2015 Fall
CREATIVE INTEGRATED DESIGN 1

AWS CloudWatch와 Lambda를 이용한 사용량 모니터링 시스템

2015. 12. 18

Team: K
컴퓨터공학부 2008-11763 황호성
자유전공학부 2010-13446 정형식

Contents

- Overview
- Problem & Goal
- Result
  - Architecture
  - Tag based
  - Log storage
- Demo

Overview

- **프로젝트명**

  Sauron(Monitor everything!)

- **프로젝트 목표**

  AWS CloudWatch와 Lambda를 이용해 클라우드 서비스에 대한 자원 사용 현황을 모니터링하는 시스템을 만든다.

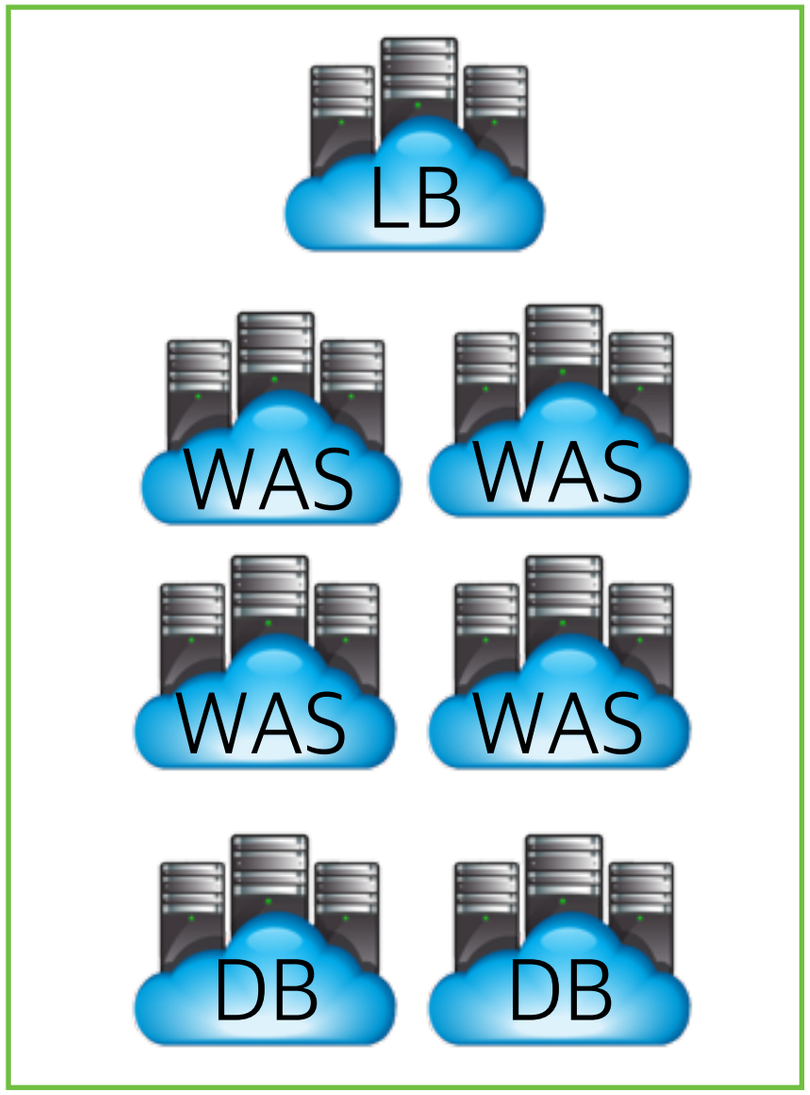
- **모니터링 대상 서비스**

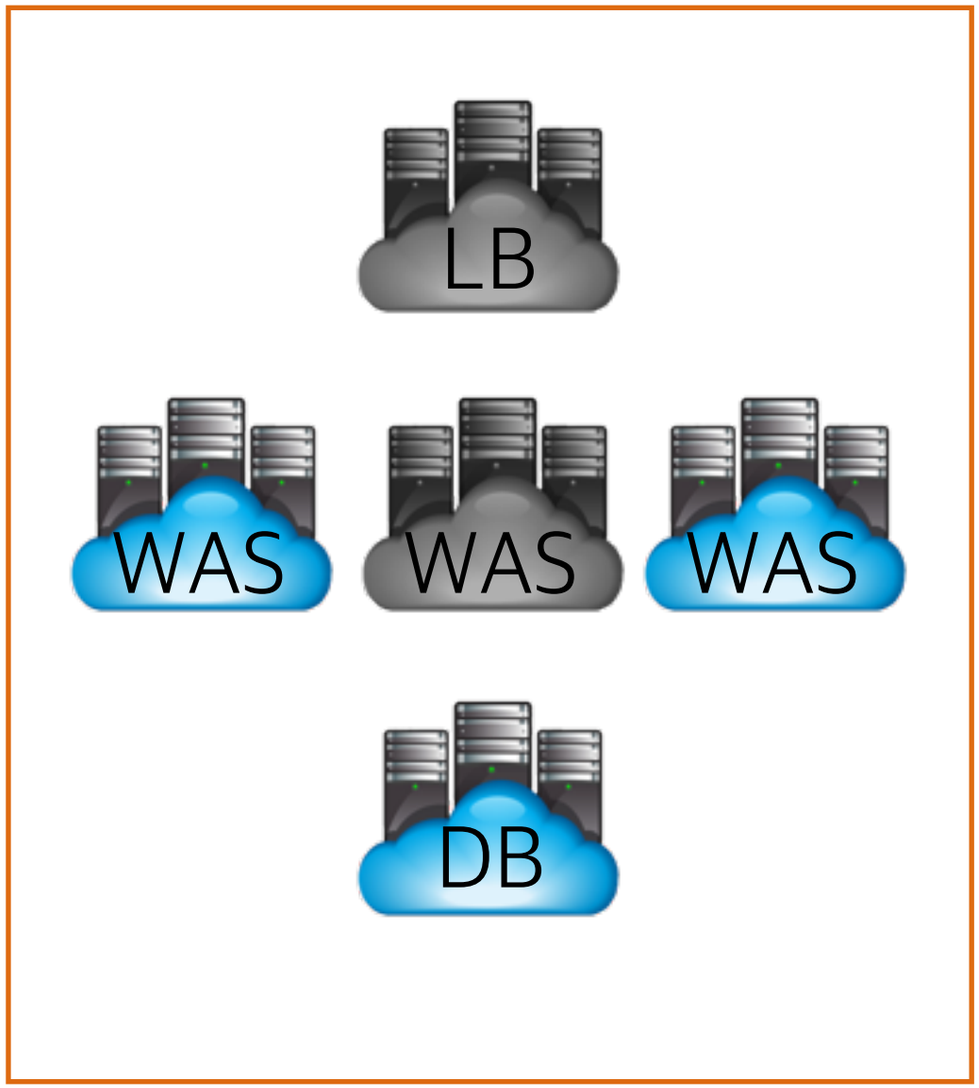
  AWS의 클라우드 서비스를 이용해 서비스하고있는 리소스군
  (ELB-로드밸런서, EC2-서버, RDS-데이터베이스)

Problem(1)

- 클라우드 상의 여러 개의 서비스, 수많은 인스턴스
- 평면적인 모니터링 시스템으로는 장애발생시 원인에 대한 overview를 갖기 어려움

Goal(1)

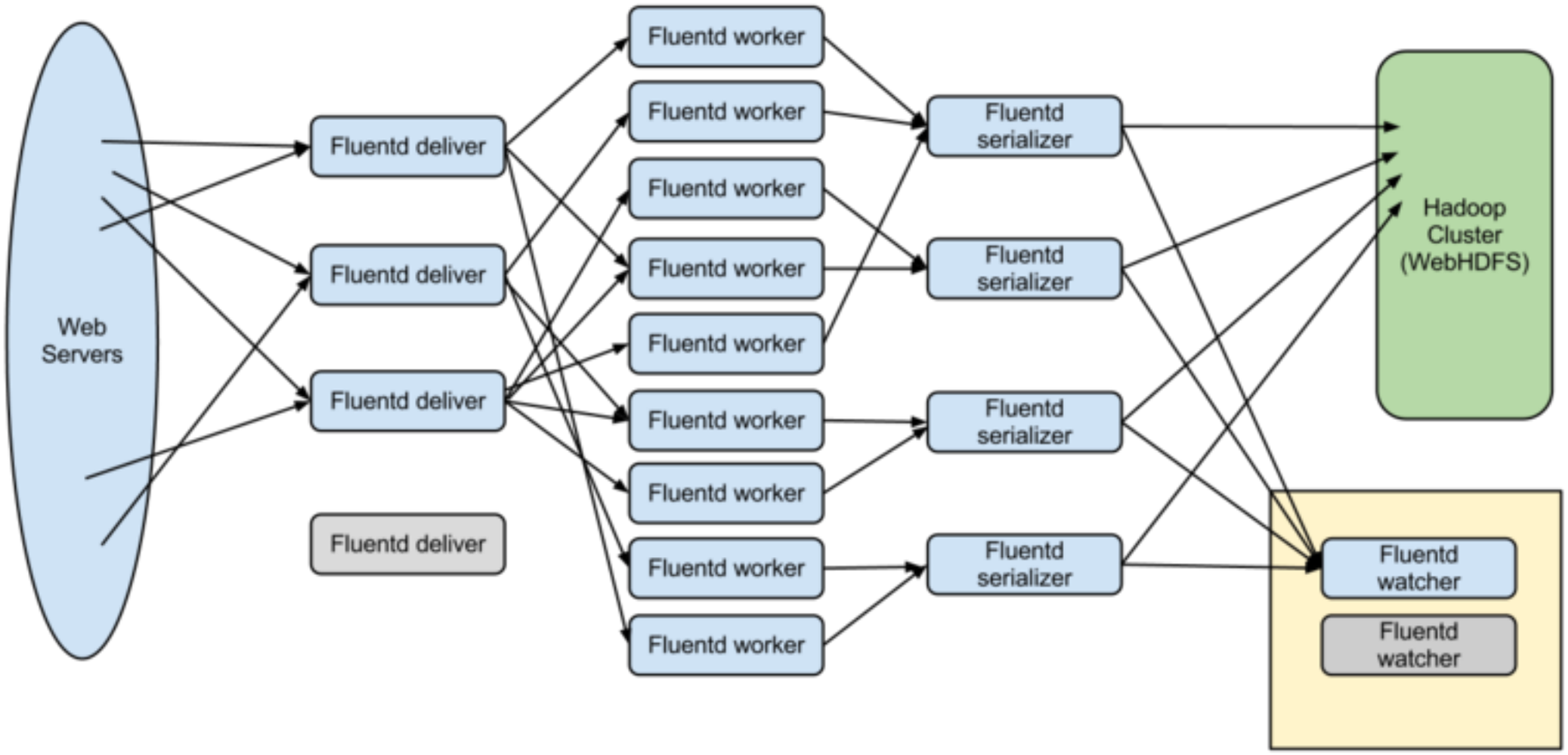




- 서비스 스택 구성을 감안한 모니터링 시스템

amazon web services

Problem(2)

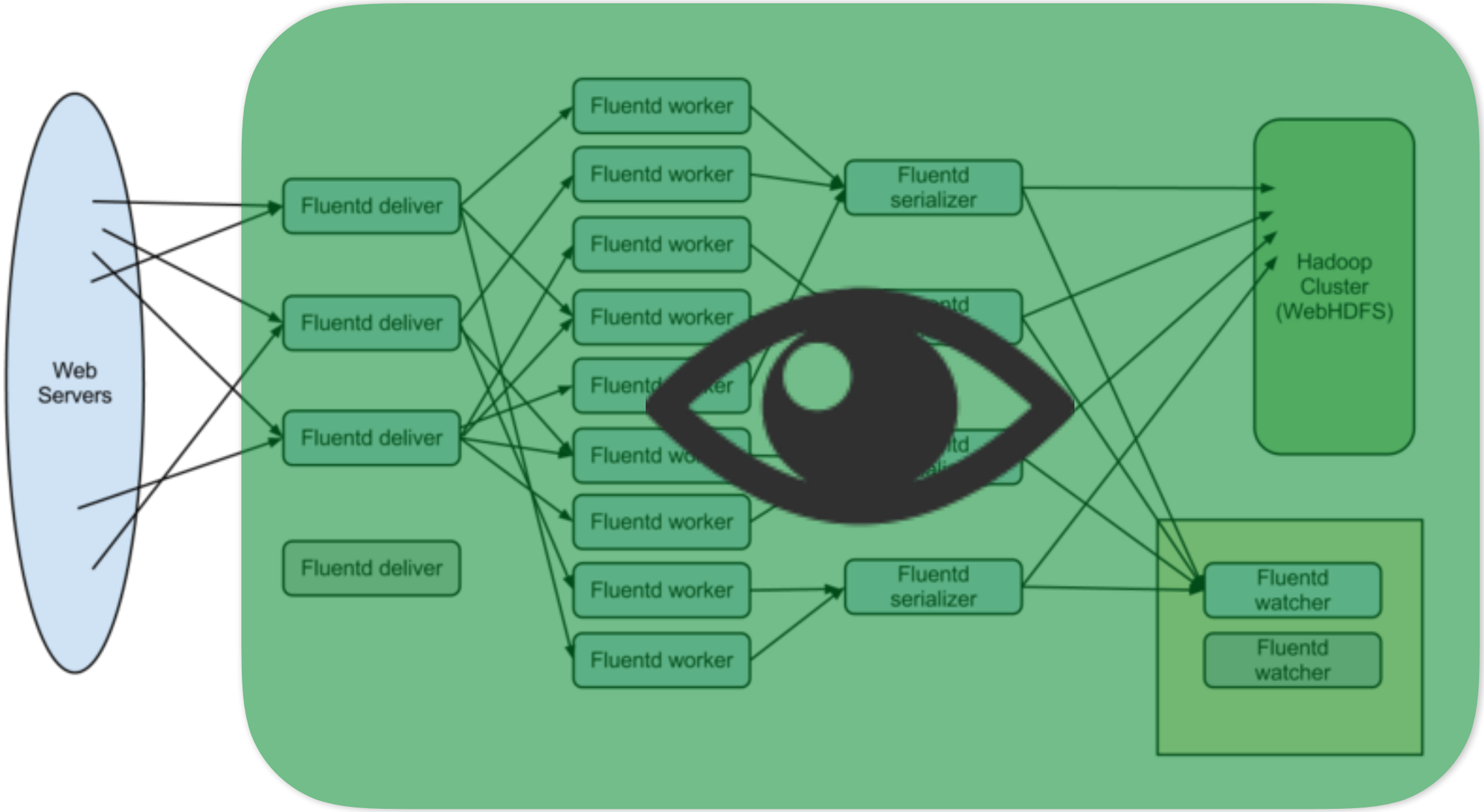


"too many points of failure"

시스템을 관리하기 위한 시스템을 관리

amazon web services

Goal(2)



가용성이 확보된 AWS 서비스의 조합으로
관리비용 최소화

amazon web services™

Our Goal

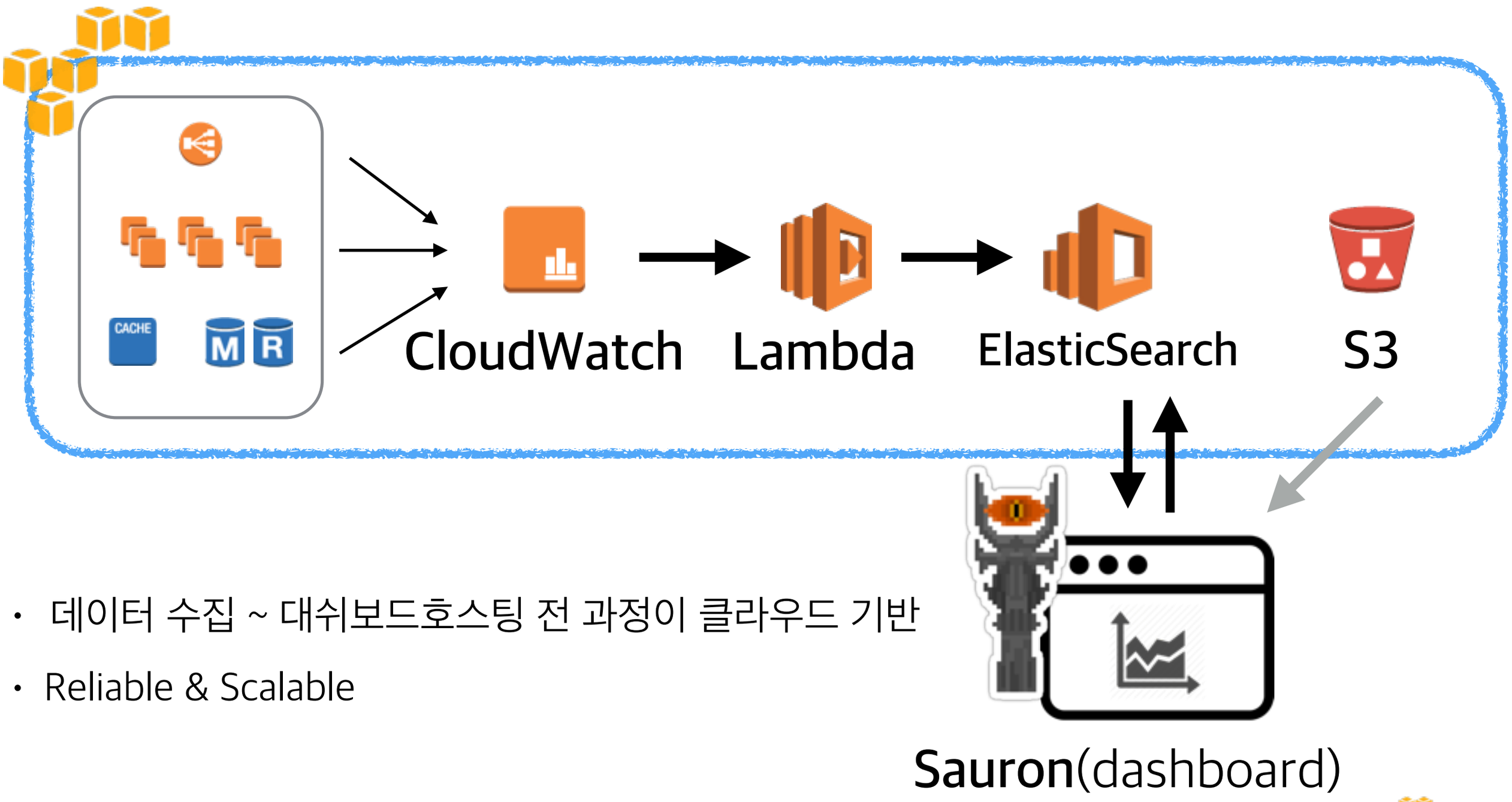
1. 인스턴스의 구성이 쉽게 파악되고
2. 관리비용이 최소화된

클라우드 서비스 모니터링 시스템

amazon web services™

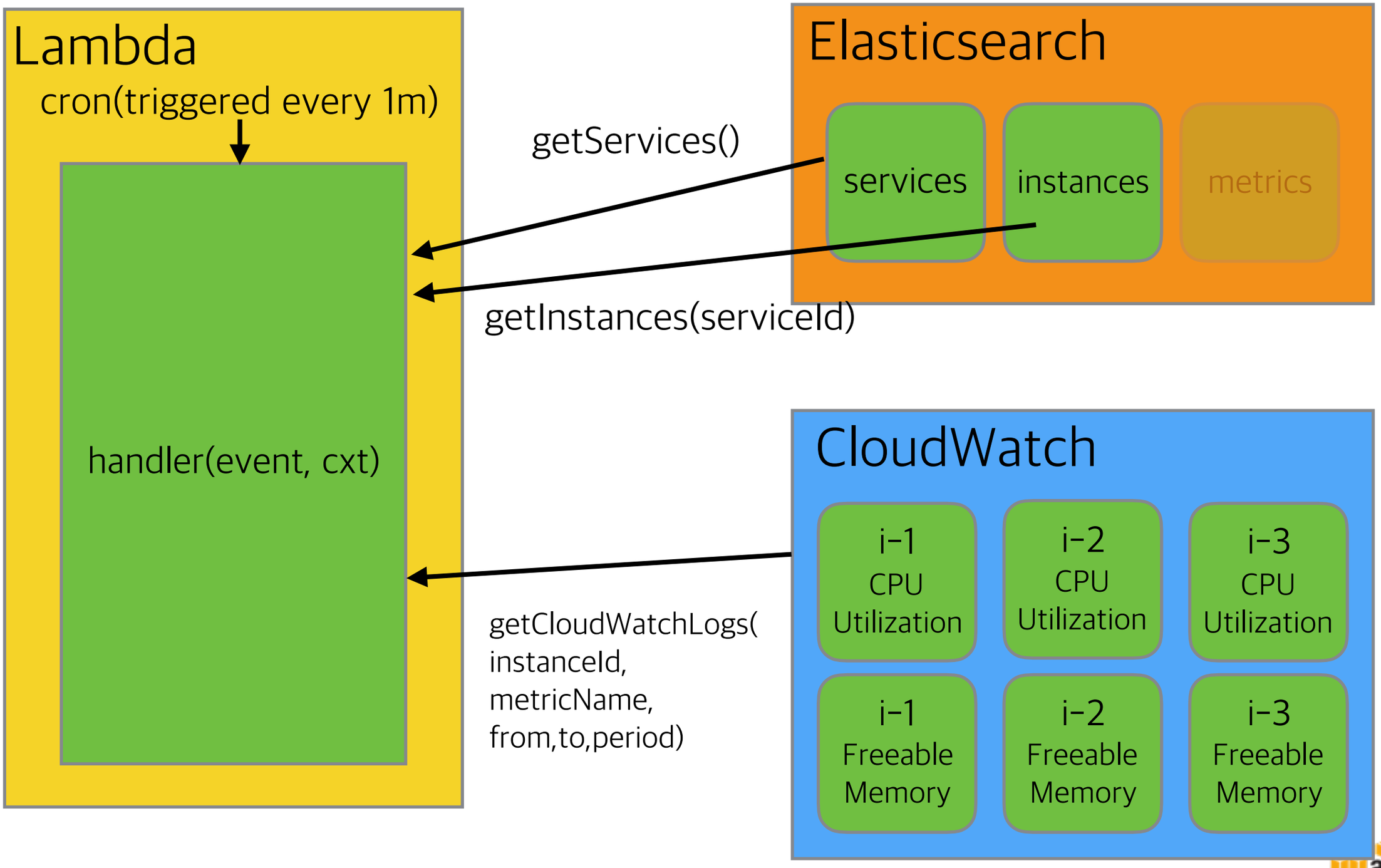
Our result(1):

Cloud-native architecture

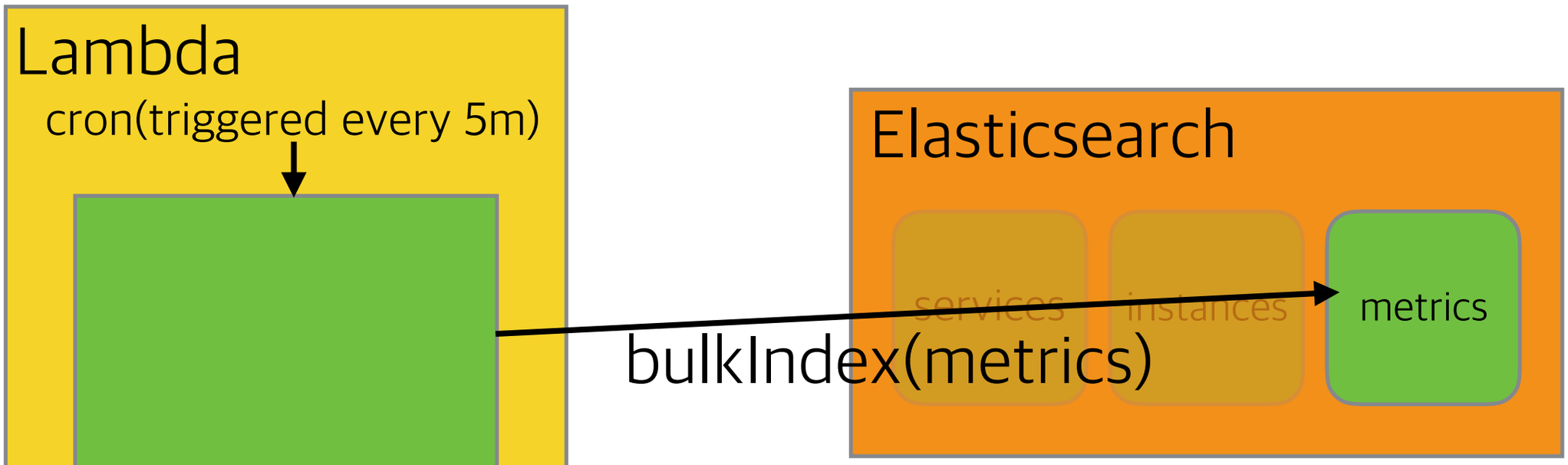


- 데이터 수집 ~ 대쉬보드호스팅 전 과정이 클라우드 기반
- Reliable & Scalable

Implementation Spec – 1) Collect



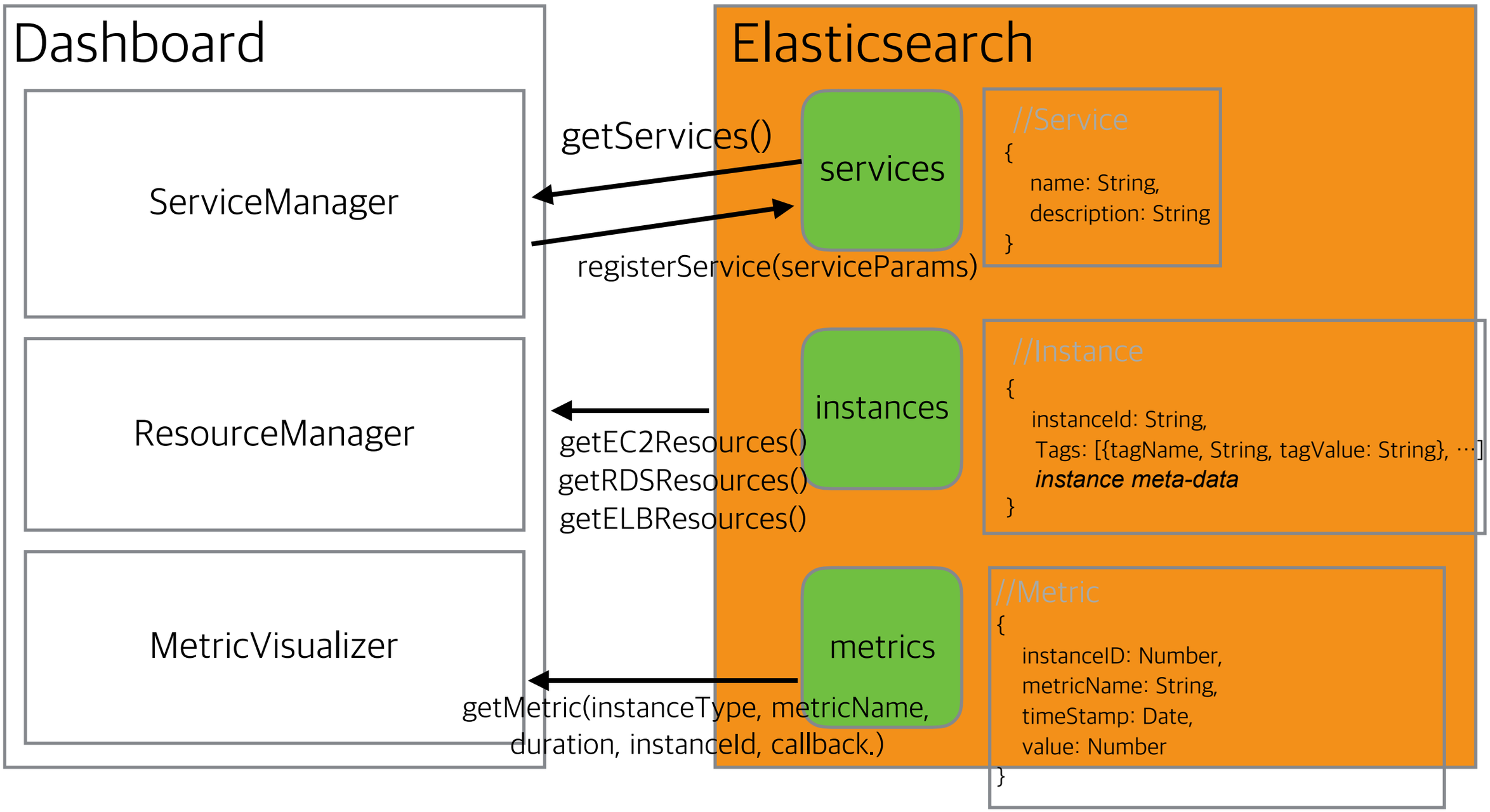
Implementation Spec – 2) Analyze



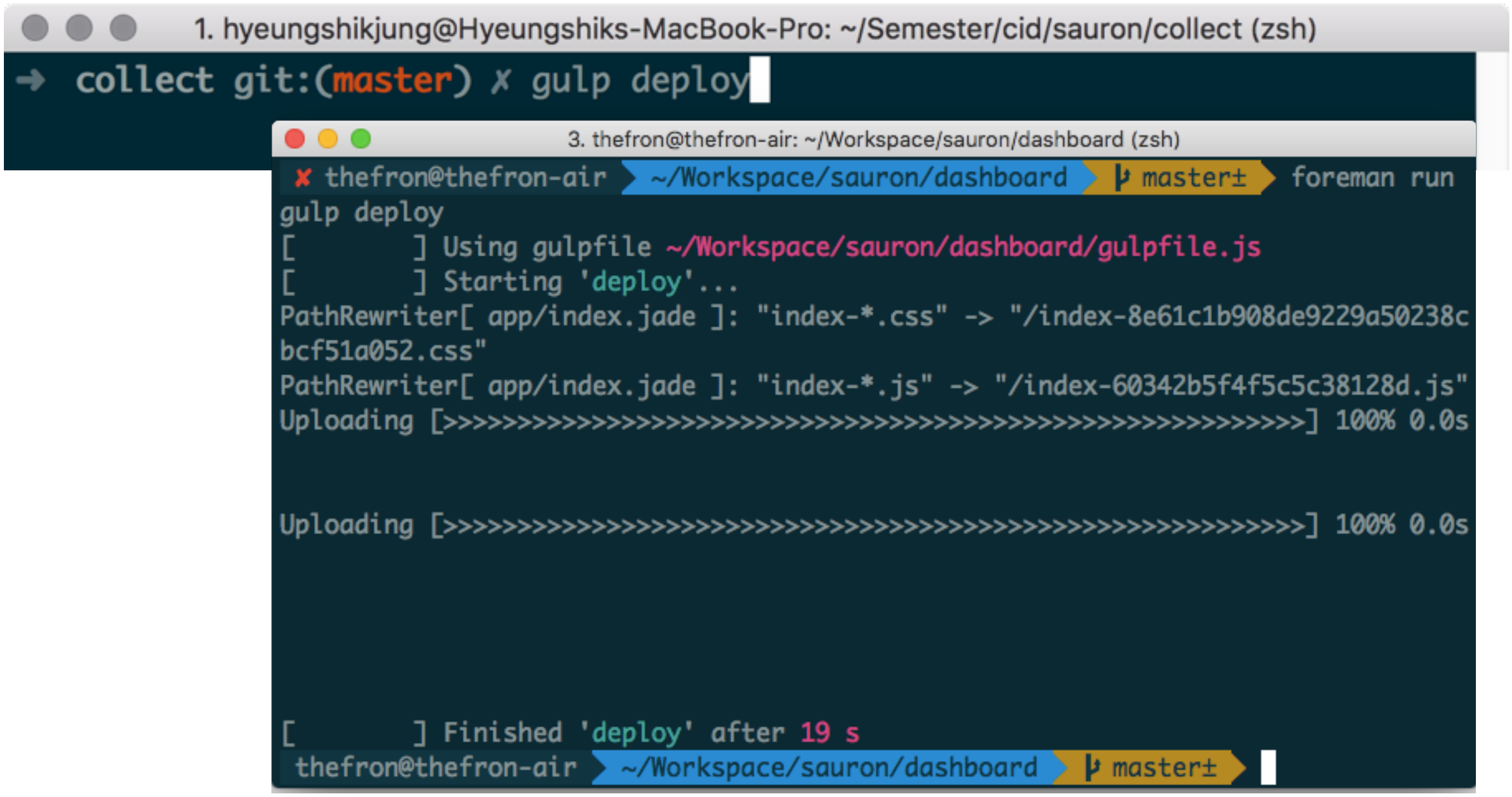

```
//example JSON to be indexed

{
    instanceID: “i-b873e3bd”,
    metricName: “CPUUtilization,
    timeStamp: "2015-12-17T06:00:00.000Z"
    value: 0.5
}
```

Implementation Spec – 3) Visualize



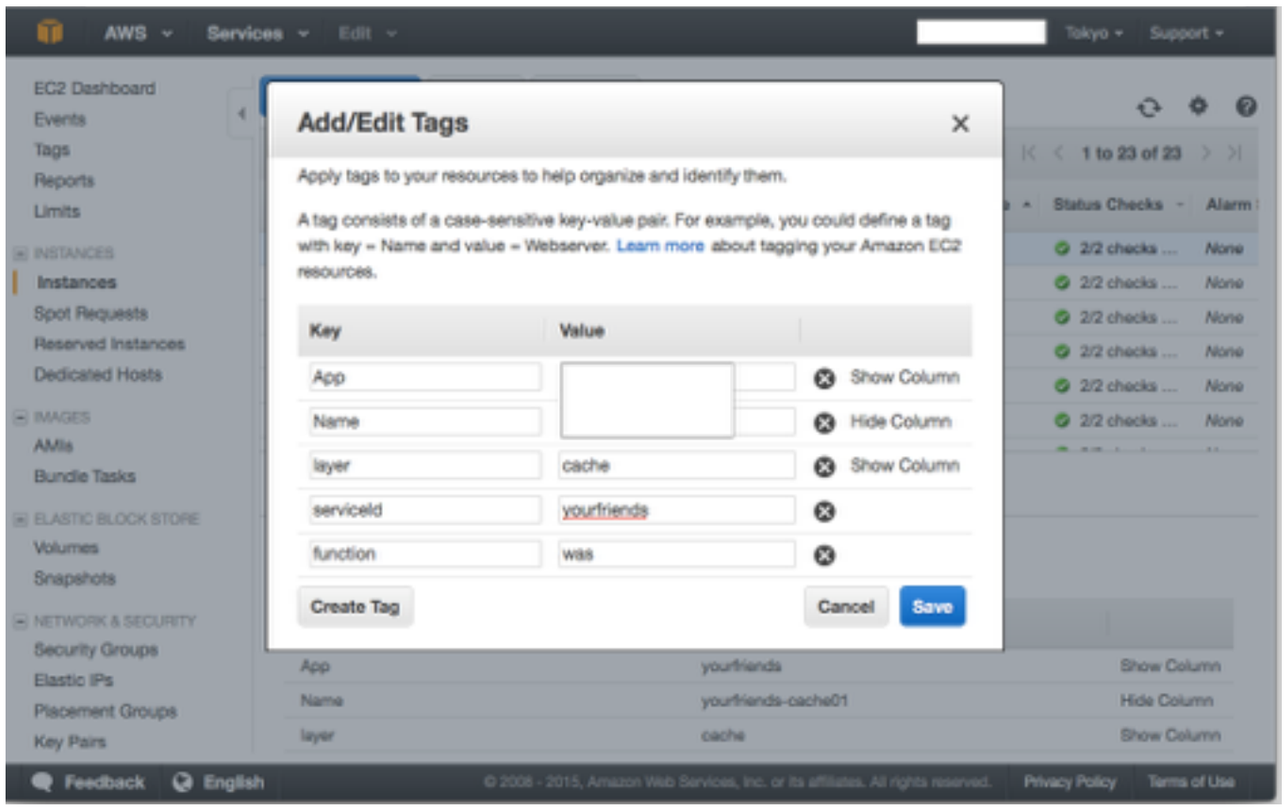
Dead Simple Deployment



amazon web services™

Our result(2):

#Tag-based monitoring



- AWS콘솔에서 편집가능한 Tag를 이용하여 분류
- ‘서비스’별 인스턴스 현황을 한눈에 모니터링

amazon web services

Our result(3):

Elasticsearch as log storage

- 처음에 DynamoDB에 denormalize된 metrics를
  바로 저장하려했으나, Elasticsearch Service 출시로 방향 전환

- DynamoDB를 쓸 경우 raw metric이 아닌
  Denormalized View를 저장해야 함 => 람다의 complexity 증가

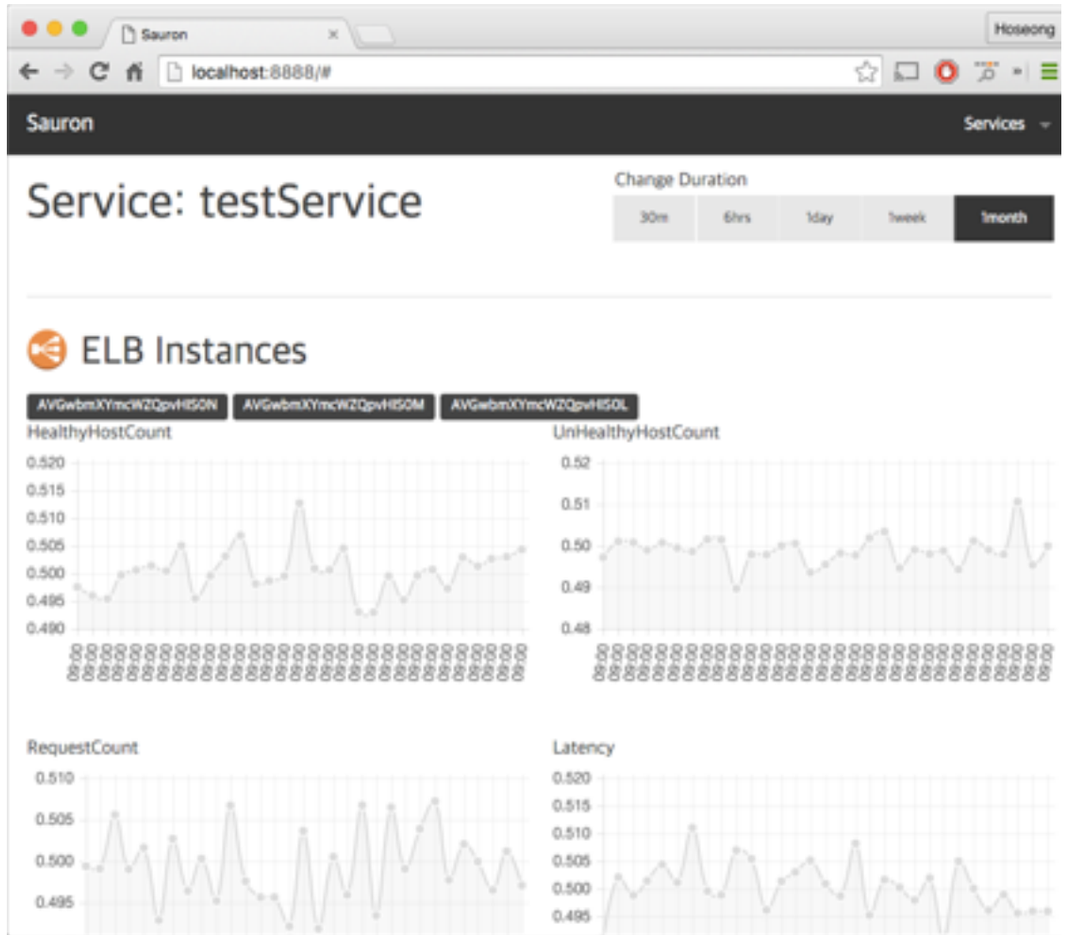
- 현재 Raw metric을 Elasticseach에 indexing하고
  대쉬보드에서 aggregation query를 이용해서 뷰 생성
  => 대쉬보드 기능 추가/변경 용이

Our result(3):

Elasticsearch as log storage

- 성능지표를 테스트하기 위해서 랜덤 메트릭을 발생시켜
  ES에 인덱싱하는 스크립트 작성
- 리소스 12개, 메트릭 16종류, 1분 단위로 한 달 분량의 메트릭 생성


```
{
  - _shards: {
        total: 10,
        successful: 10,
        failed: 0
    },
  - _all: {
      - primaries: {
          - docs: {
                count: 2767680,
                deleted: 0
            },
          - store: {
                size_in_bytes: 306639988,
                throttle_time_in_millis: 74007
            },
```

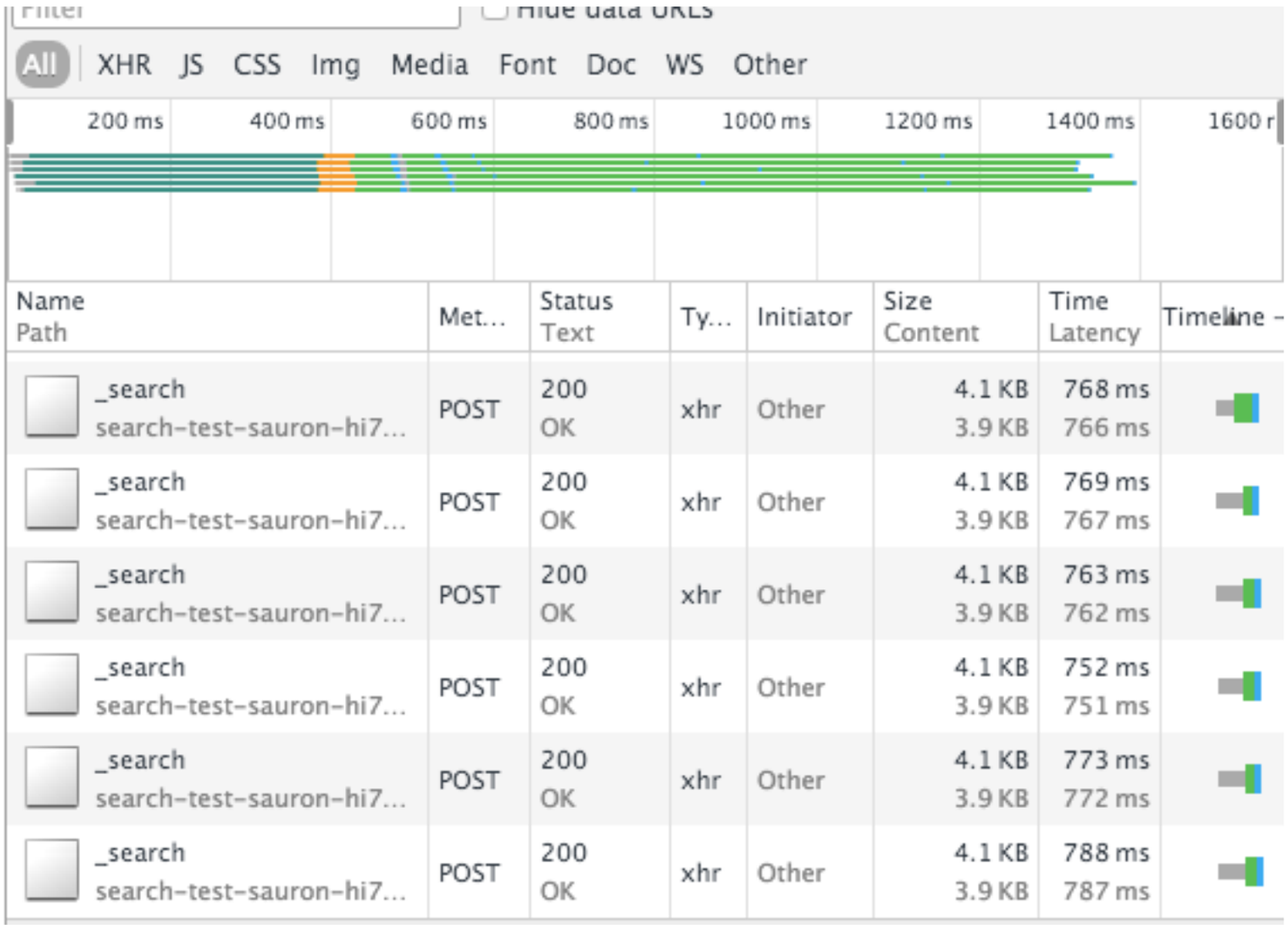



amazon web services

Our result(3):

Elasticsearch as log storage

- 14개의 메트릭이 한 달 resolution으로 1초 내에 갱신



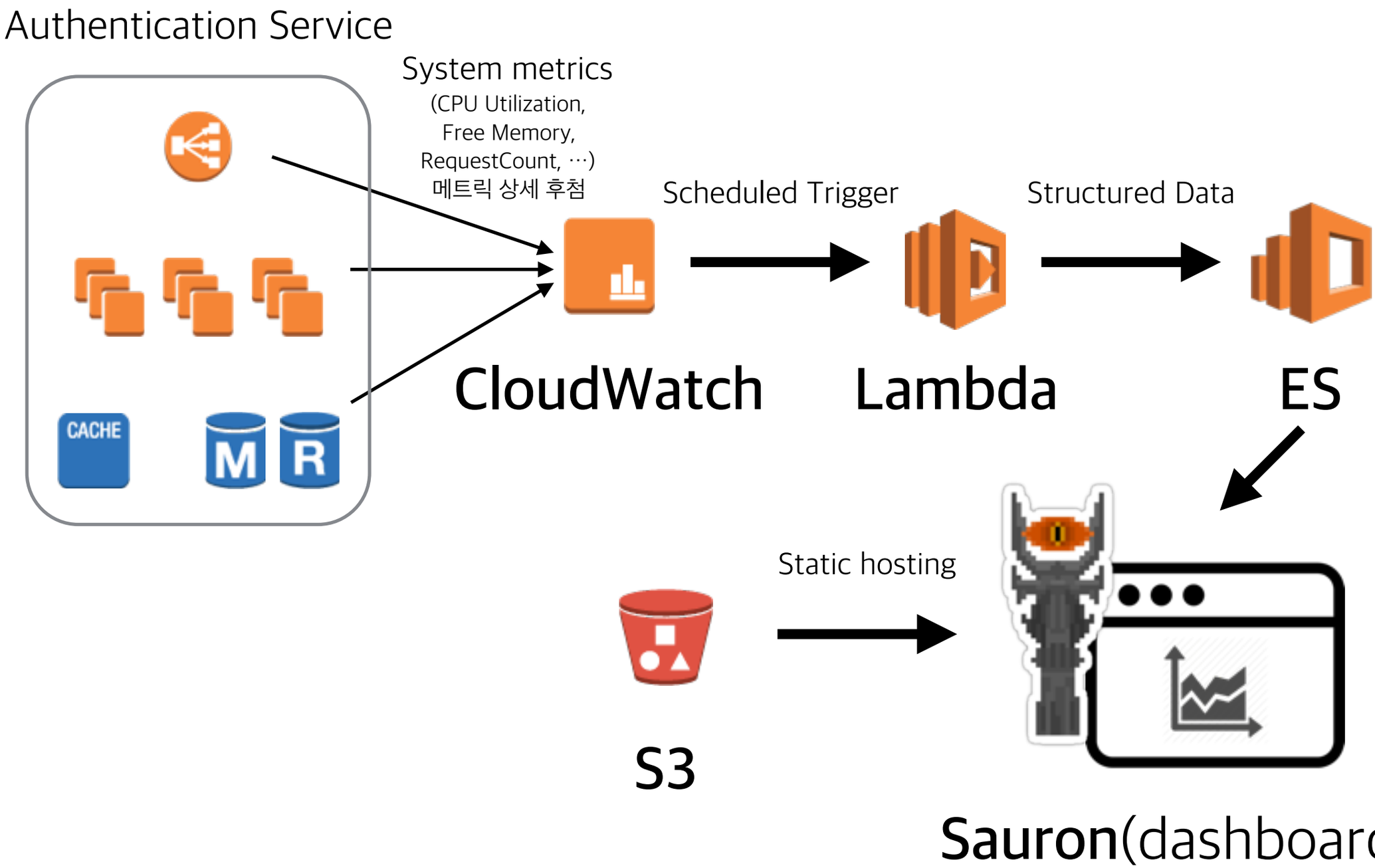
Demo

감사합니다

황호성 thefron@snu.ac.kr
정형식 cescc@snu.ac.kr

Appendixes

Architecture – Diagram



Goal/Problem

- 기존 솔루션(nagious, cacti, …)은 머신 단위로 시스템 상태를 수집하기 때문에 개별 서비스화된 클라우드 자원을 모니터링하기에 부적합.

- 기존 솔루션은 서비스 규모가 커지면 관리 비용이 매우 커짐.

- AWS의 자원의 각종 시스템 메트릭은 CloudWatch 라는 서비스를 통해 API화돼 있으나 서비스 스택 전체의 현황을 한눈에 보기 쉽지 않음.

amazon web services

Goal/Problem

- 기존 솔루션(nagious, cacti, …)은 머신 단위로 시스템 상태를 수집하기 때문에 개별 서비스화된 클라우드 자원을 모니터링하기에 부적합.

- 기존 솔루션은 서비스 규모가 커지면 관리 비용이 매우 커짐.

- AWS의 자원의 각종 시스템 메트릭은 CloudWatch 라는 서비스를 통해 API화돼 있으나 서비스 스택 전체의 현황을 한눈에 보기 쉽지 않음.

amazon web services

Goal/Problem

- 클라우드 자원에 최적화된 모니터링 시스템 구현
- RDBMS나 별도 서버 인스턴스를 이용하지 않고 AWS 서비스만으로 관리 비용을 최소화
- CloudWatch의 짧은 로그 리텐션을 극복

Requirements

- Sauron 시스템에 모니터링할 서비스 스택을 등록하는 기능
- 태깅을 이용해 서비스 스택에 하위 인스턴스 자동 할당하는 기능
- Lambda를 이용해 실시간으로 CloudWatch의 로그를 분석해 AWS ES에 인덱싱하는 기능
- 대시보드에서 서비스그룹, 기능별로 인스턴스 시각화
- 기능별, 인스턴스별로 메트릭을 차트로 표현하는 기능

Approach

Tag를 이용해 시스템 리소스를 논리그룹으로 표현

ServiceId=authentication
Function=LB

ServiceId=authentication
Function=WAS

ServiceId=authentication
Function=DB

ServiceId=bidding
Function=LB

ServiceId=bidding
Function=WAS

ServiceId=bidding
Function=DB

Approach

Serverless & Scalable

CloudWatch: CPU 사용률, 네트워크 사용량, DB 상태 등 사용자가 AWS 에서 사용하는 다양한 서비스들의 로그를 수집, 저장

Lambda: 업로드한 코드 조각을 미리 지정된 이벤트에 응답하여 실행하는 방식으로 확장성이나 보안에 신경쓰지 않고 백엔드 구축 가능

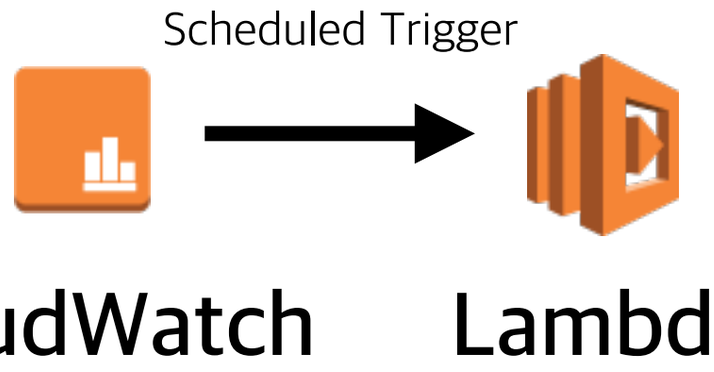
Amazon ES: 오픈소스 검색/분석 툴인 ElasticSearch를 서비스 형태로 사용할 수 있음.

S3: 객체 스토리지. 서버 로직없는 웹서비스 호스팅 가능

Development Environment

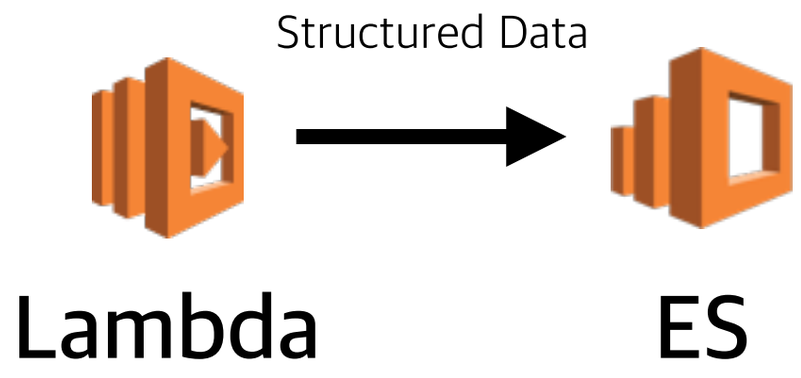
- Any environment supports Node.js
- AWS stack: CloudWatch, Lambda, S3, ES, AWS SDK
- Dashboard stack
  - Javascript: ES6 + Backbone.js(MVC Framework)
  - CSS using SCSS
  - Build and manage application using webpack & gulp

Architecture – 1) Collect



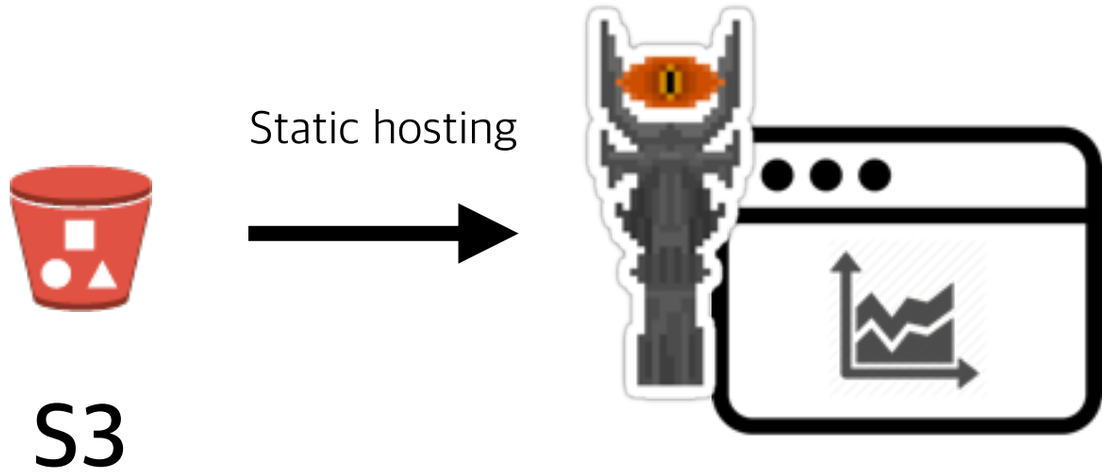
- Sauron이 관리하는 인스턴스 목록 조회
- Lambda를 이용해 수집해야하는 인스턴스에 대한 CloudWatch 메트릭 조회

Architecture – 2) Analyze



- Collect에서 쿼리한 메트릭에 인스턴스와 태깅 정보를 추가한 ES 문서로 변환 및 Bulk 인덱싱

Architecture – 3) Visualize

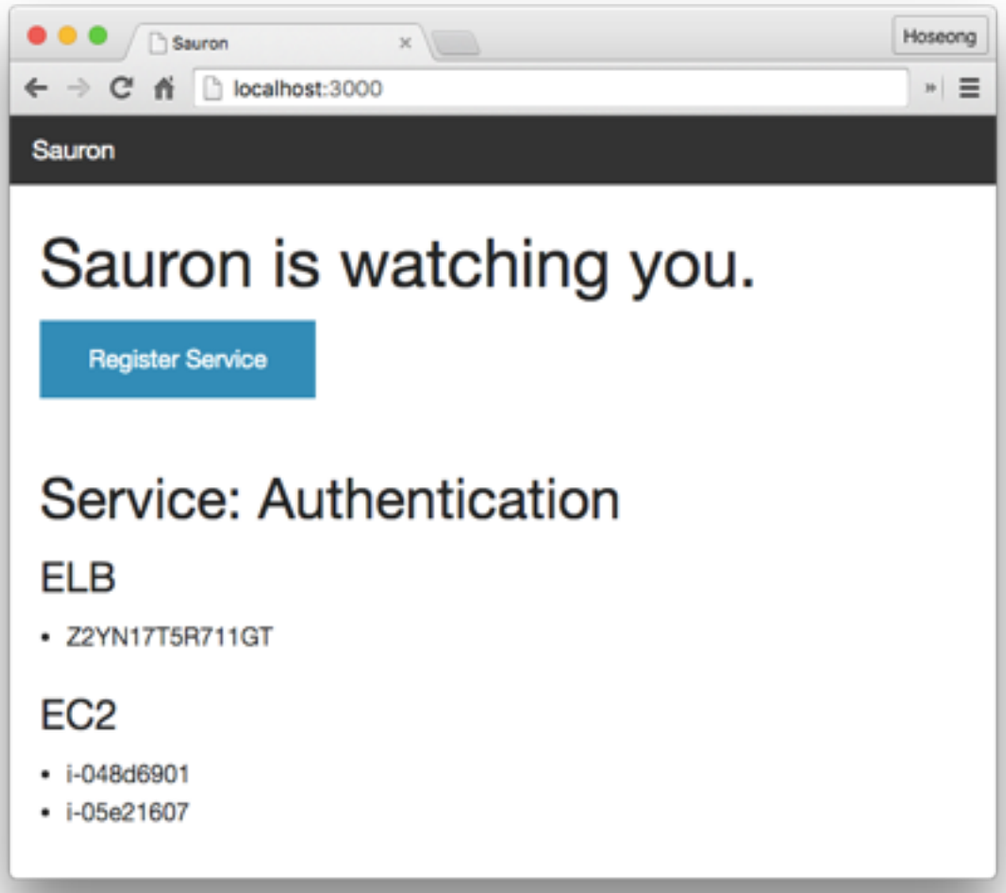


- Sauron에 등록된 서비스 그룹 조회 및 관리
- 인스턴스를 기능단위로 묶고, 기능단위를 묶어 전체 서비스 그룹으로 시각화
- 기능 묶음별, 개별 인스턴스별 시스템 지표를 차트로 시각화

Current Status

- 람다를 이용해서 Cloudwatch에 있는 로그를 조회 및 ES에 저장

- Dashboard 프론트엔드 부트스트래핑
  - 서비스 등록
  - 서비스 하위 인스턴스 조회



Further Plan

- 서비스 내 인스턴스 갱신 기능
- 메트릭 조회 쿼리 작성
- 대시보드에서 서비스 스택 트리구조 시각화
- 기능별, 인스턴스별 메트릭 차트로 표현
- 메트릭 조회 기간 설정
- 실시간 차트 갱신

Division and Assignment of Work

구분	항목	담당자
Collect	CloudWatch 알림 조건 설정 및 SNS 연동	황호성
	Lambda로 위 SNS를 구독 & metric을 파싱	황호성
Analyse	저장할 metric structure 정의	정형식
	Lambda로 metric aggregation	정형식
Visualize	Dashboard Bootstrapping	황호성
	Write ES Queries /w aggregation	정형식
	Chart Visualisation	정형식

Schedule

내용	9월				10월				11월				12월	
	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	1주	2주
Background Study (AWS Services)	▶	▶	▶											
Background Study (Monitoring Solutions)			▶	▶										
Collect log data using CloudWatch&Lambda				▶	▶									
Design Data Structure					▶	▶								
Save logs into Elasticsearch in defined						▶	▶							
Background Study (Elasticsearch, Visualisation library)						▶	▶							
Show real-time data in text								▶	▶					
Show long-retention data within given period									▶	▶				
Show datas in graph and chart										▶	▶	▶		

Appendix

1. 수집 메트릭 상세 명세

1. 수집 메트릭 명세 - 로드밸런서

- HealthyHostCount(count): 해당 로드밸런서에 속한 서버 중 동작하고 있는 서버의 수
- UnHealthyHostCount(count): 해당 로드밸런서에 속한 서버 중 동작하지 않고 있는 서버의 수.해당 메트릭이 높아지면 장애 발생의 우려가 높음
- RequestCount(count): 로드밸런서가 처리하고 있는 단위시간당 리퀘스트의 수. 현재 로드밸런서에 걸리고 있는 부하를 나타내는 지표
- Latency(ms): 리퀘스트에 대한 응답이 나갈 때 까지 걸리는 지연시간을 의미하며, 지연시간이 길어지면 어플리케이션 서버나 데이터베이스 등에서 병목현상 혹은 장애가 발생했을 확률

1. 수집 메트릭 명세 – 웹서버

- CPUUtilization(%): 해당 서버의 cpu 사용량. 해당 지표가 높아지면 병목이 발생하거나 요청이 증가했을 확률

- NetworkIn(Bytes): 해당 서버로 유입되는 네트워크 트래픽량. 해당 지표가 높아지면 요청이 급증했음을 의미

- NetworkOut(Bytes): 해당 서버에서 바깥으로 나가는 네트워크 트래픽량

1. 수집 메트릭 명세 - 데이터베이스

- CPUUtilization(%): 해당 데이터베이스 인스턴스의 cpu 사용량
- FreeableMemory(Bytes): 해당 데이터베이스의 여유 메모리양
- ReadThroughput(Bytes): 디스크에서 초당 read하고 있는 평균 Byte
- WriteThroughput(Bytes): 디스크에 초당 write하고 있는 평균 Byte
- SwapUsage(Bytes): 해당 데이터베이스에서 발생하고 있는 스왑 공간의 크기

감사합니다

황호성 thefron@snu.ac.kr
정형식 cescc@snu.ac.kr