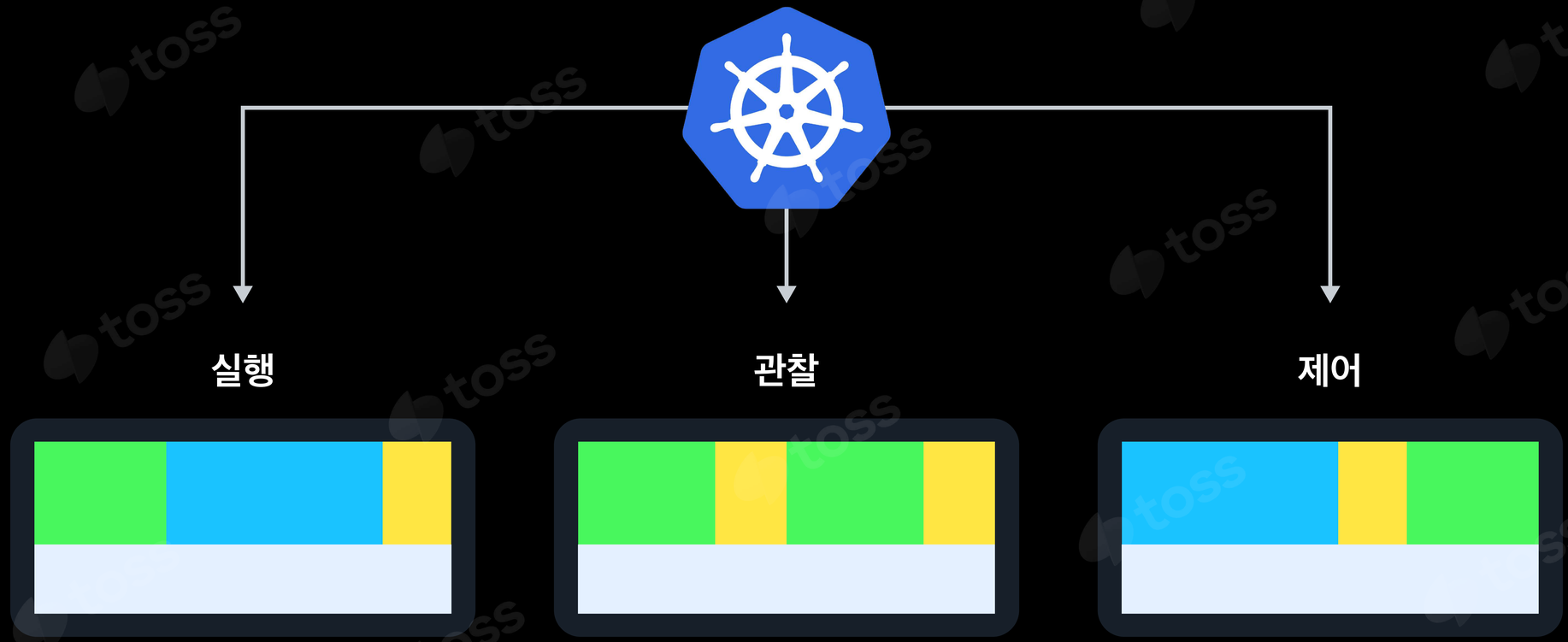
SLASH 24

Kubernetes CPU 알뜰하게 사용하기

김동석 이재성
DevOps Team Leader

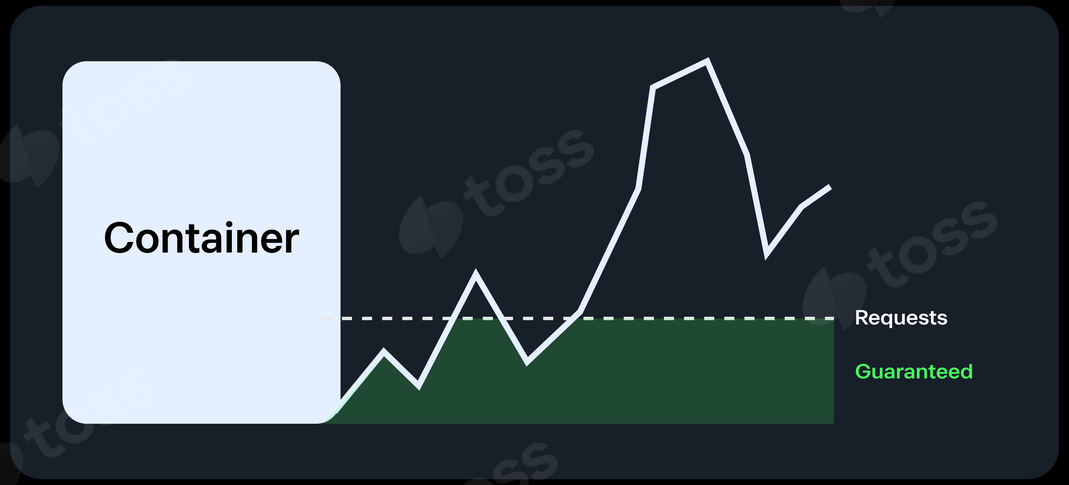
toss

Kubernetes



CPU Requests

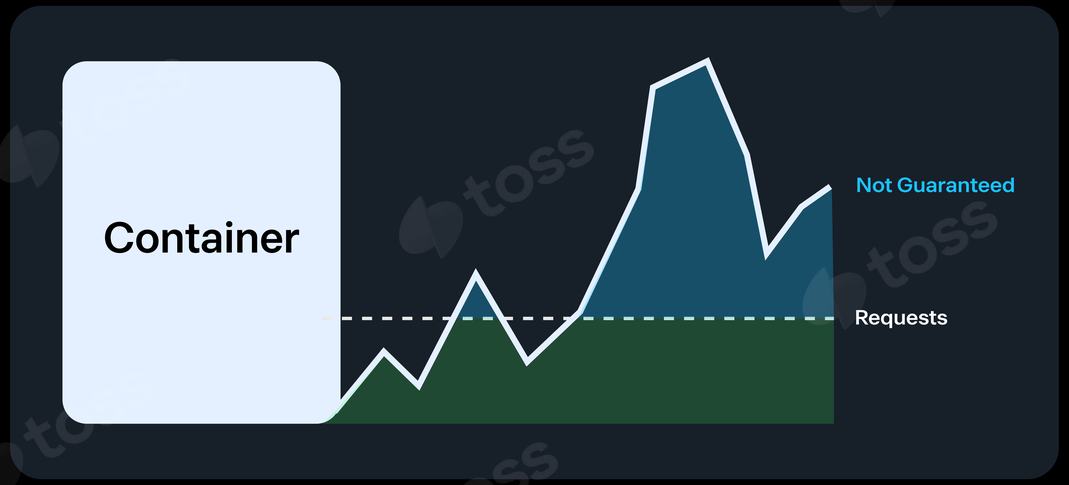
최소 보장량



CPU Requests

최소 보장량

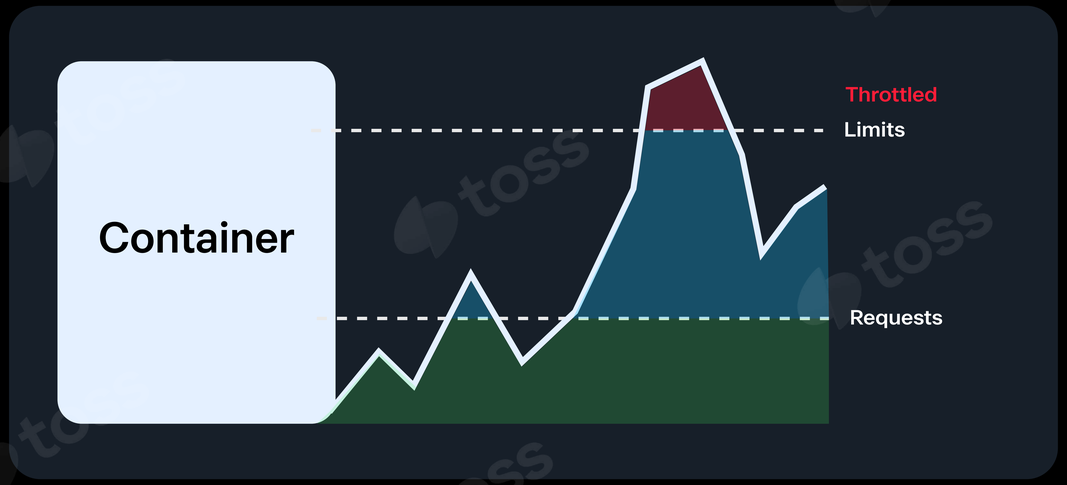
초과 가능



CPU Limits

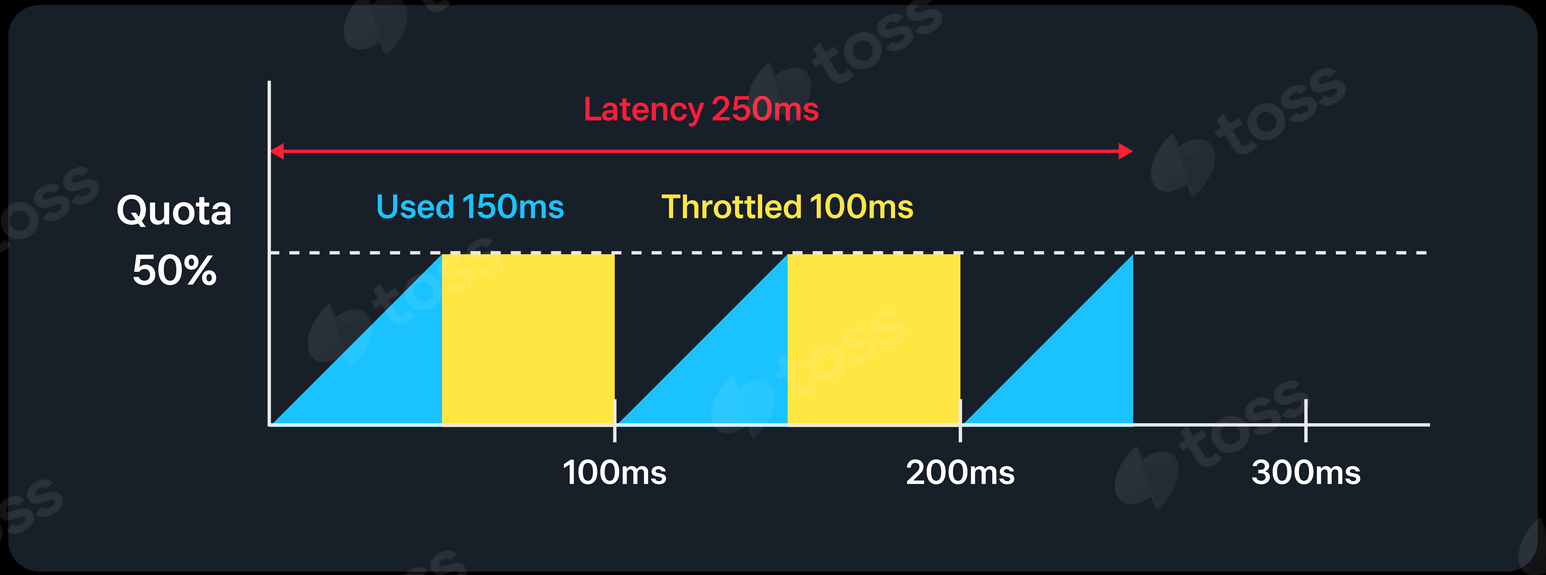
최대 허용량

Throttling



CPU Throttling

CPU를 할당받지 못하여 기다림



자원 최적화

서비스 안정성 유지

자원 할당량 최소화

자원 사용량 최소화

자원 사용량 분산

자원 최적화

서비스 안정성 유지

자원 할당량 최소화 ∝ 비용

자원 사용량 최소화

자원 사용량 분산

자원 최적화

서비스 안정성 유지

자원 할당량 최소화

자원 사용량 최소화

자원 사용량 분산

Kubernetes CPU 최적화

CPU Throttling 방지

CPU Requests/Limits 최소화

CPU 사용량 최소화

CPU 사용량 분산

Kubernetes CPU 최적화

CPU Throttling 방지

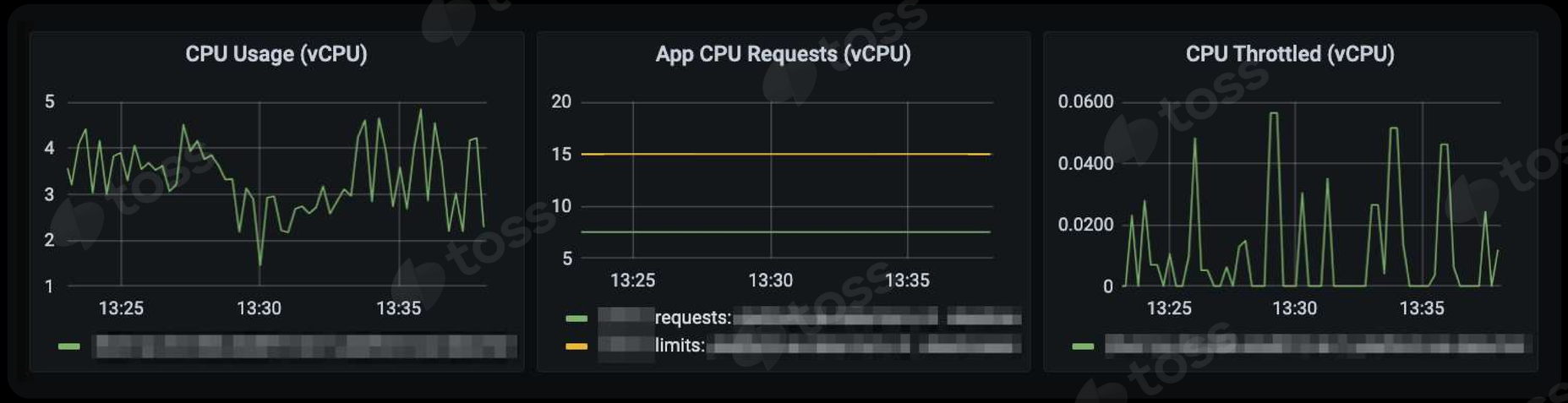
CPU Requests/Limits 최소화

CPU 사용량 최소화

CPU 사용량 분산

CPU 지표

CPU 사용량 추이가 Limits보다 낮아도 Throttling 발생



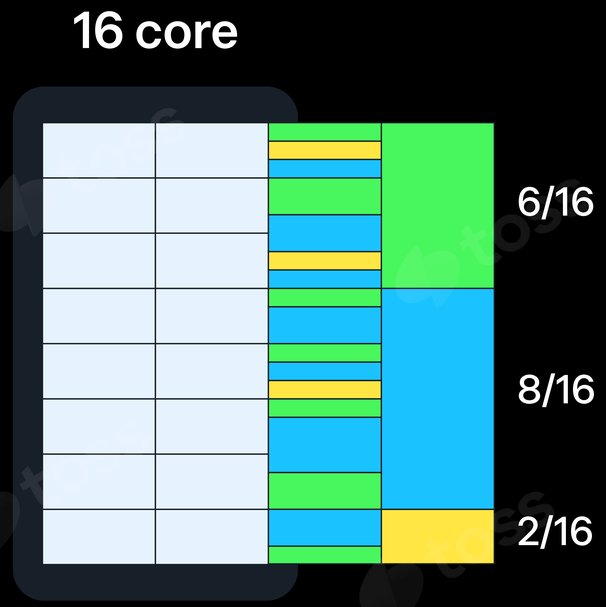
CPU 사용량 < 5 < Requests 7.5 < Limits 15

CFS

Completely fair scheduler

Worker node의 모든 CPU 사용

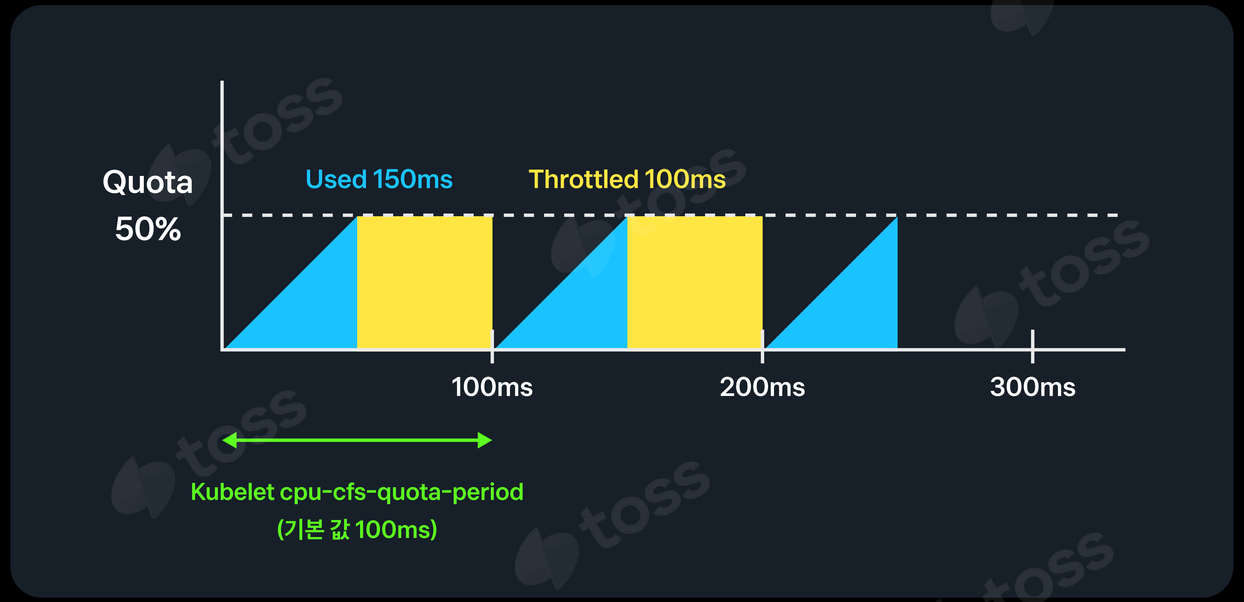
Time slice를 할당량 비율대로 분배



CFS

원인

CFS quota 동작 방식



CFS

완화

Kubelet cpu-cfs-quota-period 낮추기

방지

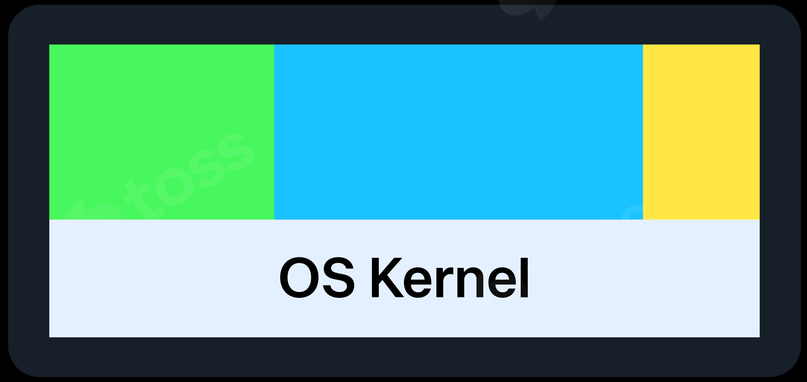
Kubelet cpu-cfs-quota 비활성화

NO LIMIT

병렬성 설정

Container CPU core 인식 개수

= Worker node CPU core 전체 개수



병렬성 설정

CPU core 개수에 연동

과도한 부하

예시

JVM ActiveProcessorCount, Java11+ Container-awareness

Node Jest numWorkers, pnpm workspace-concurrency

Go GOMAXPROCS, automaxprocs

Kubernetes CPU 최적화

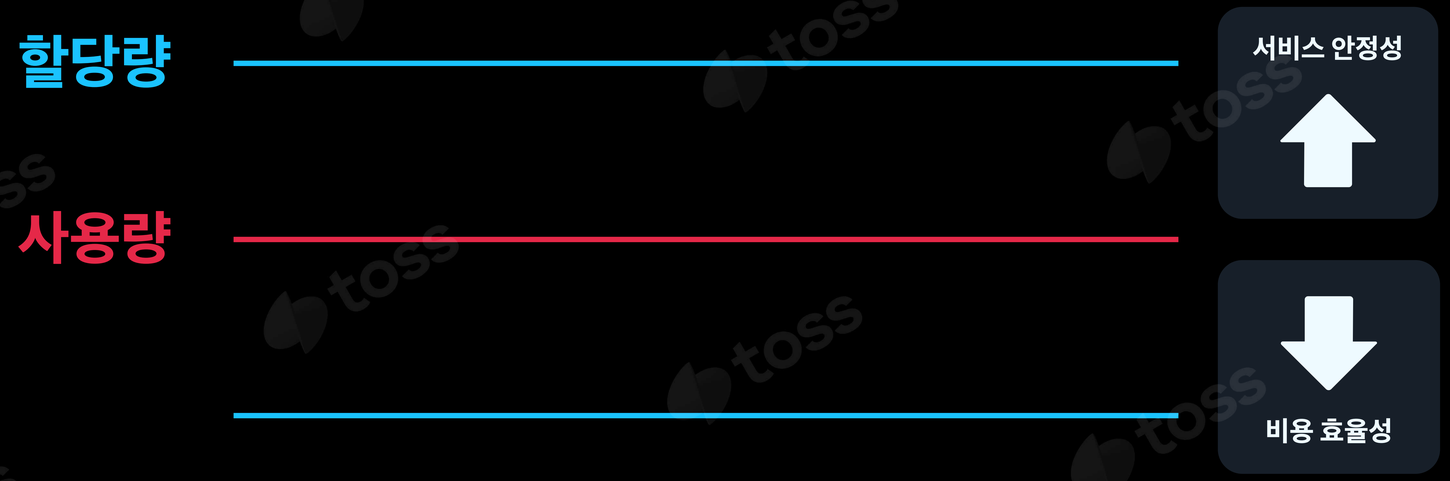
CPU Throttling 방지

CPU Requests/Limits 최소화

CPU 사용량 최소화

CPU 사용량 분산

Right sizing



정책

Workload	Requests	Limits
대고객 서비스와 직접 연관	=~ 하루 최대 사용량 ×2	NO LIMIT
나머지	=~ 하루 최대 사용량	> Requests

정책

Active-Active
Data Center

Workload	Requests	Limits
대고객 서비스와 직접 연관	=~ 하루 최대 사용량 ×2	NO LIMIT
나머지	=~ 하루 최대 사용량	> Requests

정책

SLASH 2021 이항령
토스 서비스를 구성하는 서버 기술

Workload	Requests	Limits
대고객 서비스와 직접 연관	=~ 하루 최대 사용량 ×2	NO LIMIT
나머지	=~ 하루 최대 사용량	> Requests

정책

Workload	Requests	Limits
대고객 서비스와 직접 연관	=~ 하루 최대 사용량 ×2	NO LIMIT
나머지	=~ 하루 최대 사용량	> Requests

정책

Workload	Requests	Limits
대고객 서비스와 직접 연관	=~ 하루 최대 사용량 ×2	NO LIMIT
나머지	=~ 하루 최대 사용량	> Requests

Auto scaling

Cloud 환경

자동으로 할당량 조정

Auto scaling

IDC 환경

Server 즉시 증설 불가

고부하 시 자원 과점유

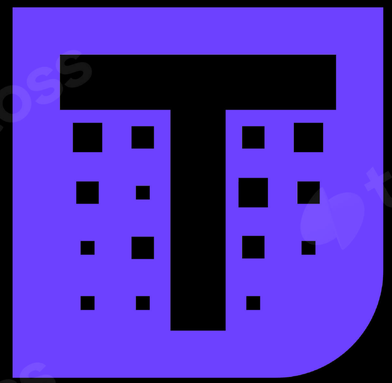
Projection

최대 부하에 맞추어 산정

사용량 변화에 따라 조정

Alert 활용

Metric



Alert

할당량이 너무 낮아서 여유 부족

할당량이 너무 높아서 자원 낭비

Alert

기준 Metric

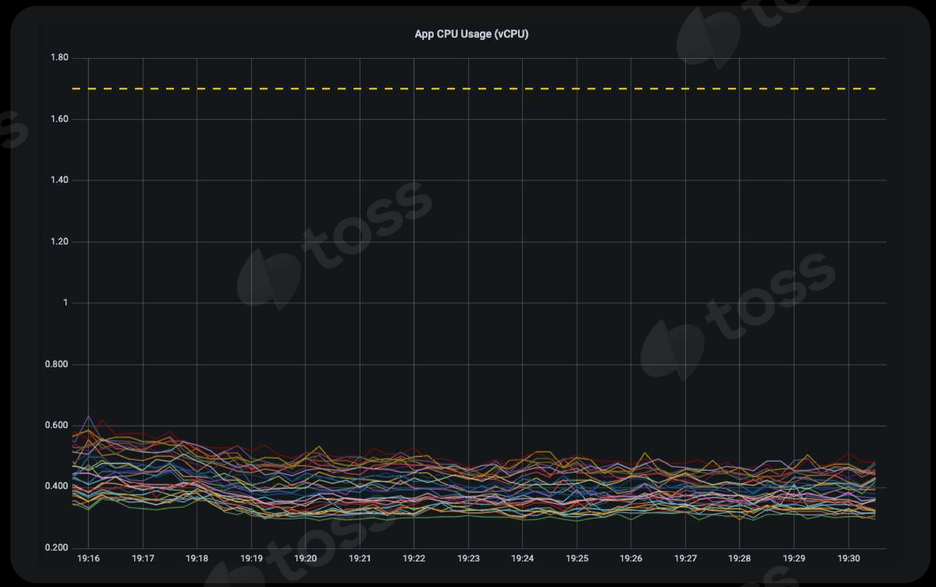
Noise 제거

기준 Metric

Requests 대비 사용량

irate(container_cpu_usage_seconds_total[1m])

/ kube_pod_container_resource_requests_cpu_cores



Noise

정확하지 않은 Alert

중요하지 않은 Alert

Noise 제거

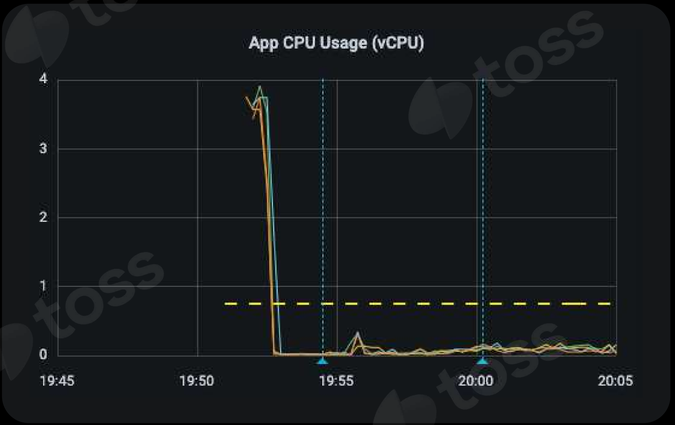
문제

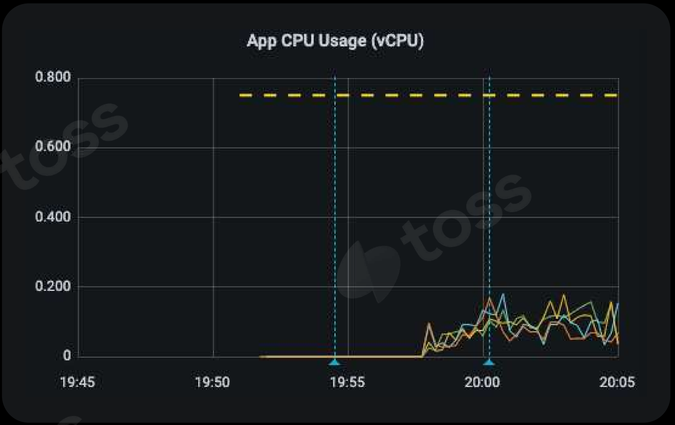
Kotlin+Spring Warm-up

해결

PromQL 로 초기 400초 무시

clamp(time() - kube_pod_created - 400, 0, 1)





Noise 제거

문제

새벽 시간대 ,주말

해결

최근 1주 동안의 최대 사용량

max_over_time(container_cpu_usage_seconds_total[7d:1m])

PromQL 성능

문제

7일 max_over_time 부하

해결

시간 해상도?

PromQL 성능

해결

max_over_time 중첩

```
container_cpu_usage_seconds_max
    = max_over_time(container_cpu_usage_seconds_total[30m:1m])
max_over_time(container_cpu_usage_seconds_max[7d:15m])
```

Kubernetes CPU 최적화

CPU Throttling 방지

CPU Requests/Limits 최소화 IDC

CPU 사용량 최소화

CPU 사용량 분산

Kubernetes CPU 최적화

CPU Throttling 방지

CPU Requests/Limits 최소화 Cloud

CPU 사용량 최소화

CPU 사용량 분산

Right Sizing 사례 소개

클라우드

오버프로비저닝 → 즉시 비용 낭비

IDC에서 처럼 할당 최적화

토스는 신규사업/서비스에 클라우드 활용

리소스 fragmentation

Right Sizing for Cloud

리소스 fragmentation

워커노드가 너무 작아서 리소스가 효율적이지 못할 수 있음

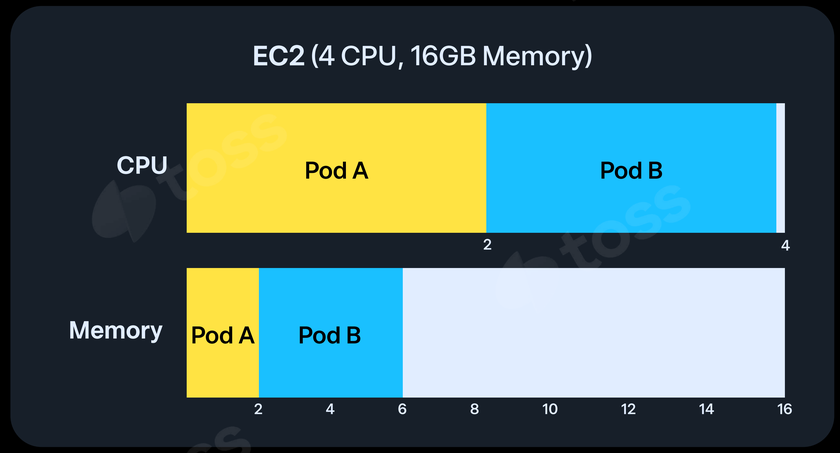
리소스 할당의 fragmentation

작은 vm(ec2)에 메모리 16기가, cpu 4코어 인 ec2 가정

이때 cpu를 4코어만 할당하였지만,

메모리가 6기가 할당하여 다른 컨테이너가 못뜰때

메모리는 10기가가 모두 낭비될 수 있습니다



Right Sizing for Cloud

리소스 fragmentation && daemonset 효율화

daemonset의 경우 모든 워커노드에 뜨기 때문에
워커노드 스펙업을 하면 daemonset이 덜 뜸

특정 클러스터가 노드 사이즈가 작았음

노드를 4배로 스케일업 → 노드 전체 사이즈가 40%가 줄었음

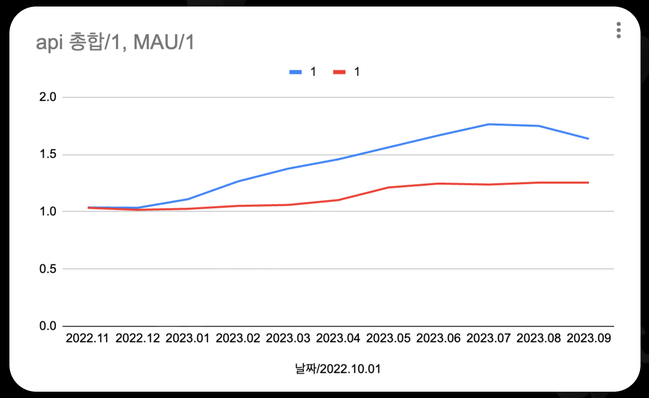
40%정도의 리소스 효율화가 되었음

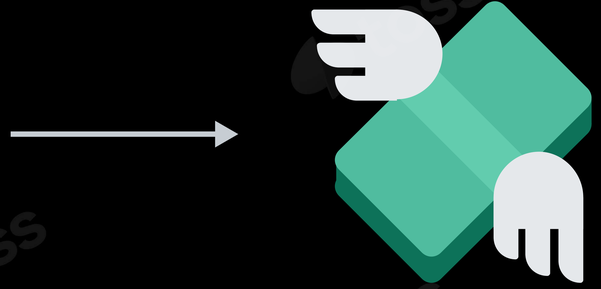
CPU Optimization

MAU 20%씩 증가

매년 40%씩 트래픽/CPU 증가

트래픽/CPU가 MAU 증가의 제곱으로 증가 → 비효율 발생



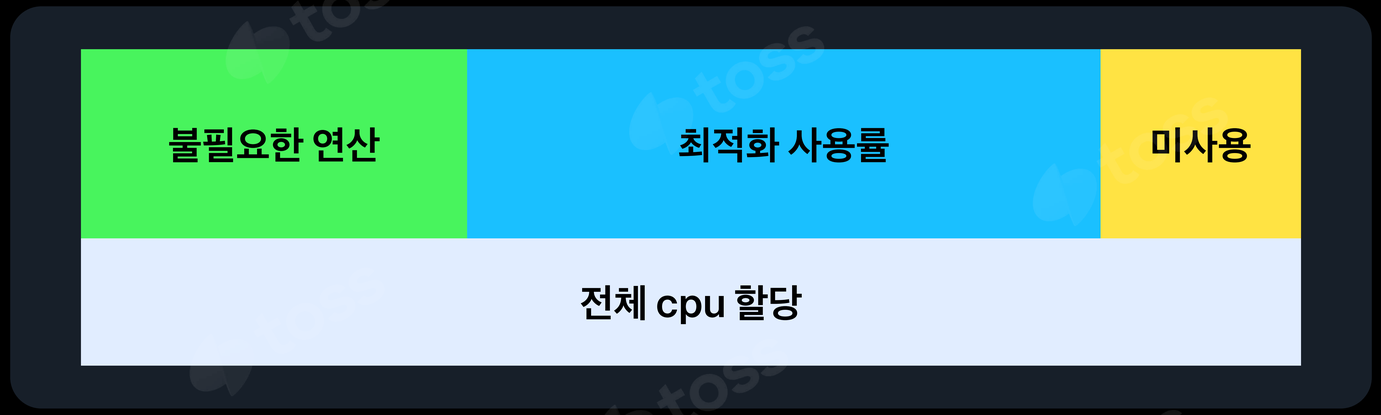


CPU Optimization

cpu 최적화

할당과 사용률의 갭을 최적화하는 것 뿐만 아니라

사용률 자체도 비효율적으로 사용하는게 없는지 파악해야됨



CPU Optimization

영역

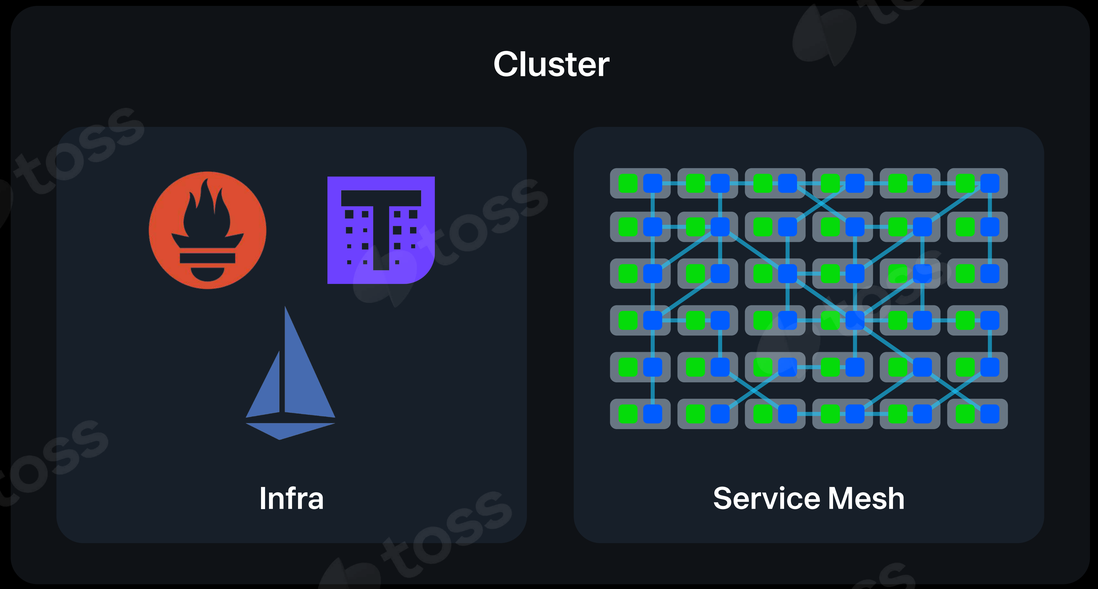
클러스터 운영을 위한 컴포넌트

(istio, thanos, prometheus, logstash 등등)

→ 인프라 레이어

사일로에서 개발하는 마이크로 서비스 튜닝

→ 서비스 레이어

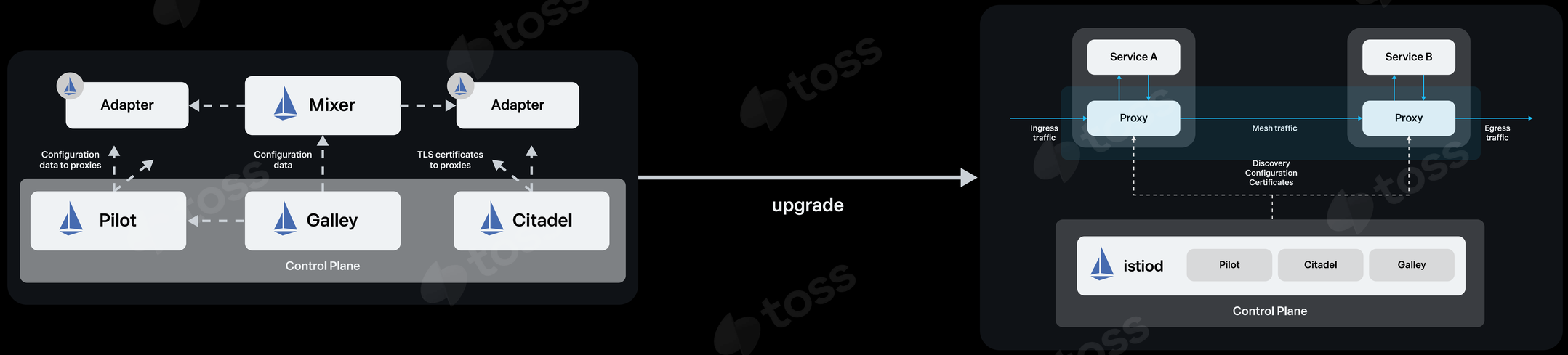


Optimization for infra

infra에서 리소스 사용률이 높은 컴포넌트

Istio (전체 클러스터의 15%차지)

- 일정버전 이상부터 telemetry가 사라짐

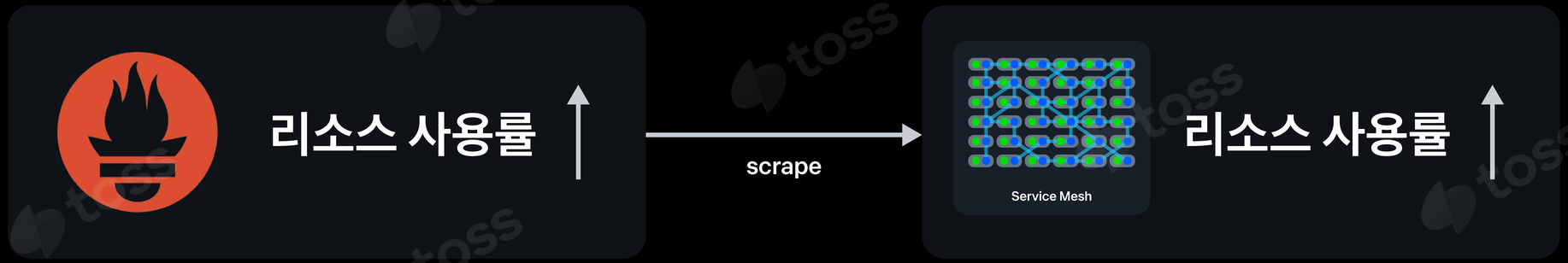


Optimization for infra

istio 업그레이드 이후 비효율

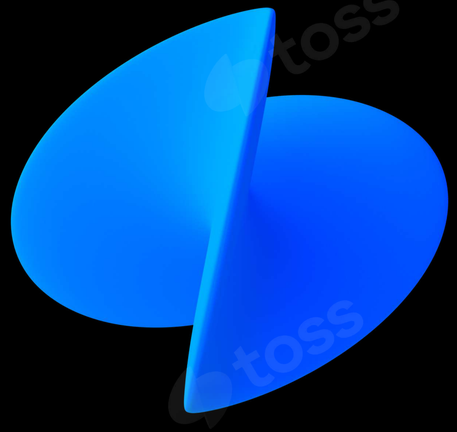
모든 envoy의 리소스 사용량 증가

메트릭 카디널리티 증가로 프로메테우스 리소스 사용량 증가



Optimization for infra

그래서 만들었습니다...



toss mixer

Optimization for infra

toss-mixer

Envoy als를 통해 메트릭, 로그 콜렉팅

Optimization for infra

기존 istio telemetry의 비효율성

prometheus의 메트릭 효율화

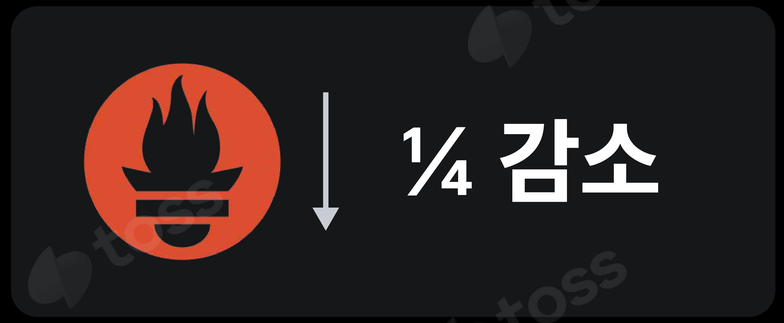
기존 istio telemetry의 구현 비효율

telemetry json serialize

- Simd 아키텍처 미활용
- 중복된 json serialize

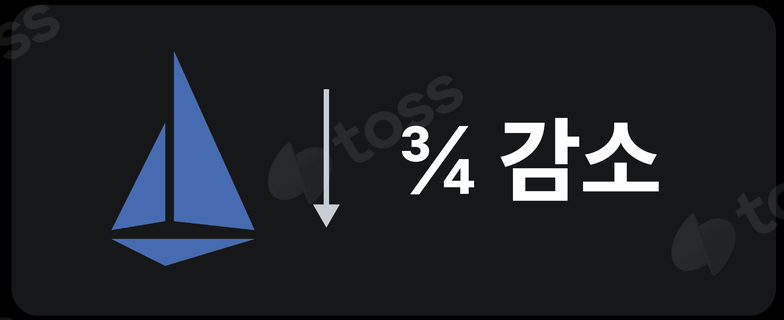
Optimization for infra

toss-mixer 도입 결과



CPU 73 core → 52 core

Memory 501GB → 310GB



CPU 420 core →79 core

Memory 174GB → 95GB

Optimization for infra

infra에서 리소스 사용률이 높은 컴포넌트

prometheus && thano

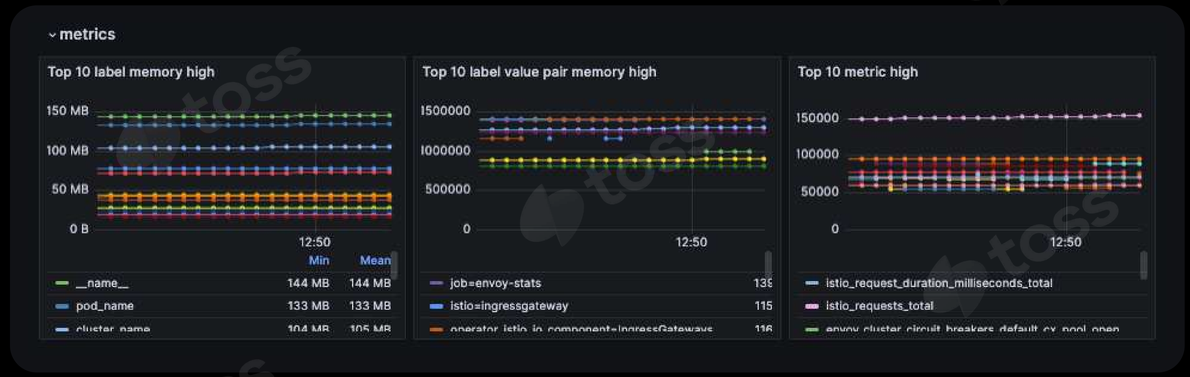
1. 메트릭 gc (TSDB status)
2. 업그레이드
3. golnag 최적화(GOGC, GOMEM, GOMAXPROCS)

Optimization for infra

infra에서 리소스 사용률이 높은 컴포넌트

prometheus && thano

메트릭 gc (TSDB status)



Optimization for infra

infra에서 리소스 사용률이 높은 컴포넌트

prometheus && thano

업그레이드 2.31 → 2.39 (cpu 40% 절감)

업그레이드 2.41 → 2.45 (cpu 30% 절감)

CPU Optimization

영역

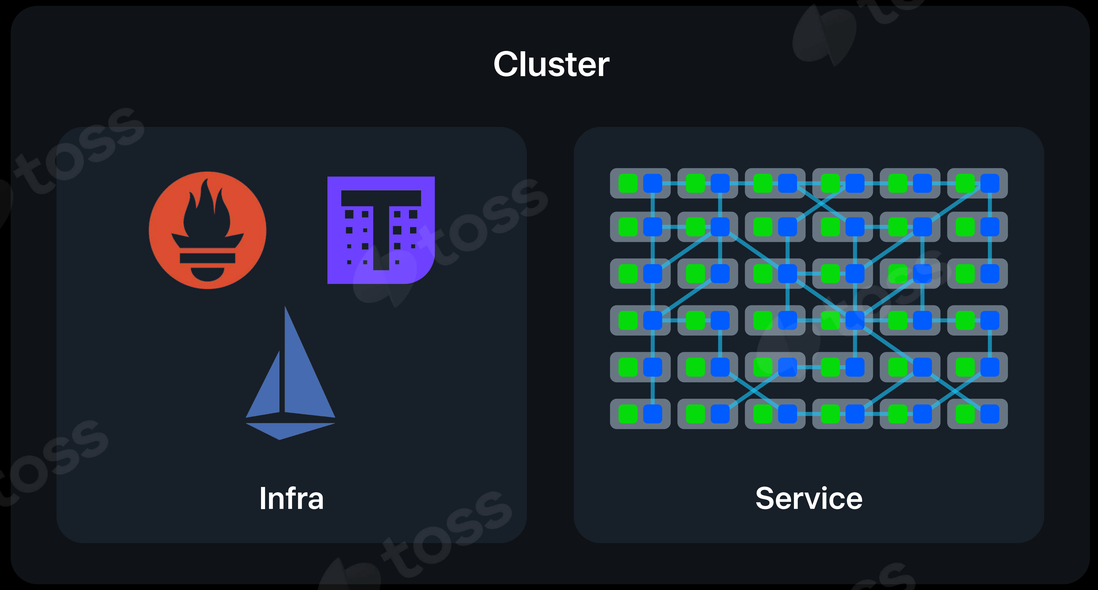
클러스터 운영을 위한 컴포넌트

(istio, thanos, prometheus, logstash 등등)

→ 인프라 레이어

사일로에서 개발하는 마이크로 서비스 튜닝

→ 서비스 레이어 (25년 slash에..)

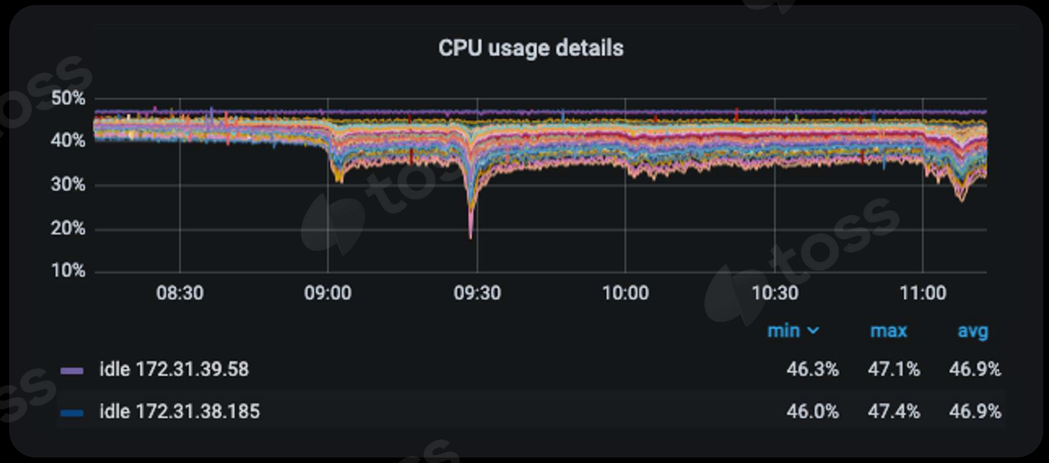


CPU 부하 분산 이슈

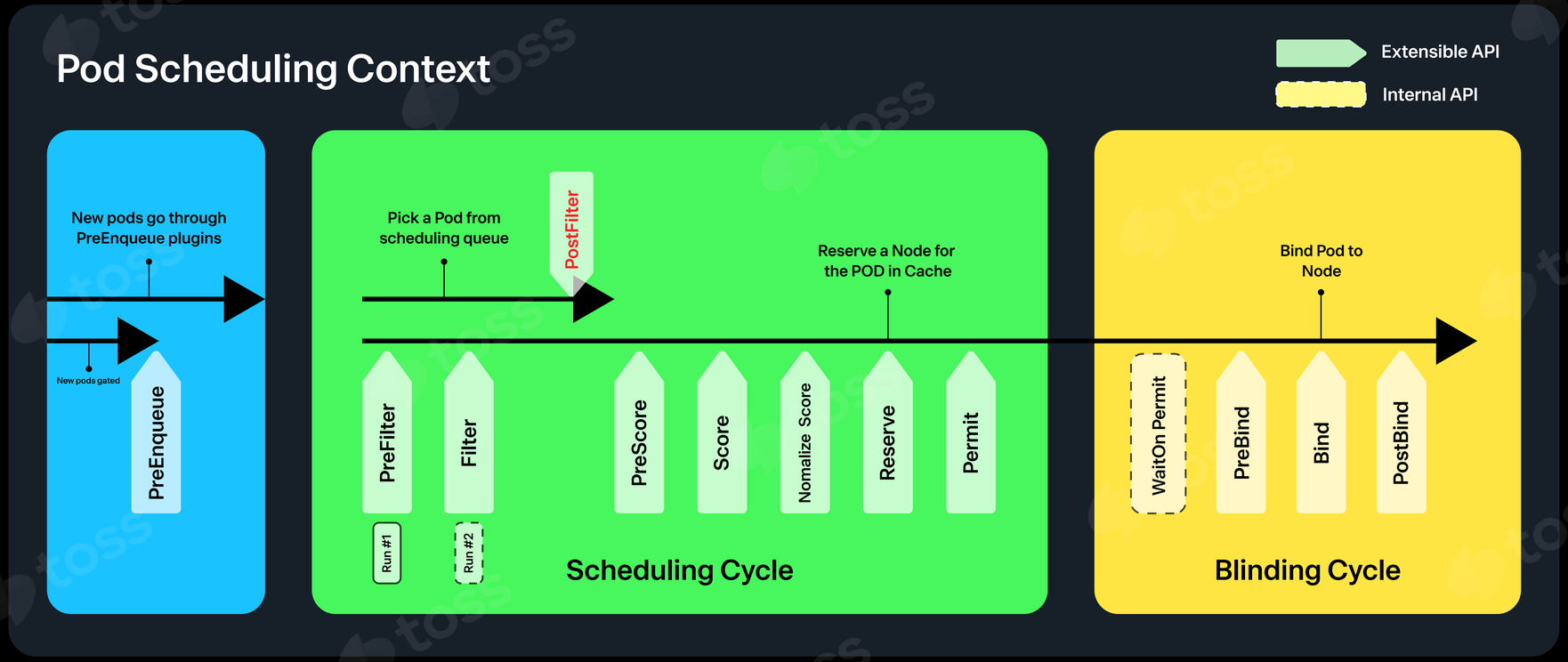
노드별 리소스 사용 괴리율 발생

→ 특정노드 과부하 로 서비스 지연

최저 ↔ 최고 cpu 사용률 괴리율 = 40%p

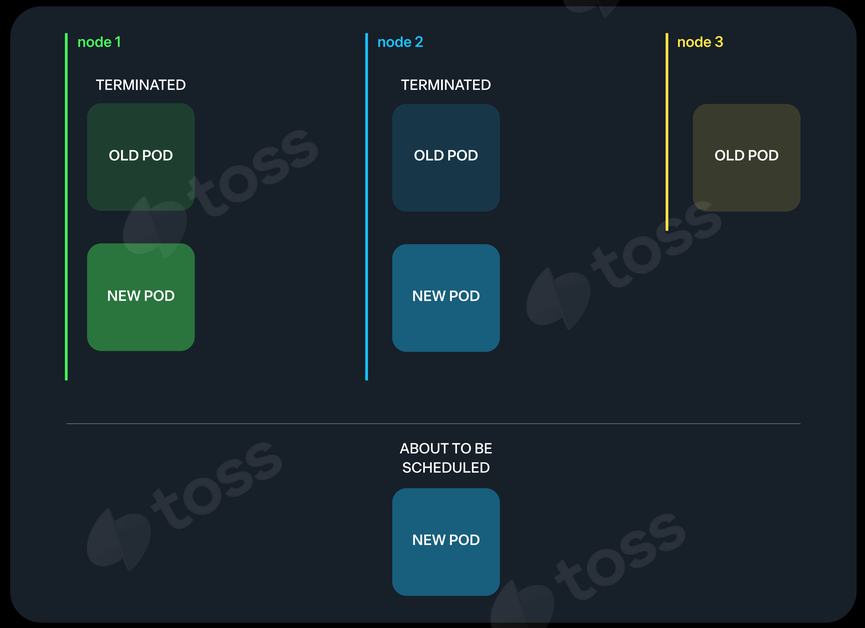


Scheduler 분배의 문제점



Topology Spread Constraint 활용

무조건 같은 갯수로 노드에 분산시켜줌



부하 분산 개선

괴리율 40%p → 20%p 로 개선

앞으로

부하 base의 scheduler

부하 skew 조정

2023년...

CPU 최적화

1. CPU Requests/Limits Right Sizing를 통해 오버프로비저닝 방지
2. 노드 스펙을 올려 리소스 fragmentation 방지
3. 불필요한 연산을 줄이고, 모니터링 아키텍처를 개선해 CPU 사용량 최소화
4. topology aware를 통해 CPU 사용량을 골고루 분산

SLASH 24