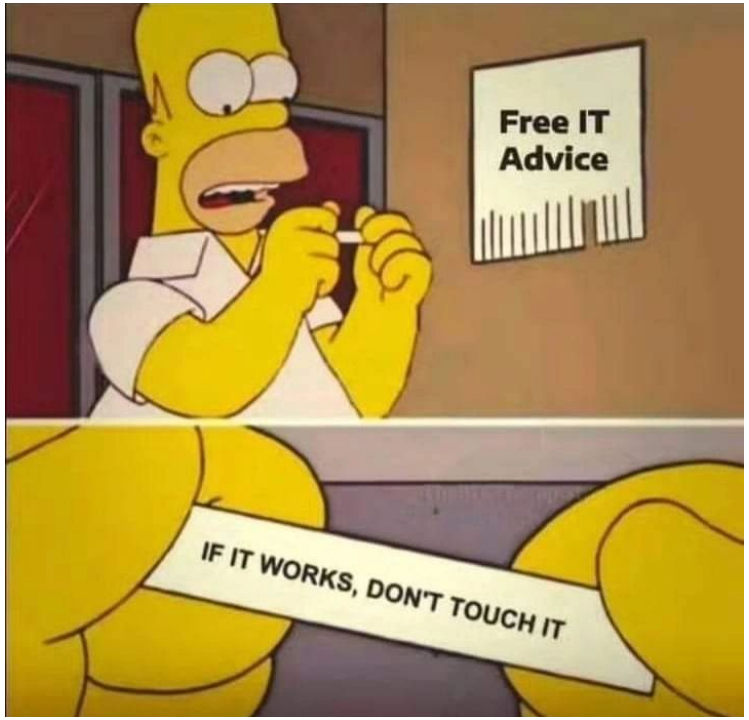
빠르게 성장한 아이디어스



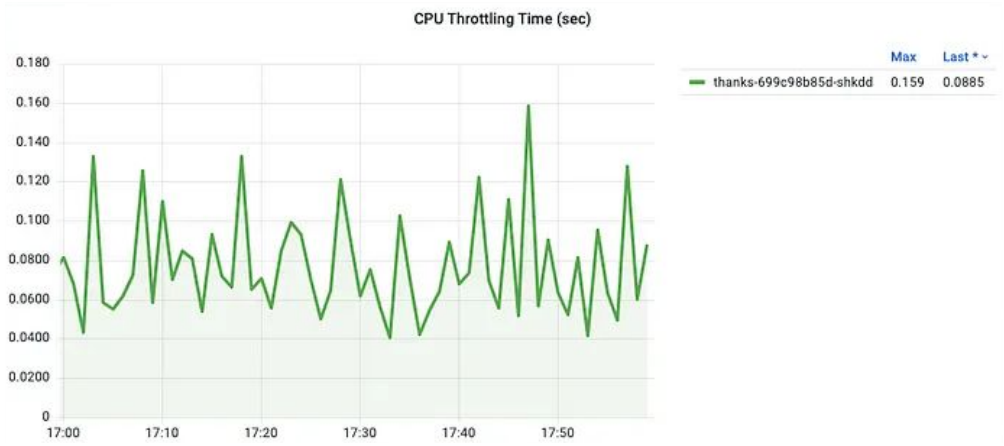
재설계가 필요한 Image 서버 구조

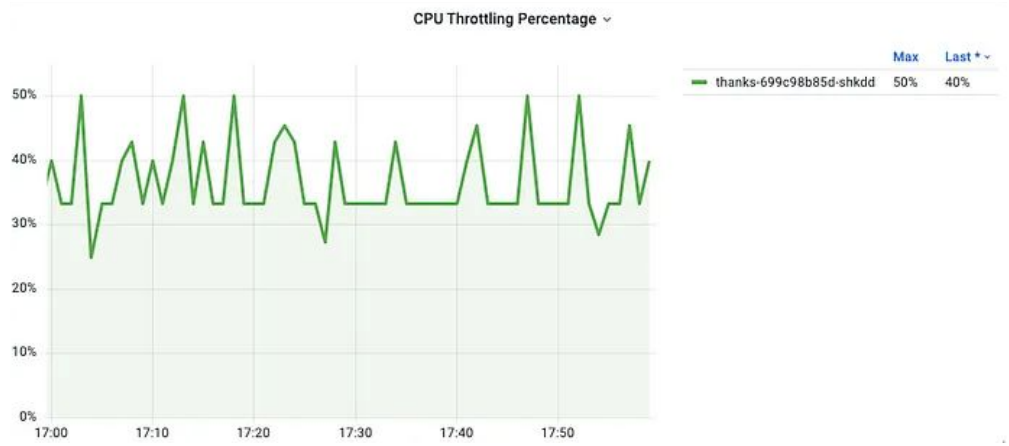


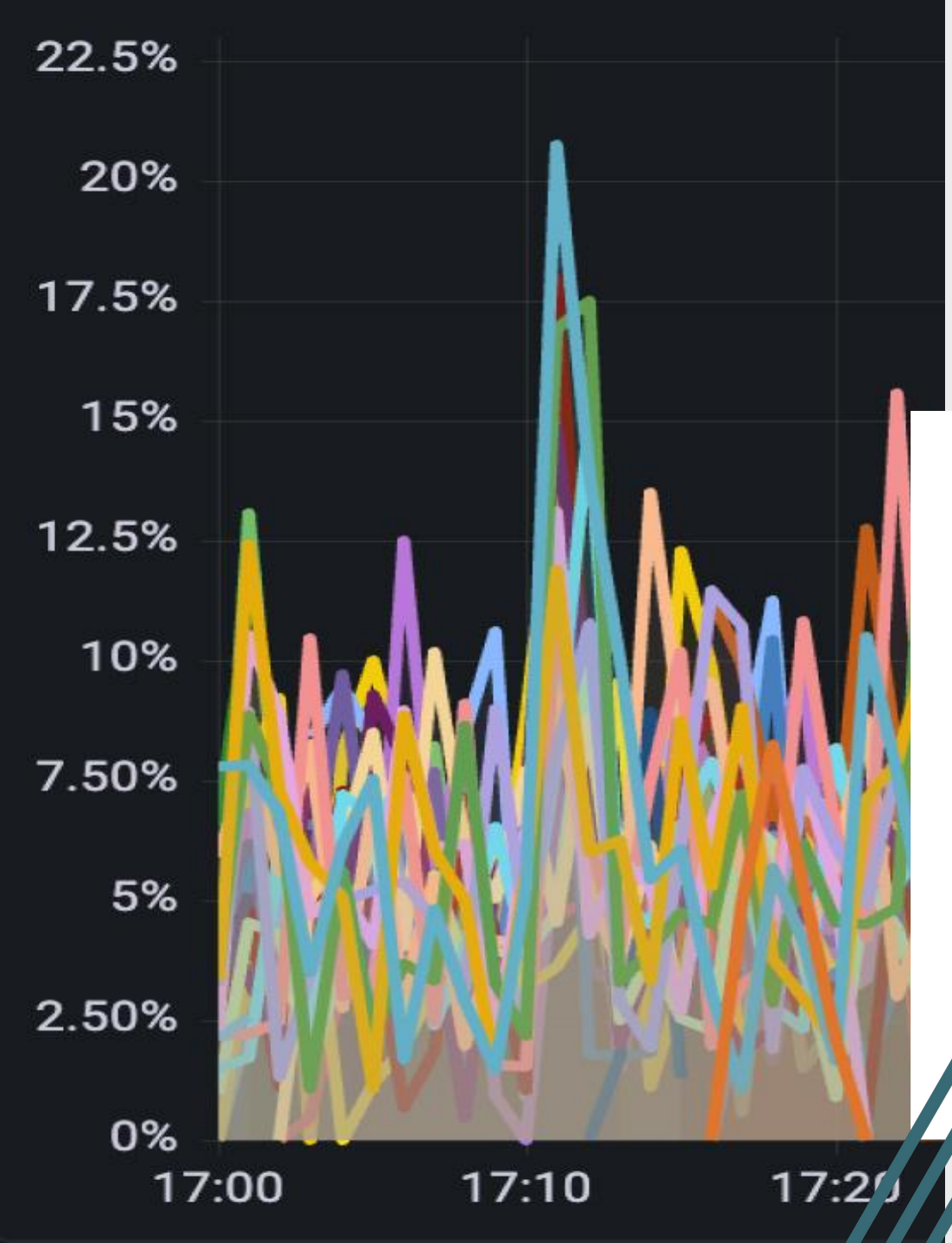
그래도 잘 돌아가니까..



간헐적으로 발생하는 CPU Throttling







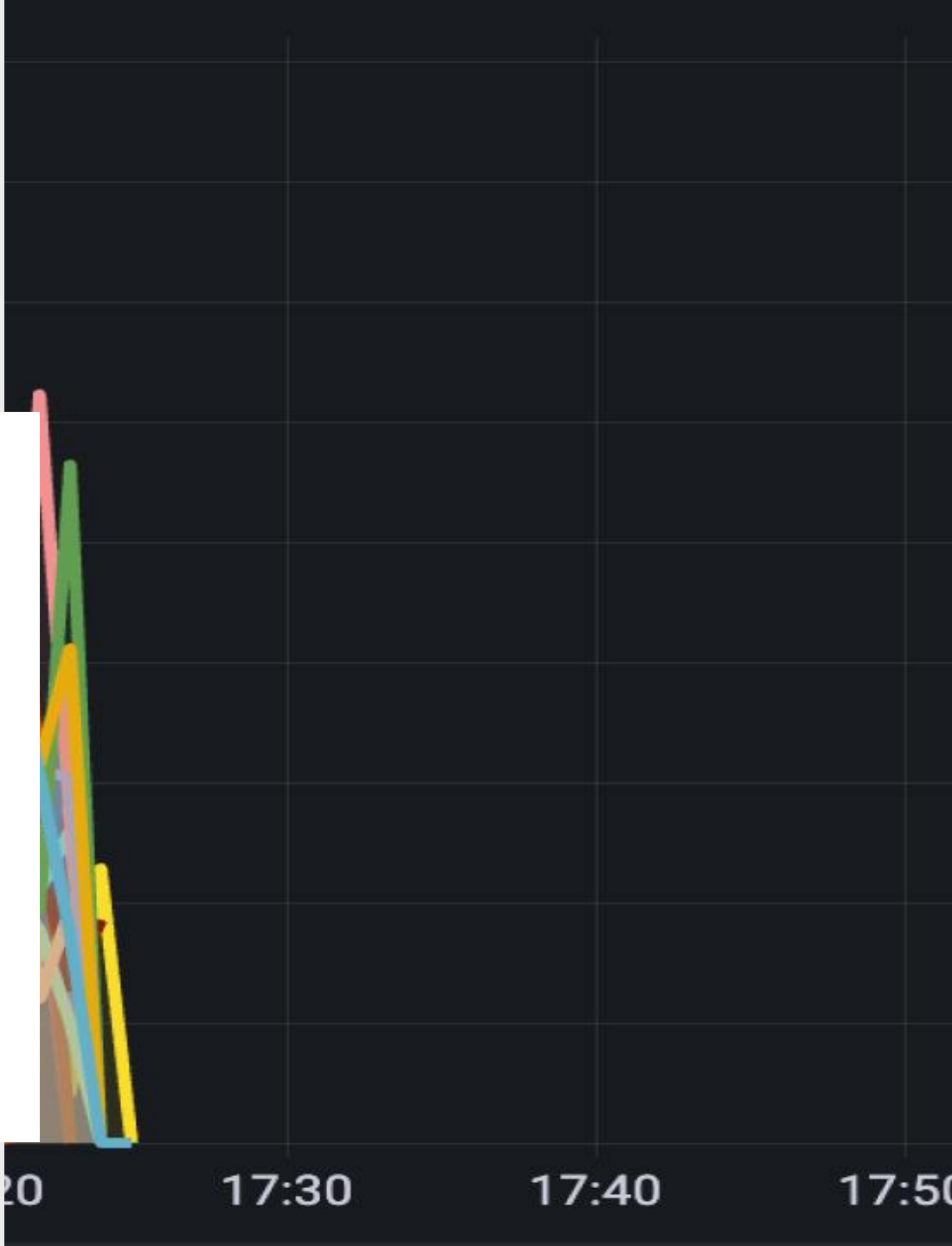
발생 원인

- **CDN cache 정책 만료**로 인해
  Image resize API 요청이 급격히 증가
- 초기 서비스 대비 전반적인 **요청량 증가**
- 고화질 이미지 **요청량 증가**

해결 방법

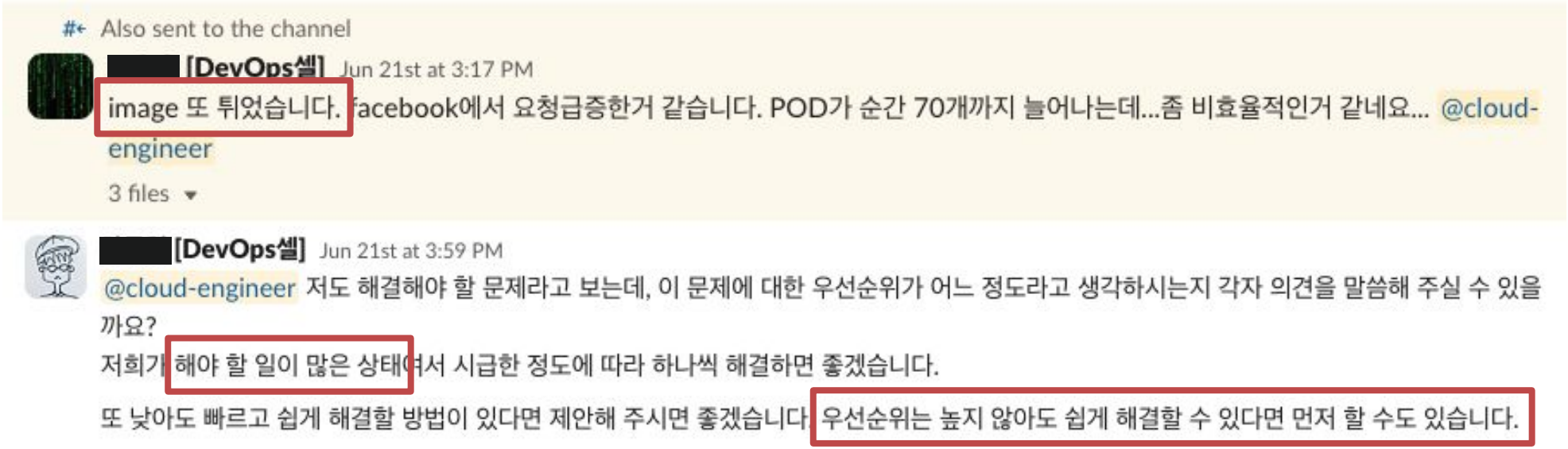
- **Overprovisioning** 인프라 설정
- CPU Limit 변경(1500m → **제한없음**)

백패커팀 테크블로그 - Kubernetes Pod Resources: CPU Limit



여전히 잘 돌아가니까..

나야, Image 서버 👋


Also sent to the channel
[DevOps셀] Jun 21st at 3:17 PM
image 또 튀었습니다. facebook에서 요청급증한거 같습니다. POD가 순간 70개까지 늘어나는데...좀 비효율적인거 같네요... @cloud-engineer
3 files
[DevOps셀] Jun 21st at 3:59 PM
@cloud-engineer 저도 해결해야 할 문제라고 보는데, 이 문제에 대한 우선순위가 어느 정도라고 생각하시는지 각자 의견을 말씀해 주실 수 있을까요?
저희가 해야 할 일이 많은 상태여서 시급한 정도에 따라 하나씩 해결하면 좋겠습니다.
또 낮아도 빠르고 쉽게 해결할 방법이 있다면 제안해 주시면 좋겠습니다. 우선순위는 높지 않아도 쉽게 해결할 수 있다면 먼저 할 수도 있습니다.

쏟아지는 일들의 우선순위 판별하기

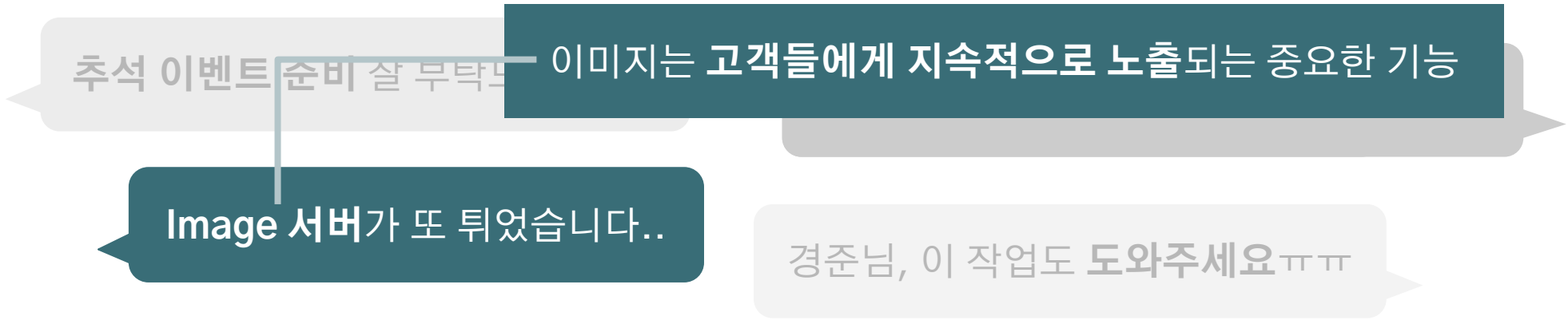
추석 이벤트 준비 잘 부탁드립니다!!

ISMS 보안 결함 대응 꼭 해주셔야 합니다.

Image 서버가 또 튀었습니다..

경준님, 이 작업도 **도와주세요**ㅠㅠ

쏟아지는 일들의 우선순위 판별하기



쉽게 해결할 수 있다고 생각한 누군가 == 나 🤦‍♂️

아이디어스 10주년 기념: New Image 서버

Image 서버 개선 작업

문제 정의

아이디어스 Image 서버에서 **리소스를 비효율적으로 사용**하고 있는데,

최소한의 인력으로 **빠르게 개선**할 수 있는 방법은 무엇인가?

목표

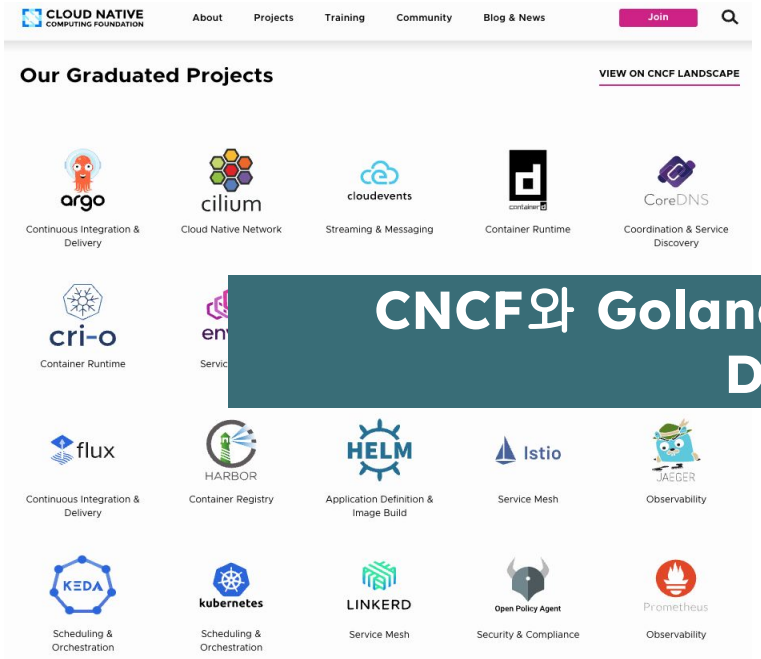
- **Image 서버 성능 개선하기**
  - 리소스(CPU, Memory) 사용률 2배 이상 개선하기
  - 기존 구성을 유지하여 애플리케이션 개발 디펜던시 제거하기

조건

- **성능이 우수할 것**
  - 현재 서버보다 우수한 성능이어야 함
- **유지보수 담당인 데브옵스팀원들에게 익숙한 언어일 것**
  - Golang: 데브옵스팀에게 익숙한 언어 / Java: 개발팀에게 익숙한 언어
- **커뮤니티 성숙도가 높을 것**
  - 진입장벽이 낮아 누구나 유지보수 할 수 있어야 함

기술스택 정하기

데브옵스팀에게 익숙한 Golang



CNCF와 Golang에 관심이 많은 백패커 DevOps팀

기술스택 정하기

우수한 성능의 Golang

특징	PHP	Golang	비교
동시성 실행 모델	단일 스레드, 블로킹 모델	**경량 스레드, 고루틴 사용**	**Golang** > PHP
메모리 사용량	높음	**낮음**	**Golang** > PHP
실행 속도	느림	**빠름**	**Golang** > PHP

기술스택 정하기

golang 백엔드 프레임워크

프레임워크	성숙도 (Github Star, Fork 수 등)	성능	기타 특징
Gin	**매우 성숙**	보통	**좋은 문서화**
Echo	성숙	보통	다양한 플러그인을 통한 확장 용이성
Fiber	중간	높음	빠른 성능

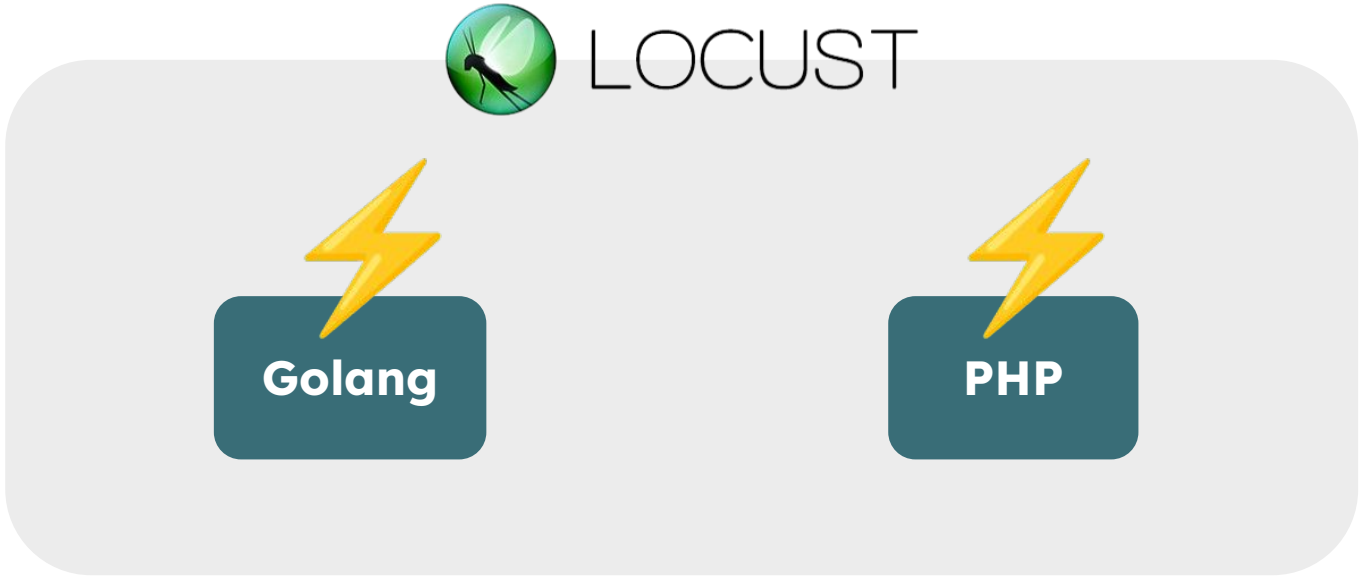
테스트하기

가설 세우기

- **Golang을 사용하면, Image 서버의 속도가 더 빨라질 것이다.**
  - 일반적으로 인터프리터 언어인 PHP보다 컴파일 언어인 Golang의 실행속도가 빠르기 때문
- **Golang을 사용하면, Image 서버의 리소스 사용량이 더 적어질 것이다.**
  - 정적 컴파일과 경량 스레드를 지원하기 때문

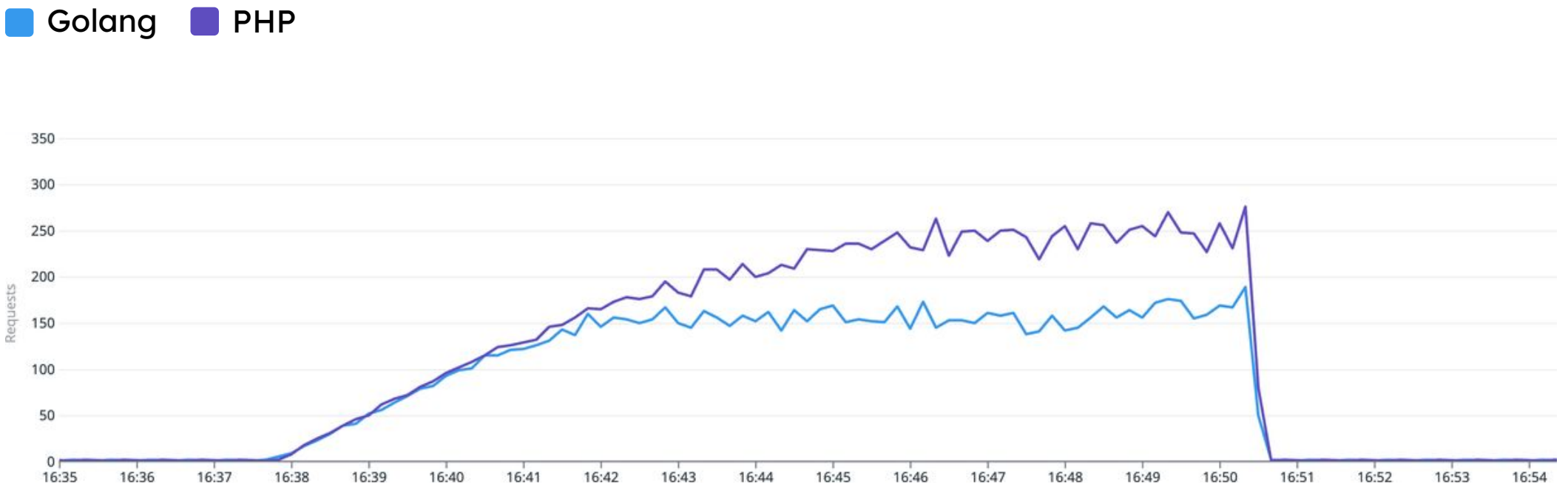
테스트하기

부하 테스트 구성



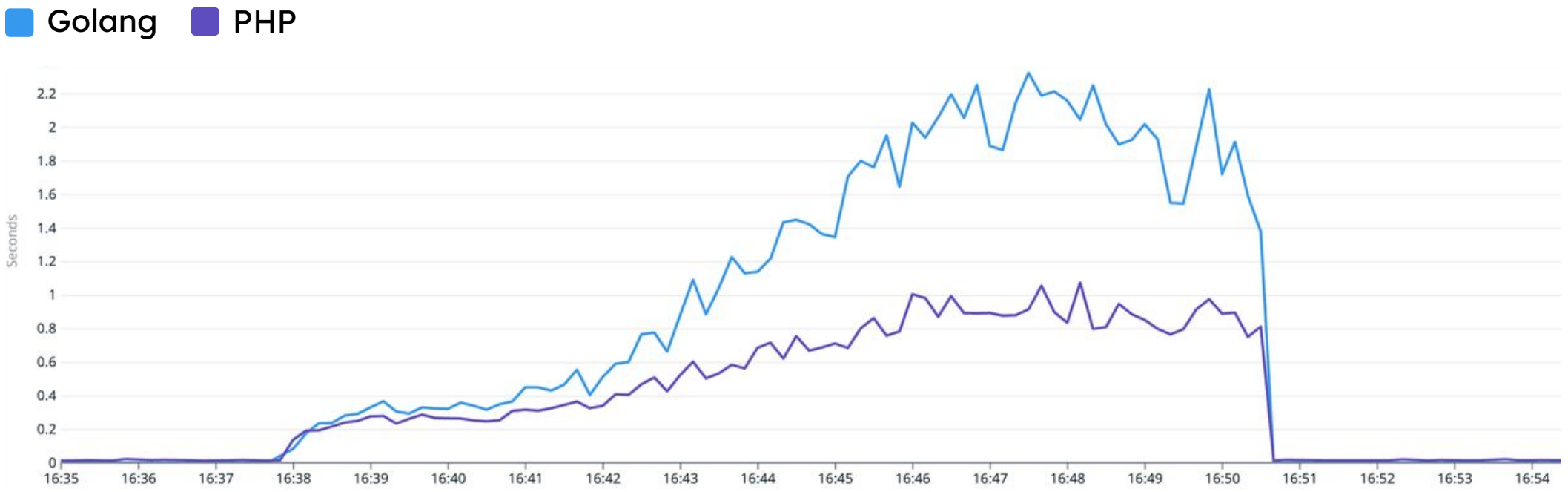
테스트하기

테스트 결과: Requests

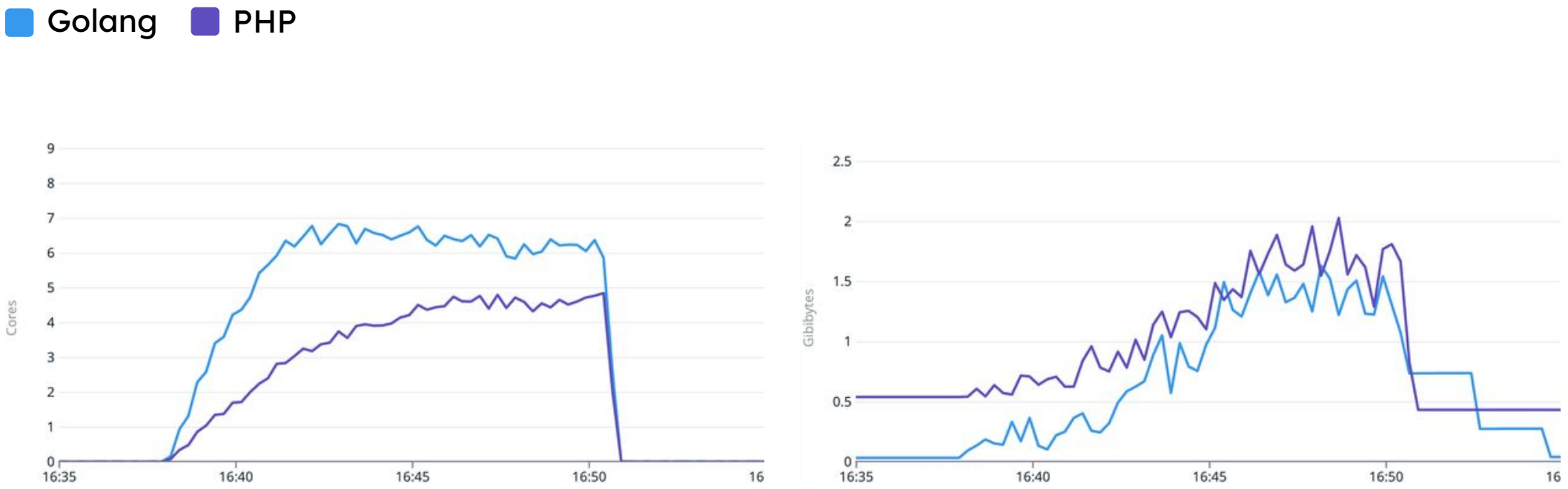


테스트하기

테스트 결과: Latency



테스트 결과: CPU, Memory



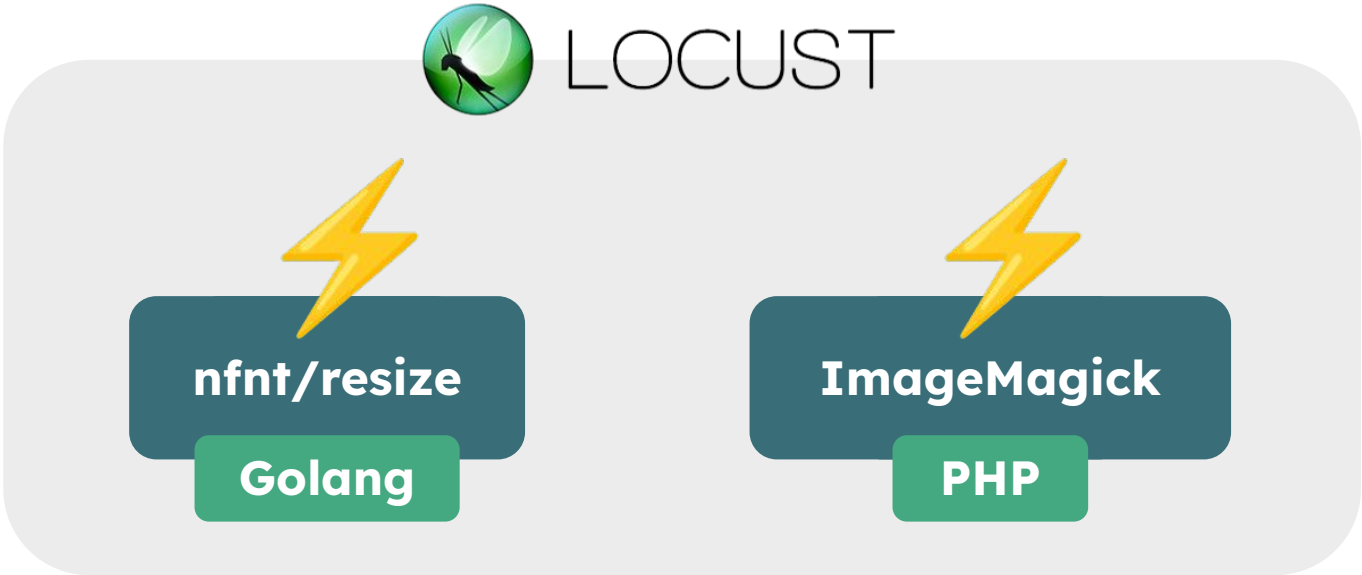
테스트하기

가설 검증 결과

- **Golang을 사용하면, Image 서버의 ~~속도가 더 빨라질 것이다.~~**
  ⇒ **PHP**로 만든 Image 서버의 **속도가 더 빨랐음**
- **Golang을 사용하면, Image 서버의 ~~리소스 사용량이 더 적어질 것이다.~~**
  ⇒ **PHP**로 만든 Image 서버의 **리소스 사용량이 더 적었음**

테스트하기

가설 검증 비교조건 오류



테스트하기

가설 검증 이미지 라이브러리 비교

특징	nfnt/resize (Golang)	ImageMagick (PHP)
주요 언어	Golang	**C**
메모리 사용량	낮음	높음
처리 속도	느림	빠름
대용량 이미지 처리	부적합	**적합**
최적 사용 사례	간단한 리사이징	**고성능 대량 처리**

테스트하기

가설 검증 이미지 라이브러리 비교

특징	nfnt/resize (Golang)	ImageMagick (PHP)	libvips (Golang)
주요 언어	Golang	C	C
메모리 사용량	낮음	높음	매우 낮음
처리 속도	느림	빠름	매우 빠름
대용량 이미지 처리	부적합	적합	매우 적합
최적 사용 사례	간단한 리사이징	고성능 대량 처리	고성능 대량 처리

테스트하기

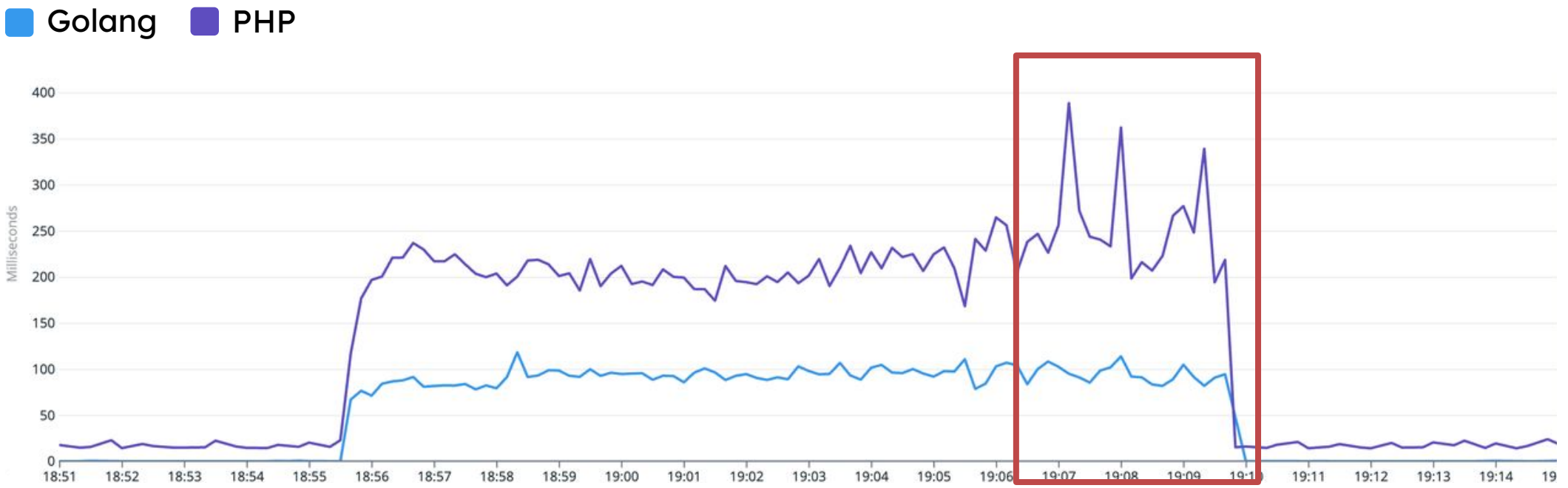
비교 조건을 바꿔서 2차 테스트 진행

테스트하기

2차 테스트 결과: Requests

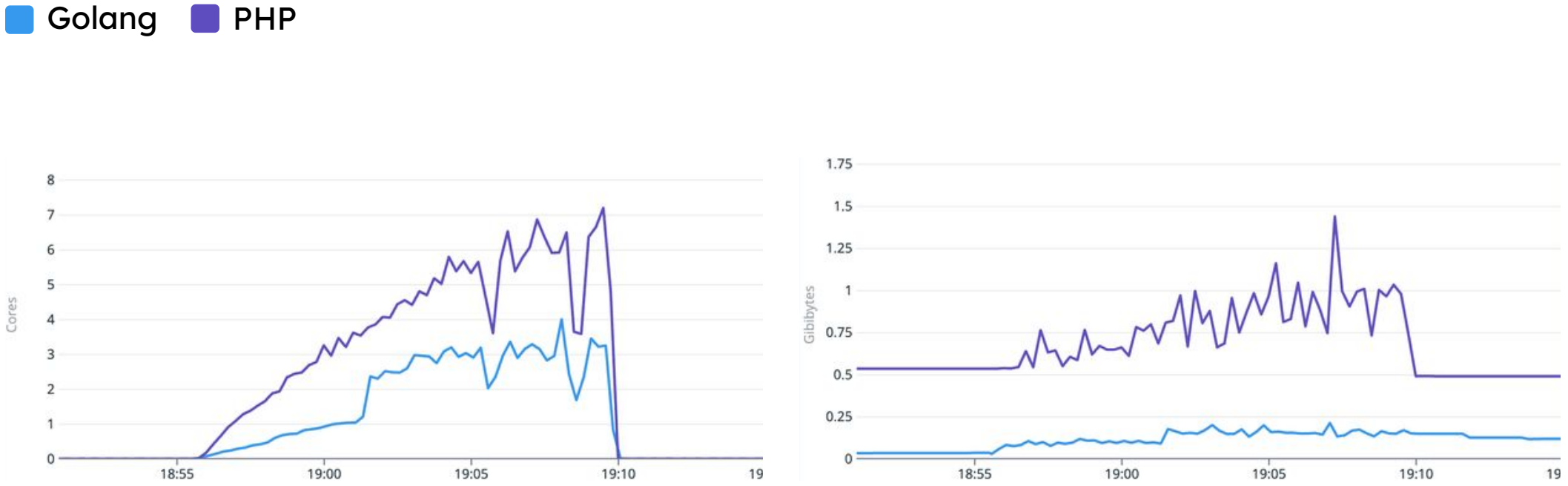
테스트하기

2차 테스트 결과: Latency



테스트하기

2차 테스트 결과: CPU, Memory



테스트하기

2차 테스트 결과

- 전반적으로 기존 PHP의 성능보다 **최적화한 Golang의 성능이 더 좋았음**
- **평균 Latency가 약 50% 감소**했으며, **요청량이 증가해도 안정적인 응답 속도**를 보였음
  - **Latency(Max 기준)**: 388.64ms → 118.24ms **(약 70% 개선)**
  - **CPU(Max 기준)**: 7.1core → 4core **(약 44% 개선)**
  - **Memory(Max 기준)**: 1.44GiB → 217MiB **(약 85% 개선)**

배포 전략

- 문제 발생 시 **빠른 롤백**이 가능한 환경 구성하기
- **최소한의 리소스**로 만드는 것이 목표이므로 **검증도 신속하게** 진행하기
- 안정성을 위해 **점진적으로 배포**하기

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 annotations:
     ...
 http:
    timeout: 65s
  - match:
      - uri:
          prefix: /image/files
    retries:
      attempts: 0
    route:
      - destination:
          host: image-blue
        weight: 90
      - destination:
          host: image-v2-blue
        weight: 10
    timeout: 65s
...
```

- **Canary 방식**으로 **트래픽 전달**하기
- **Istio Virtural Service**를 통해 **동일한 Endpoint**에 **Weight 조정**하기

⇒ 문제 발생시, **바로 롤백 가능**

배포하기

배포 안내

PROD로 image-v2 10% 트래픽 흘리고 있습니다. 🙏
cc. @cloud-engineer

> 백경준 [DevOps셀]
> @developers
> 안녕하세요. DevOps셀입니다.
> DEV/QA 현재 개선된 image api가 배포되었습니다.
> 변경 내용
> - `GET image/file/{fileName}` 엔드포인트 한정
>   - 파일 다운로드/리사이즈 성능 개선 기능 배포
>
> DEV/QA 이용하시면서 이미지가 다르게 보인다거나 기존에 동작했던 image-api 문제가 발생한다면 #g_cell_devops 로 문의주세요. 🙇
> cc. @devops

성능 개선 결과

성능 비교: Resource

- **비용 및 사용률: 74.7% 개선**
- **요청당 평균 vCPU 사용량:**
  - **vCPU**: 87cores -> 22cores (**65vCPU 감소**)
  - **Memory**: 174GiB -> 44GiB (**130GiB 감소**)

성능 개선 결과

성능 비교: Latency

- **평균 응답 시간**: 280ms → 159ms **(121ms 감소, 43.21% 개선)**
- **p50 평균 응답 시간**: 224ms → 124ms **(100ms 감소, 44.64% 개선)**
- **p75 평균 응답 시간**: 371ms → 175ms **(196ms 감소, 52.84% 개선)**
- **p99 평균 응답 시간**: 919ms → 613ms **(306ms 감소, 33.30% 개선)**

성능 개선 결과

성능 비교: CI/CD

- **CI/CD 실행 시간**: 4m 18s → 1m 7s **(191s 감소, 74.03% 개선)**
- **빌드 크기**: 900MB → 120MB **(780MB 감소, 86.67% 개선)**

Golang 컨테이너 환경 최적화 Tip

GOMAXPROCS

```
func GOMAXPROCS(n int) int
```

GOMAXPROCS sets **the maximum number of CPUs that can be executing simultaneously** and returns the previous setting. It defaults to the value of runtime.NumCPU. If n < 1, it does not change the current setting. This call will go away when the scheduler improves.

https://pkg.go.dev/runtime#GOMAXPROCS

GOMAXPROCS 설정 참고

- **GO 1.5 >=**
  - default: **node 전체의 vCPU = GOMAXPROCS**
  - 16vCPU node의 경우, GOMAXPROCS=16으로 설정됨
- **GO 1.5 <**
  - default: **1 = GOMAXPROCS**

GOMAXPROCS 임의 설정하기

1. **automaxprocs(Uber의 오픈소스)**
   - CPU Limit 값을 기준으로 GOMAXPROCS 값을 자동 설정
2. **환경변수 설정(GOMAXPROCS)**

Golang 컨테이너 환경 최적화 Tip

automaxprocs로 임의 설정하기

```
go get -u go.uber.org/automaxprocs
```

설치하기

```
import _ "go.uber.org/automaxprocs"
```

사용하기

```
func main() {
  // Your application logic here
}
```

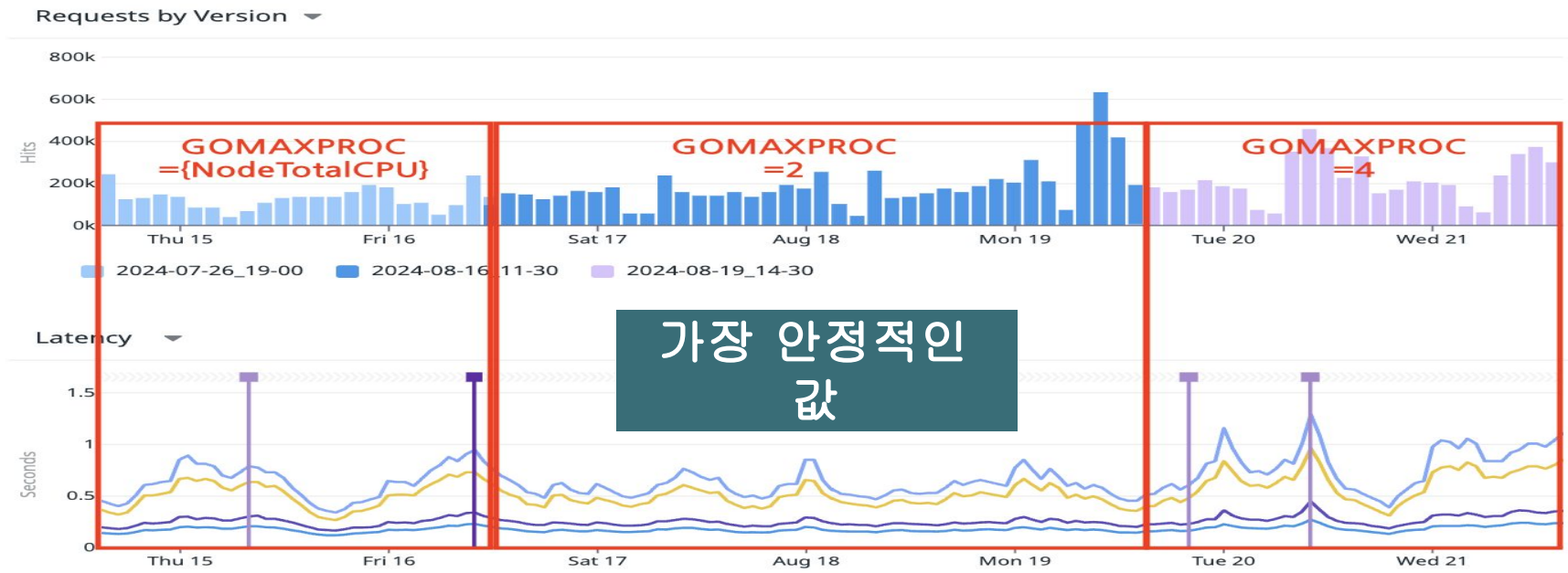
Golang 컨테이너 환경 최적화 Tip

환경변수로 임의 설정하기

```yaml
apiVersion: v1
kind: Pod
...
    resources:
      limits:
        cpu: 1500m
    env:
    - name: GOMAXPROCS
      valueFrom:
        resourceFieldRef:
          resource: limits.cpu
```

Golang 컨테이너 환경 최적화 Tip

GOMAXPROCS 설정값 테스트



안정적인 GOMAXPROCS 설정값 찾기

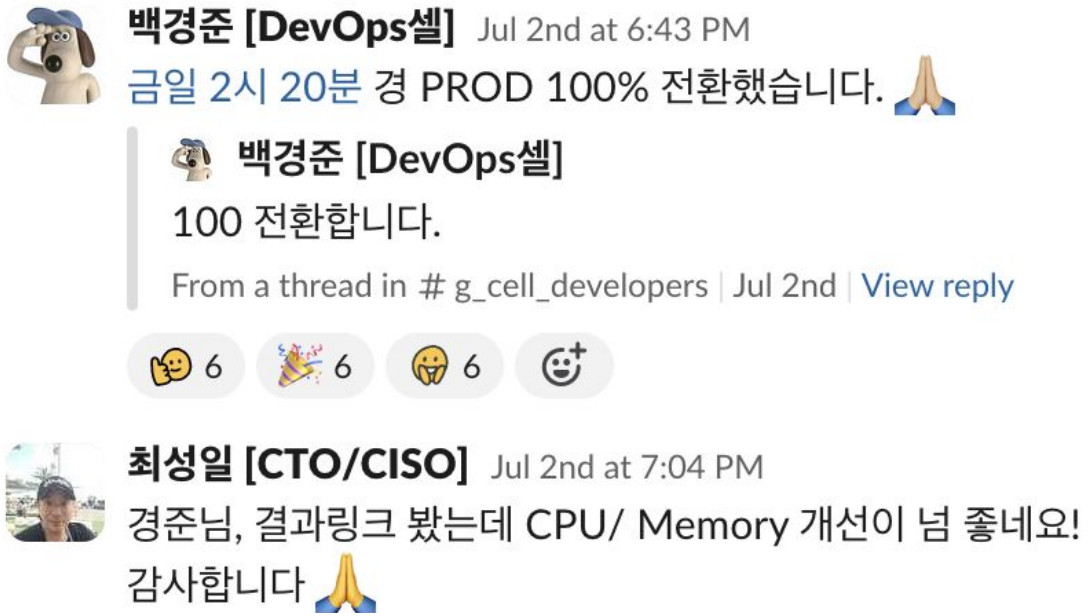
- **설정값에 따라 수치에 차이가 발생하는 이유:**
  할당된 CPU 수(CPU Limit)가 런타임에서 사용하는 CPU 최대 수(GOMAXPROCS)보다 작기 때문에 Context Switch 부하 발생

⇒ **GOMAXPROCS의 적정값 할당**은 **P95, P99 등의 Latency를 개선**해줄 수 있음

Image 서버 개선 이후의 이야기

앞으로의 과제

성공적으로 개선한 Image 서버



앞으로의 과제

Image 서버 교체는 단기안일 뿐

앞으로의 과제

Image 서버 아키텍처 재설계

- **이미지 제공용 서버, 이미지 관리용 신규 Image API를 분리하기**
- **Amazon S3 Intelligence Tier 도입을 통한 스토리지 비용 최적화하기**
  - 데이터의 액세스 패턴에 따라 자동으로 티어 설정을 하는 가장 비용 효율적인 S3 스토리지 티어
- **이미지 확장자 변경 및 마이그레이션 전략 준비하기**
  - 대부분의 브라우저에서 지원하는 고품질, 저용량의 ACIF 파일 포맷 도입하기
  - 구버전 Image 서버를 CF와 S3로 전환하기

앞으로의 과제

Image 서버 아키텍처 재설계

- **이미지 제공용 서버, 이미지 관리용 신규 Image API를 분리하기** — **진행중**
- **Amazon S3 Intelligence Tier 도입을 통한 스토리지 비용 최적화하기**
  - 데이터의 액세스 패턴에 따라 자동으로 티어 설정을 하는 가장 비용 효율적인 S3 스토리지 티어
- **이미지 확장자 변경 및 마이그레이션 전략 준비하기**
  - 대부분의 브라우저에서 지원하는 고품질, 저용량의 ACIF 파일 포맷 도입하기
  - 구버전 Image 서버를 CF와 S3로 전환하기

Image 서버 아키텍처 재설계: AS-IS

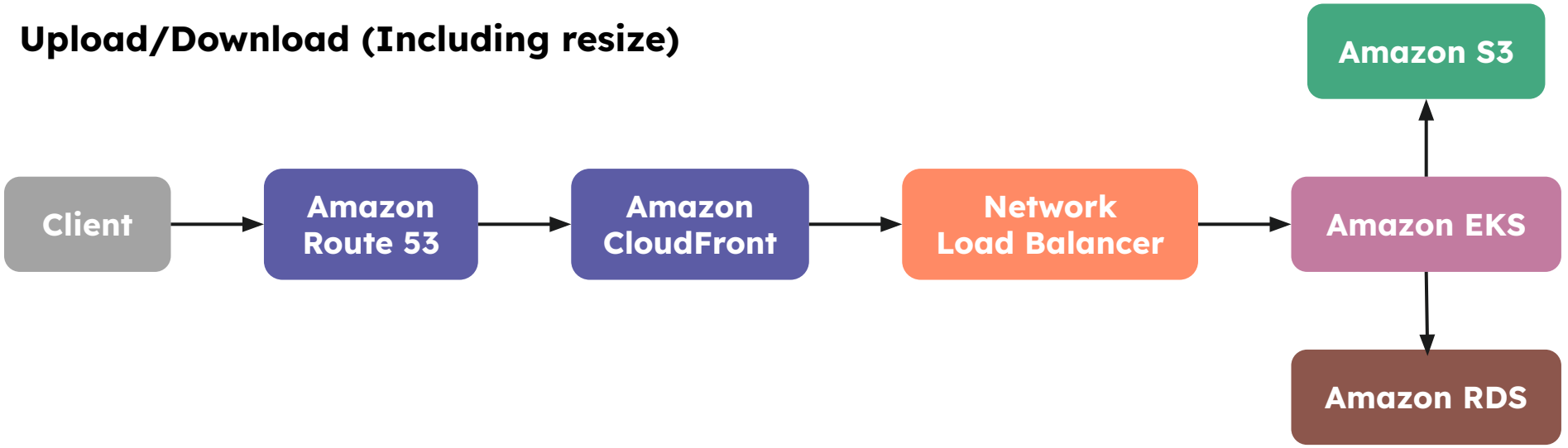


Image 서버 아키텍처 재설계: TO-BE

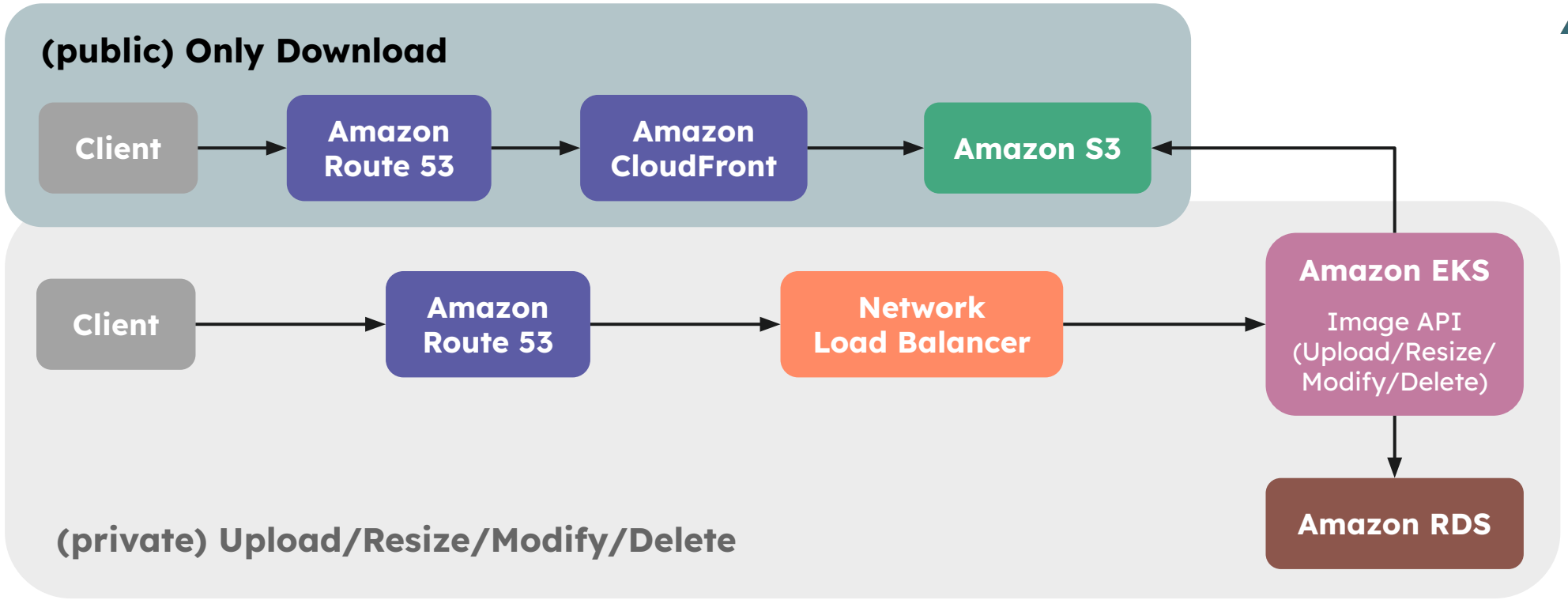


Image 서버 아키텍처 재설계

신규 Image API 배포 후 첫 요청 사례

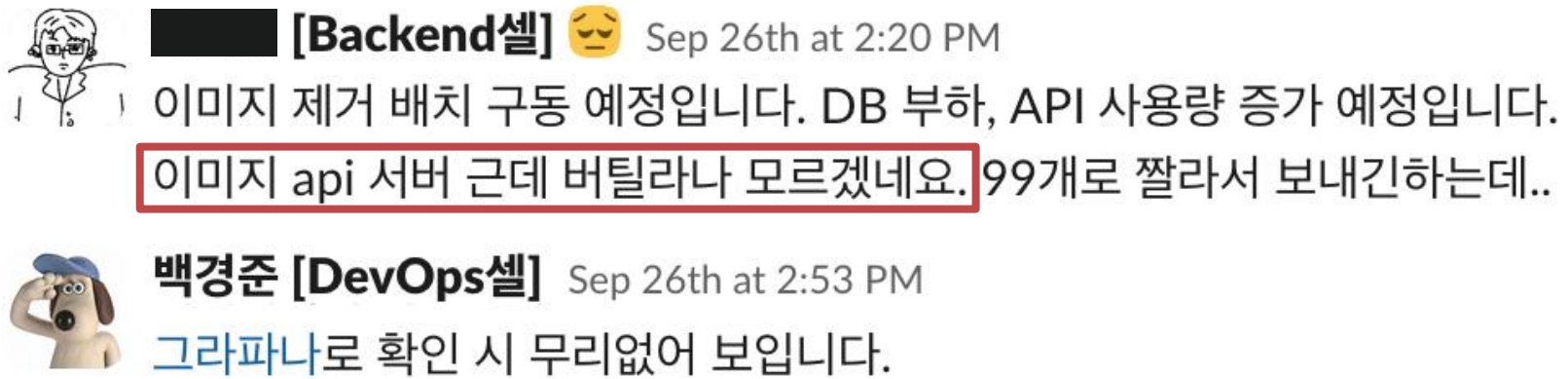


Image API 응답 상태 코드

Code	Description
200	OK Example Value | Model

```
{
  "failed_count": 0,
  "failed_files": [
    "string"
  ],
  "succeeded_count": 0,
  "succeeded_files": [
    "string"
  ]
}
```

207	not found file is success Example Value | Model

```
{
  "failed_count": 0,
  "failed_files": [
    "string"
  ],
  "succeeded_count": 0,
  "succeeded_files": [
    "string"
  ]
}
```

- **200**: 전체 성공
- **207**: 하나라도 실패할 경우

Image 서버 아키텍처 재설계

분당 4천 건의 요청에도 안정적인 서버

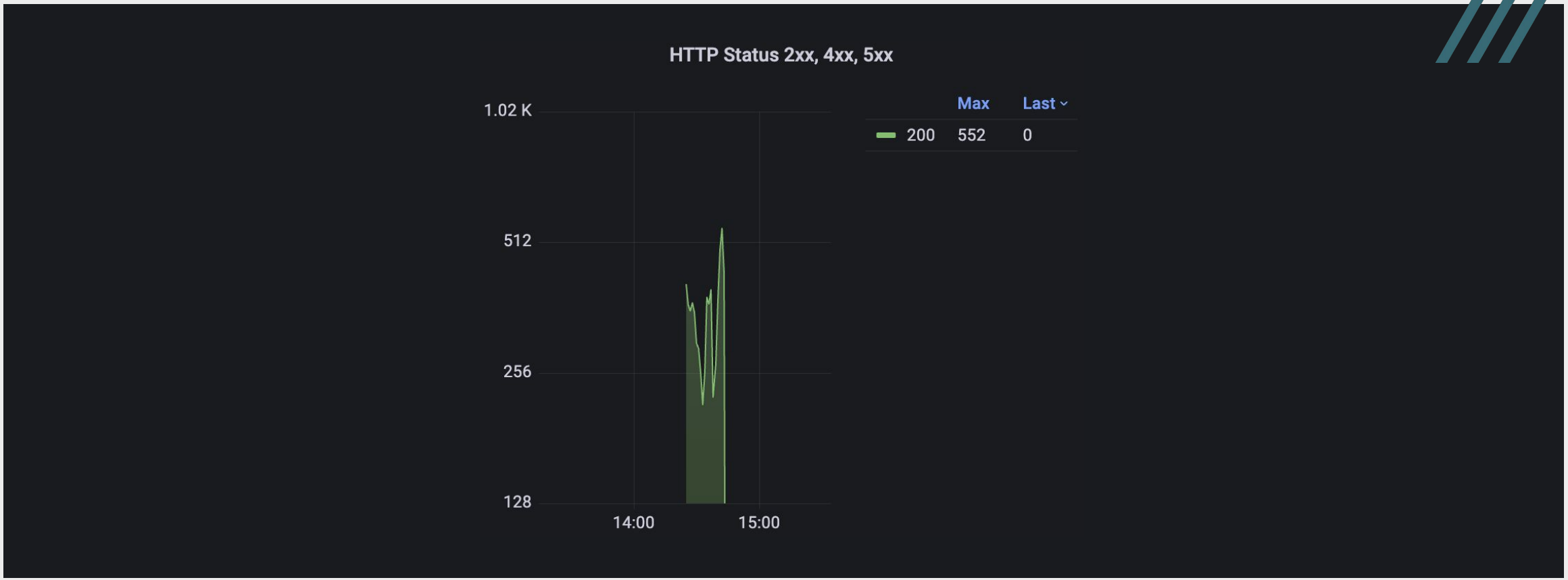


Image 서버 아키텍처 재설계

오토스케일링 없는 안정적인 서버

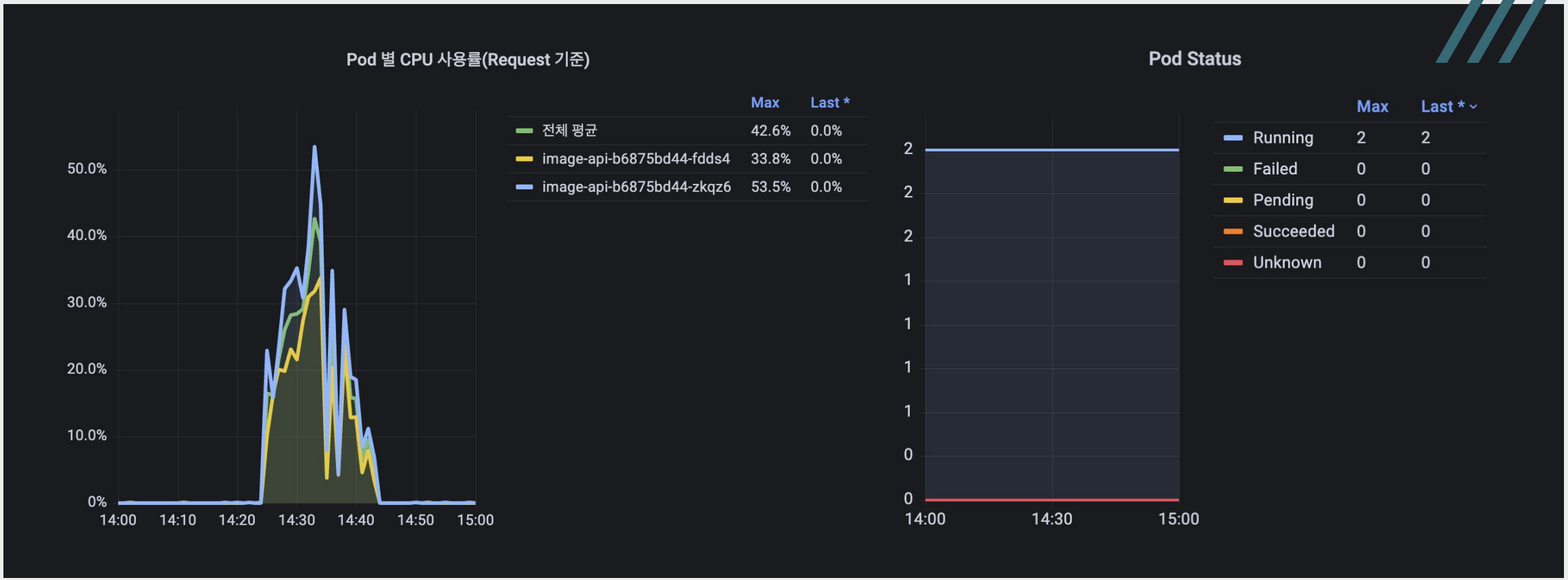


Image 서버 아키텍처 재설계

개선 작업은 계속됩니다

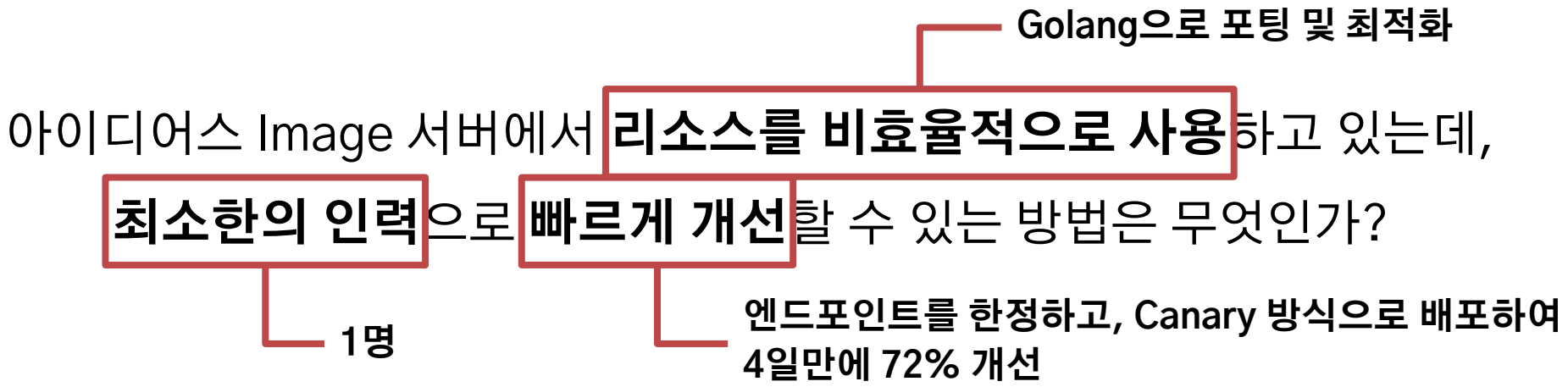
마무리

문제 정의

아이디어스 Image 서버에서 **리소스를 비효율적으로 사용**하고 있는데,

최소한의 인력으로 **빠르게 개선**할 수 있는 방법은 무엇인가?

문제 해결


Golang으로 포팅 및 최적화
아이디어스 Image 서버에서 리소스를 비효율적으로 사용하고 있는데,
최소한의 인력으로 빠르게 개선할 수 있는 방법은 무엇인가?
1명
엔드포인트를 한정하고, Canary 방식으로 배포하여
4일만에 72% 개선

Tidy First!

문제 해결 방안이 복잡해 보여도 작은 것부터 점진적으로 개선해 보세요

Join Backpackr!

창작자 생태계를
함께 만들어 나갈
배낭여행자를 찾습니다 🥾

Thank you!