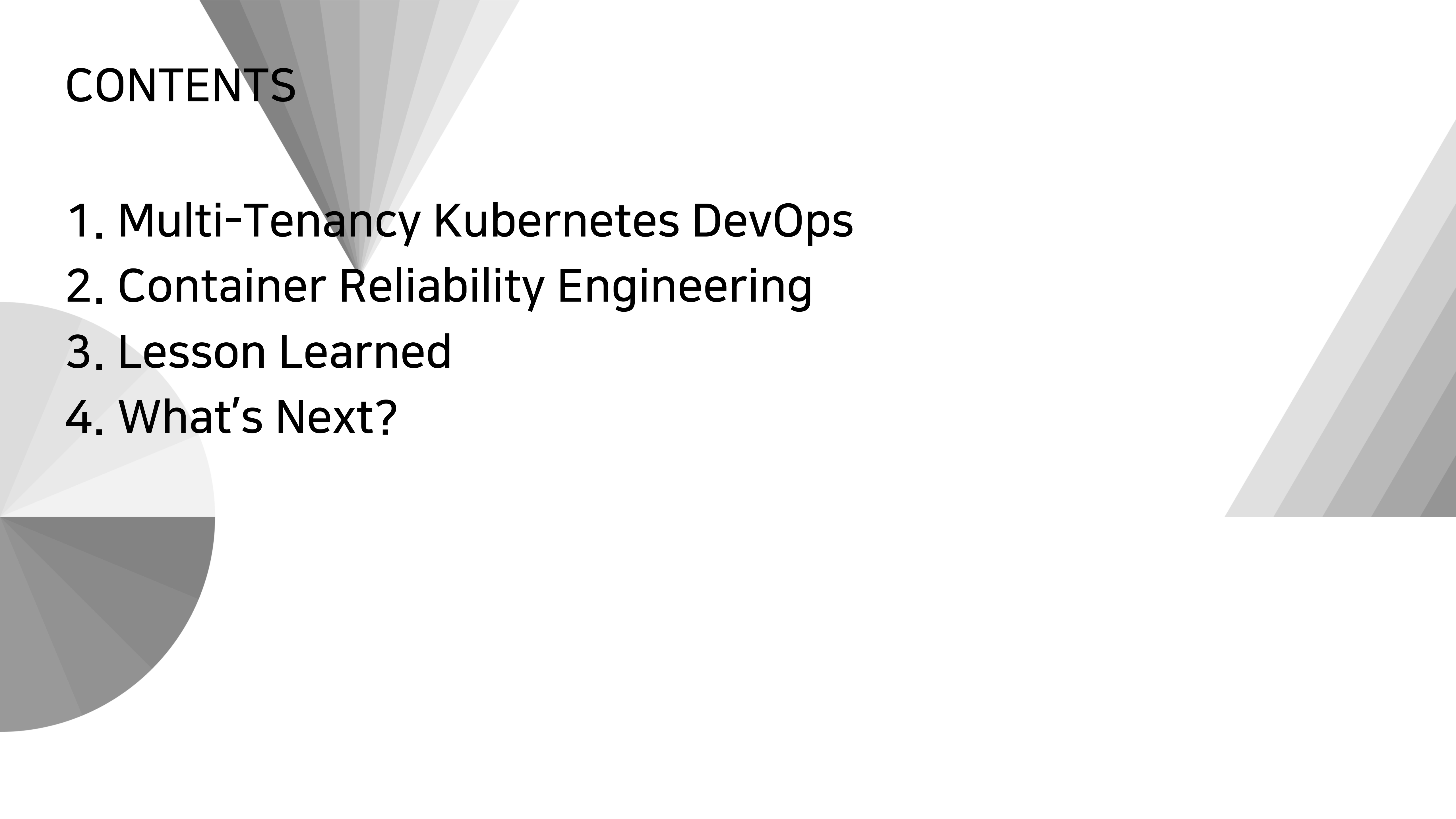


Container SRE

(Site Reliability Engineering)

황영석 NAVER / Computing Platform



CONTENTS

1. Multi-Tenancy Kubernetes DevOps
2. Container Reliability Engineering
3. Lesson Learned
4. What's Next?

1. Multi-Tenancy Kubernetes DevOps

1.1 Multi-Tenancy Kubernetes Remind

하나의 Kubernetes를 Namespace로 나눠서 사용자에게 제공하는 것

“Multi-Tenancy”

“Providing **isolation** and
fair **resource sharing** between
multiple users and their workloads
within a cluster.”

“Multi-Tenancy Kubernetes”

“Providing **isolation** and
fair **resource sharing** between
multiple “namespaces” and their workloads
within a “kubernetes”.”

[참고]

- Deview 2019 Multi-Tenancy Kubernetes on Bare-Metal Servers (네이버 컨테이너 클러스터, 이종현)
- Multi-Tenancy in Kubernetes: Best Practices Today, and Future Directions: <https://www.youtube.com/watch?v=xygE8DbwJ7c>

1.1 Multi-Tenancy Kubernetes Remind

Multi-Tenancy는 효율적이지만 사용에 제한이 생긴다



Pros

- 모든 사용자가 Kubernetes 자체를 위한 서버를 보유할 필요가 없다.
- 모든 사용자가 Kubernetes에 여유 리소스를 보유할 필요가 없다.
- 모든 사용자가 Kubernetes를 직접 운영할 필요가 없다.



Cons

- 어느정도 신뢰성 있는 사람들끼리만 사용할 수 있다.
- 사용자가 Kubernetes의 모든 기능을 다 사용할 수 없다.
- Kubernetes가 매우 커지고 사용자도 많기 때문에 잘 운영하기 힘들고 할 일이 많다.

1.2 대규모 클러스터

클러스터가 커질수록 다양하고 복잡한 이슈들이 발생

- 관리해야하는 클러스터 노드 수가 많음
k8s 버전 업그레이드만 하려해도 한달 넘는 일정을 잡고 순차적으로 진행해야 한다
- 관리 서버의 과부하로 인한 장애 발생 가능성 높음
많은 수의 Node와 Pod으로 인해 ETCD, api-server에 많은 부하를 주게 된다
- 여러 요인에 의해 클러스터 노드 간 리소스 불균형 발생
Long running pod, 리소스 사용량이 많은 pod, 스케줄러의 분배 불균형으로 노드 간 편차가 심하다

1.3 첫번째 심각한 장애 사건을 만났을 때

NCC.AAA 클러스터 장애 공지 드립니다.

발생 시간과 클러스터

- xxxx.xx.xx 06:00 ~ xxxx.xx.xx 23.45 (17시간)
- AAA 클러스터

발생 원인

- 클러스터 내 다수의 dockerd 가 사용하는 메모리가 급증하였습니다.

발생 현상

- 일부 노드 메모리 부족으로 인한 서비스 응답 지연/Hang/OOMKill
- 문제 노드 서비스 제외 작업에 의한 Pod 재시작

작업 내용

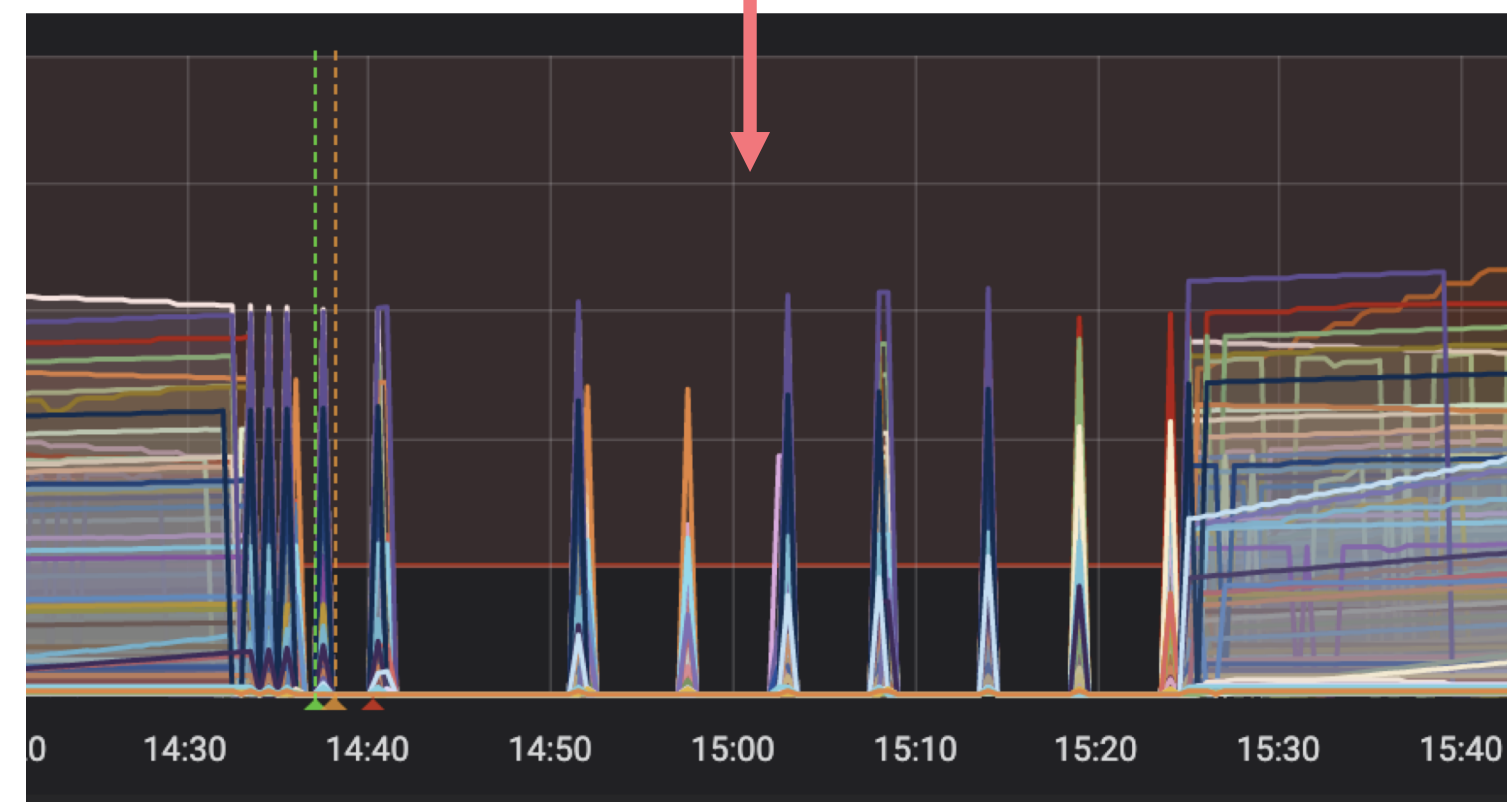
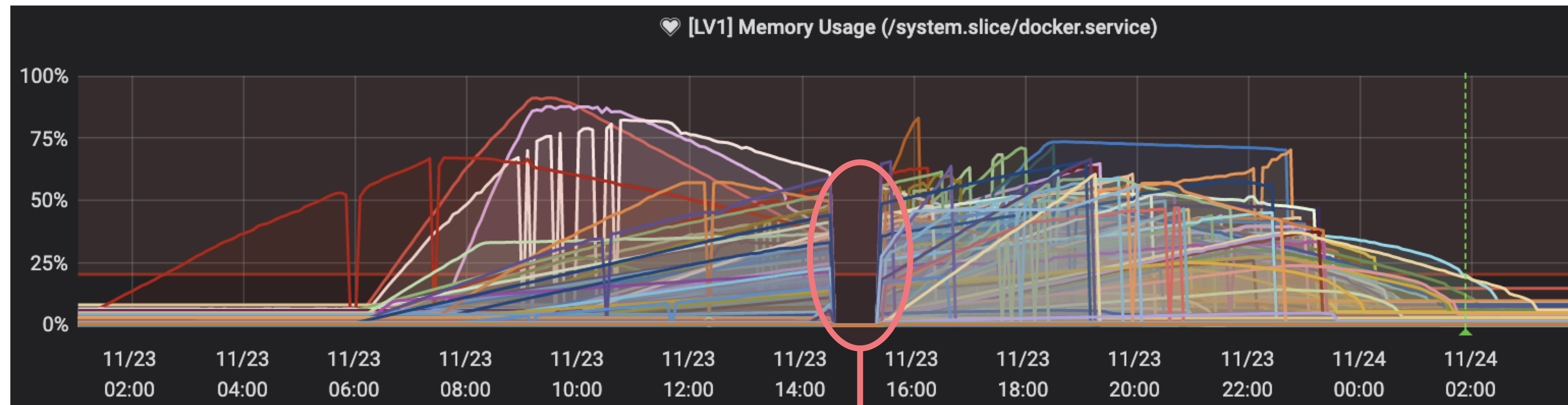
- xxxx.xx.xx ~ 10:00 : 일부 노드 메모리 부족이 확인되어 Drain & Restart 실행
- xxxx.xx.xx ~ 12:30 : 일부 노드가 아니라 AAA 다수 노드에서 발생 확인
- xxxx.xx.xx ~ 15:00 : 문제 노드들 Drain & Restart 로 Clear, 노드 증설 준비
- xxxx.xx.xx ~ 16:00 : 문제 노드로 인한 api-server 장애로 메모리 설정 변경 후 복구
- xxxx.xx.xx ~ 21:00 : 증설 예정 노드 긴급 증설
- xxxx.xx.xx ~ 23:45 : 추가 문제 노드들 Drain & Restart 진행

토요일 새벽에 발생

장애 인지 후 팀 전체가 비상 소집되어 장애 복구 투입
팀의 존폐를 위협 받은 순간

1.3.1 클러스터 노드 연쇄 장애 현상

노드마다 연쇄적으로 dockerd 메모리 급증



클러스터 사용자 서비스 Pod에도
연쇄적으로 hang 또는 oom killed로
Pod 재시작 영향 받음

1.3.2 클러스터 노드 연쇄 장애 원인

docker 진영에서 보고되어 있는 관련 이슈

<https://github.com/moby/moby/issues/40183>

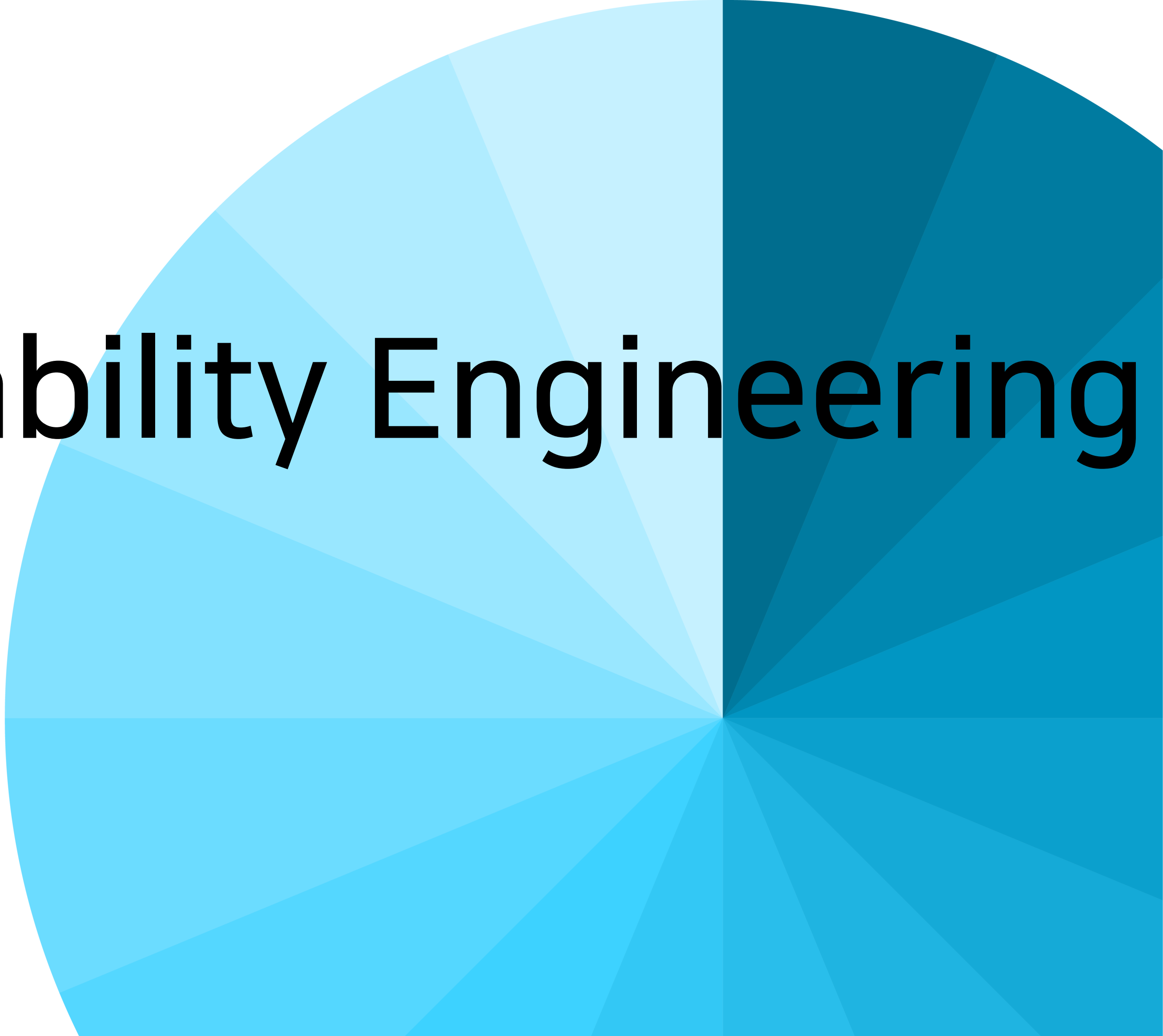
- cgroup OOM Killed가 발생할 때, container가 죽지 않는 상태가 된다
- dockerd가 hostconfig.json과 config.v2.json 파일을 무한히 쓰면서 write io 가 높아지고 이로인해 메모리 급증
- kernel 버전 **4.19.77**, 5.2.19 and 5.3.4 or above 에서 fixed

1.4 포스트모템을 통해 우리의 현실을 회고

현재 DevOps 전반을 돌아보고 부족한 부분을 바로 잡는 시간을 가짐

- 평소에 알람을 **너무 자주 받아서** 알람에 대한 피곤함이 상승했었다
- 장애시 발생한 알람 **패턴이 자주 발생하던 알람과 비슷**해서 장애 인지가 늦었다
- 운영이나 장애 **담당자가 없고** 일부 팀원이 대부분 작업을 커버하고 있었다
- 릴리즈 프로세스가 있었지만 **공통된 컨센서스**가 이뤄진 운영 프로세스가 없었다
- 장애가 발생한 노드 복구를 **수작업**으로 하다 보니 복구 시간이 오래 걸렸다
- 현재 클러스터의 노드나 리소스 **증설 시점**을 인지할 수 있는 알람과 대시보드가 있으면 좋겠다
- 특정 노드에 부하가 쏠리는 현상 등 **잠재적인 장애 유발 요인**을 찾아서 제거해야 한다

2. Container Reliability Engineering



2.1 SRE 기본 원칙

구글에서 소개하는 SRE 기본 원칙 요약

모니터링

- 알람은 최대한 false positive를 줄인다

릴리즈 엔지니어링, 변경관리 체계

- 서비스가 안정적으로 실행되려면, 안정적인 릴리즈 프로세스가 있어야 한다

2.1 SRE 기본 원칙

구글에서 소개하는 SRE 기본 원칙 요약(con.)

토일(toil) 제거, 비상대응 체계

- 엔지니어링에 집중한다
- 운영 시간은 최대 50% 넘지 않게 하고, 코딩하는데 사용한다

자동화

- 운영 중인 시스템에서 수작업을 최소화하려는 노력이 필요하다.
- 반복되는 운영 수작업은 하지 말자
 - " 애초에 수작업을 싫어하는 사람들로 팀을 구성해서 시스템 운영을 맡기고, 자동화가 필요한 부분은 직접 개발해서 운영하도록 한다"

2.2 NCC Reliability Engineering

SRE는 옵션이 아니라 필수이다

매주 1회 SRE TIME

- SRE 세미나 진행
- 주간 클러스터 운영 관련 이슈 공유
- 이슈 중심으로 안정적인 클러스터 운영을 위한 토론
- 운영 정책과 변경 사항들에 대한 컨센서스를 공유하는 시간
- 대규모 클러스터를 안정적으로 운영하기 위한 고민
- 포스트모템

2.2.1 알람 체계 변경

운영 대응이 필요한 알람만 받자

- 알람 기준을 운영 작업 필요 여부로
- LV1, LV2, LV3 알람 레벨 조정
- SMS 제외, 메신저로만
- 대응 알람과 리포트성 알람으로 채널 분리
- 리포트성 알람은 메일로만
- 통합 운영 리포트, 일일 이벤트 추이 리포트

알람 레벨

- LV1: 5분내 대응
- LV2: 업무시간 내 대응
- LV3: 한주에 한번 추이 확인

알람 대응 정비

- 장애 담당자(Master/Backup)가 처리
- SMS 는 제외, 메신저로만 일원화
- LV3 는 메일로만 전송
- 개별 노드의 장애는 Pause, Drain 으로 처리
 - Pause, Drain 의 비율에 따라 LV1 처리하고 나머진 다음 업무일 오전 Noti 로 처리
 - Pause 나 Drain 은 자동화
- 전체 서비스에 영향 받는 것만 LV1 으로 처리
 - 예) api-server 가 모두 죽음 (접근 불가), 노드의 5% 이상이 제외됨 등

2.2.2 릴리즈 체계 변경

배포 등급별로 릴리스 프로세스를 다르게

- 속도와 안정성을 모두 득하는 방향으로
- 다같이 합의하에 공통의 컨센서스 가지는 것이 중요

배포 절차

1. 배포 담당자는 배포 템플릿 내용을 작성해서 release 에 등록
2. 배포 등급 LABEL TAG
3. MINOR 등급 배포는 특별한 제한 없이 셀프 배포합니다 : release 이슈 등록, TTS 공지 후 배포

배포 등급 Label	배포 영향도	배포 정책
MAJOR	서비스에 큰 영향을 줄수 있음. (Pod 재시작, 단절)	MAJOR 등급 배포 절차로 진행
MINOR	서비스에 영향이 없거나 크지 않음 (순단)	셀프 배포
HOTFIX	서비스에 이미 영향을 주거나 줄것으로 예상됨	긴급 배포로 진행

MAJOR 등급 배포

- 개발에서 1주 이상 정상 확인 -> 서비스 개발에서 1주 이상 정상 확인 -> REAL CLUSTER 배포
1. MAJOR 등급 배포는 반드시 리뷰어를 1인 이상 지정하고, 최소 2인 1조로 배포 진행
 2. TTS 공지 후 배포
 3. 전체 사용자 공지가 필요하다고 판단되면 post 공지 요청
 4. 배포 진행 내용을 댓글 기록: 1인은 배포 진행, 1인은 모니터링 및 히스토리 기록
 5. 배포 완료 후 모니터링 진행: 배포관련 기본 테스트 진행, 해당 클러스터 DailyQA 정상 확인, 확인한 내용 댓글 작성
 6. 배포 중 개선이 필요한 부분이나 공유할 내용 있으면 포스트모템 작성후 팀내 공유

HOTFIX 등급 배포

- 기본 원칙은 무조건 긴급 배포로 진행
1. 가능하면 리뷰어를 1인 이상 지정하고, 최소 2인 1조로 배포 진행
 2. 배포 진행 내용을 댓글 기록: 1인은 배포 진행, 1인은 모니터링 및 히스토리 기록, TTS 등록
 3. 배포 완료 후 모니터링 진행: 배포관련 기본 테스트 진행, 해당 클러스터 DailyQA 정상 확인, 확인한 내용 댓글 작성
 4. 전체 사용자 공지가 필요하다고 판단되면 리더에게 요비 post 공지 요청
 5. 이슈가 해결된 부분이나 개선이 필요한 부분은 포스트모템 작성 후 팀내 공유

MINOR 등급 배포

1. release 이슈 등록 : MINOR LABEL TAG
2. TTS 공지 후 배포

배포가 여러 업무일동안 지속되면

1. 자동 스크립트에 의해 진행되는 것은 전체 수행 기간을 TTS 1건으로 등록
2. 수작업이 있으면 1일 단위로 끊어서 TTS 1건으로 재등록
3. Release 이슈는 한건으로 작성 가능하나 수작업이 있는 경우 그 내용을 1일 단위로 comment 해야 한다.

배포 중 장애가 발생하거나 이슈가 발생하면

- 기본 원칙은 무조건 포스트모템 작성
- 필요시 장애 보고서 작성하고, 요비 post 를 통해 전체 사용자 공지
- 포스트모템은 주단위로 다같이 보고 의견 교환하는 형식으로 (주간회의, SRE TIME 에서)

2.2.2 릴리즈 체계 변경

클러스터 업데이트 정책 설정

- 변경 작업이 있을 때 장애 발생 확률이 가장 높다
- 관리하는 노드와 Pod의 수가 많이 때문에 운영 컴포넌트 업데이트는 주의가 필요하다

업데이트 원칙

- MAJOR 는 노드 1대씩 Drain 작업을 원칙으로 한다.
- Running 중인 Pod/Container 에 영향이 없으면 서비스 영향도를 고려하여 Drain 하지 않을 수 있다.
 - Master/ETCD/LB 노드는 1대씩 작업 한다.
 - Worker 는 전체 노드의 25% 이하만 작업 한다.
- 모든 작업 은 변경 후 알람을 수신할 수 있는 대기 시간을 둔다.
 - Worker 1대 작업 은 1분
 - Master/ETCD/LB 1대, Worker 여러대 작업 은 10분

변경사항 체크

- Kubernetes 의 버전 변경은 [ChangeLog](#) 와 [Admin Guide](#) 를 확인한다.

사용자 공지 정책

- Pod/Container 재시작 필요하면 2주 전 사용자에게 미리 공유해야 한다.
- MAJOR 는 TTS 등록시 전체 사용자 메일 수신 설정한다.
- MINOR 는 TTS 등록시 수신자는 팀 메일 수신 설정한다.

2.2.3 운영 대응 체계 변경

운영 업무는 무조건 50%이하로 나머지는 코딩 시간

- 2인 1조(Master/Sub)로 2주간 운영 담당자 역할
- 모든 사용자 문의와 클러스터 알람 대응
- 비상 상황 발생시 운영 오퍼레이터 담당
- 운영 포인트

				담당	0		담당	1		담당	2		담당	3		황영석	4	
포인트 현황					1295	1120		935	950		900	1080		1105	1100		955	940
정상 대응 여부				84	13.21	13.33	71	13.16	13.38	69	13.04	13.01	84	13.15	13.09	70	13.64	13.42
2020-04-14	1	1	1		0	0		0	0	1	10	10		0	0		0	0
2020-04-15	3	1	1		0	5		5	5	1	20	20		5	0		0	0
2020-04-16	1	1	1		0	0		0	0	1	10	10		0	0		0	0

2.2.4 자동화

한 번 이상 수행되는 모든 수작업은 자동화해야 한다

현재 자동화로 돌고 있는 작업들

- 커널 업그레이드 + k8s 버전 업그레이드
- 비정상 노드 서비스 제외 처리
- reboot 된 노드 서비스 투입
- dockerd 메모리 증가 시 복구
- PLEG 동기화 지연/실패 시 복구

목적

drain 또는 reboot이 필요한 upgrade를 가능한 한번에 처리
PM 등록은 자동화 할 수 있도록 되어있습니다

대상

- Master, LoadBalancer node는 수작업으로 진행
- 나머지 Worker를 자동으로 진행
 - 1시간에 1대 정도

확인 필요

- Master는 1.15.7 Worker는 1.14.10 일때 문제가 없는지 확인
- ES, Kafka의 처리 방안

작업

- k8s 1.15.7 upgrade
- kernel 4.19.93 upgrade
- tini apply

2.2.5 장애 대응

장애 발생시 역할 분담이 중요하다

- 클러스터 점검 영역 별로 담당자 지정
- 오퍼레이터는 해당 주간 운영 대응 담당자

장애 대응 역할 담당

클러스터 외부

- Build/Deploy, Ceph, ES/Kafka 등 외부 연동 서비스 점검
- 사용자 문의에 대한 증상 확인 및 대응

클러스터 내부

- Master, ETCD 등의 관리 노드의 상태, 정상 동작 유무, 리소스 사용량 등
- Worker 노드의 리소스 사용량 및 스케줄링 상태
- Grafana/Prometheus 등의 제공 서비스 이상 유무 점검

네트워크

- Pod to Pod 네트워크, SWLB, NAT 의 상태, 정상 동작 유무, 리소스 사용량 등

점검 항목

- 현재 정상 동작 여부와 장애 현상에 대한 힌트 찾기
 - 장애에 의한 파급도 분석 및 리포트
-

2.2.6 NCC SLIs/SLOs

클러스터 안정성을 표현해주는 정량적인 지표를 측정하고 관리

- 목표 SLO: 99.5%

API call latency

SLI	SLO
모든 리소스에 대해 단일 object 에 대한 mutating API 호출에서 최근 5분 단위 99분위 latency 측정	per cluster-day 의 99분위 latency 가 <= 3s
모든 리소스에 대해 read-only API 호출에서 최근 5분단위 99분위 latency 측정	per cluster-day 의 99분위 latency 가 <= 1s

- mutating : POST, PUT, DELETE, PATCH
- read-only : GET requests without watch=true, non-streaming

Pod startup latency

SLI	SLO
스케줄 가능한 stateless pods 의 startup latency, 최근 5분단위 99분 위 latency 측정	per cluster-day 의 99분위 latency 가 <= 5 s
스케줄 가능한 stateful pods 의 startup latency, 최근 5분단위 99분 위 latency 측정	per cluster-day 의 99분위 latency 가 <= 10 s



[참고]
 k8s 공식 SLIs/SLOs: <https://github.com/kubernetes/community/blob/master/sig-scalability/slos/slos.md>

2.3 DevOps 변화

어느새 1년이 되어 팀을 돌아보니 자연스럽게 녹아있는 SRE

- 운영 담당자는 **꼭 필요한 알람만** 받고 있고 메뉴얼 대로 빠른 대응을 하고 있다
- 한 두 사람에게 집중되었던 사용자 문의 및 운영 대응의 로드가 **팀 전체로 분산**되었다
- 변경된 릴리즈 체계에 대한 **공통의 컨센서스**가 있고 의도대로 정착되어 운영되고 있다
- 자주 발생하는 알람 대응은 대부분 **자동화**되어 수작업 대응 회수가 획기적으로 줄었다
- 클러스터 **증설 시점을 예측**할 수 있는 알람 및 대시보드를 구성해서 활용 중이다
- 매일 발송되는 **통합 운영 리포트, 이벤트 리포트**를 통해 클러스터 전체 상황을 한 눈에 파악하고 있다
- 장애 발생시 일사분란하게 움직여서 **빠르게 복구**를 하고 있다

2.4 그러면 잠재적인 장애 유발 요인은?

다시 Multi-Tenancy Kubernetes의 대규모 클러스터 이슈에서

- 관리해야하는 클러스터 노드 수가 많음
k8s 버전 업그레이드만 하려해도 한달 넘는 일정을 잡고 순차적으로 진행해야 한다
완화 방법: **릴리즈 프로세스를 잘 만들고 수작업을 자동화한다**
- 관리 서버의 과부하로 인한 장애 발생 가능성 높음
많은 수의 Node와 Pod으로 인해 ETCD, api-server에 많은 부하를 주게 된다
완화 방법: **증설 시점을 예상해서 미리미리 증설하고 클러스터 업데이트 프로세스를 잘 만든다**
- 여러 요인에 의해 클러스터 노드 간 리소스 불균형 발생
Long running pod, 리소스 사용량이 많은 pod, 스케줄러의 분배 불균형으로 노드 간 편차가 심하다
완화 방법: **3. Lesson Learned** 챕터에서 고민했던 내용 설명 한다

3. Lesson Learned

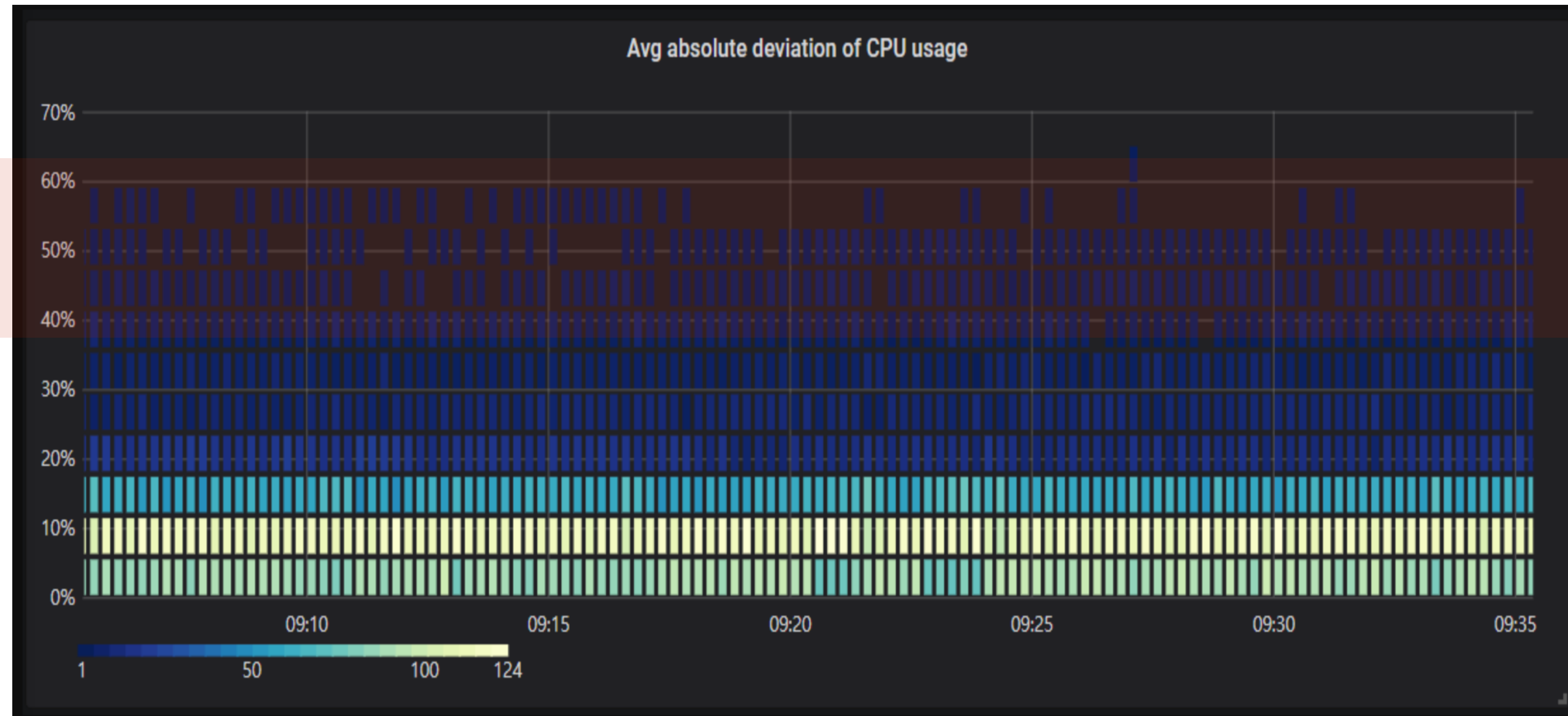
3.1 "Push the Limits" (한계를 뛰어넘다)

Multi-Tenancy K8s를 운영하면서 SRE 관점에서 해결하려고 했던 이슈들

- 여러 요인에 의해 클러스터 노드 간 리소스 불균형 발생
Long running pod, 리소스 사용량이 많은 pod, 스케줄러의 분배 불균형으로 노드 간 편차가 심하다
갑자기 **트래픽이 폭증하는 이벤트 상황**에서 특정 노드의 리소스가 고갈되어 장애로 이어질 수 있다
- Kubernetes 이벤트 관리
놓치고 지나간 k8s 이벤트에서 **장애 발생 유발 단서**를 미리 얻을 수 있지 않을까?
클러스터 사용자에게도 중요한 이벤트를 알려주면 장애를 미연에 방지하는데 사용할 수 있지 않을까?
- 기타 이슈: HPA 성능 이슈
지진이나 연예인 이슈와 같이 **짧은 시간에 대규모 트래픽**이 들어오는 상황에서도 빠르게 대응 가능하도록 충분히 빠른 오토스케일이 필요하다

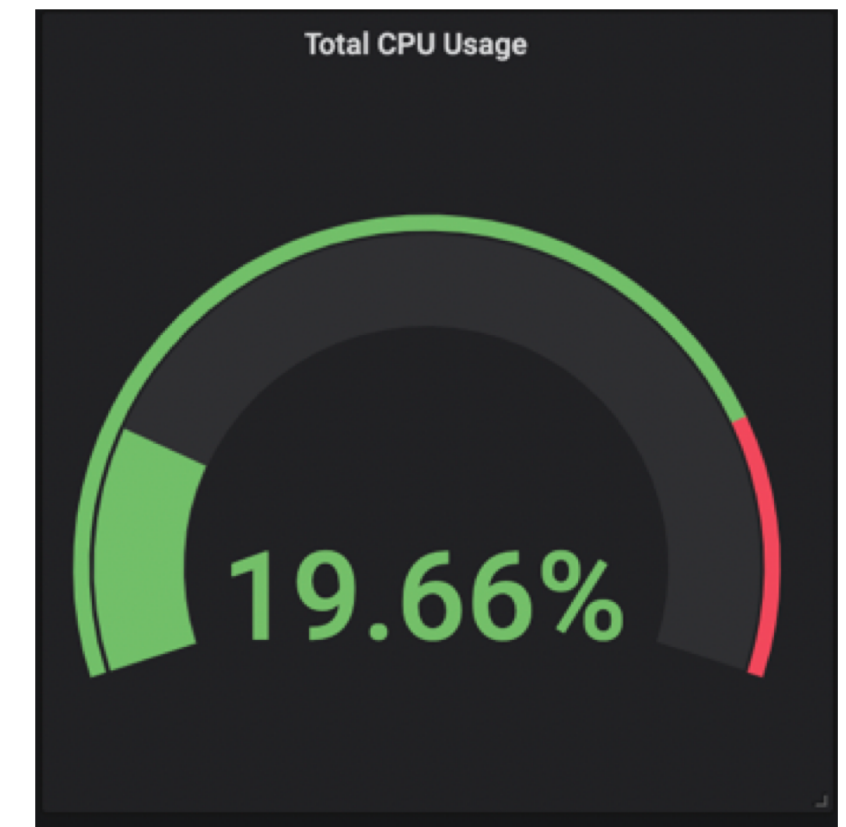
3.2 Pod Re-Balancing

클러스터 노드 간 리소스 불균형 현상



편차가 40%이상
나는 노드에 주목

평균 CPU 사용량



TOP 8 Nodes

CPU ▾
82.42%
77.99%
76.36%
74.36%
73.05%
71.66%
70.30%
70.24%

CPU 사용량 리소스 편차 그래프

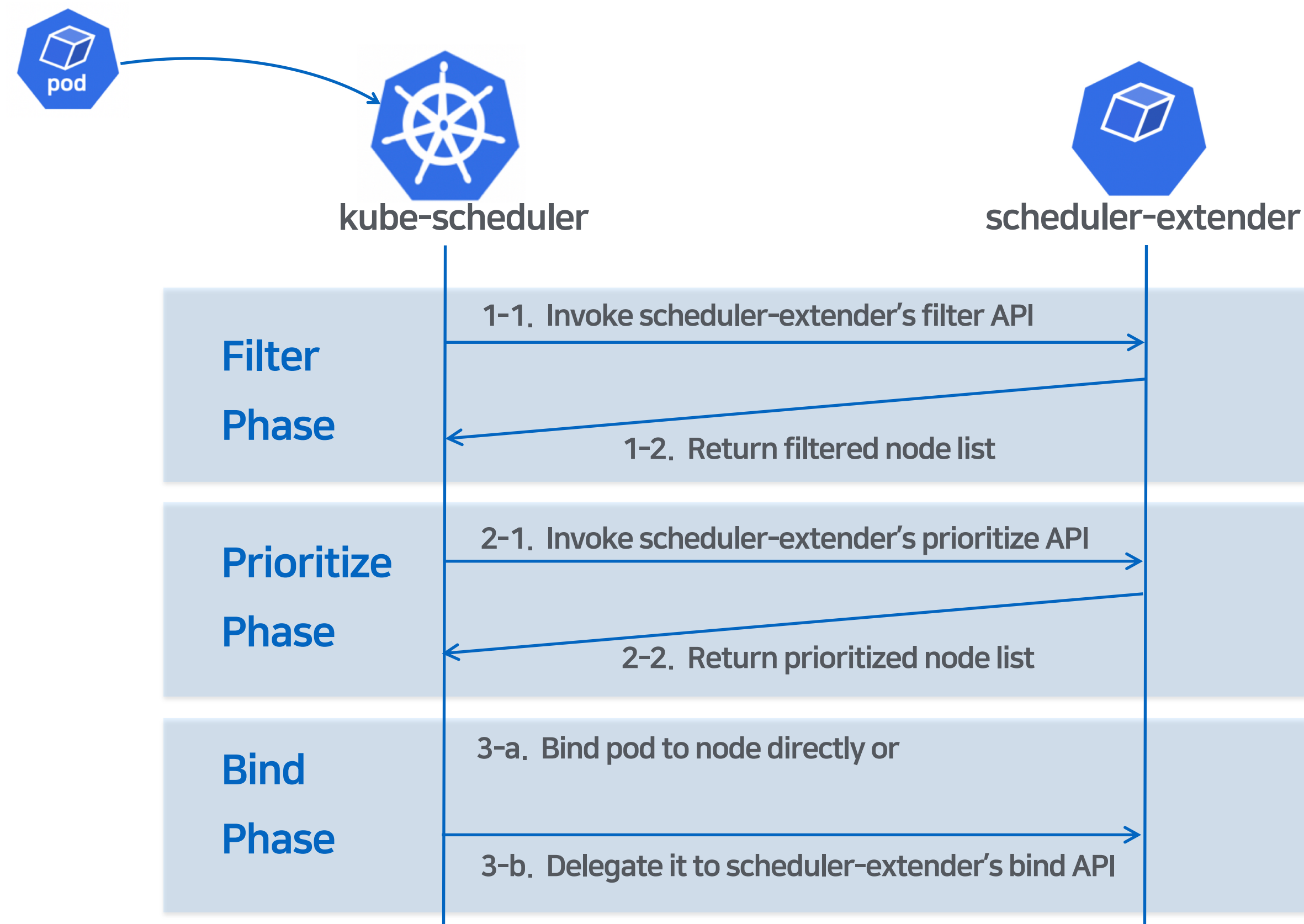
3.2 Pod Re-Balancing

클러스터 노드 간 리소스 불균형을 해소시키는 방법

- Scheduler Extender
리소스 사용량이 높은 노드는 피해서 Pod을 스케줄링 한다
- Pod Auto Rebalancer
그래도 리소스 사용량이 높은 노드가 있으면 Pod을 다른 노드로 이동시킨다

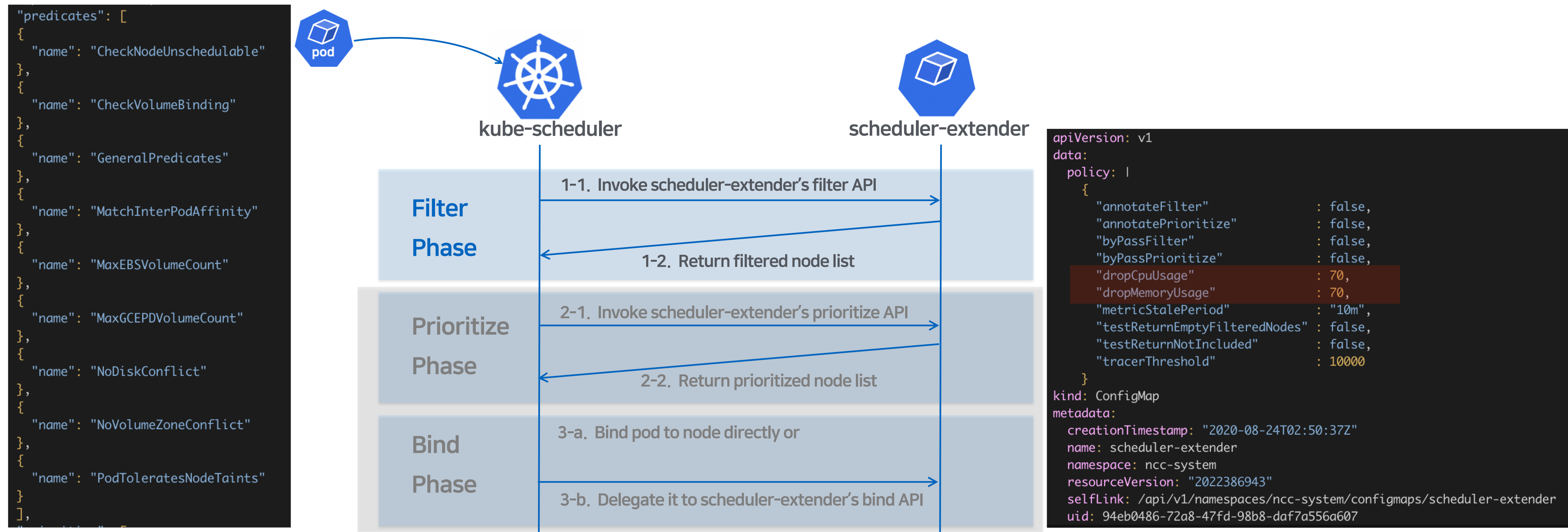
3.2.1 Scheduler Extender

리소스 사용량이 높은 노드는 피해서 Pod을 스케줄링한다



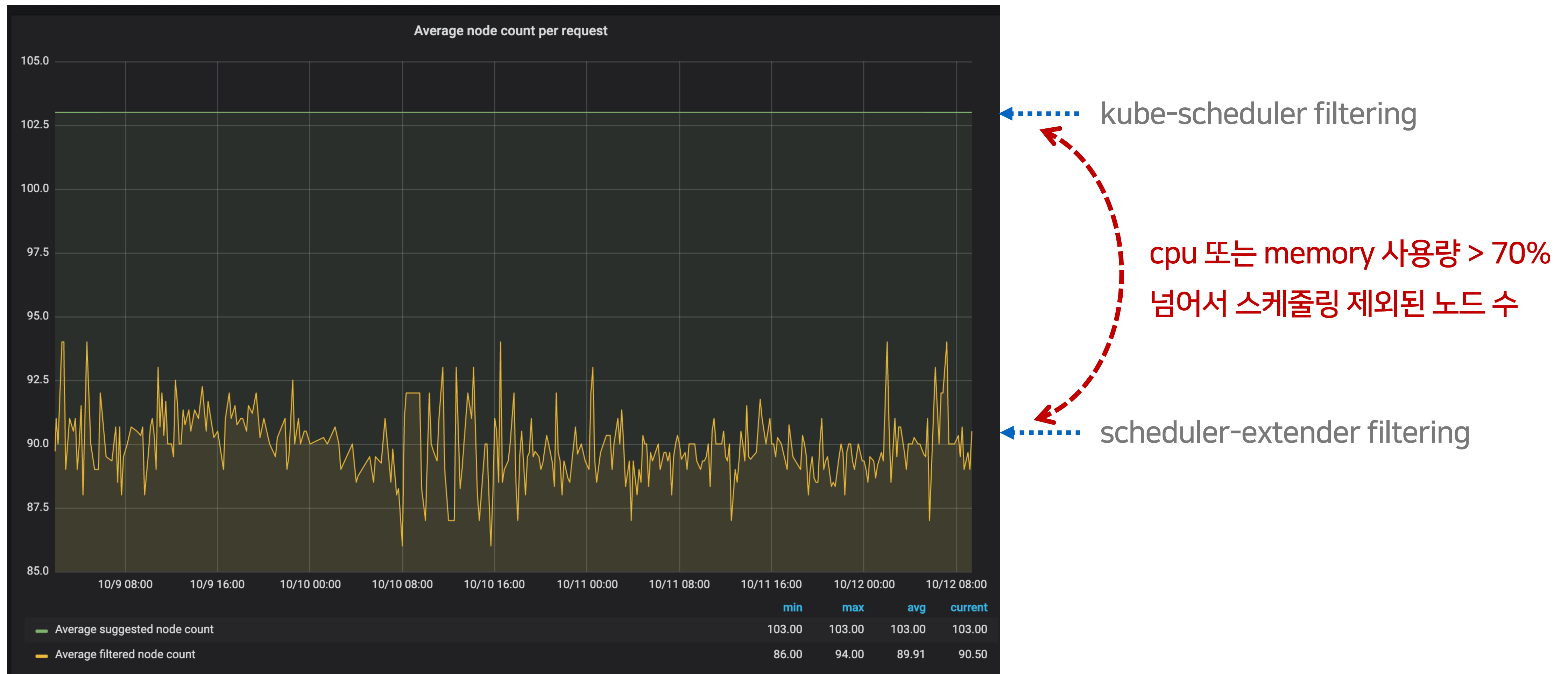
3.2.1 Scheduler Extender

리소스 사용량이 높은 노드는 피해서 Pod을 스케줄링한다



3.2.1 Scheduler Extender

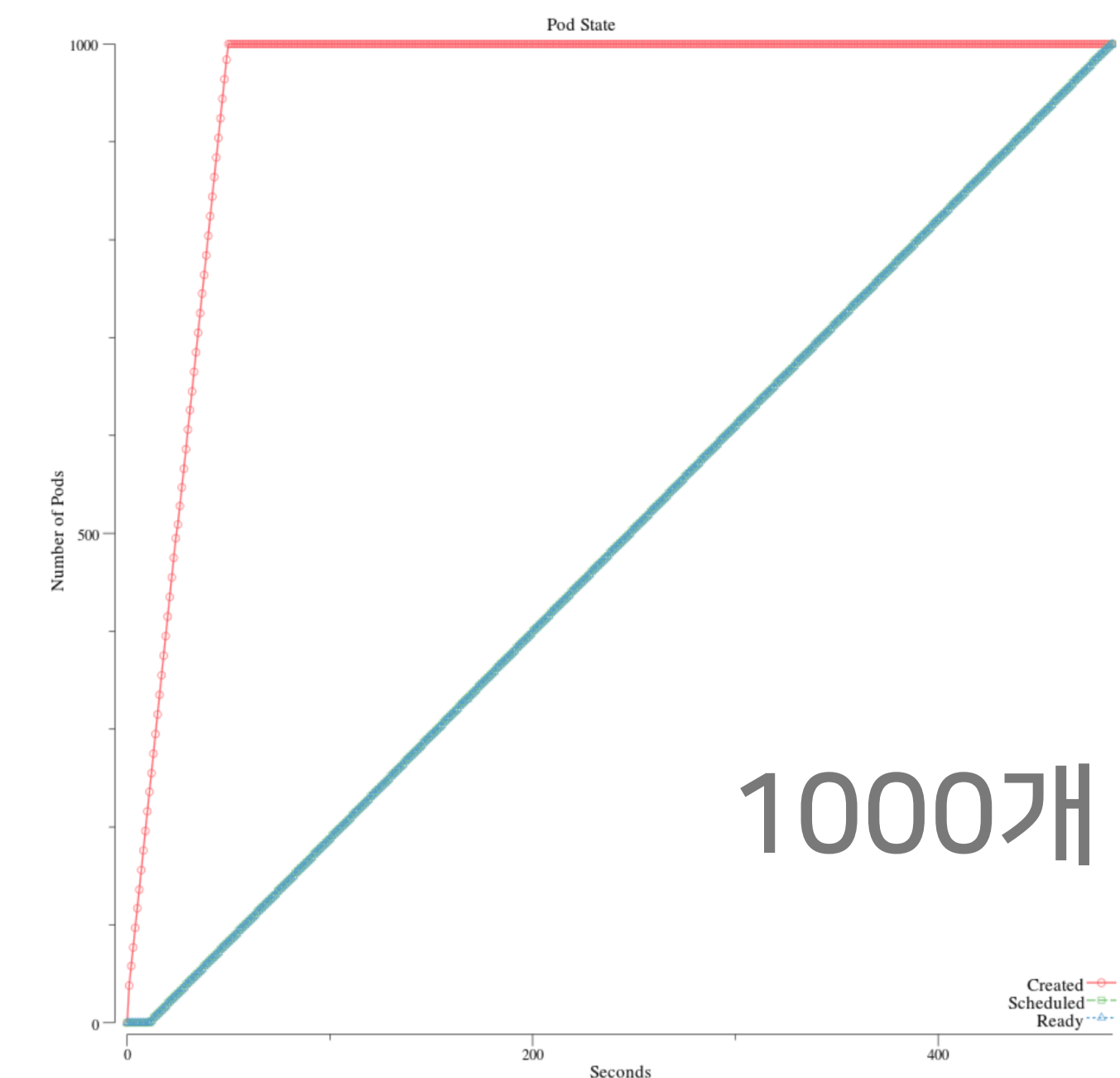
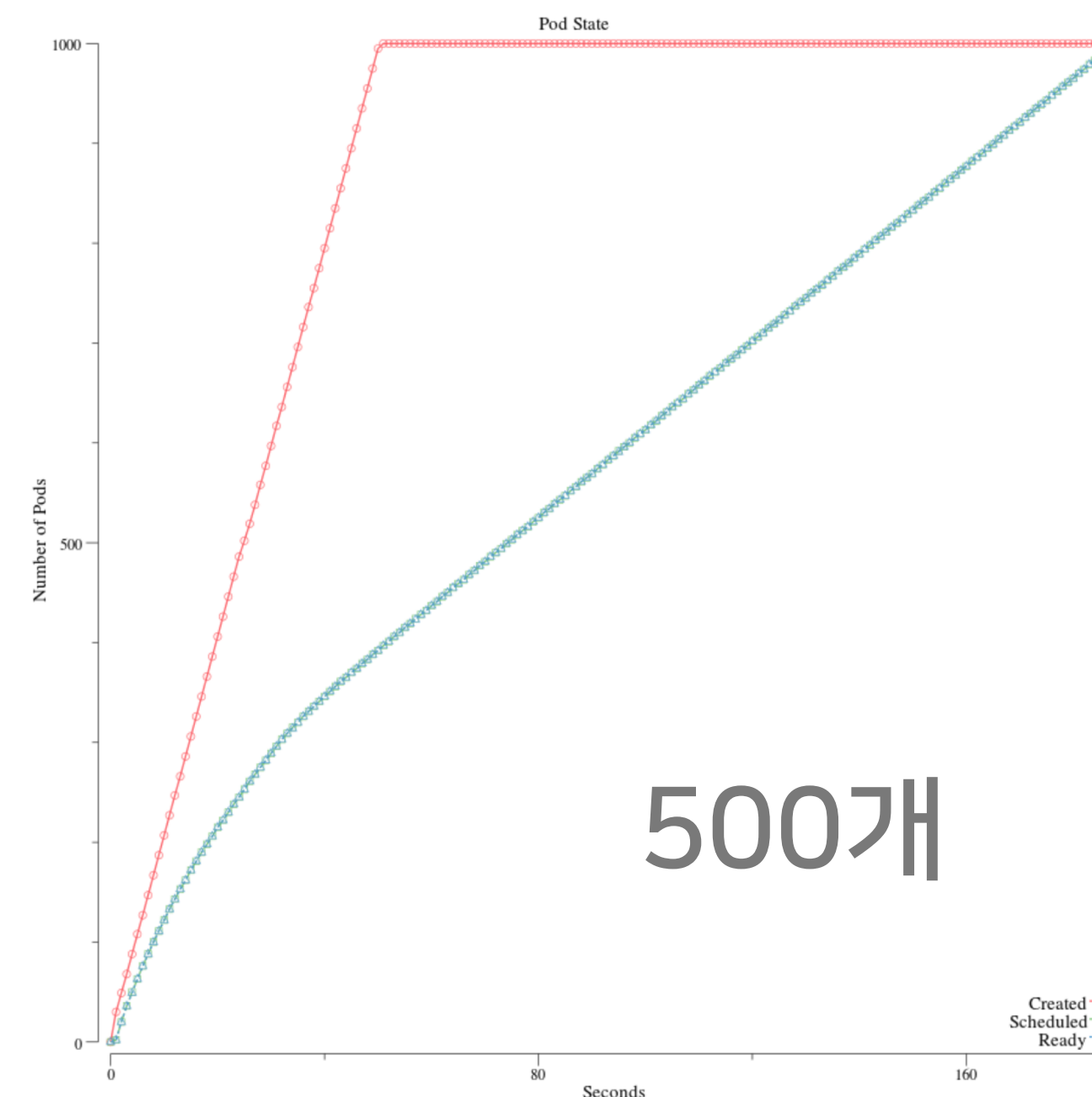
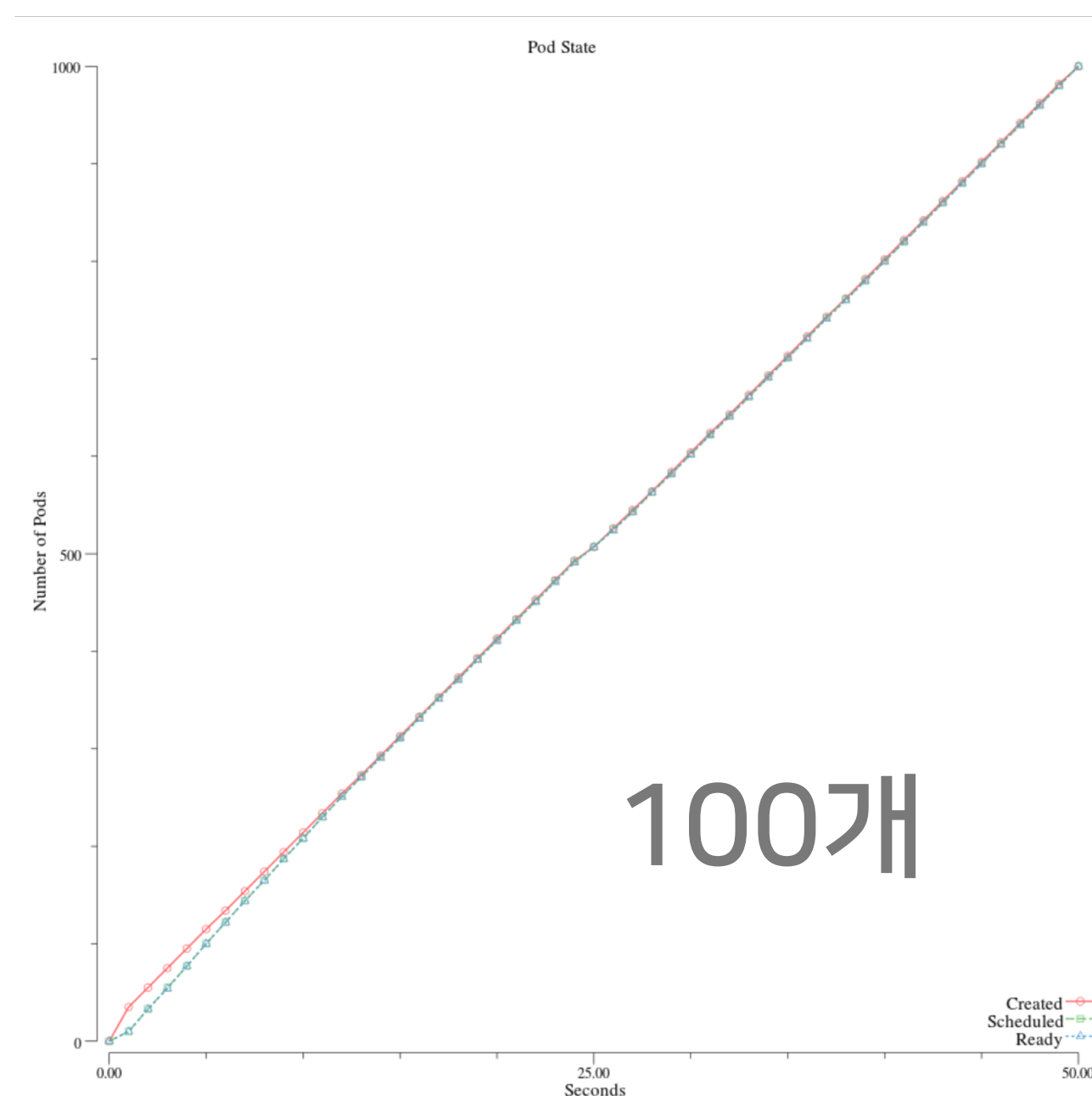
scheduler-extender를 통해 필터링된 결과



3.2.1 Scheduler Extender Tip

Kubernetes Scheduler Extender 성능 이슈

- 성능 이슈: low performance to schedule pod via scheduler extender (<https://github.com/kubernetes/kubernetes/issues/72634>)
- Both scheduler and extender need lots of time to Marshal/Unmarshal JSON



3.2.1 Scheduler Extender Tip

Kubernetes Scheduler Extender 성능 확인

- 300대 규모 Worker 노드 클러스터에서 테스트
- 노드 1대 정보 120kb 데이터 -> 300대 약 35MB
- json encoding/decoding 시간 + 네트워크 35MB x 2
- pod 1개 스케줄링 처리하는데 평균 1초 정도 소요

3.2.1 Scheduler Extender Tip

Kubernetes Scheduler Extender 튜닝 포인트

- scheduler-extender에서는 node cache 사용
- kube-scheduler <-> scheduler-extender 데이터는 node name list만 전송
- 노드 300개일 때 데이터 크기: 4504 bytes (약 4KB)
- pod 1개 스케줄링 처리하는데 99분위 평균 **72ms** 소요

3.2.2 Pod Auto Rebalancer

리소스 사용량이 높은 노드에서 Pod을 다른 노드로 이동시킨다

ReplicaSet

- cpu: 1시간에 4회
- memory: 1시간에 1회

조건

- "NodeCpuUsagePctThreshold": 80
- "NodeMemoryUsagePctThreshold": 80
- "TopNodes": 10
- "TopPods": 0.7

Project BE-Evictor-Trigger-Cronjob

매개변수가 필요한 빌드입니다.

CLUSTER_NAME	
TARGET_CREATED_BY_KIND	☐ DRY_RUN replicaset
TARGET_RESOURCE	cpu 대상을 선택할 리소스 기준. (eg. cpu : cpu 사용률이 높은 노드에서 cpu 사용률이 높거나 낮은 pod 선택)
SCHEDULE	*/15 * * * *
EVICTON_ORDER	desc
TOP_NODES	10
TOP_PODS	0.7
ALLOWED_STS_POD	^esfarm-cluster-(.*)-(datahdd)-[1-9][0-9]*(.*)-storm[0-9]*-supervisor-[0-9][0-9]*\$
NODE_CPU_PCT_THRESHOLD	80
NODE_MEM_PCT_THRESHOLD	80
NODE_PODS_PCT_THRESHOLD	85
NODE_NETWORK_USAGE_THRESHOLD	734003200
RELEASE_REASON	

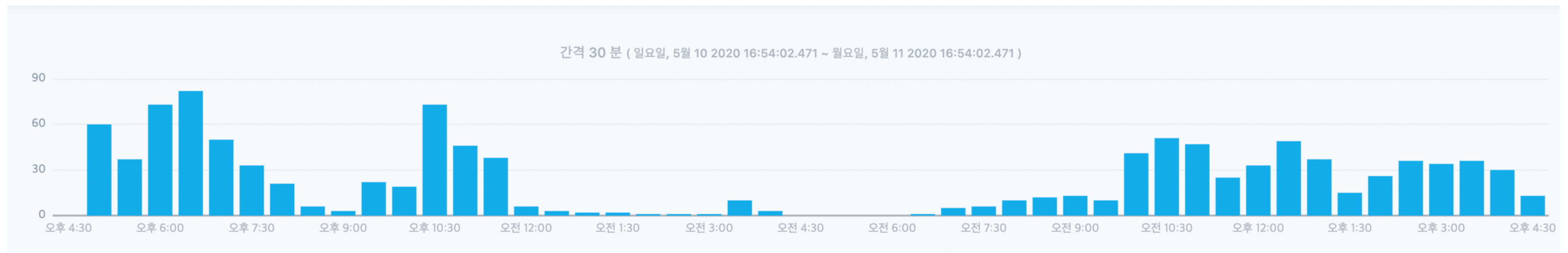
왜 이 배포 작업을 수행했는지 간단히 기술해 주세요.
작업 메일을 받은 사람들이 어떤 작업이 진행되었는지 파악하기 쉽도록요.

3.2.2 Pod Auto Rebalancer

하루 동안 Pod이 Rebalancing 된 횟수

🔍 projectName:"NCC-EVICTOR" AND body:"Succeed"

🔍 검색 ➕ 추가 📅 최근 24시간



```
I0511 16:45:04.689772      6 log_json.go:10] ===== Target Nodes =====
I0511 16:45:04.689806      6 log_json.go:15] {
  "worker1614": {
    "Name": "worker",
    "Description": "cpu_84.58333333317812"
  },
  "worker0816.ncc": {
    "Name": "worker0816.ncc",
    "Description": "cpu_82.69166666657354"
  }
}

I0511 16:45:05.817884      6 log_json.go:10] ===== [worker1614.ncc] Target Pods =====
I0511 16:45:05.818063      6 log_json.go:15] {
  "autocomplete-autocomplete-1-5-3-f566cd8d4-ndx9k": {
    "Name": "autocomplete-autocomplete-1-5-3-f566cd8d4-ndx9k",
    "Namespace": "naver"
  },
  "batch-naver2-r1-76cf9b8f79-4ksrx": {
    "Name": "batch-naver2-r1-76cf9b8f79-4ksrx",
    "Namespace": "naver"
  }
},
... (중 략)
```

```
I0511 16:45:05.824107      6 manager.go:154] [naver|autocomplete-autocomplete-1-5-3-f566cd8d4-ndx9k] Succeed to evict pod
I0511 16:45:05.831563      6 manager.go:154] [naver|batch-naver2-r1-76cf9b8f79-4ksrx] Succeed to evict pod
I0511 16:45:05.841561      6 manager.go:154] [naver|lapi-green-6c57967558-4pqvb] Succeed to evict pod
I0511 16:45:05.850908      6 manager.go:154] [shivaldeployment-6f8c788f44-5zfz6] Succeed to evict pod
I0511 16:45:05.858334      6 manager.go:154] [inflreal-v1-0-42-200511-163956-c5fb76dd5-c5zhx] Succeed to evict pod
I0511 16:45:05.870918      6 manager.go:154] [stglapi-7fb8b7dfb7-dcjs9] Succeed to evict pod
I0511 16:45:05.881871      6 manager.go:154] [cloudlapi-release-0-0-1-0-200511-161448-6867689sft65] Succeed to evict pod
I0511 16:45:05.881901      6 manager.go:158] [worker1614.ncc] 7 pods are evicted
```


3.2.2 Pod Auto Rebalancer

사용자에게 Pod Rebalancing 결과를 제공

NCC > Eviction History

Rebalanced pods

	node	pod
16:15:15		notout-egg-001-65bc9d5cf8-d42bw
		notout-egg-001-65bc9d5cf8-d4f5h
		notout-egg-001-65bc9d5cf8-ng4jz
16:00:22		notout-egg-001-65bc9d5cf8-6xrm2

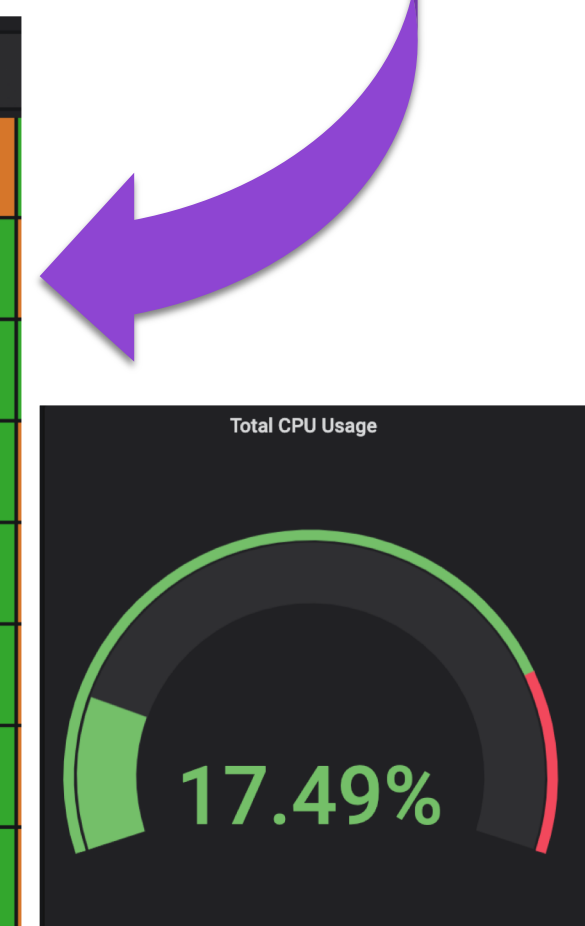
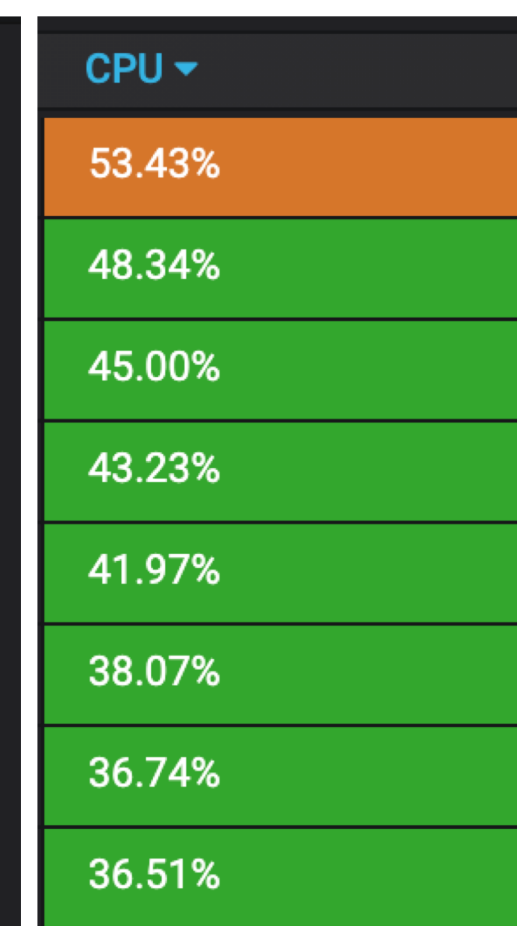
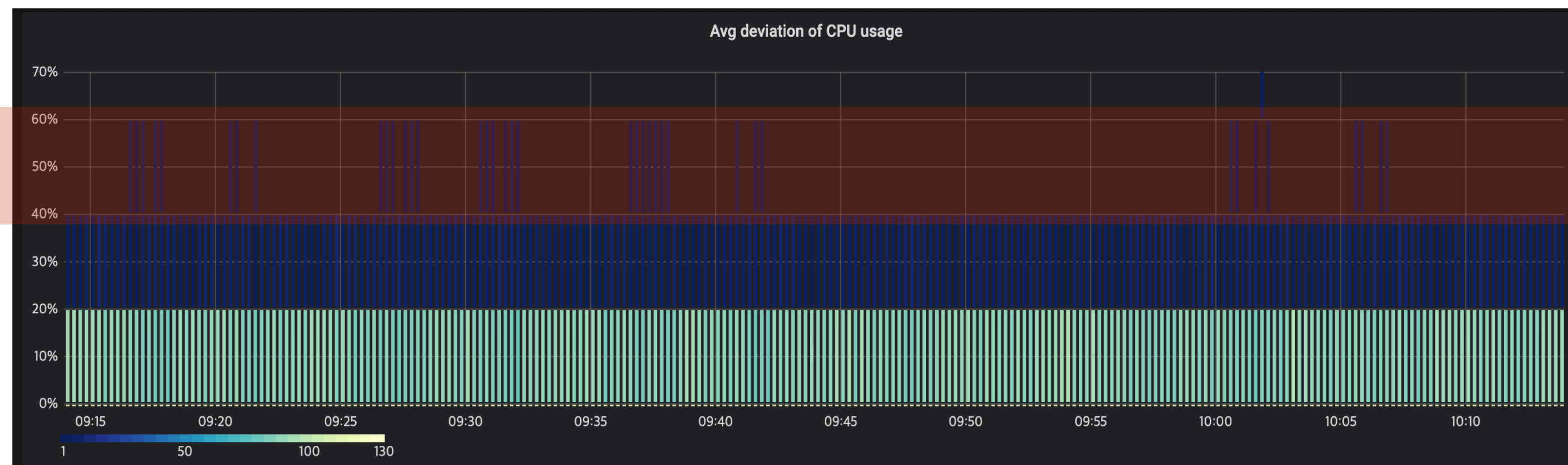
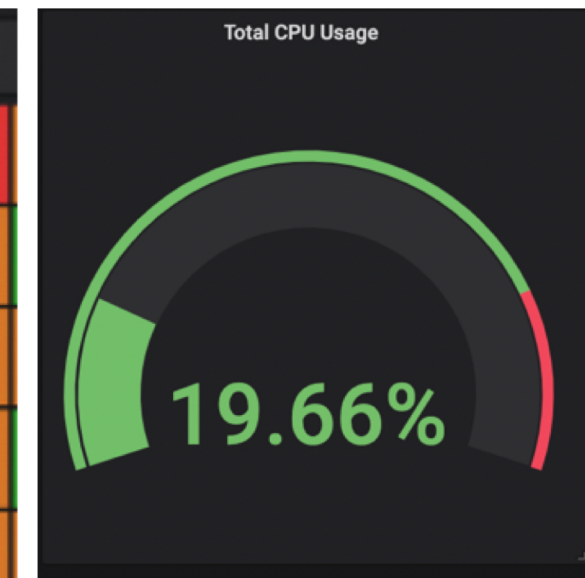
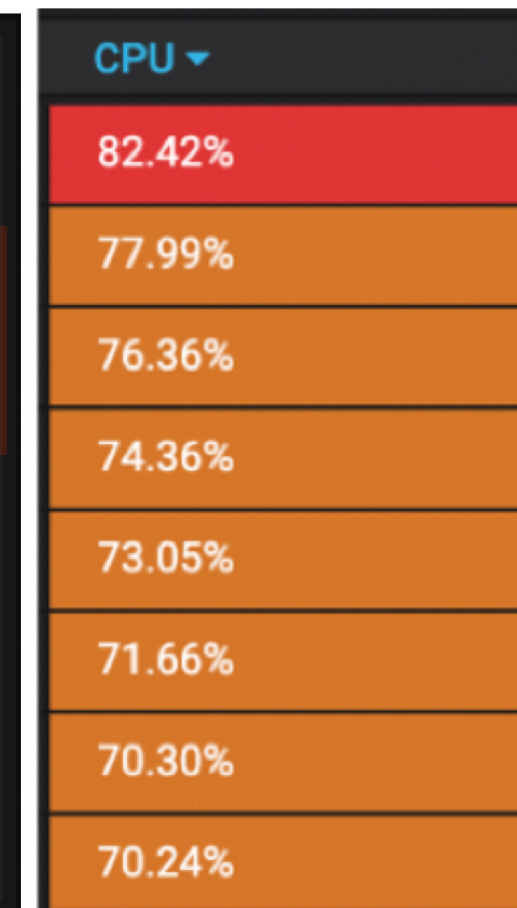
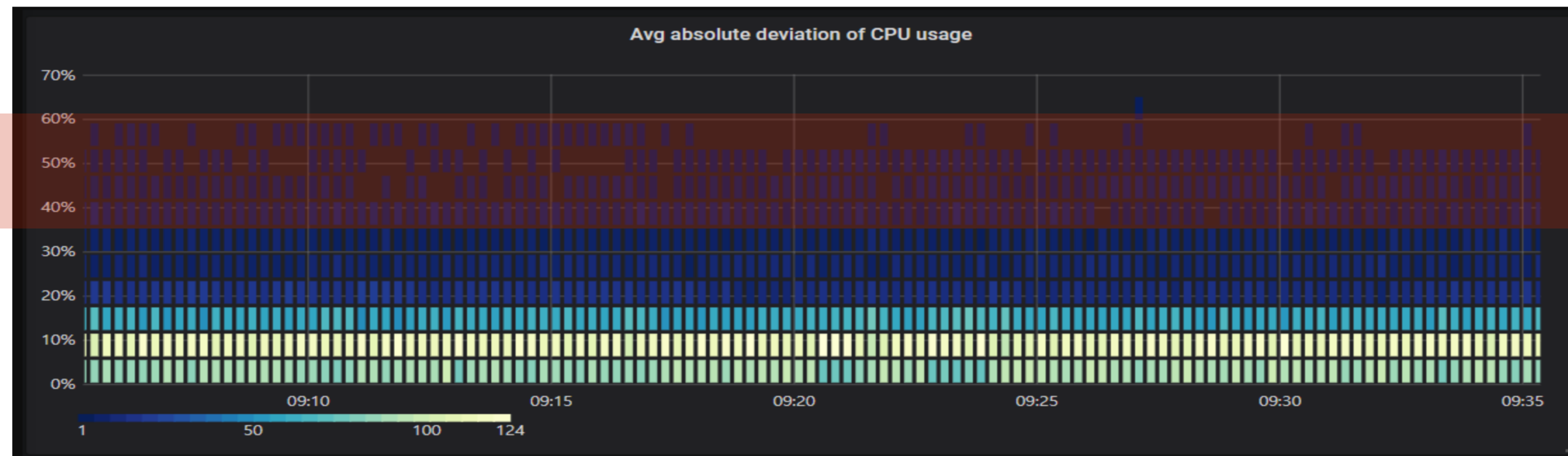
Evicted/Killed pods

LastTimestamp	Namespace	Pod	Message	Reason
16:45:33+09:00	ncc-be	notout-egg-001-65bc9d5cf8-d42bw	Stopping container shell	Killing
16:31:14+09:00	ncc-be	notout-egg-001-65bc9d5cf8-d4f5h	Stopping container shell	Killing
16:20:56+09:00	ncc-be	qa-dns-rule-test-sts-0	Stopping container http	Killing
16:15:15+09:00	ncc-be	notout-egg-001-65bc9d5cf8-ng4jz	Stopping container shell	Killing
16:00:22+09:00	ncc-be	notout-egg-001-65bc9d5cf8-6xrm2	Stopping container shell	Killing

NCC Rebalancing rule에 의해, 다른 노드에서 운영되도록, 현재 노드에서 제거된 Pods list 입니다.

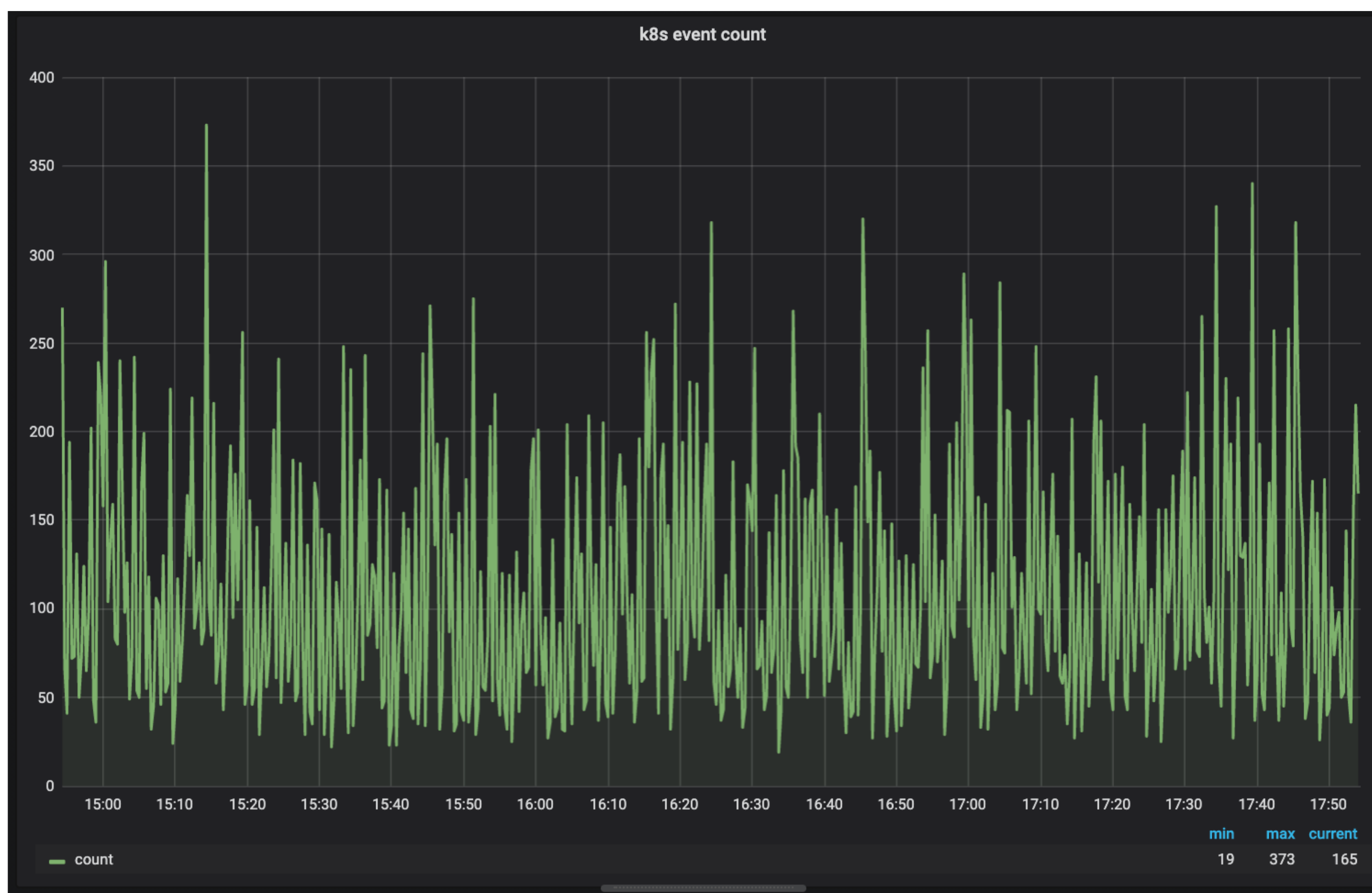
3.2.3 Pod Re-Balancing 결과

클러스터 노드 간 리소스 불균형이 많이 해소됨



3.3 Kubernetes 이벤트 관리

Multi-Tenancy K8s 특성상 다양한 이벤트가 많이 발생한다



- 중요한 이벤트를 놓치지 않고 캐치할 필요가 있다
- 반복되는 이벤트, 주기적으로 발생하는 이벤트, 짧은 시간 많이 발생하는 이벤트
- 이벤트 추이도 추적할 필요
- 클러스터 운영, 서비스에서 모두 의미 있는 이벤트 일 수 있다

3.3 Kubernetes 이벤트 관리

Kubernetes 이벤트에서 장애 발생 유발 가능성이 있는 단서를 찾는다

- sentry

유사한 로그나 이벤트를 하나의 티켓으로 관리하고 발생 횟수 등을 시각화해서 보여주는데 뛰어난 툴

- sentry-kubernetes

Kubernetes 클러스터의 warning이나 error를 sentry로 전송하는 역할

Errors and warnings in Kubernetes often go unnoticed by operators. Even when they are checked they are hard to read and understand in the context of what else is going on in the cluster. `sentry-kubernetes` is a small container you launch inside your Kubernetes cluster that will send errors and warnings to Sentry where they will be cleanly presented and intelligently grouped. Typical Sentry features such as notifications can then be used to help operation and developer visibility.

(우리랑 비슷한 고민)

[참고]

sentry: <https://sentry.io/welcome/>

sentry-kubernetes: <https://github.com/getsentry/sentry-kubernetes>

3.3 Kubernetes 이벤트 관리

sentry + github으로 관리

- 운영 대응이 필요한 이슈만 필터링해서 등록
- 특정 노드에서만 발생하는 이벤트는 모아서
- 빈도수가 많은 이벤트는 표시해서 확인
- 중복되는 이슈는 닫고 댓글로 기록
- 서비스 담당자가 알아야 되는 이벤트는 정리해서 리포트로 전송(예정)

(github에 자동 등록된 이슈 샘플)

☐

!

[aaa][worker2508] 2020-10-14 노드 이벤트

aaa

worker2508

#2894 opened 6 hours ago by sentry-app-cp

bot

☐

!

[ccc][18460] 2020-10-14 Deleted job: jobs.batch "crow-manage-1602575400" not found

ccc

n/a

ncc-system

#2893 opened 6 hours ago by sentry-app-cp

bot

☐

!

[aaa][42577] 2020-10-14 The node had condition: [DiskPressure].

aaa

kube-system

worker2508

#2892 opened 6 hours ago by sentry-app-cp

bot

☐

!

[aaa][35515] 2020-10-14 Readiness probe failed: HTTP probe failed with statuscode: 503

aaa

kube-system

worker013

#2891 opened 7 hours ago by sentry-app-cp

bot

☐

!

[aaa][75605] 2020-10-14 Error: cannot find volume "default-token-cg8nc" to mount into container "ini...

1d 100+

1h 100+

aaa

naver

worker0525

#2890 opened 10 hours ago by sentry-app-cp

bot

☐

!

[aaa][313] 2020-10-13 Readiness probe failed: timeout: failed to connect service ":80" within 1s

1d 100+

1h 100+

aaa

naver

total 10k+

worker1111

#2885 opened 21 hours ago by sentry-app-cp

bot

3.4 HPA 성능 이슈

Multi-Tenancy Kubernetes의 HPA

- 일반적으로 HPA 운용 방식은 노드를 많이 만들어 두거나 실시간으로 노드를 늘리는 방식으로 오토스케일한다
- Multi-Tenancy 방식에서는 노드를 공유하기 때문에 빠르게 오토스케일 될 수 있다
- 문제는 클러스터 규모가 커지면서 HPA 개수가 많아지면 반응 속도가 급격히 느려진다
- 이벤트 성 이슈 등으로 빠르게 오토스케일이 되어야 하는 경우에 문제가 될 수 있다

3.4.1 HPA 처리 속도 테스트

HPA 반응시간 측정

- HPA 개수: 총 669개
- HPA 하나 생성해서 요청 주입 후 scale out 시작 시점까지 총 30회
- HPA 반응시간: **평균 79초, 최대 99초**

사용하는 metric	개수
resource - cpu	134
pod - cpu_usage	332
memory_usage_bytes	26
memory_working_set_bytes	2
network_receive_packets	8
network_transmit_bytes	1
nginx_http_requests	240
tsd_http_query	2
splicer_query_requests	2
chronos_ts_normal_connections_hpa	2
kafka_consumer_total_lag	1
custom_burrow_metric_kafka_total_lag	1
external - ncc_example_weight	1
external - nginx_http_requests	1

hpa 사용방식

- 총 669개
- 멀티 metric 3개 사용: 6개
- 멀티 metric 2개 사용: 73개
- 단일 metric 사용: 590개
- 참고로 이중에서 단일 100개는 istio : resource type cpu

타입별	개수
Resource	132
Pods	619
External	2

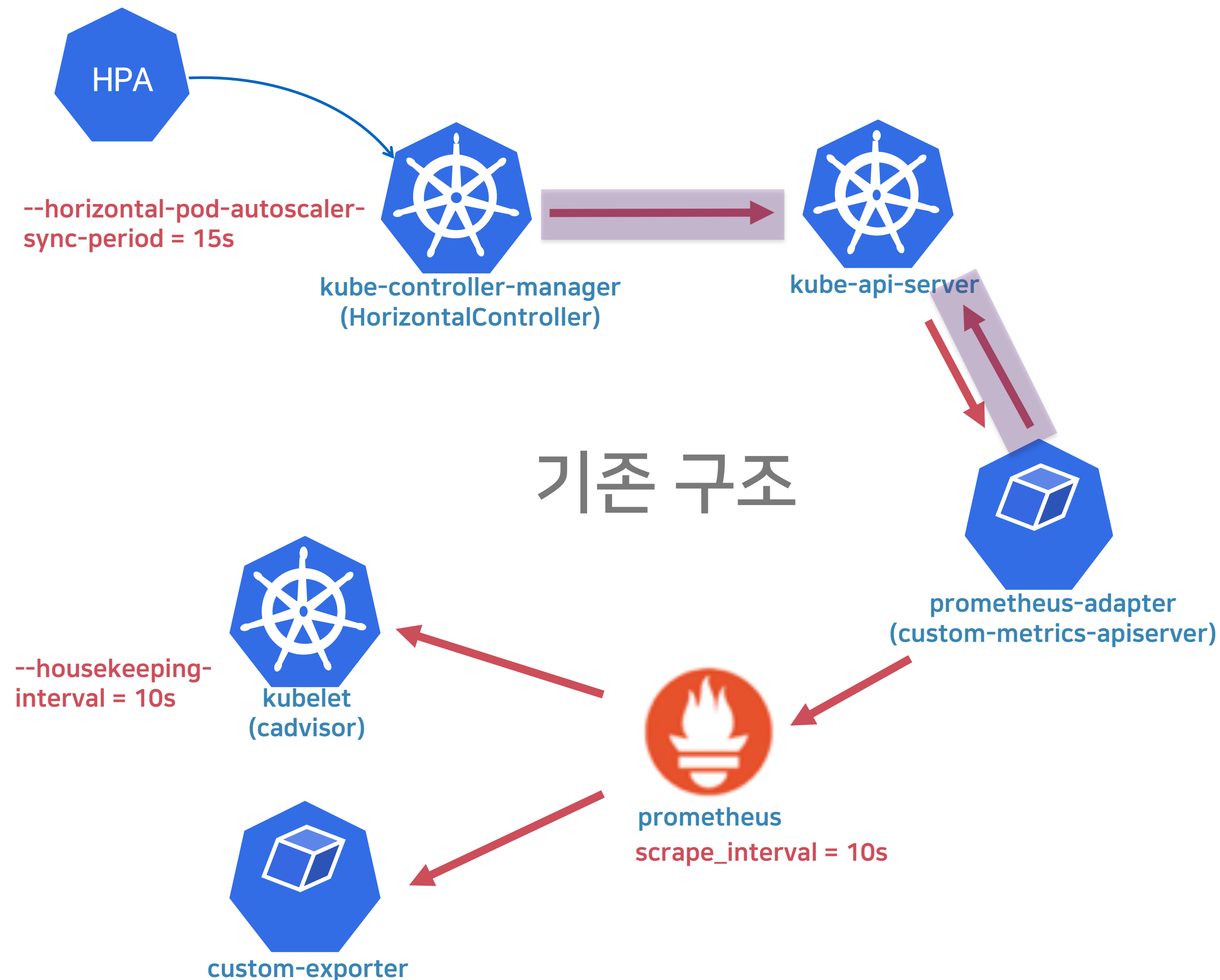
3.4.2 HPA 구조적인 한계

현재 HPA의 구조에서 처리 속도를 빠르게 하려면 한계가 있다

- HPA 컨트롤러가 **kube-controller-manager**에 **dependency**가 있기 때문에 QPS를 높이거나 Worker 수를 늘리는데 한계가 있다
- Kubernetes 컨트롤러와 분리해서 성능을 높이고 추가 기능들을 포함한 HPA를 만들어서 해결하려고 했다

3.4.3 NCC HPA

목표 처리 시간 10초에 반응하는 HPA를 개발



병목 구간

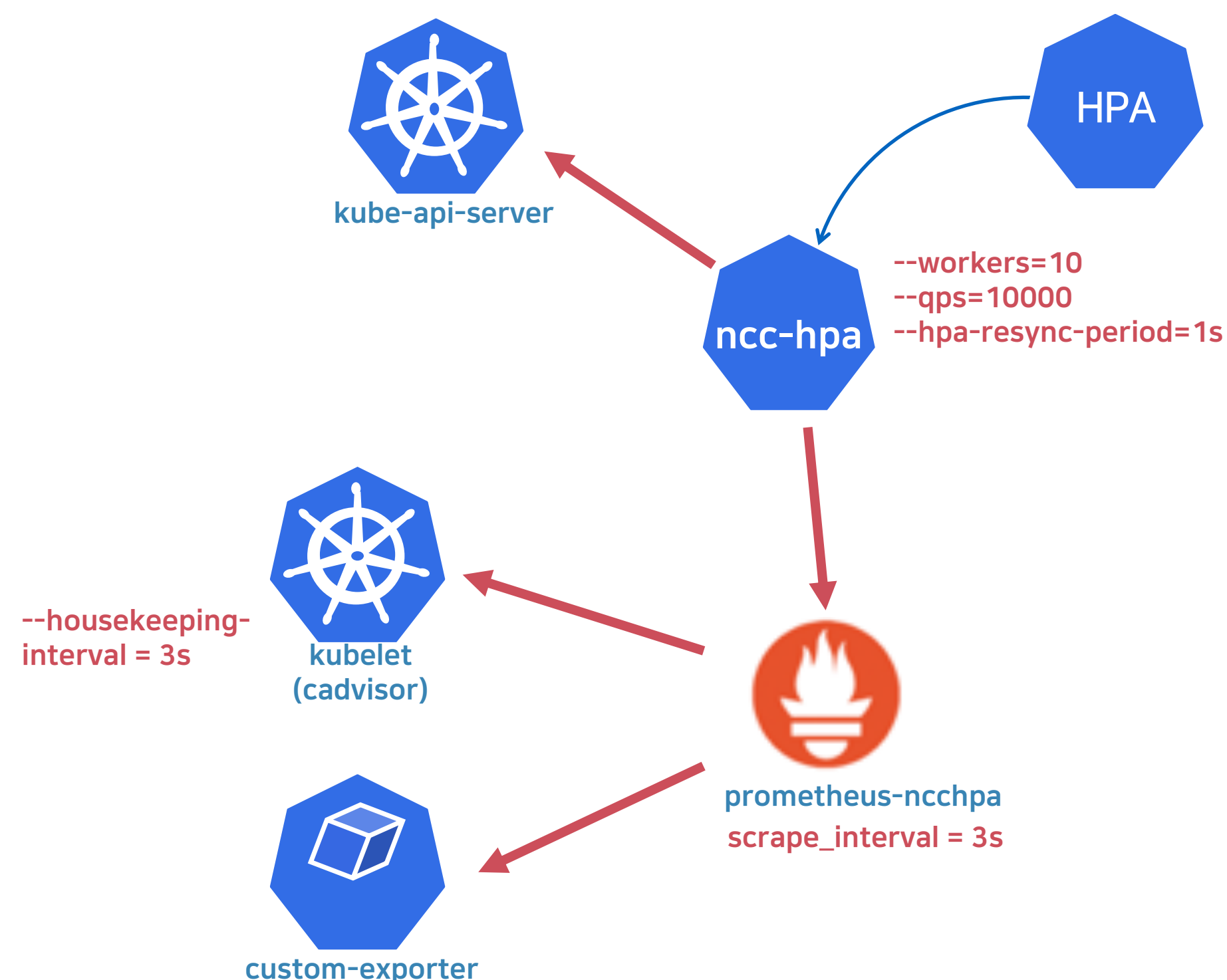
- k8s client-go default QPS: 5

HorizontalController

- single worker 사용

3.4.3 NCC HPA

목표 처리 시간 10초에 반응하는 HPA를 개발



동일한 클러스터에서 반응 시간 측정

- HPA 개수: 1000개
- HPA 반응시간: **평균 7.8초(5~10초)**

개선 포인트

- 구조 단순화와 병목 제거
- worker 수 조절
- period, interval 조절

3.4.3 NCC HPA

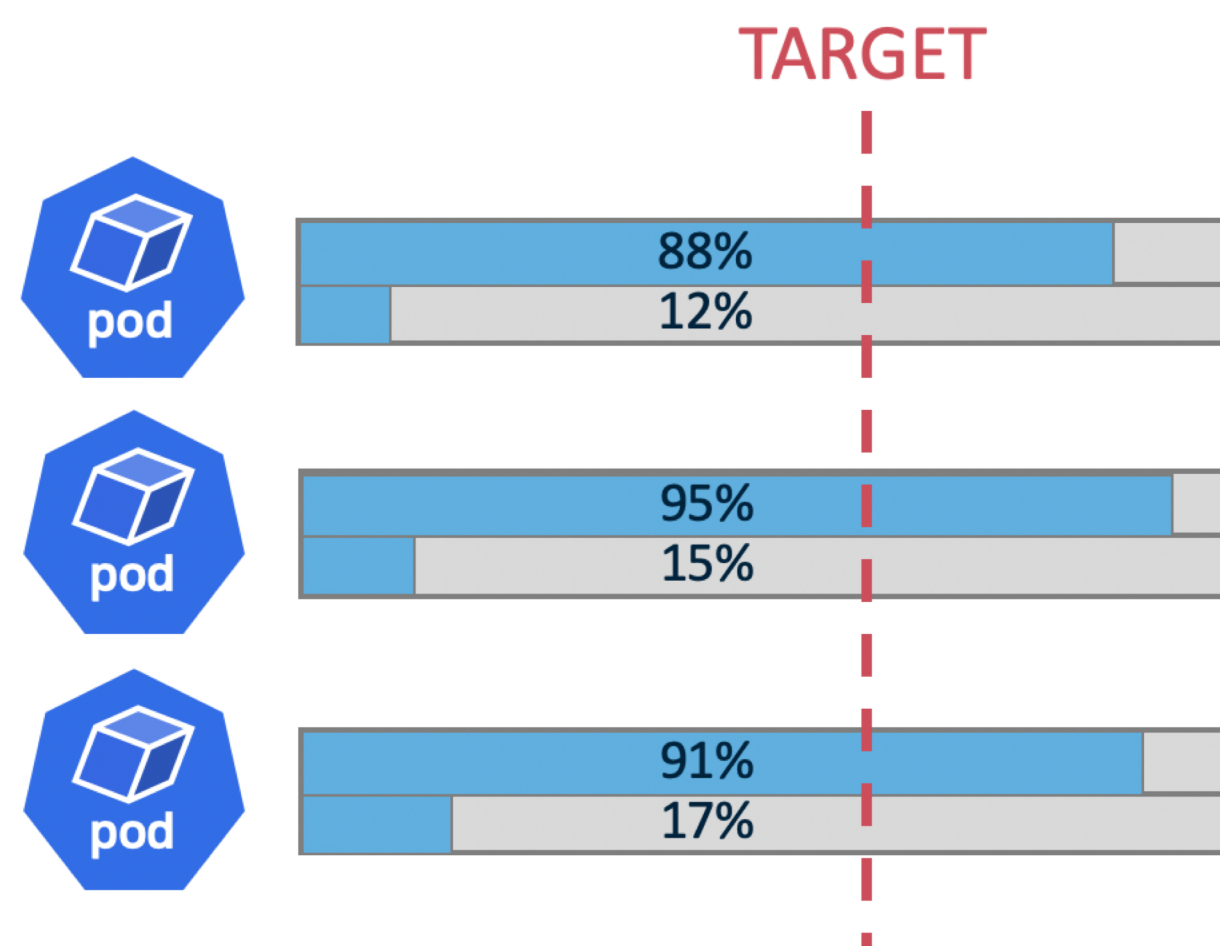
HPA 성능을 만족하면서 추가 기능은 덤으로

- Container 단위로 HPA replicas 계산
- 사용자가 직접 쿼리 가능하도록 PromQL 지원
- Metric 범위 별로 scale limit 정책을 설정 가능
- 다른 서비스의 replicas를 참조하는 cascading 방식의 오토스케일 기능
- 특정 시간에만 부하가 몰리는 서비스를 위한 예약 기능
- 메신저로 HPA 변경사항을 받아보고 최대화 명령을 내릴 수 있는 기능

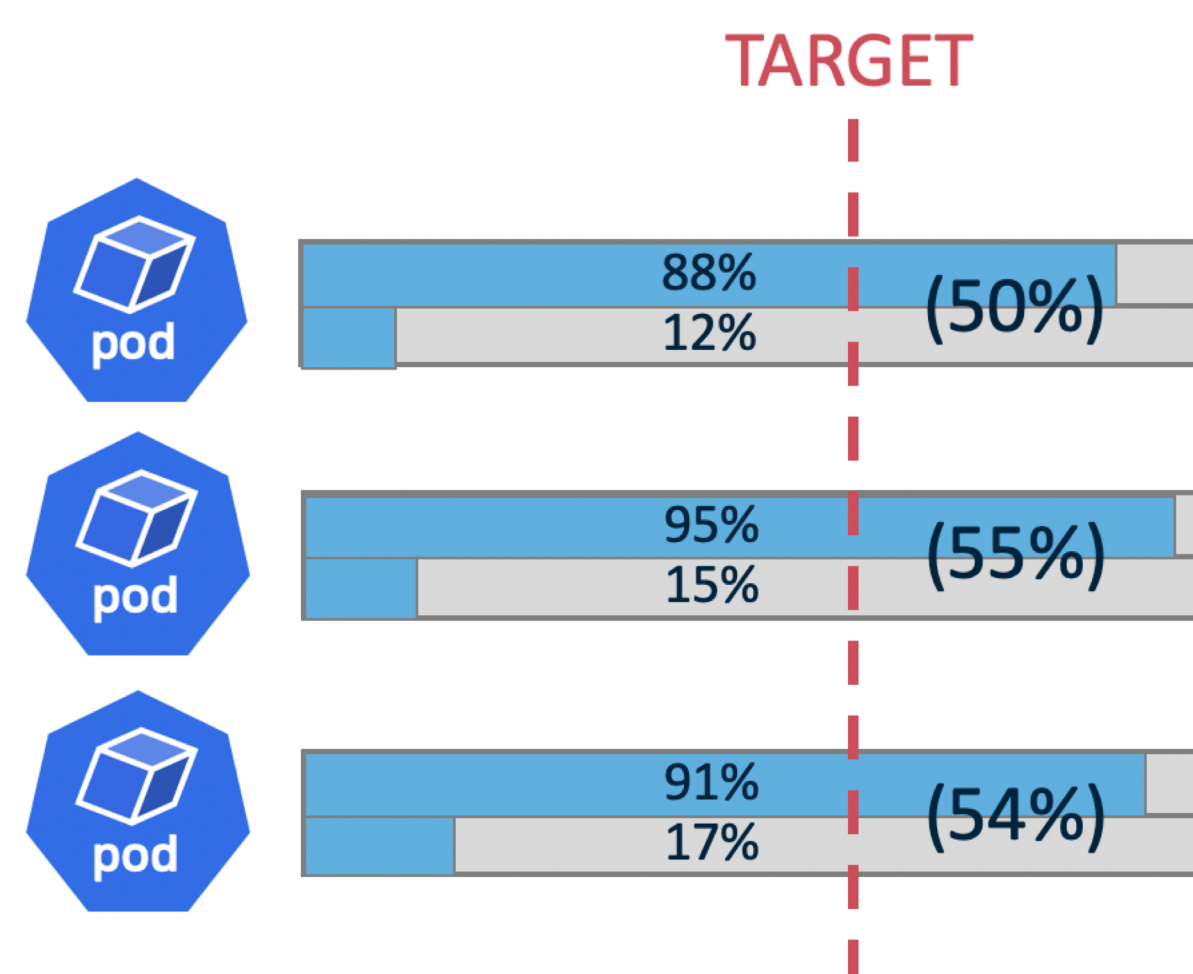
3.4.3 NCC HPA

Container 단위로 HPA replicas 계산

Containers 타입



Pods 타입



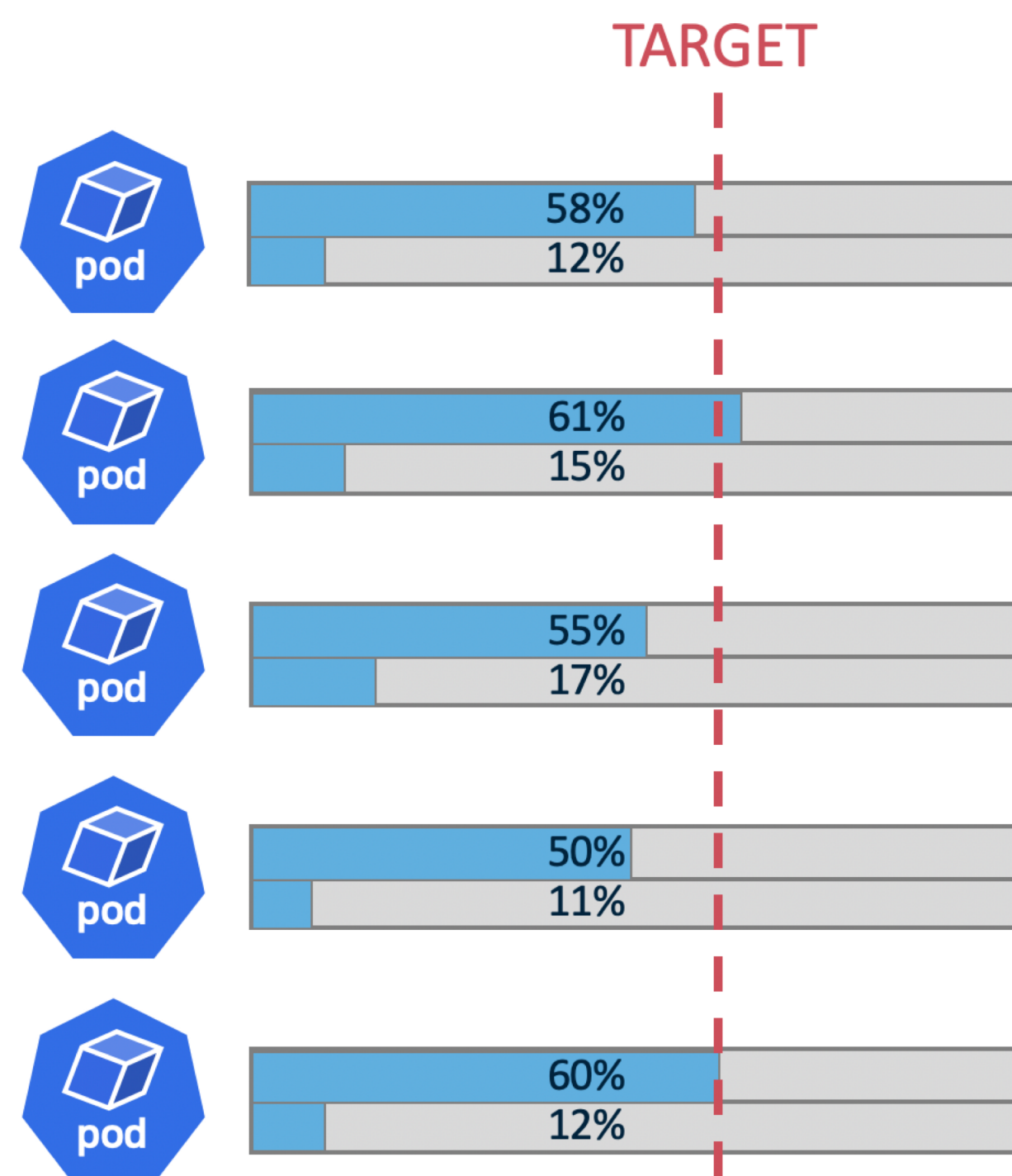
[참고] container type 오토스케일은 k8s 진영에서 아직 PR merge 전이다.

<https://github.com/kubernetes/kubernetes/pull/90691>

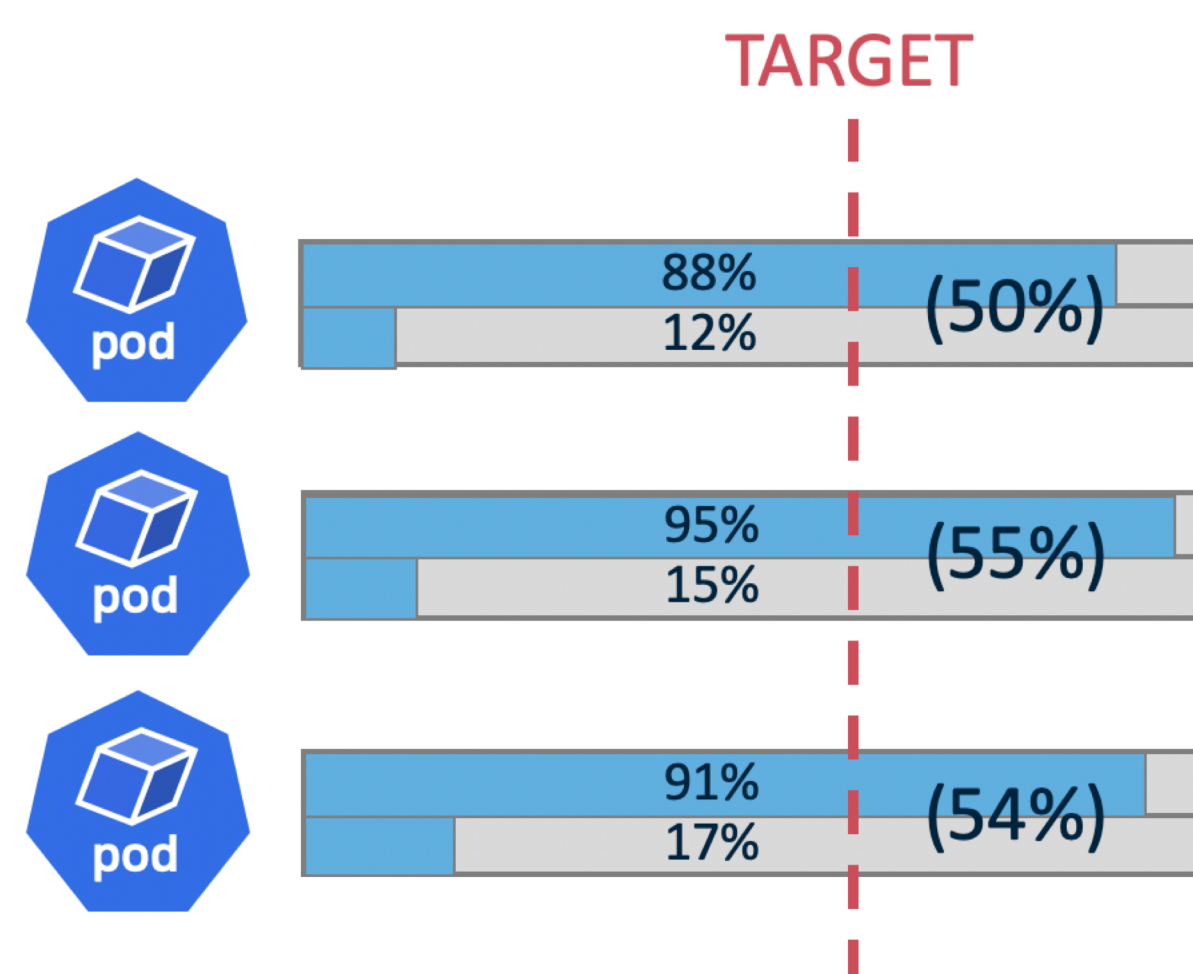
3.4.3 NCC HPA

Container 단위로 HPA replicas 계산

Containers 타입



Pods 타입



3.4.3 NCC HPA

Metric 범위 별로 scale limit 정책을 설정 가능

- behavior의 추가 기능으로 범위를 지정할 수 있다

```
behavior:
  scaleUp:
    policies:
      - type: Percent
        periodSeconds: 15
        value: 10
      - type: Percent
        periodSeconds: 15
        value: 50
        bounds:
          lower: 50
          upper: 70
          metric: cpu_usage
          type: Containers
      - type: Percent
        periodSeconds: 15
        value: 100
        bounds:
          lower: 70
          upper: 100
          metric: cpu_usage
          type: Containers
```

- default: +10% scaling
- 50 <= cpu_usage < 70: +50% scaling
- 70 <= cpu_usage < 100: +100% scaling

[참고] behavior 기능은 k8s 에서 v1.18 버전부터 베타버전으로 제공

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#support-for-configurable-scaling-behavior>

3.4.3 NCC HPA

사용자가 직접 쿼리 가능하도록 PromQL 지원

- Prometheus에서 사용하는 PromQL 쿼리로 오토스케일에 사용
- Containers Type에서 cpu_usage를 사용하는 내부 쿼리

```
sum by (pod,container)(rate(container_cpu_usage_seconds_total{container="nginx"}[1m]))
```

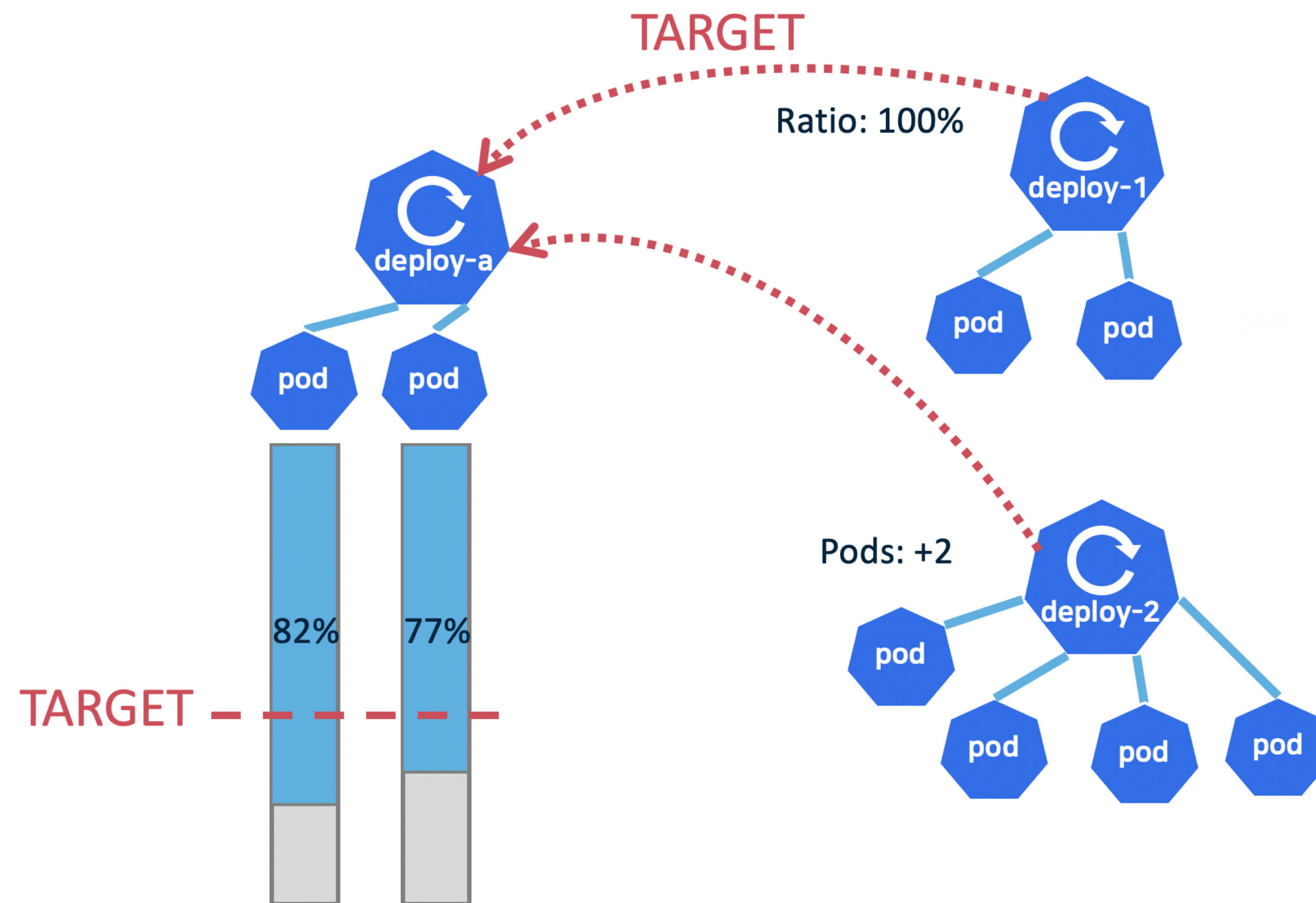
- rate range 값을 자유롭게 변경할 수 있는 장점
- external metric 을 구성해서 자유롭게 쿼리로 만들 수 있다

[참고] promql 에서 rate 함수는 범위 벡터 내에서 초당 평균 증분 값이다.

promql : <https://prometheus.io/docs/prometheus/latest/querying/functions>

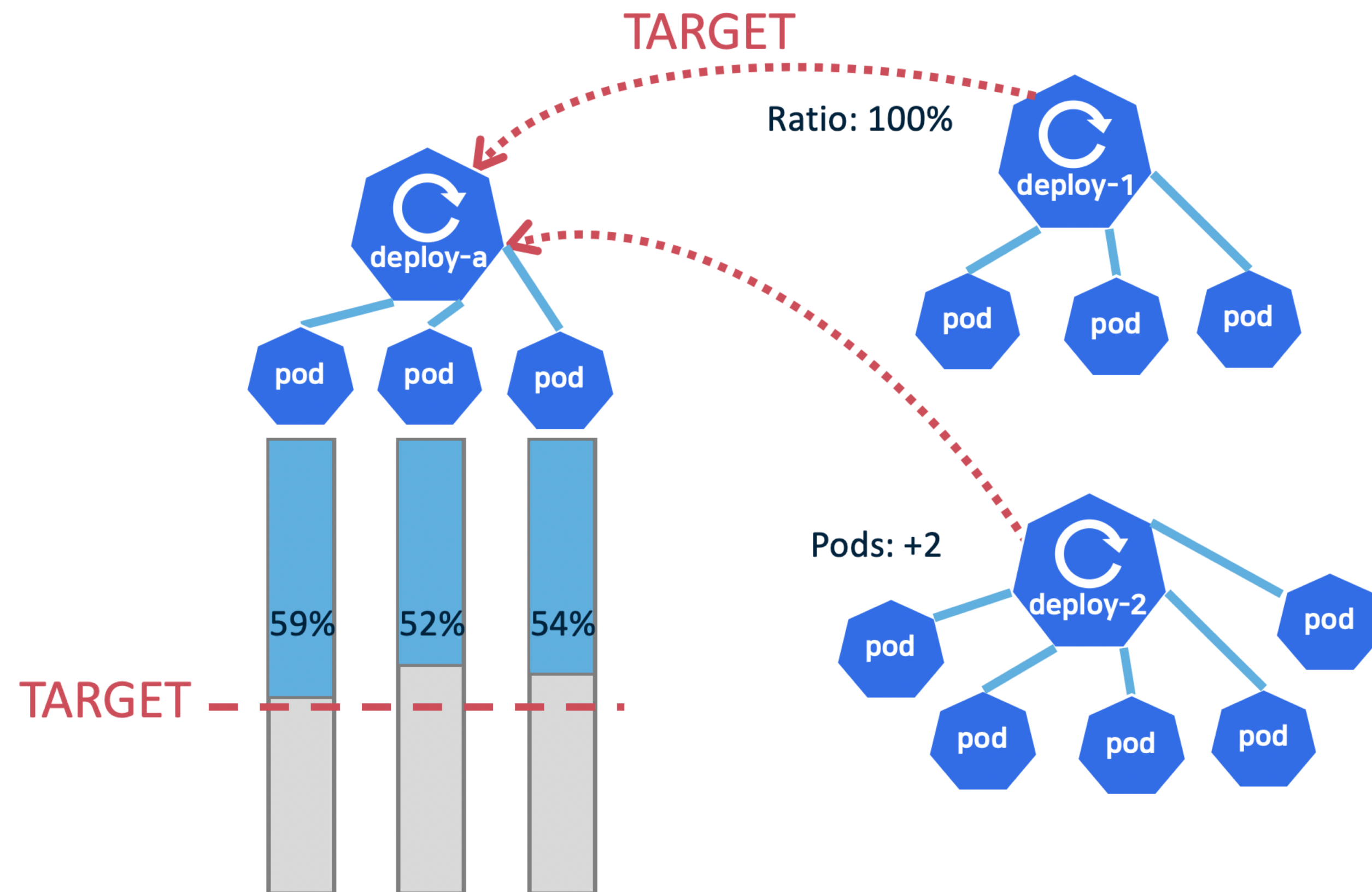
3.4.3 NCC HPA

다른 서비스의 replicas를 참조하는 cascading 방식의 오토스케일



3.4.3 NCC HPA

다른 서비스의 replicas를 참조하는 cascading 방식의 오토스케일



3.4.3 NCC HPA

특정 시간에만 부하가 몰리는 서비스를 위한 예약 기능

```
behavior:
  schedules:
  - recurrence: "0 14 * * 1-5 "
    duration: 3h
    minReplicas: 20
    maxReplicas: 40
```

(월-금 오후2시부터 5시 사이로 예약)

3.4.3 NCC HPA

메신저로 HPA 변경사항을 받아보고 최대화 명령 기능

hpa-messages-test

hpa-messages-test
멤버 3

AD1 NCC Messenger

HPA hpa-test-1의 Scale이 변경 되었습니다.
 기존 : 2 -> 변경 후 : 3
 HPA : ncc-be/hpa-test-1

```

-----
# Initial Replicas
> Min : 1, Max : 5
-----
# Maximize(On Min Max)
> False 1 5
-----
# Metrics. (Name|Target|Current)
> [*] cpu_usage|AverageValue
[1000m]|1135m
-----
# Proposed replicas
> Count : 3
> Reason : cpu_usage above target
-----
# Normalize
> Prenormalized : 3
> Stabilized : 3
> Final : 3
> Reason : DesiredWithinRange
-----

```

메시지를 입력해주세요.

AD1 NCC Messenger

NCC HPA ncc-be/hpa-test-1 에 사용 가능한 명령어

[알람 켜기](#)
[알람 끄기](#)
[Scale 최대화 켜기](#)
[Scale 최대화 끄기](#)

오전 11:07

AD1 NCC Messenger

NCC HPA ncc-be/hpa-test-1 의 최대화를 설정 하였습니다.

오전 11:08

AD1 NCC Messenger

HPA hpa-test-1의 Scale이 변경 되었습니다.
 기존 : 3 -> 변경 후 : 5
 HPA : ncc-be/hpa-test-1

```

-----
# Initial Replicas
> Min : 1, Max : 5
-----
# Maximize(On Min Max)
> True 5 5
-----
# Primitive condition matched
> Current number of replicas below
Spec.MinReplicas
-----
# Rescale
> true
-----
# Deleted
> false

```

가능한 작업 보기

오전 11:08

3.5 Lesson Learned 요약 정리

Multi-Tenancy K8s를 운영하면서 SRE 관점에서 해결하려고 했던 이슈들

- 여러 요인에 의해 클러스터 노드 간 리소스 불균형 발생

Long running pod, 리소스 사용량이 많은 pod, 스케줄러의 분배 불균형으로 노드 간 편차가 심하다
갑자기 트래픽이 폭증하는 이벤트 상황에서 특정 노드의 리소스가 고갈되어 장애로 이어질 수 있다

완화 방법: **scheduler-extender + pod auto rebalancer**

- Kubernetes 이벤트 관리

놓치고 지나간 k8s 이벤트에서 장애 발생 유발 단서를 미리 얻을 수 있지 않을까?

클러스터 사용자에게도 중요한 이벤트를 알려주면 장애를 미연에 방지하는데 사용할 수 있지 않을까?

해결 방법: **sentry + github, 클러스터 이벤트 리포트, 사용자 SRE 리포트**

- 기타 이슈: HPA 성능 이슈

지진이나 연예인 이슈와 같이 짧은 시간에 대규모 트래픽이 들어오는 상황에서도 대응 가능하도록
충분히 빠른 오토스케일이 필요하다

해결 방법: **NCC HPA**



4. What's Next?

4.1 클러스터 사용자들도 SRE

사용자들이 안정적인 서비스를 서빙할 수 있도록 SRE 리포트 제공

- 리소스 사용량 관련 리포트

리소스 사용량의 장기 추세 기반 Abnormal Detection

Deployment 또는 StatefulSet의 특정 Pod의 부하 패턴이 다른 경우 알람

- 커널 로그 및 k8s 이벤트 리포트

OOM Killed Pod 정보

간헐적인 ReadinessProbe 실패에 대해 사용자들이 인지할 수 있도록

- 비정상 동작 관련 리포트

restart가 많은 Pod 정보

사용자들이 안정적인 서비스를 서빙할 수 있도록 SRE 리포트 제공

(현재 열심히 만들고 있는 리포트 샘플)

4.2 메신저 채널

클러스터 사용자들에게 메신저 채널 제공

- 메신저 알람 채널

서비스 알람이 발생했을 때 관련 대응 담당자로 구성된 메신저 채널을 만든다
서비스 상태를 확인할 수 있고 긴급 대응할 수 있는 명령을 제공한다

- 시나리오 기반 챗봇

HPA 상태 변경 내용을 메신저로 받아보고 최대화 명령으로 긴급 대응할 수 있다 (완료)
부하 등으로 서비스 상태 이상 알람을 받았을 때 현재 상태를 조회하고 재시작, 스케일 등을 조절할 수 있다

4.3 Image Pull 시간 단축

빠른 오토스케일을 제대로 사용하려면 Image Pull 단축이 필요하다

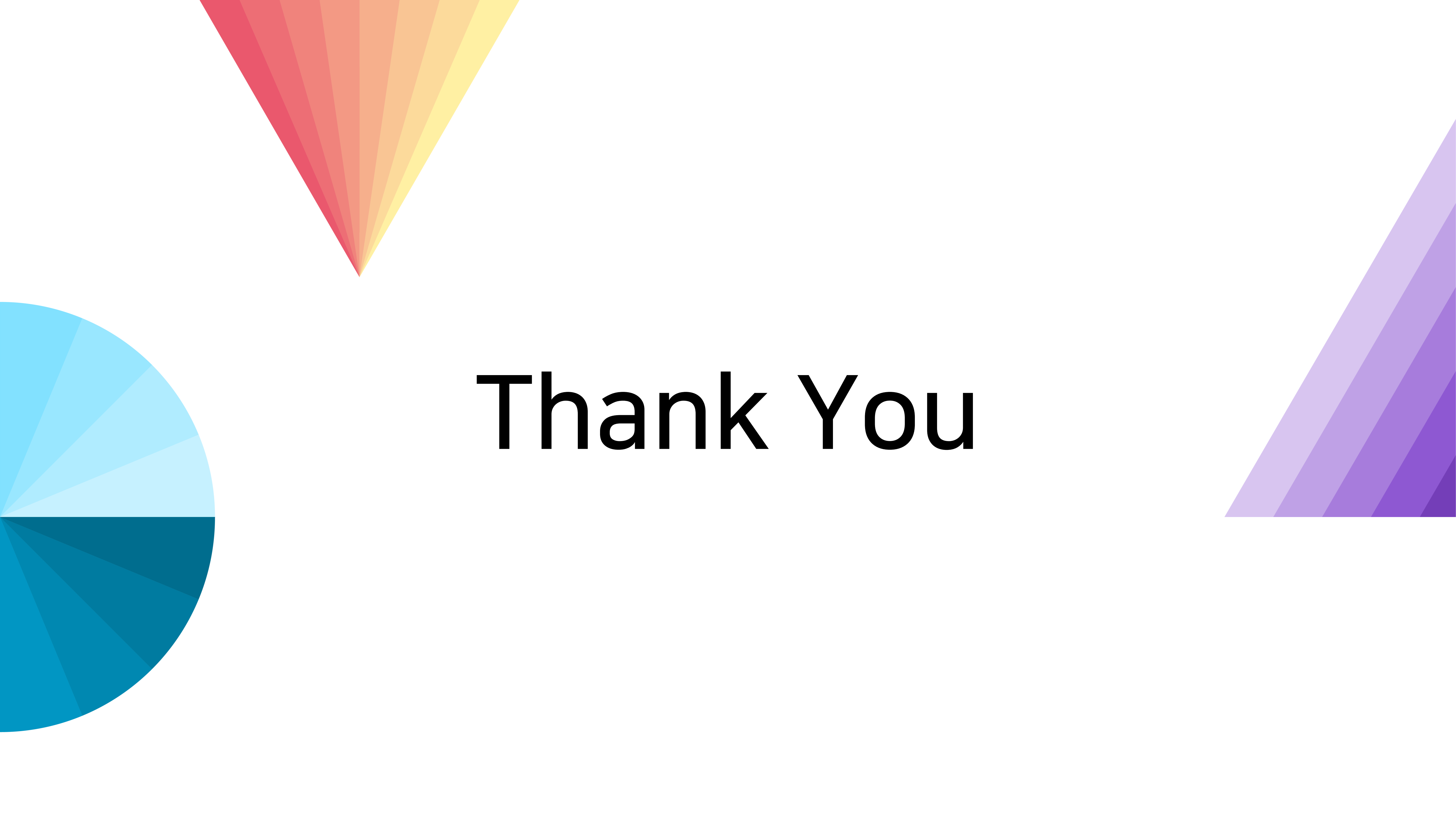
- 생각 중인 방법은 Image를 노드에 미리 받아 놓고, schedule-extender에서 우선순위를 높이는 방식으로 실제 Pod이 running 타임을 줄일 수 있을 것 같다

Time Table

시간	이벤트	소요시간	
18:07:45	Grafana 상으로 avg core 2+		
18:08:30	HPA 에서 Scale Up 이벤트 발동	약 45s	<10s 해결
18:08:30	노드에 Pod Assign	약 <1s	
18:08:31	initContainer 실행	약 <1s	
18:09:20	노드 스케줄링이후 image pull 완료	약 50s	미해결
18:09:40	readinessprobe 성공	약 20s	



Q & A



Thank You