

Multi-Tenancy Kubernetes on Bare Metal Servers

이종현

NAVER / 클라우드&스토리지플랫폼 / 컨테이너플랫폼

NAVER

CONTENTS

DEVIEW
2019

1. Introduction
2. NAVER Container Cluster
3. Lesson Learned
4. Next Step for Better World 😊

1. Introduction

1.1 컨테이너 플랫폼 팀

“Since 2015, 과연 우리는 누구인가...”

- Container Oriented

컨테이너 기술의 가치를 믿는 사람들

- Handy Man (or Woman 😊)

필요한 것은 직접 만들 줄 아는 사람들

- Minimalism People

꼭 필요한 것만 하려는 사람들



1.2 Our Design History

우리의 디자인 결정 사항 중 중요한 것들

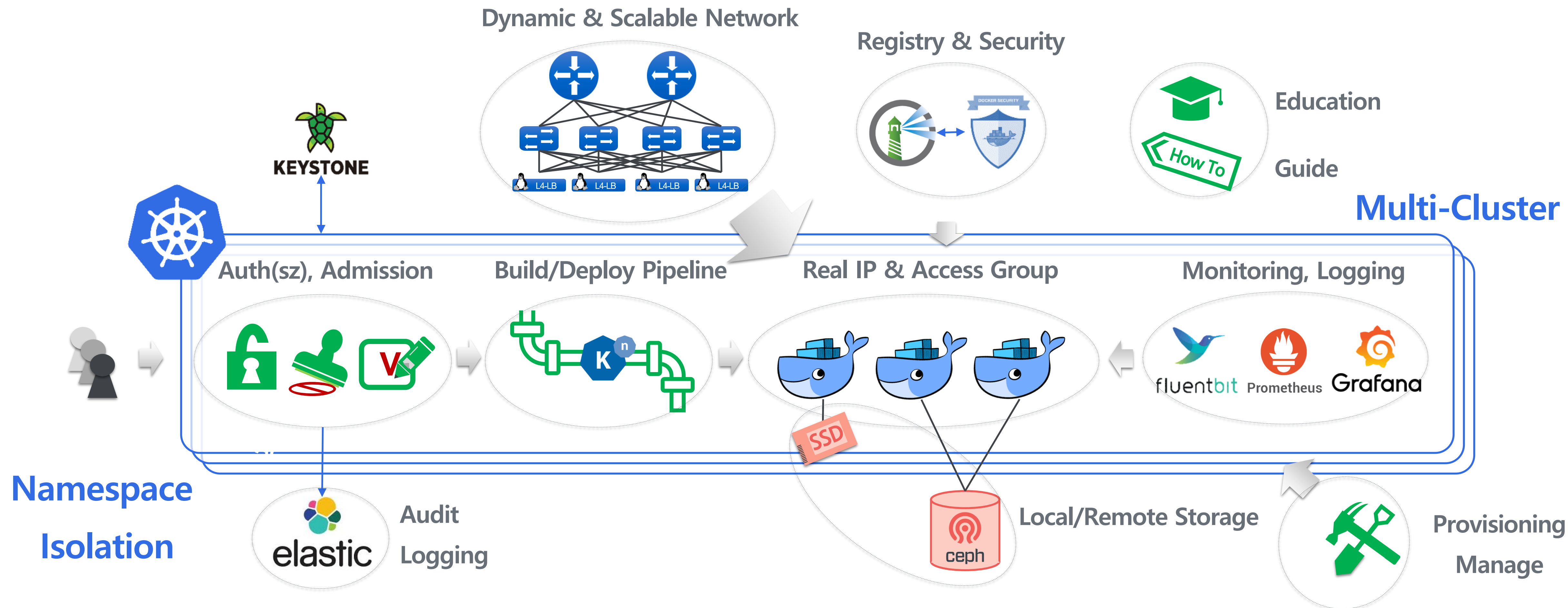
- **Container Oriented** : 컨테이너의 장점을 해치지 말자.
→ 베어메탈 + 컨테이너 오케스트레이션으로 클러스터를 구축하여 제공하자.

- **Cost Efficiency** : 가장 비용 효율적으로 문제를 해결하자.
→ 범용 서버를 사용하고 필요한 것은 직접 구축하자. (SWLB, Ceph, Admission, ...)

- **Help Containerize** : 네이버의 컨테이너 화에 필요한 것들을 찾아서 제공하자.
→ 컨테이너 클러스터 + 레지스트리, 모니터링/로깅, 도커/Kubernetes 교육, 컨테이너화 가이드, ...

1.3 NAVER Container Cluster

“Now 2019, 지금 우리는 무엇을 하고 있는가 ?”



1.4 Multi-Tenancy Kubernetes ?

우리는 Multi-Tenancy 란 이름을 몰랐다. (솔직히...)

네이버의 서비스와 플랫폼에

가장 효율적으로 컨테이너를 적용하고자 했을 뿐!

그러다 보니 사람들이

우리를 “Multi-Tenancy Kubernetes” 라고 부르더라...

1.5 Multi-Tenancy Kubernetes 의 정의

하나의 Kubernetes 를 Namespace 로 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”**.

1.6 Multi-Tenancy Trade-Off

Multi-Tenancy 는 효율적이지만 사용에 제한이 생긴다.

Pros

모든 사용자가 Kubernetes 자체를 위한 서버를 보유할 필요가 없다.

모든 사용자가 Kubernetes 에 여유 리소스를 보유할 필요가 없다.

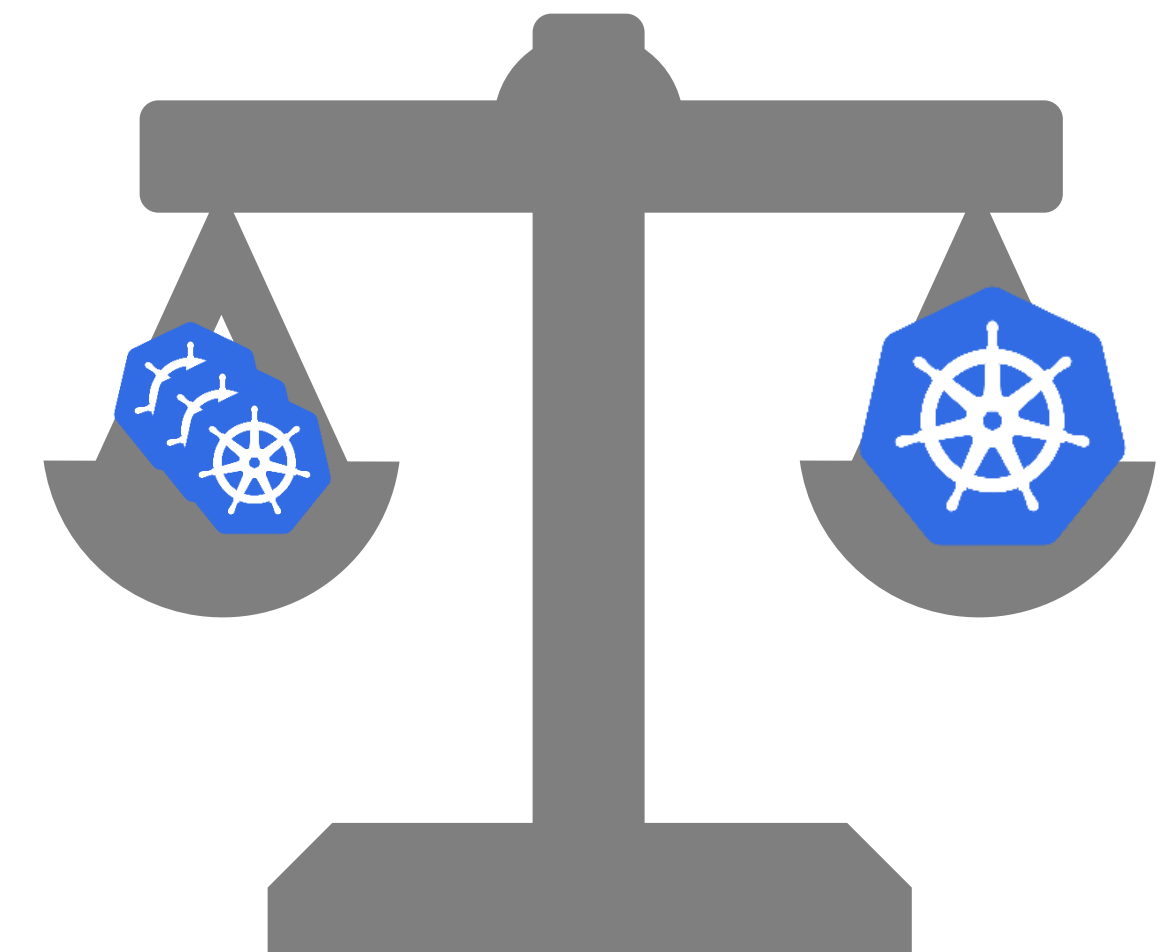
모든 사용자가 Kubernetes 를 직접 운영할 필요가 없다.

Cons

어느정도 신뢰성 있는 사람들 끼리 만 사용할 수 있다.

사용자가 Kubernetes 의 모든 기능을 다 사용할 수 없다.

Kubernetes 가 매우 커지고 사용자도 많기 때문에 잘 운영하기 힘들고 할 일이 많다.



1.7 Multi-Tenancy Kubernetes Links

보다 자세한 내용은 아래 문서나 동영상을 참고하세요~

“클러스터 다중 테넌트”

→ Google Kubernetes Engine 의 클러스터 다중 테넌트에 대한 소개

“Multi-Tenancy in Kubernetes: Best Practices Today, and Future Directions – David Oppenheimer”

→ Multi-Tenancy Kubernetes 의 종류와 구성요소 등에 대한 소개 동영상

“Multi-Tenancy Best Practices for Google Kubernetes Engine (Cloud Next '18)?”

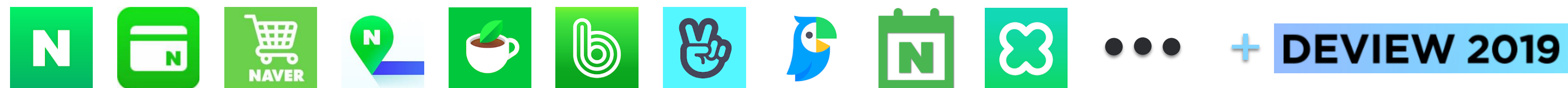
→ Multi-Tenancy Kubernetes 의 Use Case 와 구성 요소에 대한 동영상

2. NAVER Container Cluster

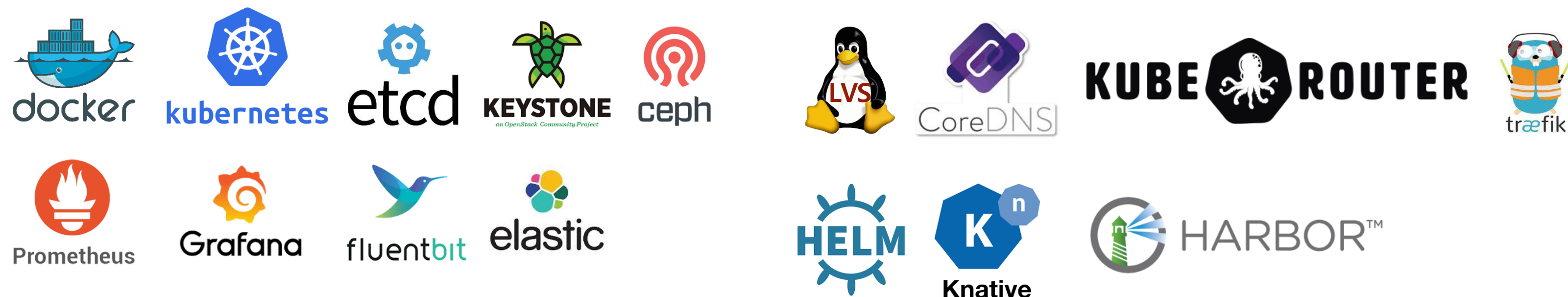
2.1 NAVER Container Cluster (NCC)

네이버를 위한 Multi-Tenancy Kubernetes 프로젝트

- 12+ Clusters
- 300+ Namespaces, 1000+ Users



- **5000+ Pods, 10000+ Containers** (in single cluster)






2.2 NCC Objectives (2019)

“Elastic” 과 “Efficiency” 의 2가지 올해 목표

- 네이버의 “Elastic” 한 서빙




연말/연초, V 이벤트 등 미리 시점을 알 수 있는 이벤트에 **쉽게** 대응
자연 재해나 연예 사건 등 미리 시점을 알 수 없는 이벤트에 **빠르게** 대응

 [elastic](#) 미국·영국 [ɪˈlæstɪk]  영국식  ★

1. (탄력이 있도록 고무 물질을 넣은) 고무 밴드
2. 고무로 된
3. 탄력 있는

- 네이버의 “Efficiency” 한 리소스 활용

컨테이너화로 개별 서비스의 여유 리소스를 **공유**할 수 있게 제공
서로 다른 리소스 사용 패턴을 **조합**하여 서버의 리소스 활용률 향상

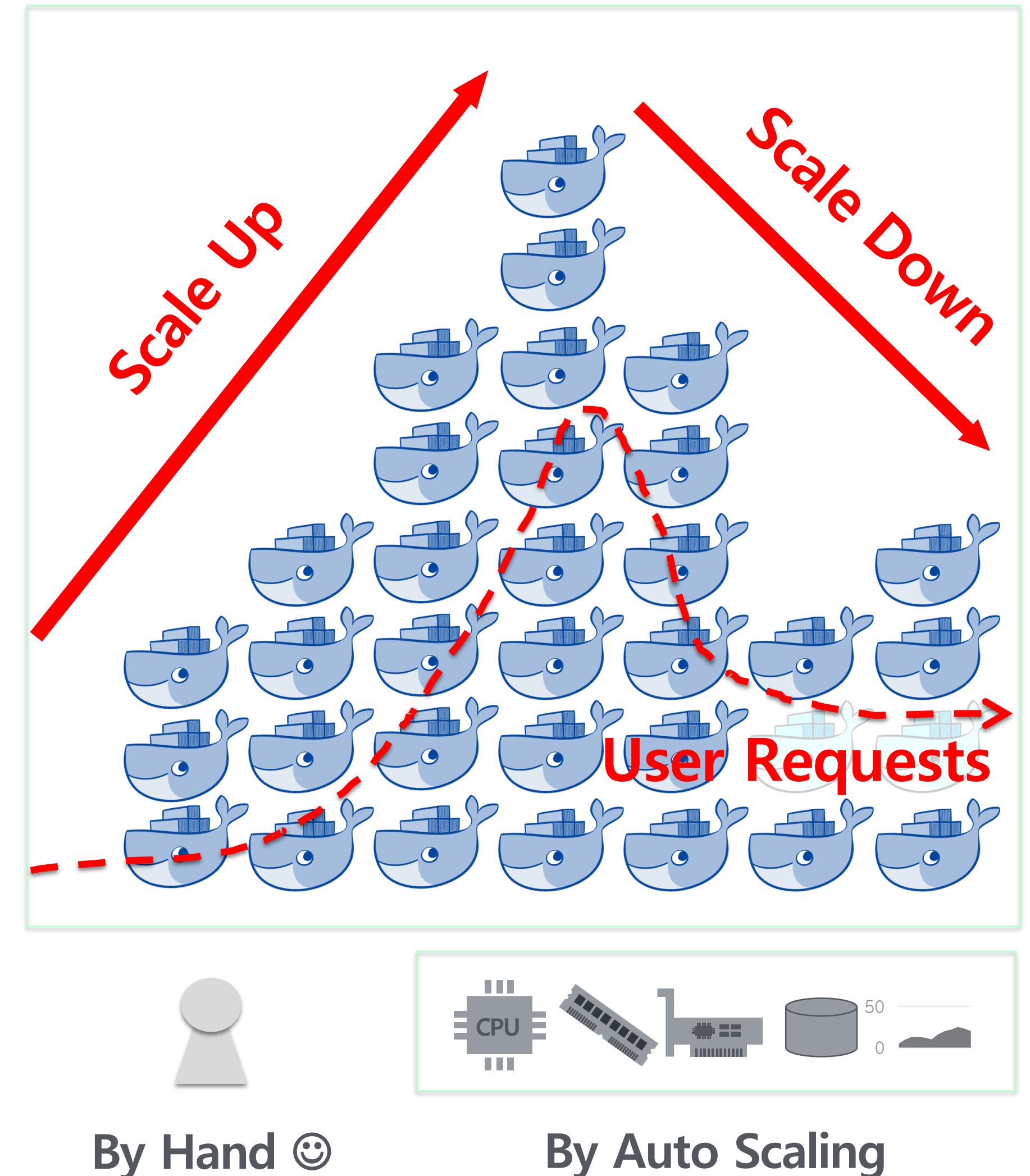
 [efficiency](#) 미국·영국 [ɪˈfɪʃnsi]  영국식  ★

1. 효율(성), 능률
2. 효율화 (방안)
3. (기계의) 효율

2.3 “Elastic” 한 서빙

대규모, 실시간, 신뢰성의 검증

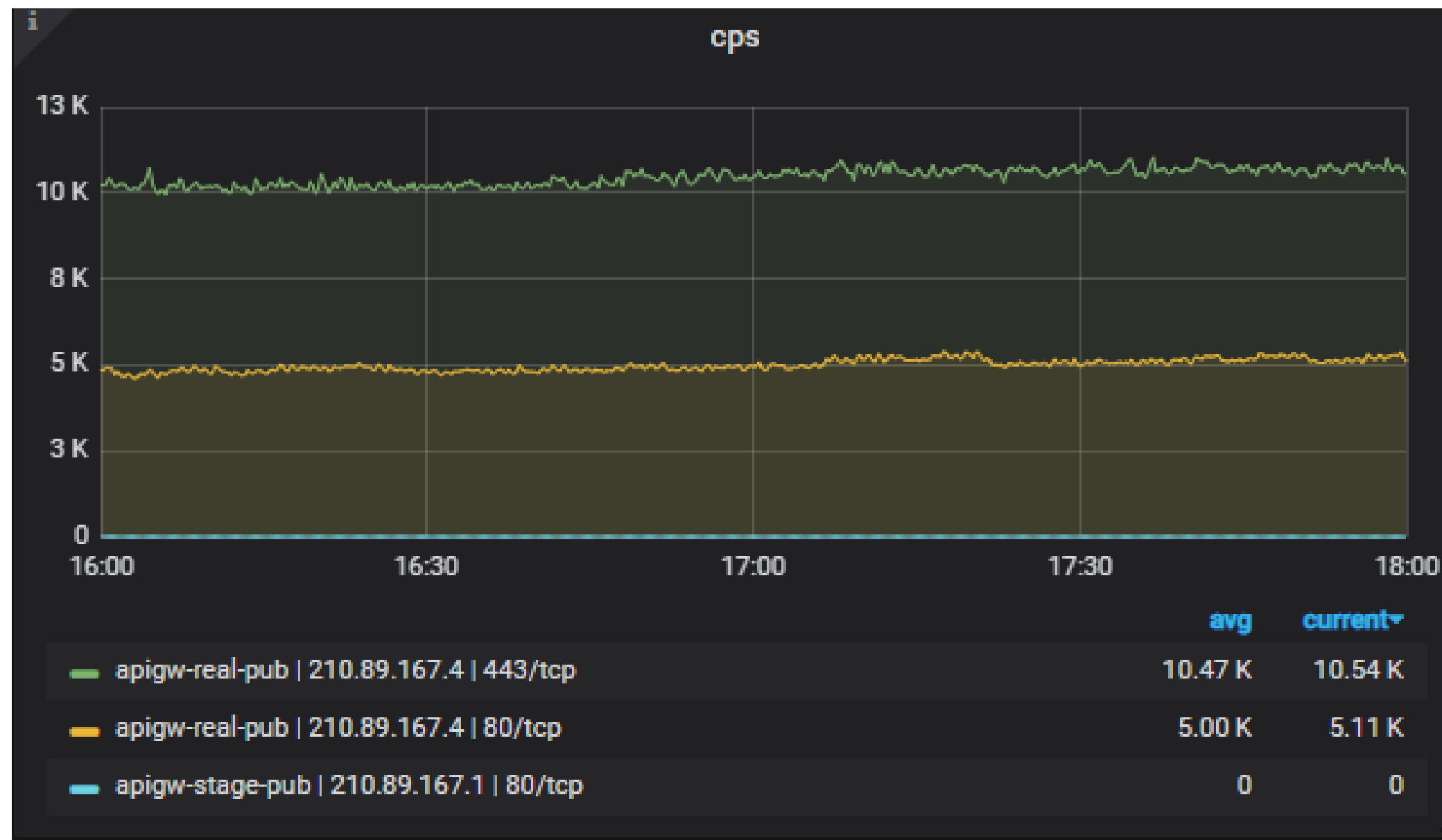
- 대규모 : api-gateway
일반 서비스의 약 25+배 (**100k** TPS, **4M** AC, **25k** CPS)
- 실시간 : 네이버 뉴스
실시간으로 빠르게 반응 (수분 내 3~4배 이상 트래픽 대응)
- 신뢰성 : 네이버 쇼핑
데이터의 신뢰성이 중요 (결제, 상거래 등에 이용)



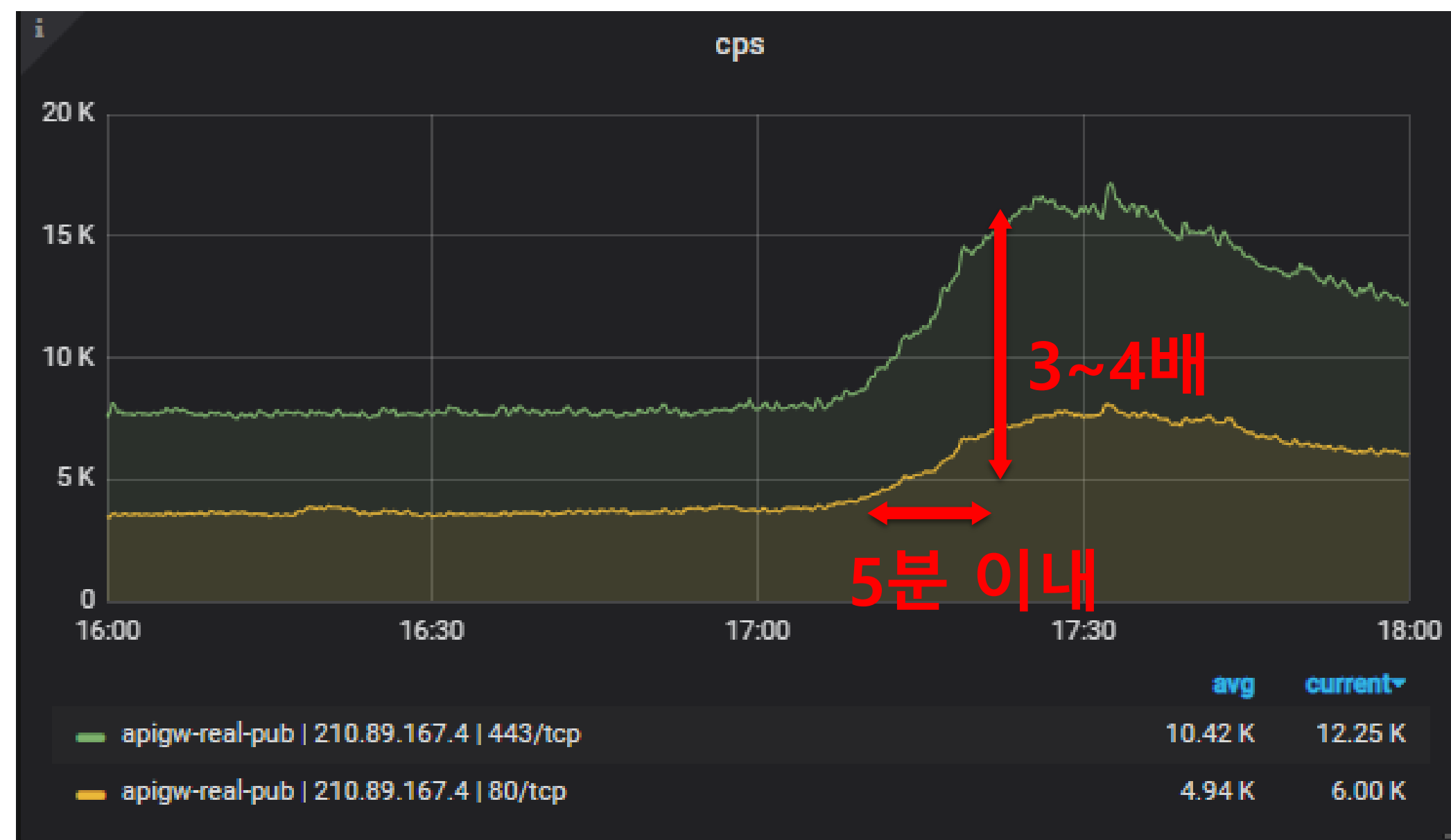
2.3.1 실시간 트레픽 대응

연예인 사건/사고 트레픽 대응의 예

평시 트레픽의 **3~4배**가 **20배** 빠른 속도로 증가



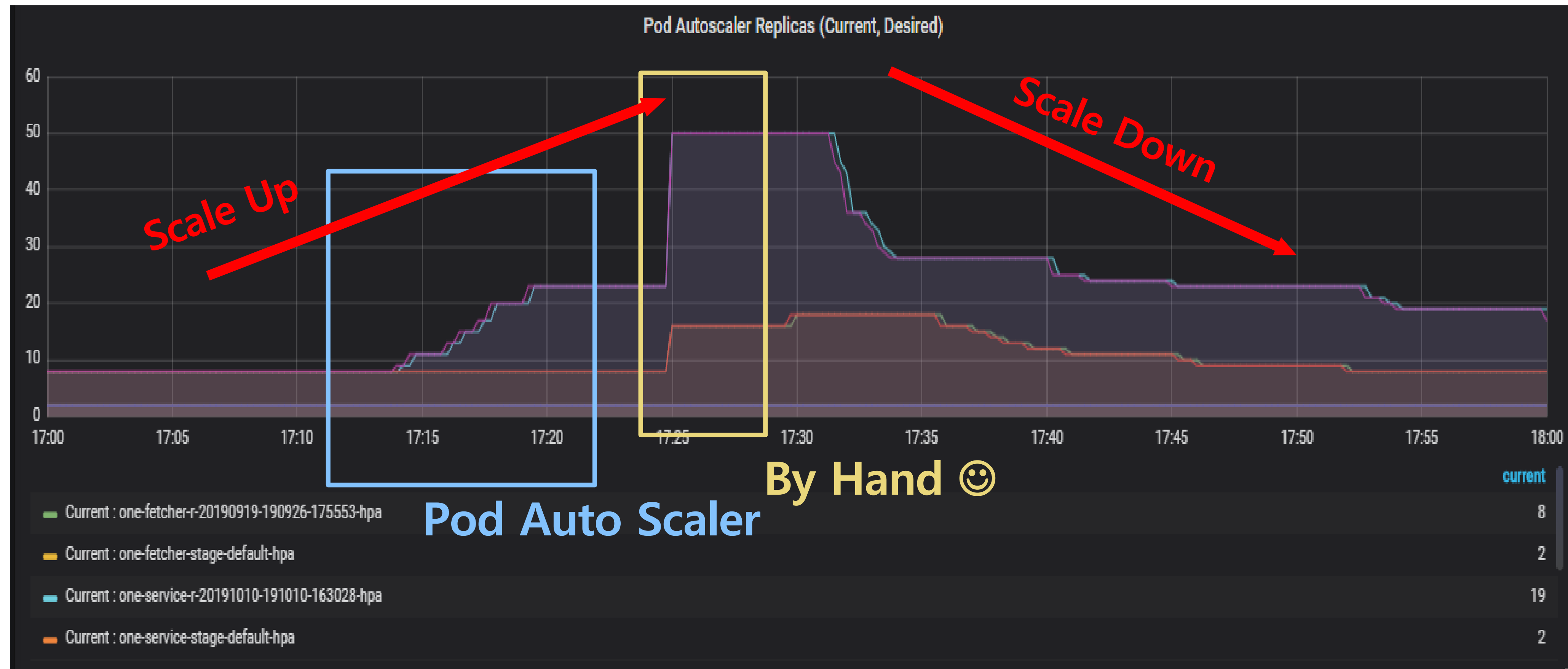
한주 전 CPS 그래프



이벤트 시점 CPS 그래프

2.3.2 Auto Scaling

Horizontal Pod Auto Scaler 와 수작업에 의한 빠른 대응



2.4 "Efficiency" 한 리소스 활용

다양한 리소스 사용 패턴의 컨테이너가 서버를 공유하여 리소스 활용률 Up!

- 리소스 요구량, 시간대별 사용량이 모두 다름

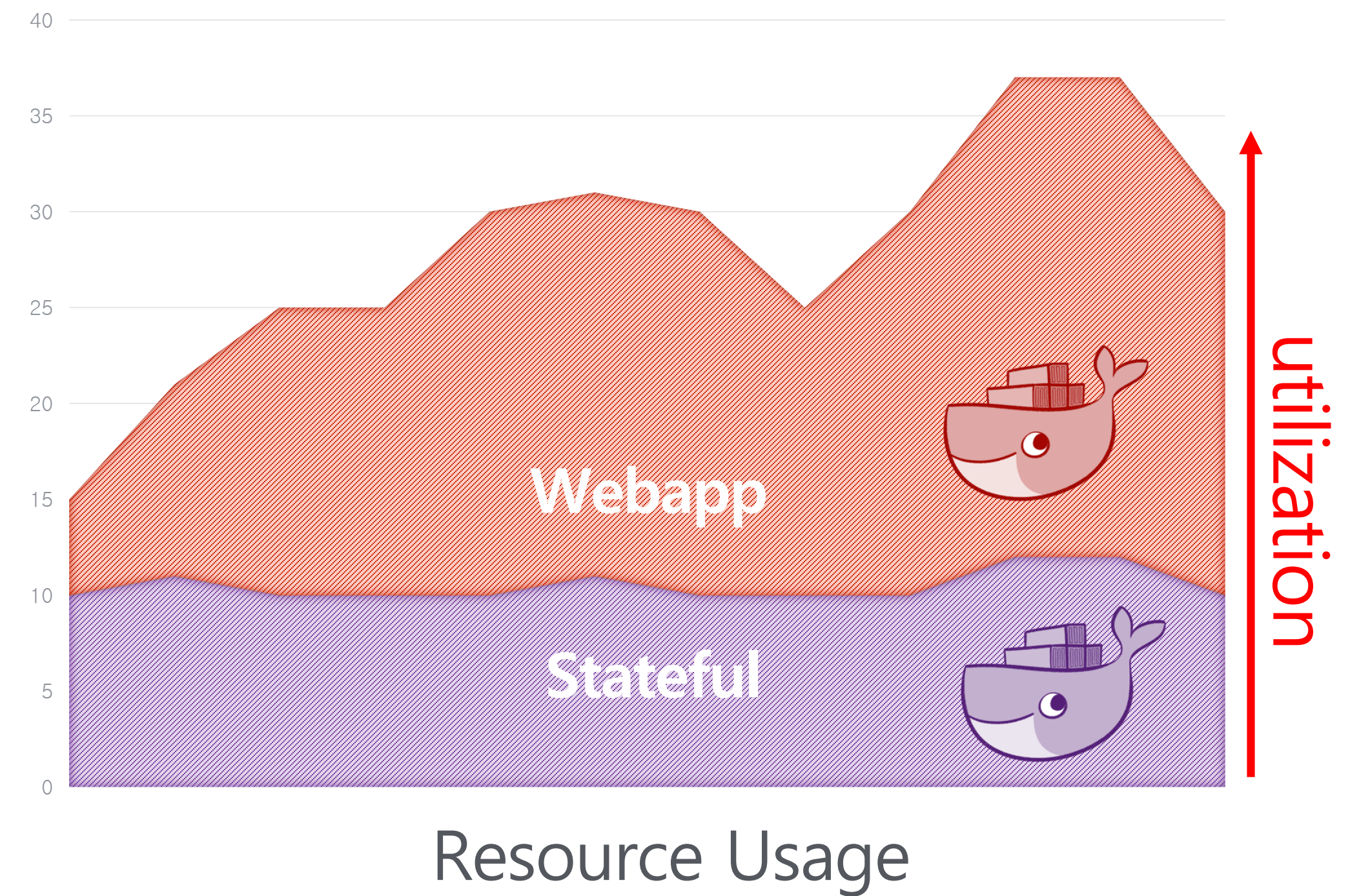
Webapp 과 Stateful (**Elasticsearch**, **Kafka**, ...)

- 여유 리소스 최소화

여유 리소스 서버를 공유하여 최소화

- 서버 대수 최소화

전체 서버 대수 요구량을 1/2 수준으로 줄임



2.4.1 “Efficiency” 의 결과

일반 네이버 서버 대비 2~4배의 리소스 사용률

Node CPU Usage (avg)



●●● 300+ Node(s)

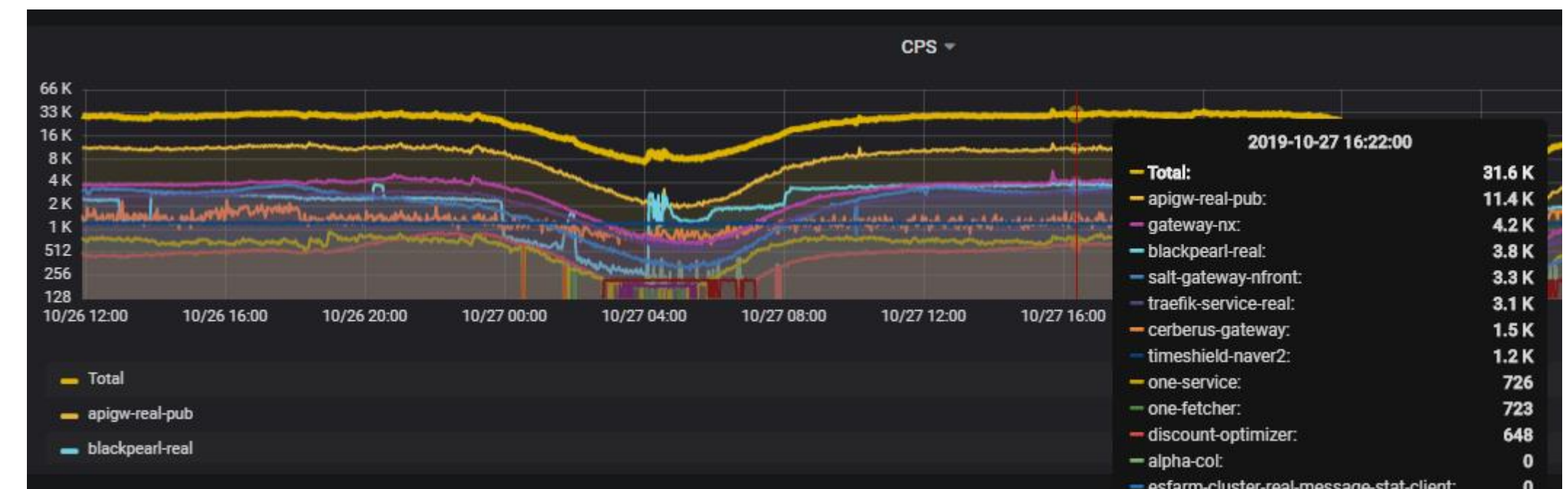


“평시 Single Cluster 평균 사용률 기준”

Busy Node

Not Busy Node

Load Balancer Usage



800+ VIP, 2m AC, 31k CPS, 3Gbps Throughput

3. Lesson Learned

3.1 "Push the Limits"

"Push the Limits" > "한계를 뛰어넘다. "



→ Multi-Tenancy Kubernetes 가 가지는 한계를 극복하자.

"Balance the Resources" > "자원의 균형을 잡다."



"Provide the Convenience" > "편의성을 제공하다. "



3.1.1 대규모 클러스터

클러스터가 커질수록 다양하고 복잡한 이슈들이 발생

- 다수의 클러스터 노드 관리

20 Node(s) Kubernetes 와 500 Node(s) Kubernetes 는 다르다.

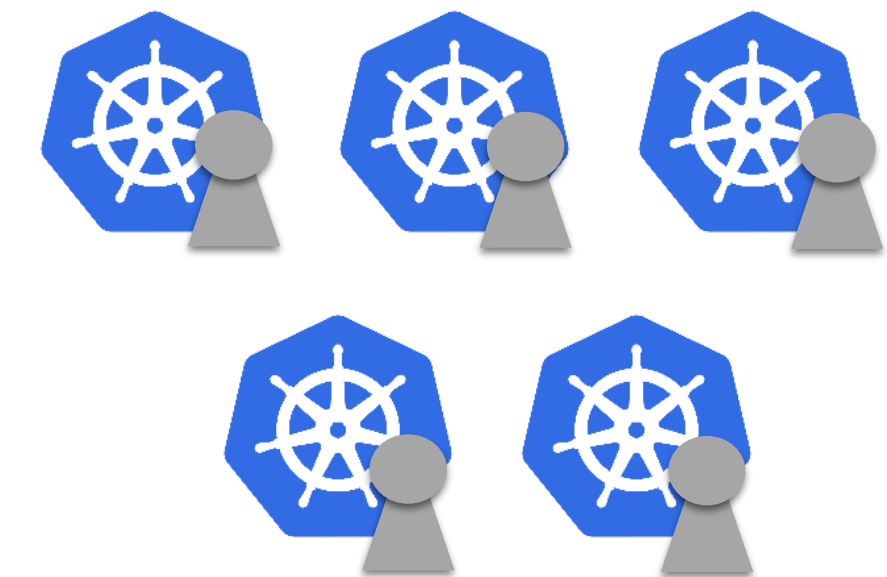
- 관리 서버의 과부하

더 많은 Node(s) 와 Pod(s) 은 ETCD, api-server 등에 더 많은 부하를 주게 된다.

- 복잡해지는 관리 시스템

클러스터가 커지면서 다양한 사용자와 시스템 관리 요구사항이 생긴다.

20+ Nodes Cluster (s)



VS



500+ Nodes Cluster(S)

3.1.2 사용의 제한 (Admission)

시스템이나 다른 사용자에게 영향을 주는 일부 API 는 허용할 수 없음!

- 클러스터에 영향을 주는 API 제한

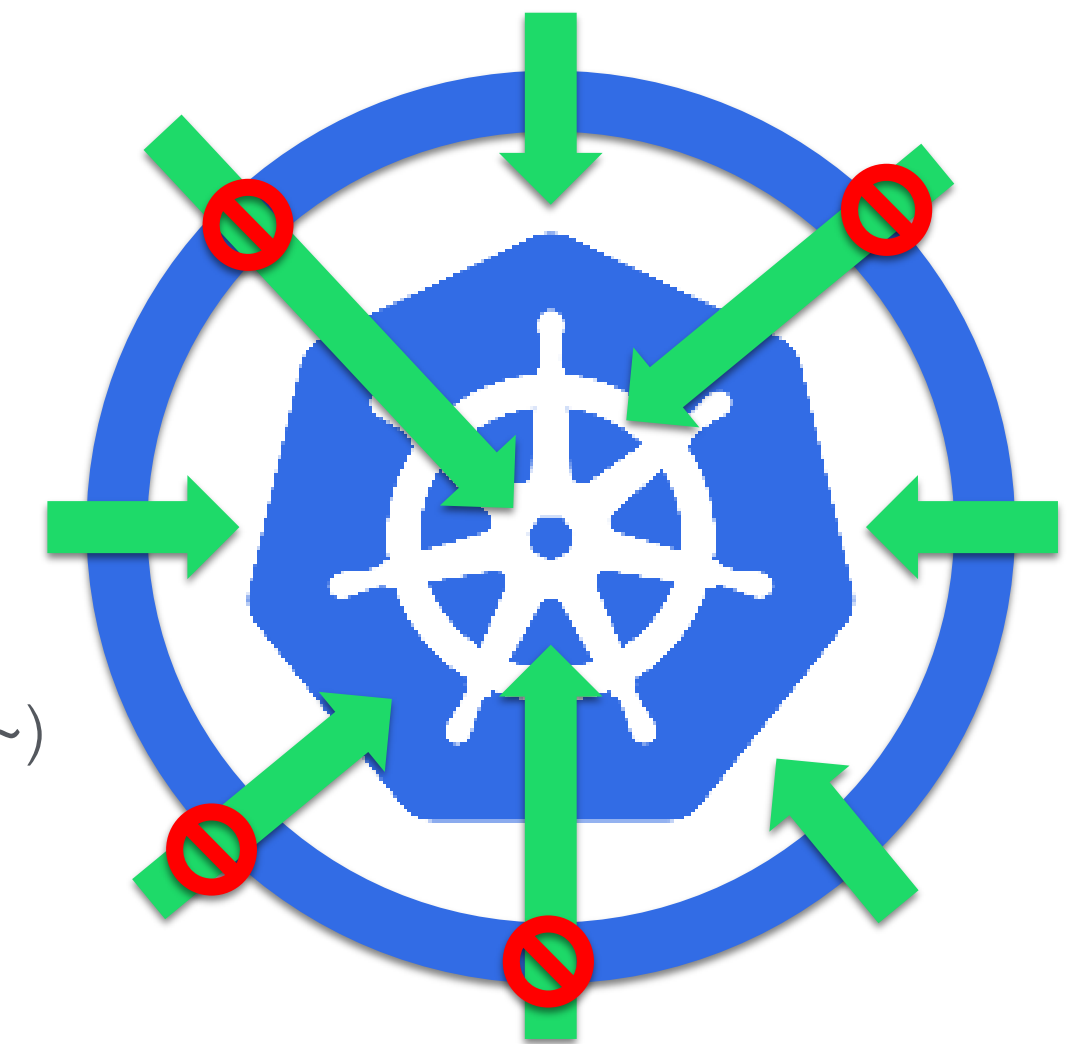
Node, Quota, System Configuration, ...

- 클러스터 전체에 적용되는 권한 제한

Role vs ClusterRole : 오픈 소스들이 ClusterRole 을 사용하는 경우가 많음 (편하니까~)

- 오픈 소스 활용의 걸림돌

“Permit” or “Private Node” or “Private Cluster” or “WAITING”



3.1.3 요구사항에 따른 스케줄링

다양한 요구사항을 Affinity/Anti-Affinity 로 해결

- 노드의 분리

“Private Node” 로 전용의 리소스 보장, IP 나 Local Storage 점유, 다른 타입의 노드 (GPU) 등에 대응

- 리소스 장애 예방

CPU, Memory, IP 등의 특정 리소스가 부족해진 노드에 대한 실시간 Anti-Affinity

- 부하의 고른 분배

같은 타입의 Pod 이 여러 Node 나 Rack 에 분산하는데 활용 (Preferred)

3.2 Balance the Resources

"Push the Limits" > "한계를 뛰어넘다. "



"Balance the Resources" > "자원의 균형을 잡다."



→ 사용자나 노드에 자원 사용이나 할당에 대한 공평한 분배

"Provide the Convenience" > "편의성을 제공하다. "



3.2.1 Runtime 의 리소스 제한

LimitRange 정책으로 Container/Pod 의 리소스를 제한

- 리소스 Request/Limit

누구나 큰 값을 원한다. 하지만...

→ 너무 큰 값은 클러스터의 효율성을 저해하기 때문에 허용할 수 없다.

→ 그렇다고 너무 작은 값은 서비스의 정상적인 수행을 방해할 수 있다.

- 리소스 별 MIN/MAX 와 Ratio 에 의한 설정

리소스 별로 허용되는 min/max 에서 사용자가 Limit 선택

→ ex. CPU : 0.1 (min) ~ 4 (default) ~ 8 (max)

사용자가 선택한 Limit 에 일정 비율로 Request 설정

→ ex. CPU : Limit 4 → Request 0.4

3.2.2 사용자 별 리소스 제한

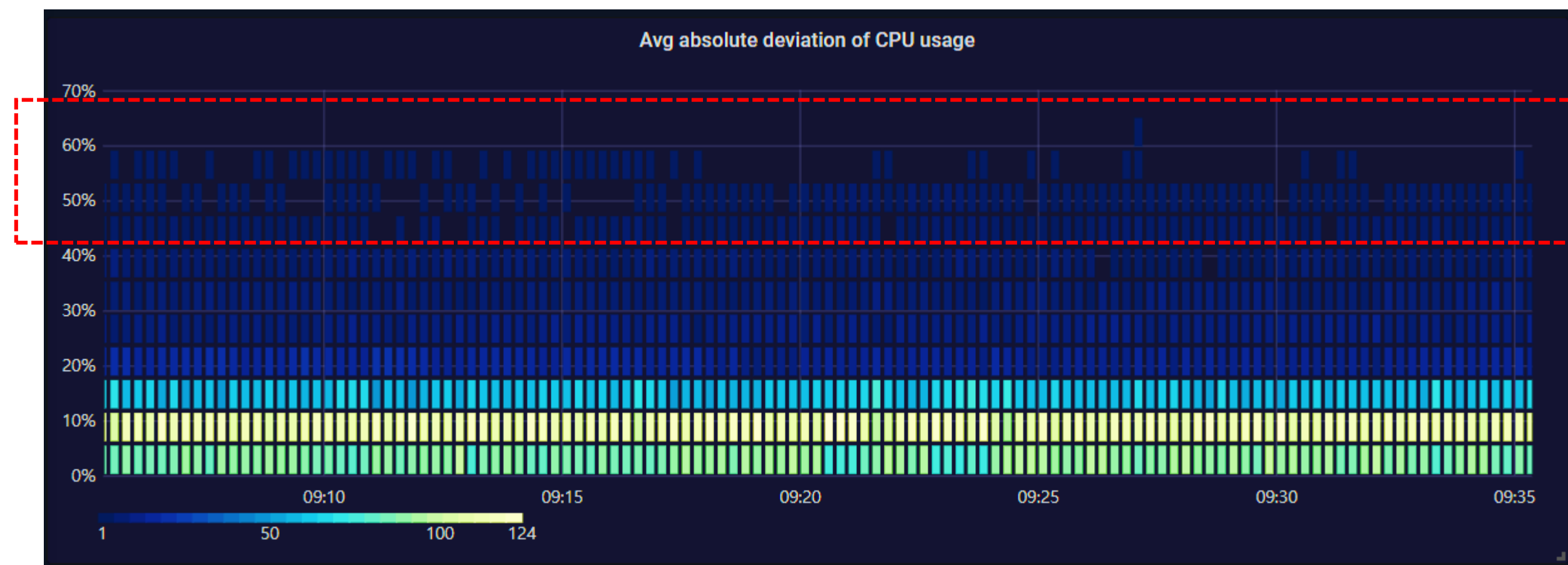
Namespace Quota 로 사용자 별 허용 리소스 제어

- 기본 3 ~ 4 Physical Machine 정도의 리소스 부여
 - 초기 별도 요청 없이 컨테이너화를 신속하게 시작할 수 있도록
 - 사용자가 증설을 요청하면 늘려주는 방식
- Quota 가이드/알림
 - Quota 관리 프로세스, Quota 알람, ...
 - 클러스터 Capacity Planning 에 활용

3.2.3 Re-Scheduling

여러 요인에 의해 클러스터 노드 간 리소스 불균형 발생

- Long Running Pod 이 많은 노드의 리소스 부족 (long running 할지는 알려주지 않으므로...)
- 사용자가 Request/Limit 을 잘 설정해주지 않음 (보통 기본값 그대로 사용)
- 이 모든걸 감안해도... 스케줄러가 아주 똑똑하지는 않은... (단순히 앞에 빈자리를 빨리 찾아주는 ...)



리소스 불균형 노드에 대한 Background Evict

리소스 편차 그래프

3.3 Provide the Convenience

"Push the Limits" > "한계를 뛰어넘다. "



"Balance the Resources" > "자원의 균형을 잡다."



"Provide the Convenience" > "편의성을 제공하다. "



→ 사용자가 Multi-Tenancy 에 대한 불편함을 최대한 느낄 수 없도록

3.3.1 자동 제공되는 모니터링

아마도… Prometheus/Grafana 는 누구나 필요할 거야…

- Prometheus/Grafana Provisioning

with 잘 정의된 대시보드 템플릿과 함께~

- 보다 편리한 알람

컨테이너 환경에 적합한 Availability, 사용자 별 튜닝이 가능한 Custom Metric

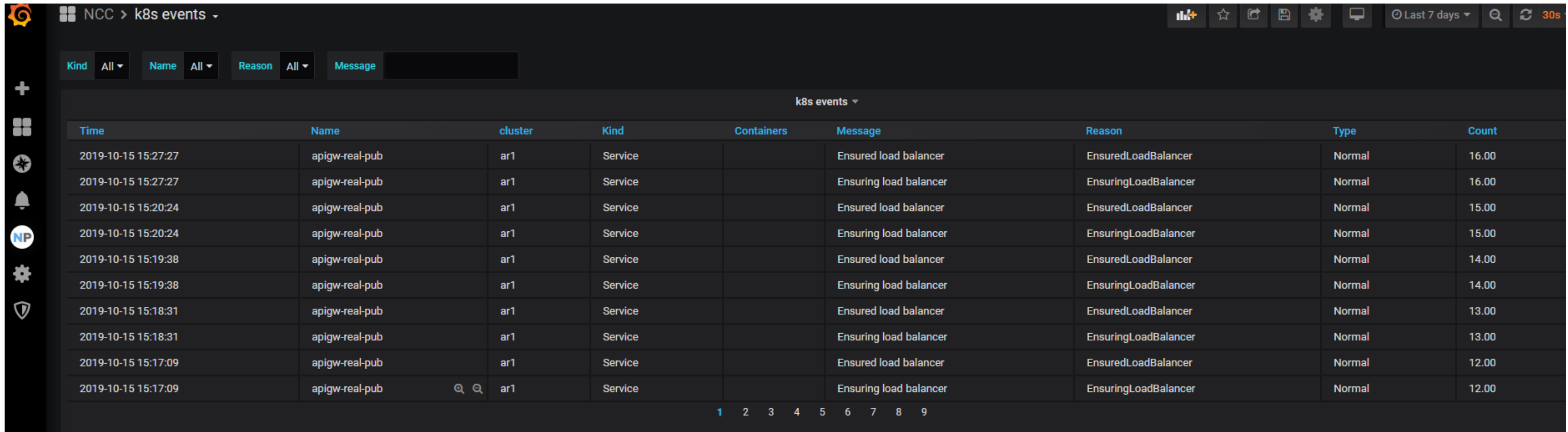
- 이슈

범용 모니터링 대시보드의 한계

자동 제공의 편리함 vs 무심결에 버려지는 리소스 → **Parking**

3.3.2 Kubernetes Event Logging

Kubernetes 의 Event 를 사용자가 직접 확인할 수 있도록 제공



The screenshot shows the 'NCC > k8s events' page. It features a table with columns: Time, Name, cluster, Kind, Containers, Message, Reason, Type, and Count. The table lists several events for the 'apigw-real-pub' service on the 'ar1' cluster, showing 'Ensured load balancer' and 'Ensuring load balancer' messages. The interface includes filters for Kind, Name, Reason, and Message, and a sidebar with navigation icons.

Time	Name	cluster	Kind	Containers	Message	Reason	Type	Count
2019-10-15 15:27:27	apigw-real-pub	ar1	Service		Ensured load balancer	EnsuredLoadBalancer	Normal	16.00
2019-10-15 15:27:27	apigw-real-pub	ar1	Service		Ensuring load balancer	EnsuringLoadBalancer	Normal	16.00
2019-10-15 15:20:24	apigw-real-pub	ar1	Service		Ensured load balancer	EnsuredLoadBalancer	Normal	15.00
2019-10-15 15:20:24	apigw-real-pub	ar1	Service		Ensuring load balancer	EnsuringLoadBalancer	Normal	15.00
2019-10-15 15:19:38	apigw-real-pub	ar1	Service		Ensured load balancer	EnsuredLoadBalancer	Normal	14.00
2019-10-15 15:19:38	apigw-real-pub	ar1	Service		Ensuring load balancer	EnsuringLoadBalancer	Normal	14.00
2019-10-15 15:18:31	apigw-real-pub	ar1	Service		Ensured load balancer	EnsuredLoadBalancer	Normal	13.00
2019-10-15 15:18:31	apigw-real-pub	ar1	Service		Ensuring load balancer	EnsuringLoadBalancer	Normal	13.00
2019-10-15 15:17:09	apigw-real-pub	ar1	Service		Ensured load balancer	EnsuredLoadBalancer	Normal	12.00
2019-10-15 15:17:09	apigw-real-pub	ar1	Service		Ensuring load balancer	EnsuringLoadBalancer	Normal	12.00

- 사용자가 이것이 자신의 로그인 것을 쉽게 인지하지 못함...

4. Next Step For Better World 😊

4.1 Isolation

Network, Resource, Runtime 에 대한 Isolation 강화

- **Virtual Private Cluster : Container Network**

Isolated Overlay Network + High Performance NAT

- **Resource Rate Limiting : Network, Storage**

Network 과 Storage 에 대한 “Adaptive Quality of Service”

- **More Isolated Runtime : kata, gVisor**

아직은 그저 주시하는 단계... (성능, 호환성, ...)

4.2 Site Reliable Engineering

쏟아지는 알람, 작업, 요청에 내년에도 살아남아야 한다!

- 보다 자동화된 운영 방법과 툴

기준 운영 방법/툴 + 자동화, GUI 기반 도구

- Kubernetes Reliable Engineering + SRE

당장은 Kubernetes 자체를 잘 관리하는 것도 매우 힘든 문제 T_T

하지만 언젠가는 Multi-Tenancy 의 장점을 잘 살려서 진정한 DevOps SRE 를 해보리라.

4.3 Cloud Native

보다 많은 Cloud Native 서비스로 네이버의 경쟁력을 향상하도록...

- 보다 많은 StatefulSets

Vitess, Redis, Mongo, ...

- GPU 에 대한 지원

GPU 를 지원하기 위해 검토 중



Q & A

We're Hiring! : dl_join_ncc@navercorp.com



Thank You

We're Hiring! : dl_join_ncc@navercorp.com

