BESPIN GLOBAL
HELPING YOU ADOPT CLOUD.

Google Cloud

#부하테스트 #GKE #AUTOSCALE #LOCUST #HPA

부하테스트 어디까지 해봤니?

a.k.a GKE와 Locust를 활용한 대규모 부하테스트

9월29일 (목) 오후 2시-3시 YouTube

발표자 소개

발표자 소개

"베스핀글로벌에서 Google Cloud 사업본부에서
GCP 클라우드 구축 및 운영 업무를 담당하고 있으며
빅데이터 파이프라인의 서비스를 위한
클라우드 시스템 안정적인 운영을 위해 노력하고 있습니다."

백승렬 차장

베스핀글로벌 Google Cloud Architect

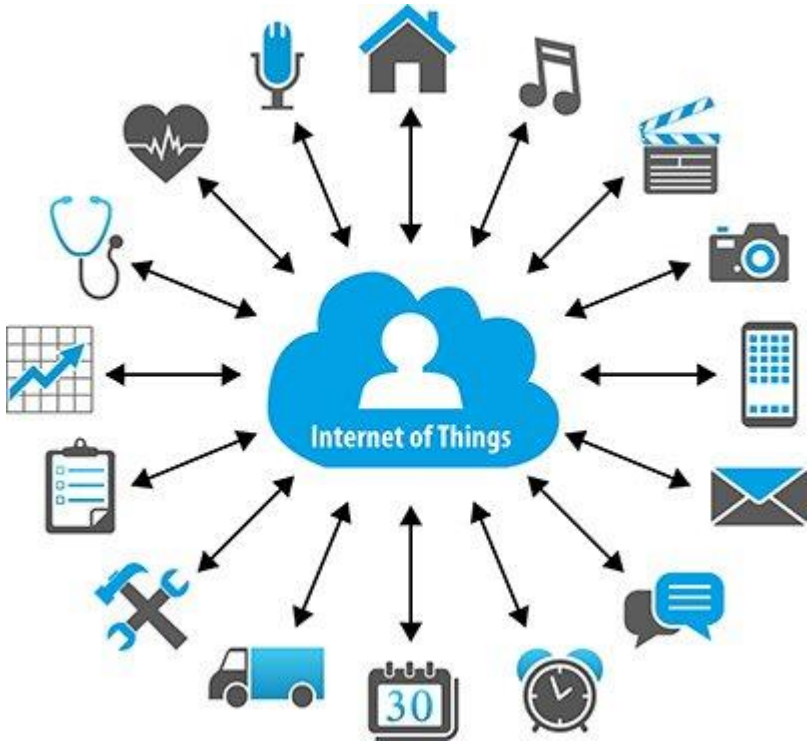
아젠다

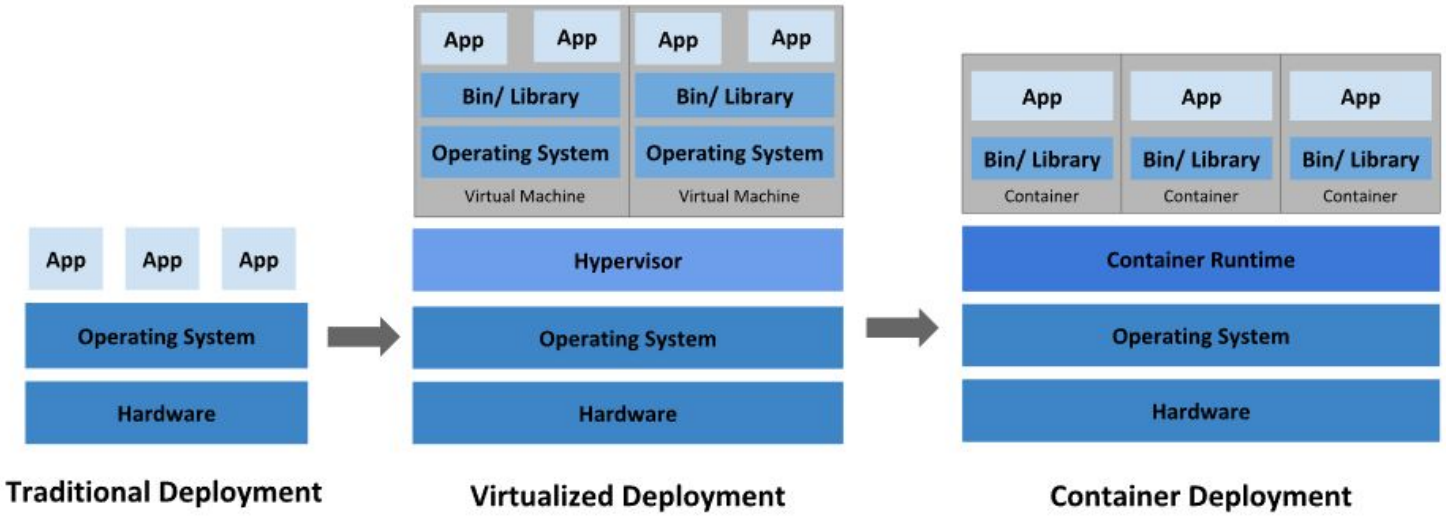
세션 구성

- 대규모 부하테스트에 대한 인프라 및 Tool 선정 (10’)
- Locust 활용한 부하테스트 (25’)
- 부하테스트를 통한 GKE Autoscale 최적화 방법 (10’)
- 베스핀글로벌 고객 사례 (5’)
- Q & A (10’)

대규모 부하테스트에 대한 인프라 및 Tool 선정

시나리오





1억개의 센서 IOT 데이터 수집 서비스

기존 사용중인 150대의 서버 -> GKE 환경 전환

BESPIN GLOBAL Google Cloud

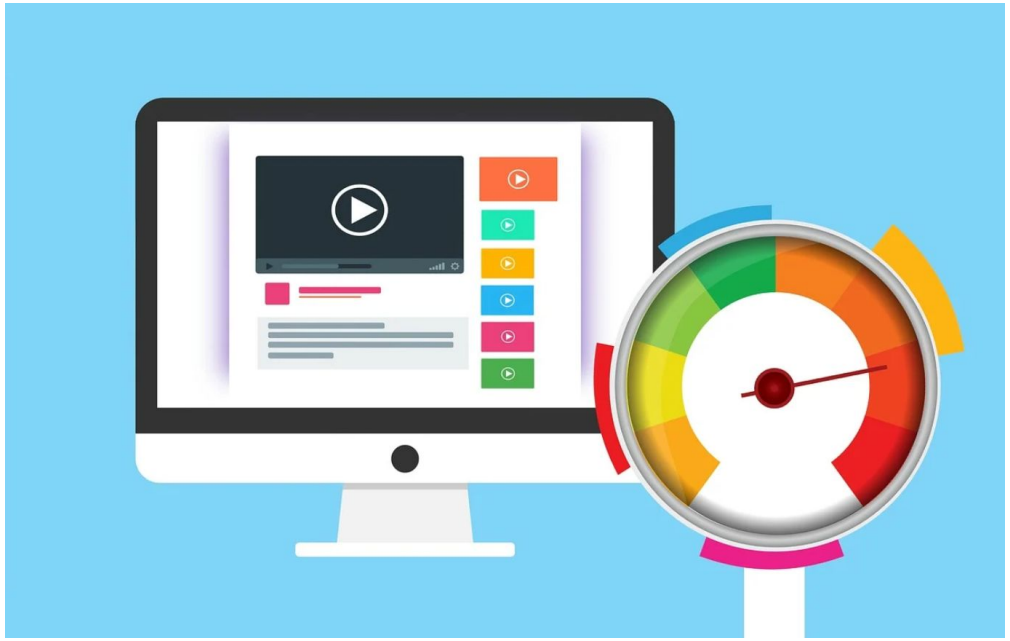
시나리오

- 대규모 트래픽을 감당할 수 있는지?
- 아키텍처가 적합한지?
- 어느 정도의 리소스가 필요한지?

시나리오

- RPS (Request Per Second) 100,000
- Latency <= 150ms
- 최소한의 리소스 비용

고려사항



[대규모 부하 테스트]

충분한 인프라 자원 확보

- 대규모 트래픽 생성 필요

유연한 인프라 자원 활용

- 쉬운 인프라 증설
- 미 사용 시 비용 발생 절감

쉽고 단순한 구성 및 관리

- 쉬운 관리 및 환경 구성
- 확장 용이성
- 테스트 환경 초기화 용이

인프라 선정

충분한
인프라
자원 확보

유연한
인프라
자원 활용

Google Kubernetes Engine

간단하게 Kubernetes를 자동으로 배포, 확장, 관리할 수 있습니다.

- 파드 및 클러스터 자동 확장
- 컨테이너 기반 네트워킹 및 보안
- 로깅 및 모니터링 통합
- 노드 15,000개까지 확장 가능

Locust

쉽고
단순한
구성 및
관리

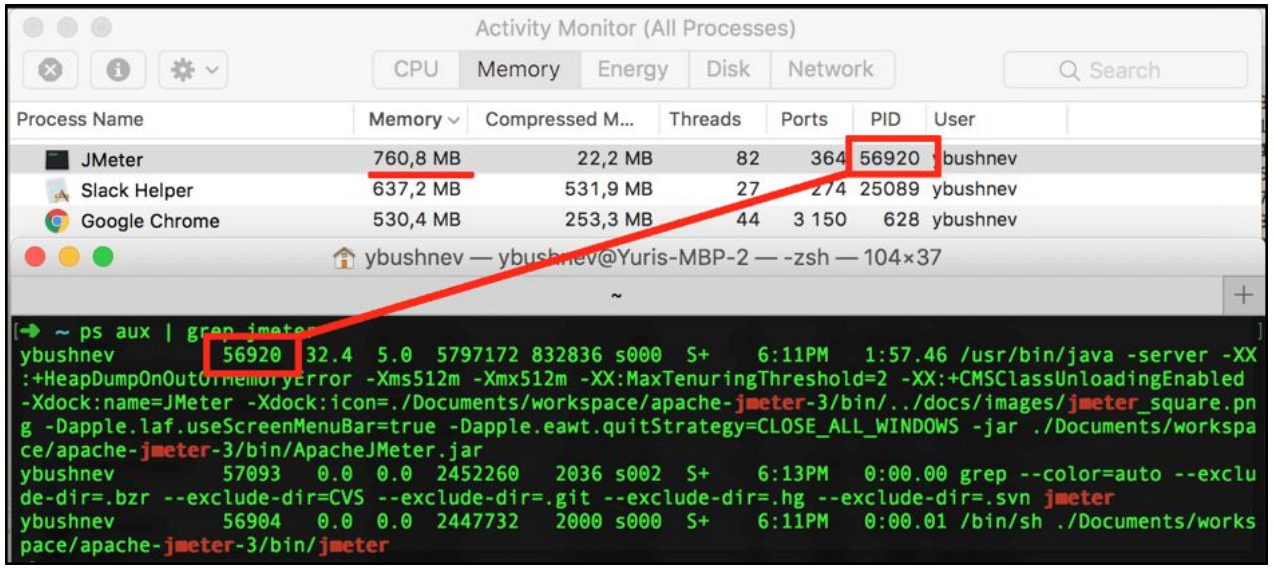
Locust는 사용하기 쉽고 스크립트 가능하며 확장 가능한 성능 테스트 도구입니다.

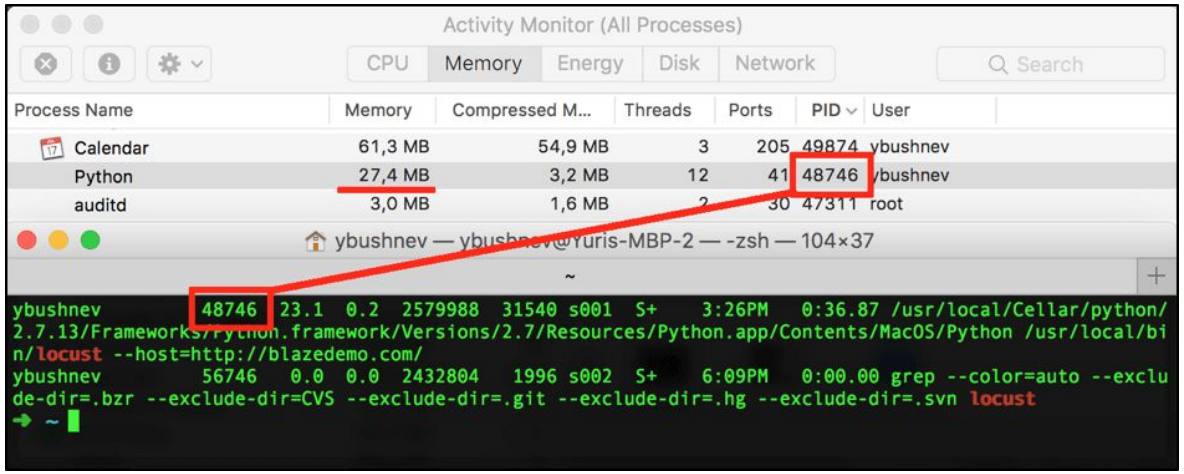
- Python으로 테스트 시나리오 작성 가능 - Test as code 지원
- 분산 및 확장 가능 - 수십만 명의 동시 사용자 시뮬레이션 지원
- 웹기반 UI 제공

Tool 선정

주요 선정 요소 : **동시 사용자 시뮬레이션 수, 적은 리소스 사용, 쉬운 확장성**

Jmeter VS Locust



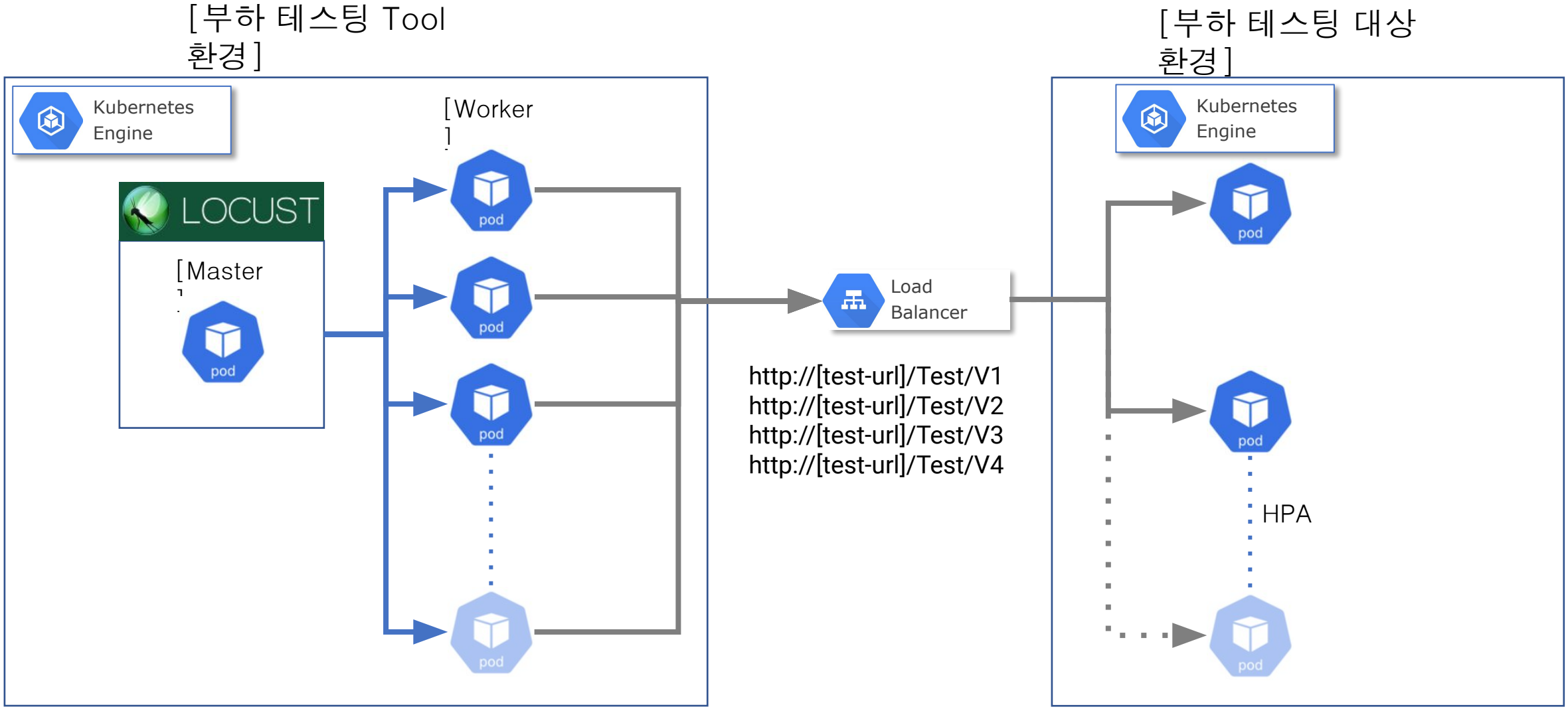


Jmeter가 Locust대비 **30배정도 메모리를 더 사용**하는 것으로 확인 되었음.

출처 : https://www.blazemeter.com/blog/jmeter-vs-locust

Locust 활용한 부하테스트

테스트 구성도



BESPIN GLOBAL Google Cloud

테스트 환경 구성 Hands-on

Step 1.
Locust용 Google Kubernetes Engine 구성하기

Hands-on

1. GKE 생성 옵션 작성

생성 옵션

Cluster basics
- Name : locust-load-tester
- Location Type : Regional
- Region : asia-northeast3

Node Pools
- Default-pool
  - Size : 2
  - Nodes
    - Series : E2
    - Machine Type : e2-standard-8

Node 계산법

최적환경에서 1CPU worker당 최대 5,000 Request지원
100,000/5,000 = 20 1cpu worker 필요.

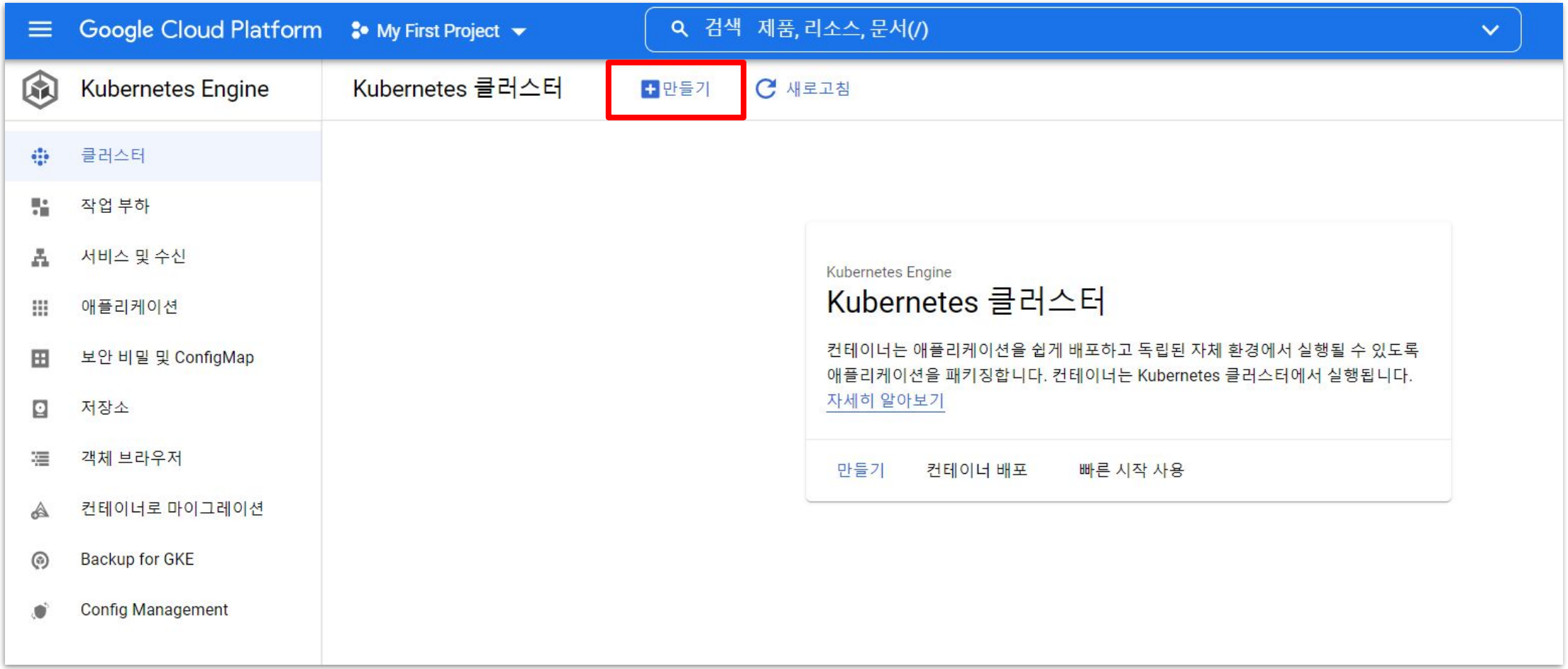
Locust Master CPU : 4
Locust Worker CPU : 80[Worker 수] x 0.5[Work별 cpu] = 40
Total : 필요 vCPU = 44개

8 [Machine Type] x 3[Zone] x 2[Node Size] = 48 vCPU

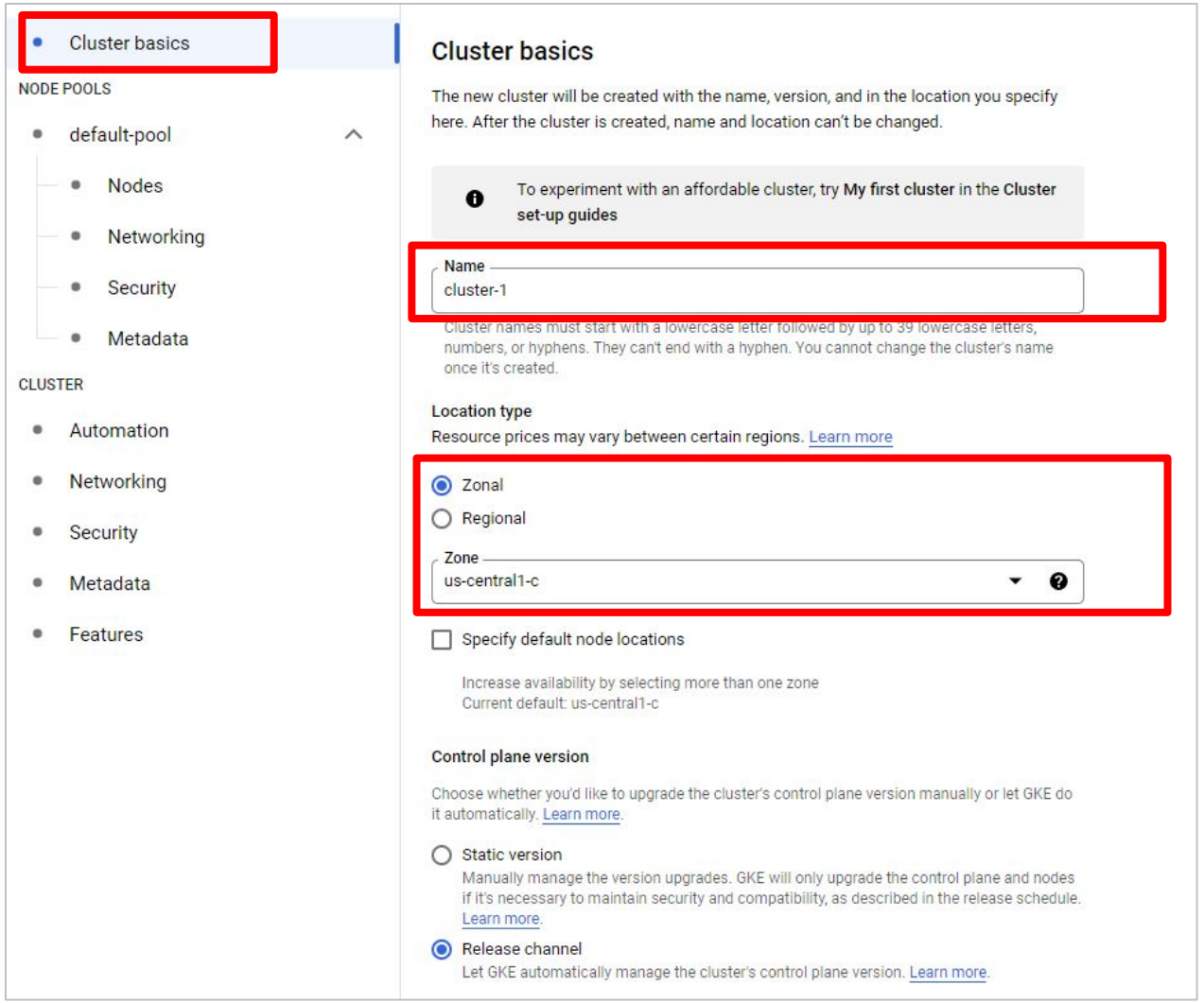
※ Node Size 조절을 통해 Locust의 필요 vCPU 개수보다 많게 생성 필요(또는 Worker별 cpu를 yaml 파일에서 조정 가능)

※ 테스트 결과 Worker 생성시 CPU를 많이 할당하는 것 보다, Worker 수를 늘리는 것이 부해 생성에 효율적

2. GKE 생성

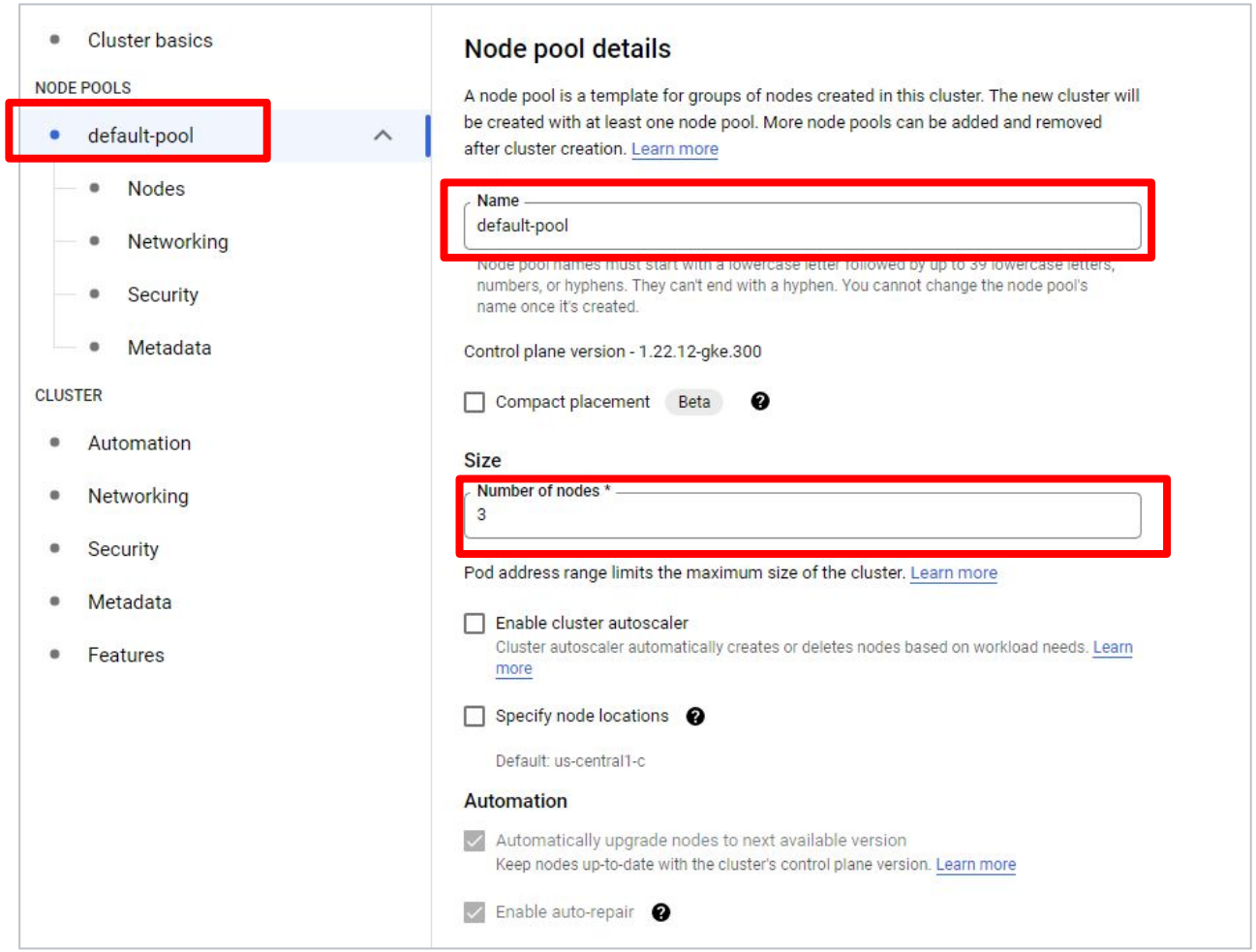

Google Cloud Platform
My First Project
검색 제품, 리소스, 문서(/)
Kubernetes Engine
Kubernetes 클러스터
만들기
새로고침
클러스터
작업 부하
서비스 및 수신
애플리케이션
보안 비밀 및 ConfigMap
저장소
객체 브라우저
컨테이너로 마이그레이션
Backup for GKE
Config Management
Kubernetes Engine
Kubernetes 클러스터
컨테이너는 애플리케이션을 쉽게 배포하고 독립된 자체 환경에서 실행될 수 있도록 애플리케이션을 패키징합니다. 컨테이너는 Kubernetes 클러스터에서 실행됩니다.
자세히 알아보기
만들기
컨테이너 배포
빠른 시작 사용

2. GKE 생성(계속)



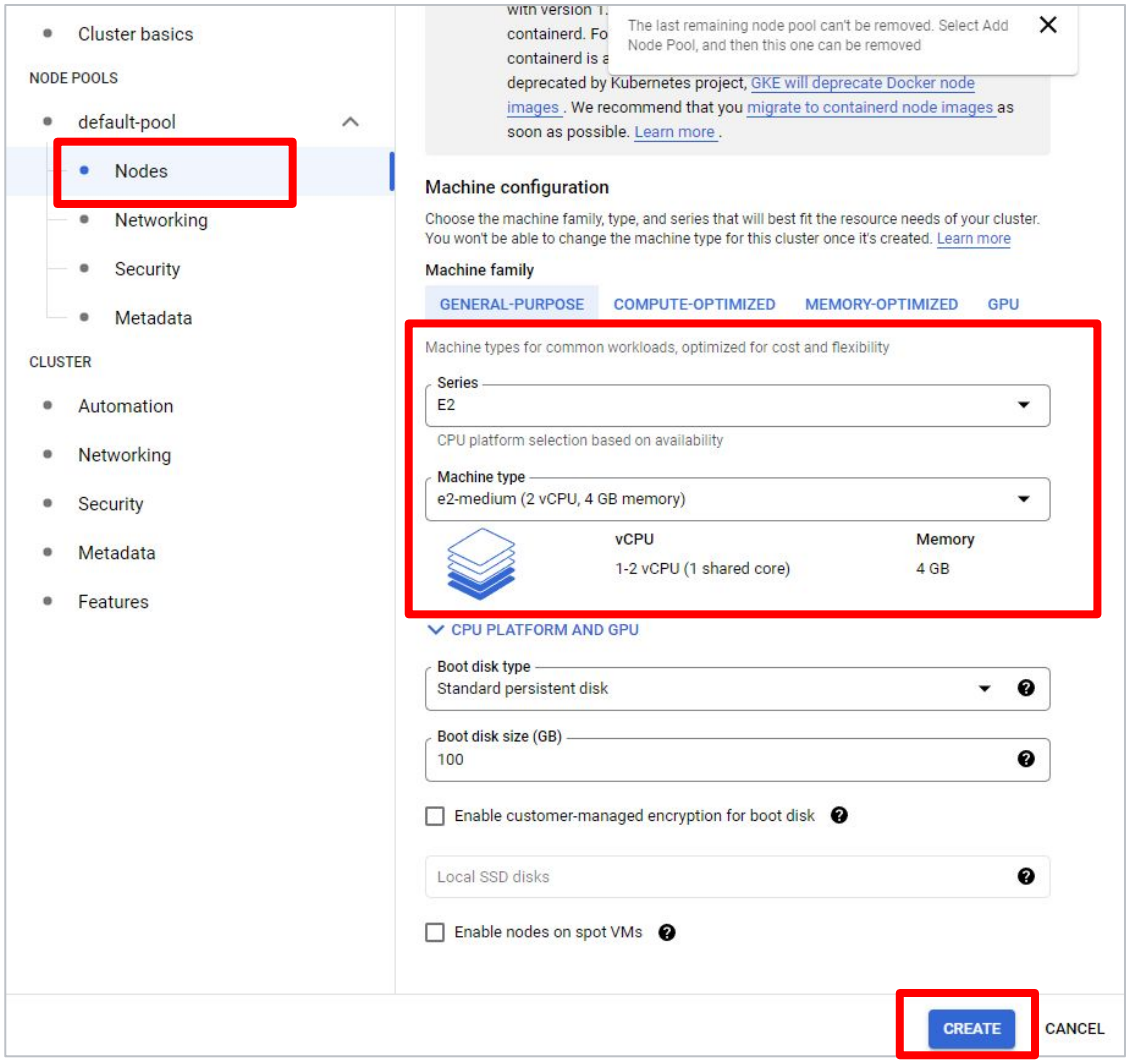
Cluster basics
- Name : locust-load-tester
- Location Type : Regional
- Region : asia-northeast3

2. GKE 생성(계속)



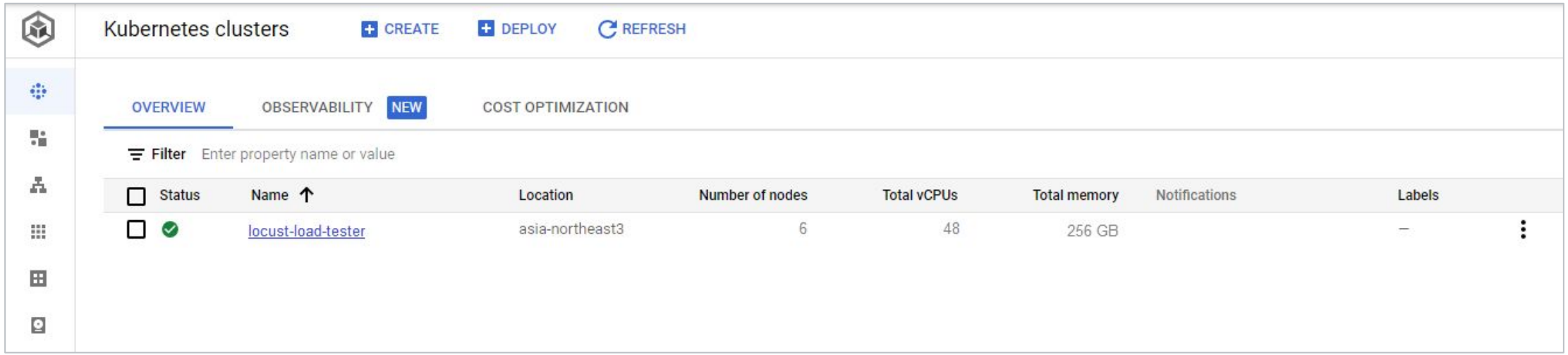
Node Pools
- Name : default-pool
  - Size : 2
  - Nodes
    - Series : E2
    - Machine Type : e2-standard-8
  - Security
    - Access Scopes
      - Allow full access to all Cloud APIs

2. GKE 생성(계속)



Node Pools
- Name : Default-pool
  - Size : 2
  - Nodes
    - Series : E2
    - Machine Type : e2-standard-8

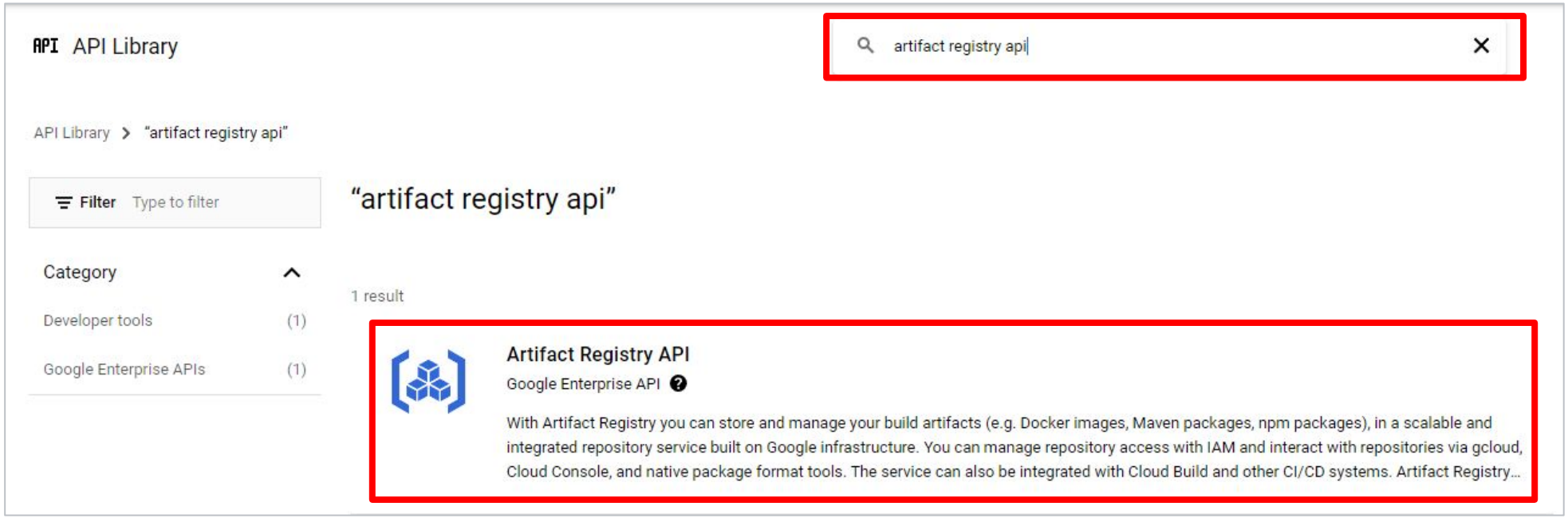
2. GKE 생성 완료

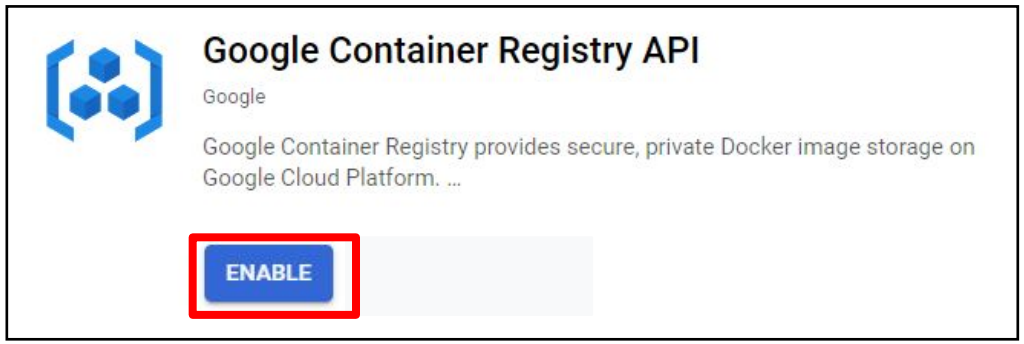

Kubernetes clusters
CREATE
DEPLOY
REFRESH
OVERVIEW
OBSERVABILITY
NEW
COST OPTIMIZATION
Filter
Enter property name or value
Status
Name
Location
Number of nodes
Total vCPUs
Total memory
Notifications
Labels
locust-load-tester
asia-northeast3
6
48
256 GB
–

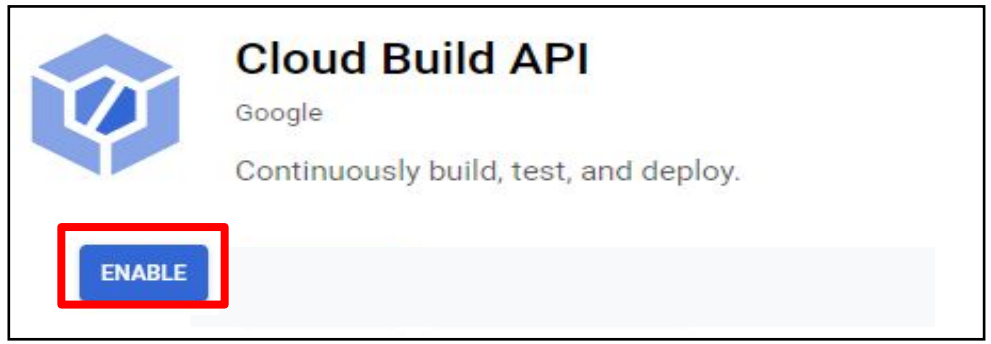
Step 2.
Locust 이미지 빌드 및 배포

Hands-on

1. 관련 API Enable





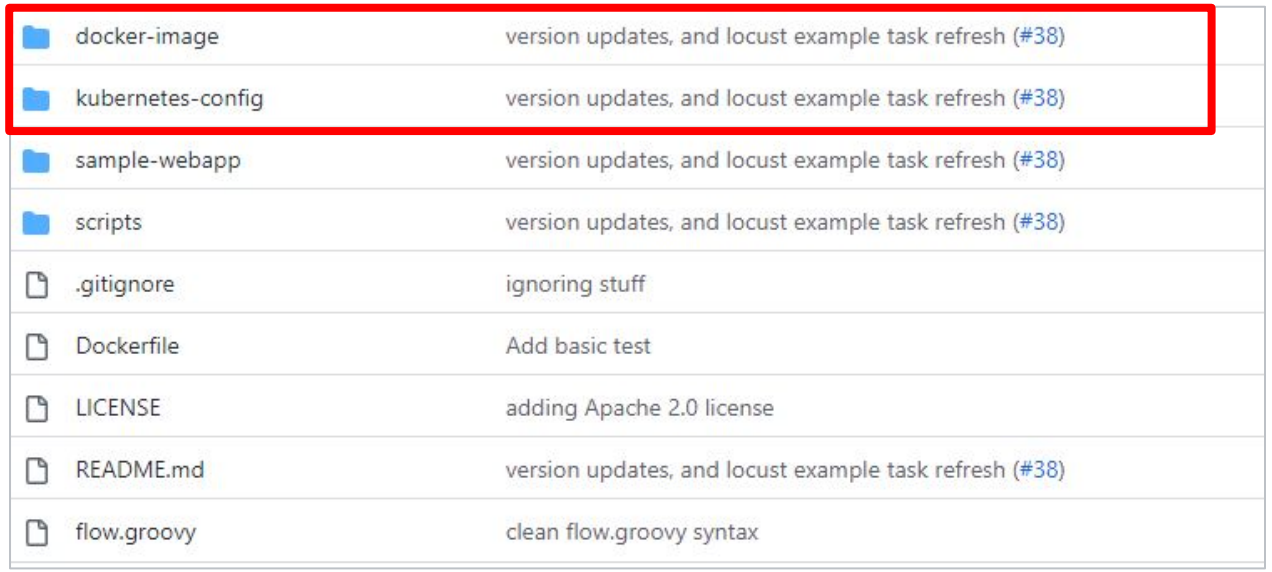


Hands-on

2. Master/Worker Image 관련 소스 다운로드

```
$ git clone https://github.com/GoogleCloudPlatform/distributed-load-testing-using-kubernetes
remote: Enumerating objects: 232, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 232 (delta 4), reused 15 (delta 3), pack-reused 214
Receiving objects: 100% (232/232), 52.70 KiB | 1.50 MiB/s, done.
Resolving deltas: 100% (109/109), done.
```

[소스 구조]



docker-image : locust 이미지

kubenetes-config : locust 배포용 yaml

Hands-on

3. Tasks.py 수정

```
$ cd distributed-load-testing-using-Kubernetes
$ cd docker-image
$ cd locust-tasks
$ vi tasks.py
```

```
import uuid

from datetime import datetime
from locust import FastHttpUser, TaskSet, task


# [START locust_test_task]

class MetricsTaskSet(TaskSet):
    _deviceid = None

    def on_start(self):
        self._deviceid = str(uuid.uuid4())

    @task(1)
    def login(self):
        self.client.post(
            '/login', {"deviceid": self._deviceid})

    @task(999)
    def post_metrics(self):
        self.client.post(
            "/metrics", {"deviceid": self._deviceid, "timestamp": datetime.now()})


class MetricsLocust(FastHttpUser):
    tasks = {MetricsTaskSet}
```

```
from locust import FastHttpUser, TaskSet, task

class MetricsTaskSet(TaskSet):
    @task(60)
    def callV1(self):
        self.client.get("/Test/V1")

    @task(1)
    def callV2(self):
        self.client.get("/Test/V2")

    @task(14)
    def callV3(self):
        self.client.get("/Test/V3")

    @task(5)
    def callV4(self):
        self.client.get("/Test/V4")

class MetricsLocust(FastHttpUser):
    tasks = {MetricsTaskSet}
```

Hands-on

3. Tasks.py 수정(소스 설명)

```python
from locust import FastHttpUser, TaskSet, task

class MetricsTaskSet(TaskSet):
    @task(60)
    def callV1(self):
        self.client.get('/Test/V1')

    @task(1)
    def callV2(self):
        self.client.get('/Test/V2')

    @task(14)
    def callV3(self):
        self.client.get('/Test/V3')

    @task(5)
    def callV4(self):
        self.client.get('/Test/V4')

class MetricsLocust(FastHttpUser):
    tasks = {MetricsTaskSet}
```

필요 라이브러리 Import

TaskSet 선언

Task 정의
60 : 1: 14 : 5의 비율로 /Test/V1~V4 까지의
API URL을 Get 하도록 설정

Taskset을 수행할 User Class 선정 (FastHttpUser 사용)

Hands-on

※ Locustfile 작성 Info

1. User Class

한명의 User로서 Task를 수행.
weight 속성을 통해 특정 User Class가 더 많은 작업을 수행하도록 설정 가능

2. HttpUser VS FastHttpUsers Class

FastHttpUsers의 경우 Core당 최대 5000 Request에 가까운 부하를 발생 시킬 수 있는 반면,
HttpUser는 Core당 850 Request를 발생 시킬 수 있는 것으로 테스트 되었음.
Tasks.py 작성 시 FastHttpUsers를 사용할 것을 권고 함

3. Task

부하 테스트 시 User가 수행하는 업무를 정의. User Class가 정의된 Task를 실행.

4. Locust Image 빌드

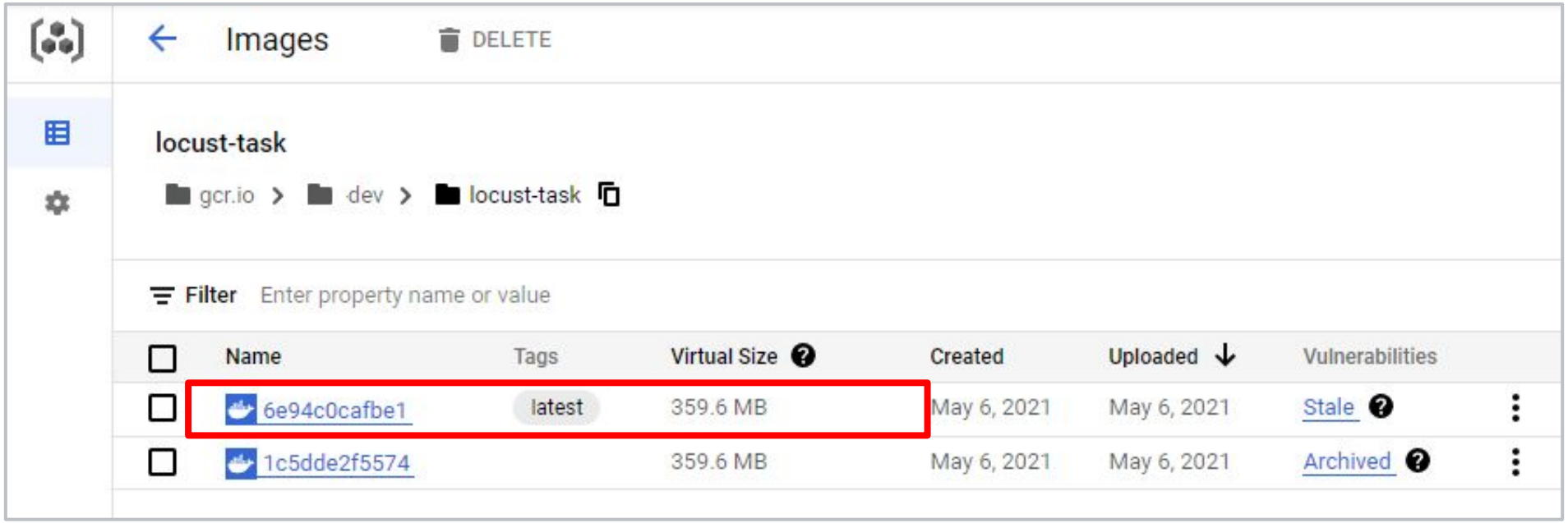
```
$ cd ../../
$ gcloud builds submit    --tag gcr.io /[PROJECT_NAME] /[IMAGE_NAME] :latest [FOLDER_NAME]
```

[PROJECT_NAME] : GCR 프로젝트 명
ex) bespin-xxxxx

[IMAGE_NAME] : 빌드 이미지 명
ex) locust-task

[FOLDER_NAME] : 빌드 폴더
ex) locust_image

정상 수행 시 GCR에 아래와 같이 이미지가 생성 됨



5. Locust 배포

① GKE 접속

```
gcloud container clusters get-credentials locust-load-tester --region asia-northeast3 --project bespin-54321
```

```
kubectl get namespace
```

② 아래 순서대로 배포 진행

<locust-master.yaml>

Hands-on

5. Locust 배포 – Master 생성

```
$ kubectl –f locust-master.yaml
```

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: locust-master
  labels:
    name: locust-master
spec:
  replicas: 1
  selector:
    matchLabels:
      app: locust-master
  template:
    metadata:
      labels:
        app: locust-master
    spec:
      containers:
        - name: locust-master
          image: gcr.io/bespin-xxxxx/locust_image:latest
          env:
            - name: LOCUST_MODE
              value: master
          ports:
            - name: loc-master-web
              containerPort: 8089
              protocol: TCP
            - name: loc-master-p1
              containerPort: 5557
              protocol: TCP
            - name: loc-master-p2
              containerPort: 5558
              protocol: TCP
```

<locust-master.yaml>

Master 배포 확인

```
$ kubectl get pods –o wide
```

```
g9562703@cloudshell:~/distributed-load-testing-using-kubernetes (clean-facility-
NAME                                READY   STATUS    RESTARTS   AGE   IP
locust-master-7fd58b6ccf-sgbbr      1/1     Running   0          78s   10.96.2.6
```

BESPIN GLOBAL Google Cloud

Hands-on

5. Locust 배포 – Service 생성

```
$ kubectl –f locust-master-service.yaml
```

```
kind: Service
apiVersion: v1
metadata:
 name: locust-master
 labels:
   app: locust-master
spec:
 ports:
   - port: 5557
     targetPort: loc-master-p1
     protocol: TCP
     name: loc-master-p1
   - port: 5558
     targetPort: loc-master-p2
     protocol: TCP
     name: loc-master-p2
 selector:
   app: locust-master
---
kind: Service
apiVersion: v1
metadata:
 name: locust-master-web
 annotations:
   networking.gke.io/load-balancer-type: "Internal"
 labels:
   app: locust-master
spec:
 ports:
   - port: 8089
     targetPort: loc-master-web
     protocol: TCP
     name: loc-master-web
 selector:
   app: locust-master
 type: LoadBalancer
```

<locust-master-service.yaml>

Service 배포 확인

```
$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.87.240.1	<none>	443/TCP	12m
locust-master	ClusterIP	10.87.245.22	<none>	5557/TCP,5558/TCP	1m
locust-master-web	LoadBalancer	10.87.246.225	<pending>	8089:31454/TCP	1m

Hands-on

5. Locust 배포 – Worker 생성

```
$ kubectl –f locust-worker.yaml
```

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: locust-worker
  labels:
    name: locust-worker
spec:
  replicas: 5
  selector:
    matchLabels:
      app: locust-worker
  template:
    metadata:
      labels:
        app: locust-worker
    spec:
      containers:
        - name: locust-worker
          image: gcr.io/bespin-xxxxx/locust_image:latest
          env:
            - name: LOCUST_MODE
              value: worker
            - name: LOCUST_MASTER
              value: locust-master
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 200m
```

<locust-worker.yaml>

Worker 배포 확인

```
$ kubectl get pods –o wide
```

```
g9562703@cloudshell:~/distributed-load-testing-using-kubernetes (clean-facility-
NAME                                READY   STATUS    RESTARTS   AGE   IP
locust-master-7fd58b6ccf-sgbbr      1/1     Running   0          78s   10.96.2.6
locust-worker-6dbd8684b6-6mflw      1/1     Running   0          38s   10.96.2.7
locust-worker-6dbd8684b6-cswtq      1/1     Running   0          38s   10.96.0.7
locust-worker-6dbd8684b6-k8dbb      1/1     Running   0          38s   10.96.1.3
locust-worker-6dbd8684b6-kb2sc      1/1     Running   0          38s   10.96.2.8
locust-worker-6dbd8684b6-qjn6q      1/1     Running   0          38s   10.96.1.4
```

5. Locust 배포 – Worker 동작 확인

```
$ kubectl logs [POD NAME]
```

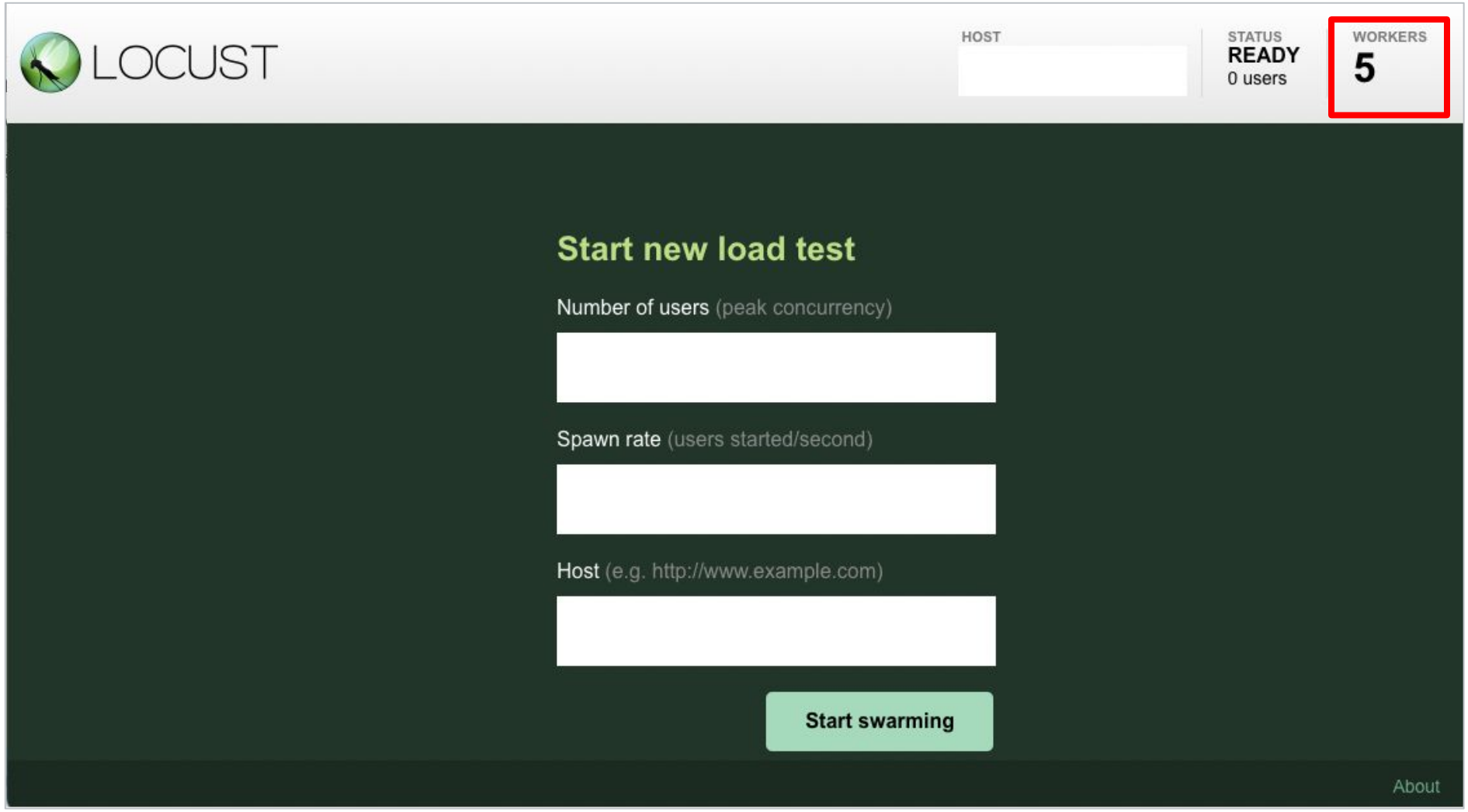
Case 1. 정상 수행되었을 경우

```
[2020-08-31 04:08:43,610] locust-worker-778ff5d7cd-jpw6k/INFO/locust.main: Starting Locust 1.1.1
```

Case 2. 비정상 수행되었을 경우

tasks.py 파일의 오류를 출력함.
해당 오류 수정 후 Worker 이미지 빌드 재실행

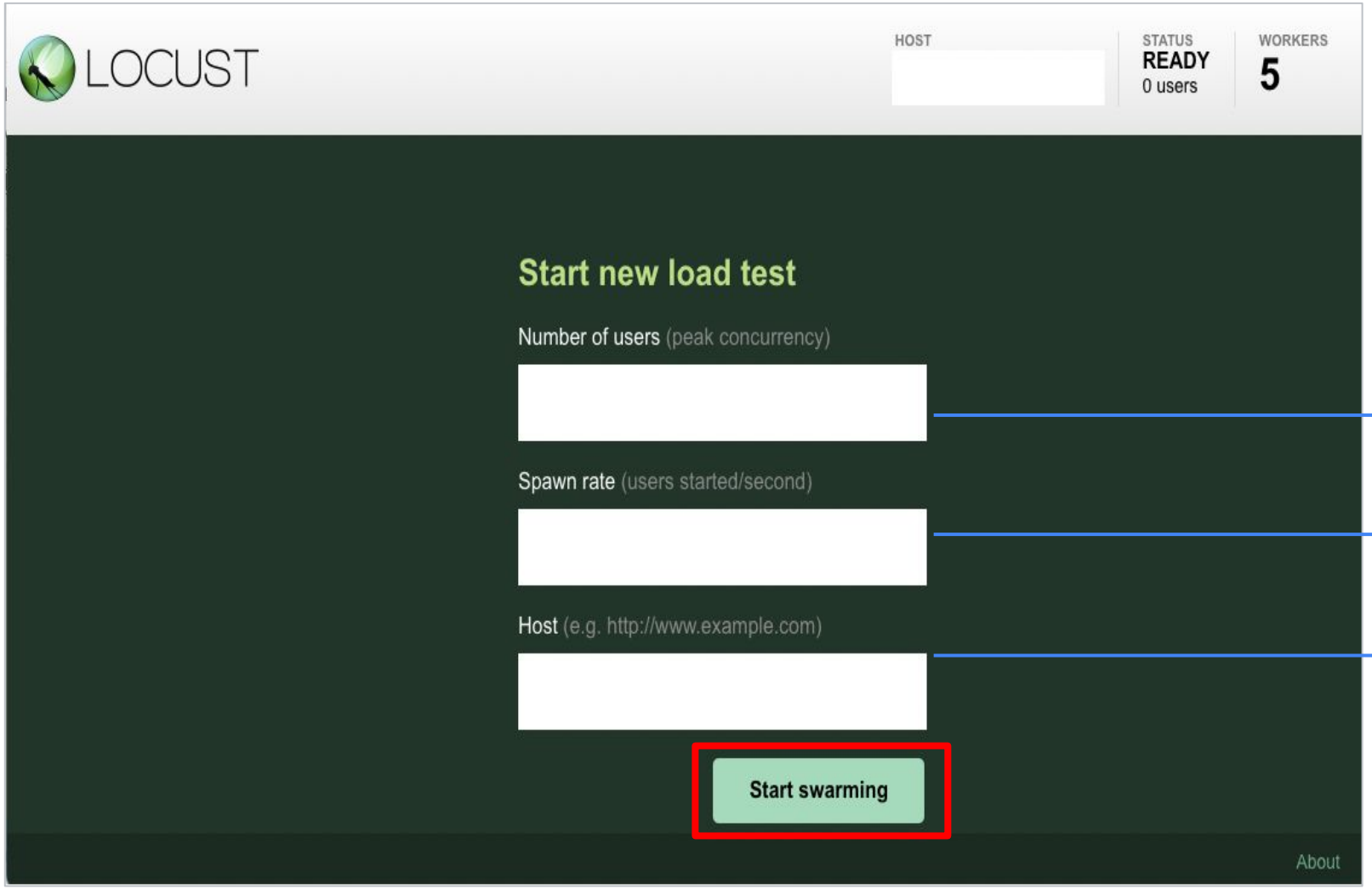
6. Locust 배포 확인 – 실행 화면



Step 4.
부하테스트 수행하기

Hands-on

1. 부하테스트 수행

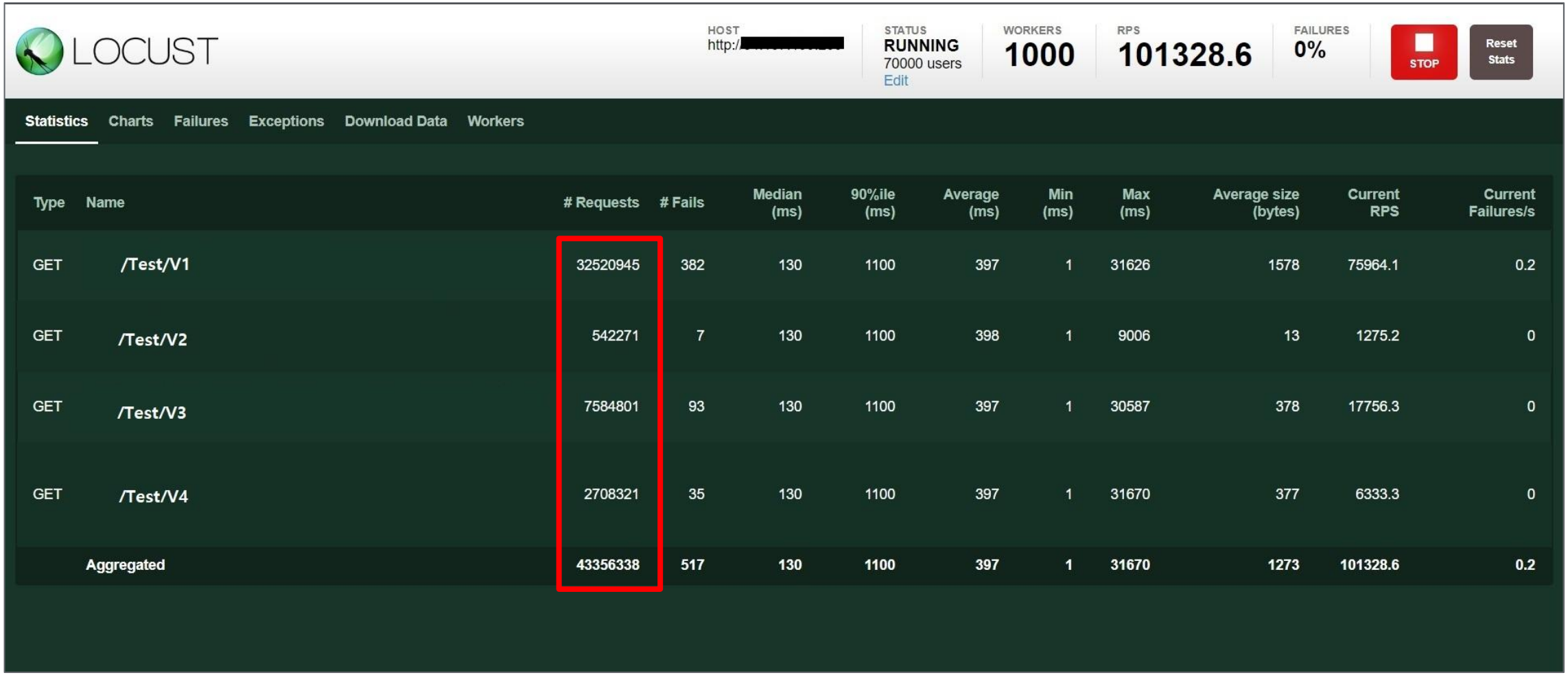


Number of users : 총 사용자 수

Spawn rate : 사용자 증가 속도 / 초

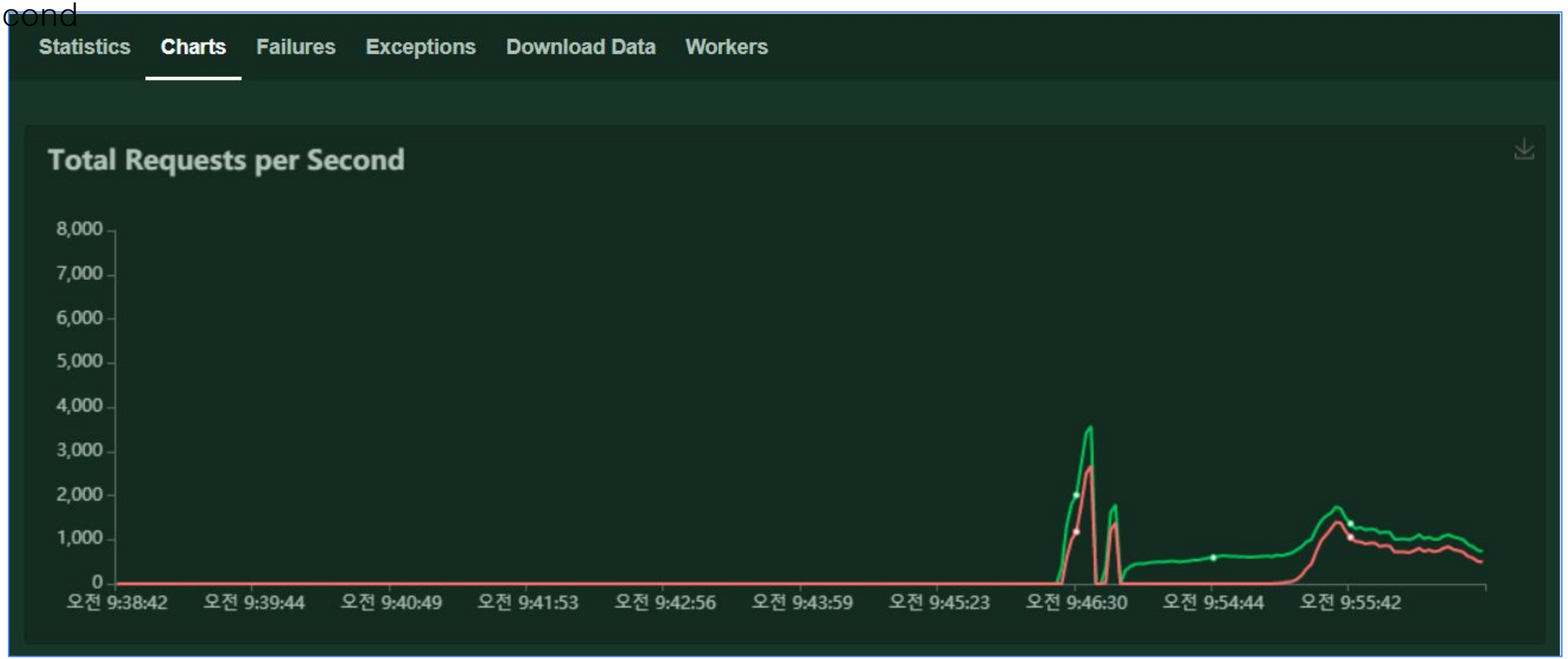
Host : 테스트 대상 URL

2. 부하테스트 수행 화면 – Statistics

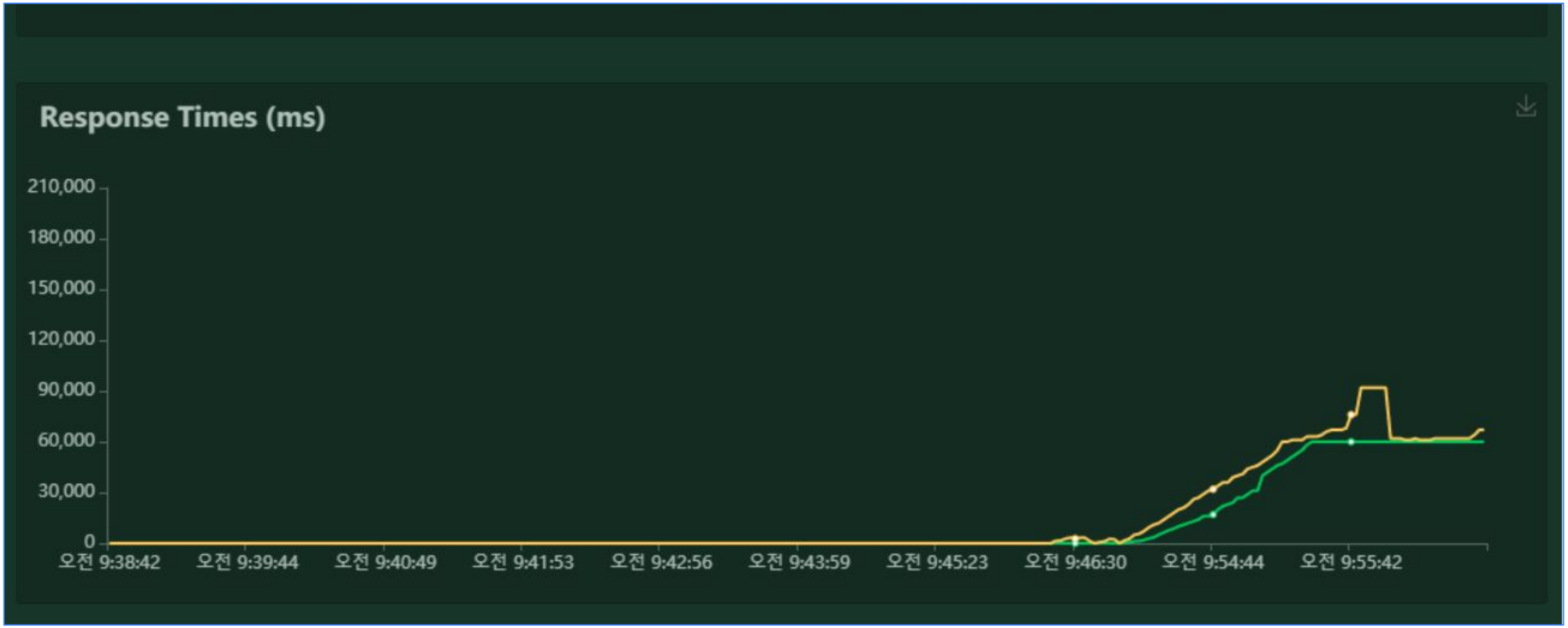


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/Test/V1	32520945	382	130	1100	397	1	31626	1578	75964.1	0.2
GET	/Test/V2	542271	7	130	1100	398	1	9006	13	1275.2	0
GET	/Test/V3	7584801	93	130	1100	397	1	30587	378	17756.3	0
GET	/Test/V4	2708321	35	130	1100	397	1	31670	377	6333.3	0
	Aggregated	**43356338**	**517**	**130**	**1100**	**397**	**1**	**31670**	**1273**	**101328.6**	**0.2**

2. 부하테스트 수행 화면 – Chart ①Total Requests per Second



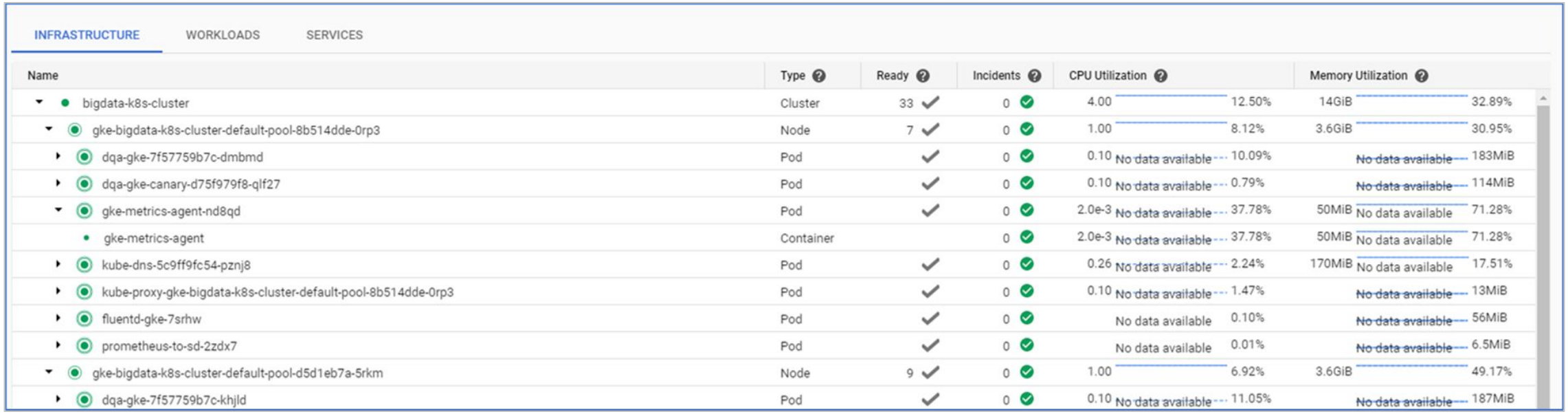
2. 부하테스트 수행 화면 – Chart ②Response Time



2. 부하테스트 수행 화면 – Chart ③Number of Users

3. 부하테스트 대상 리소스 모니터링

Monitoring > Kubernetes Engine : CPU / Memory / Network 에 대한 모니터링

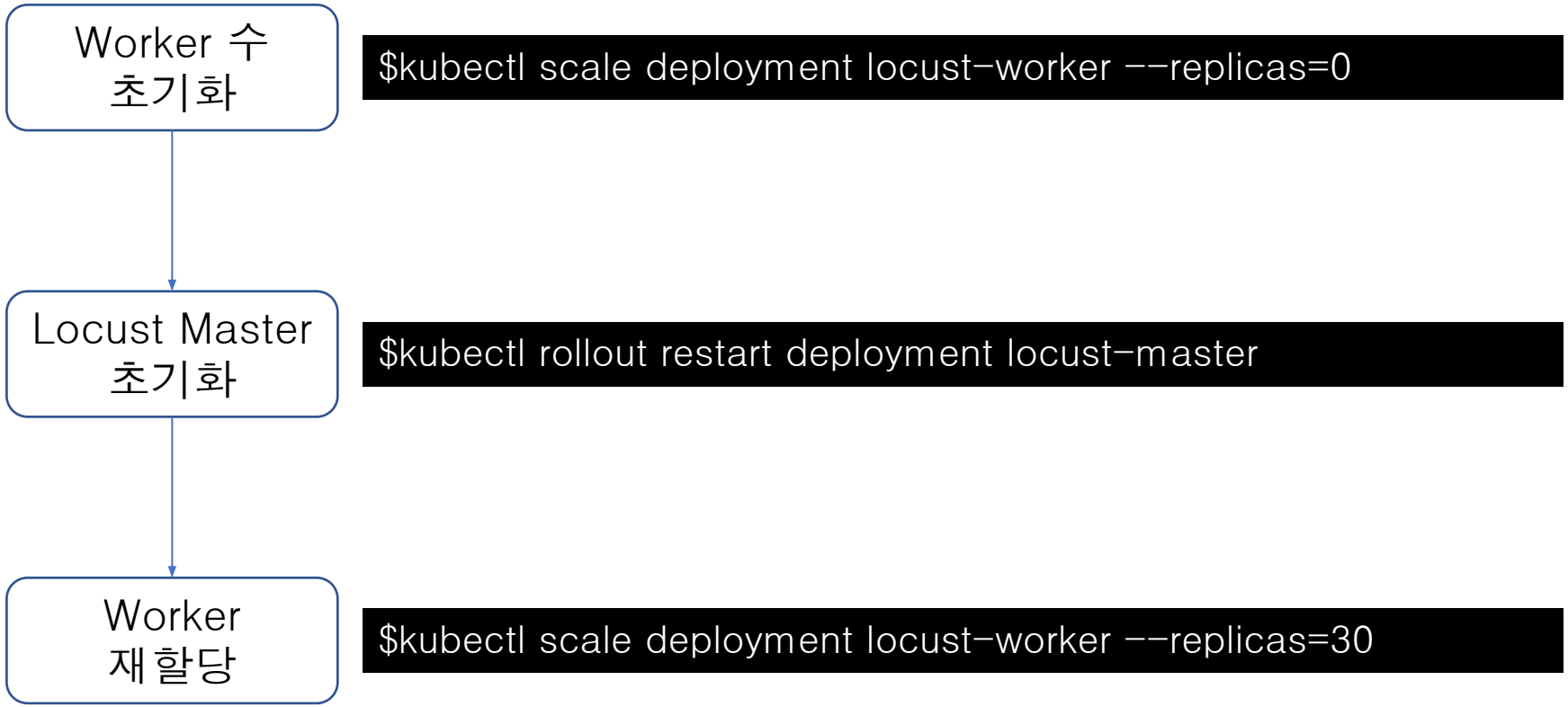


Name	Type	Ready	Incidents	CPU Utilization	Memory Utilization
bigdata-k8s-cluster	Cluster	33	0	4.00 12.50%	14GiB 32.89%
gke-bigdata-k8s-cluster-default-pool-8b514dde-0rp3	Node	7	0	1.00 8.12%	3.6GiB 30.95%
dqa-gke-7f57759b7c-dmbmd	Pod		0	0.10 No data available 10.09%	No data available 183MiB
dqa-gke-canary-d75f979f8-qlf27	Pod		0	0.10 No data available 0.79%	No data available 114MiB
gke-metrics-agent-nd8qd	Pod		0	2.0e-3 No data available 37.78%	50MiB No data available 71.28%
gke-metrics-agent	Container		0	2.0e-3 No data available 37.78%	50MiB No data available 71.28%
kube-dns-5c9ff9fc54-pznj8	Pod		0	0.26 No data available 2.24%	170MiB No data available 17.51%
kube-proxy-gke-bigdata-k8s-cluster-default-pool-8b514dde-0rp3	Pod		0	0.10 No data available 1.47%	No data available 13MiB
fluentd-gke-7srhw	Pod		0	No data available 0.10%	No data available 56MiB
prometheus-to-sd-2zdx7	Pod		0	No data available 0.01%	No data available 6.5MiB
gke-bigdata-k8s-cluster-default-pool-d5d1eb7a-5rkm	Node	9	0	1.00 6.92%	3.6GiB 49.17%
dqa-gke-7f57759b7c-khjld	Pod		0	0.10 No data available 11.05%	No data available 187MiB

설정방법 – 해당 GKE의 서비스 계정에 아래 Role 추가

Logs Writer
Monitoring Metric Writer
Stackdriver Resource Metadata Writer

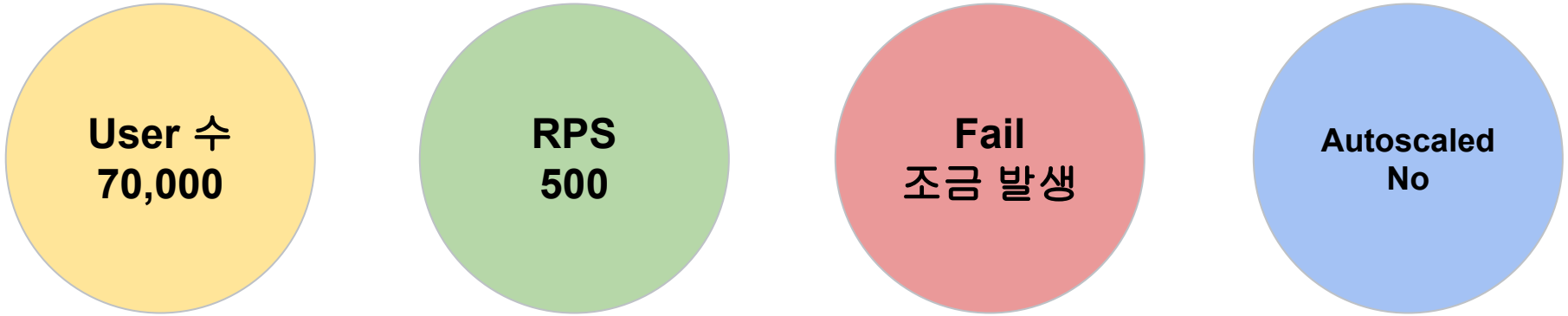
4. 부하테스트 반복 수행을 위한 Locust 초기화 Flow



부하테스트를 통한
GKE Autoscale 최적화 방법

BESPIN GLOBAL Google Cloud

Case 1. RPS가 목표치에 도달하지 않을 경우

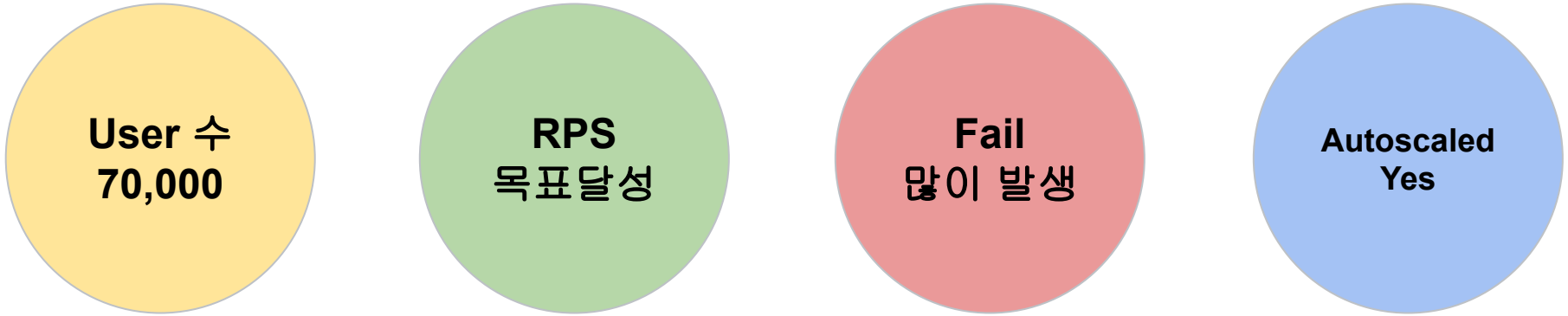


Test User수가 충분히 큼에도 불구하고 Autoscale 없이 RPS가 목표치에 도달하지 않는다면, 부하를 일으키는 Worker의 수가 부족한 경우입니다.

아래 명령어를 이용하여 Worker 수를 늘려주세요

```
$kubectl scale deployment locust-worker --replicas=[Worker 수]
```

Case 2. RPS목표를 달성했지만 Fail이 많이 발생하는 경우

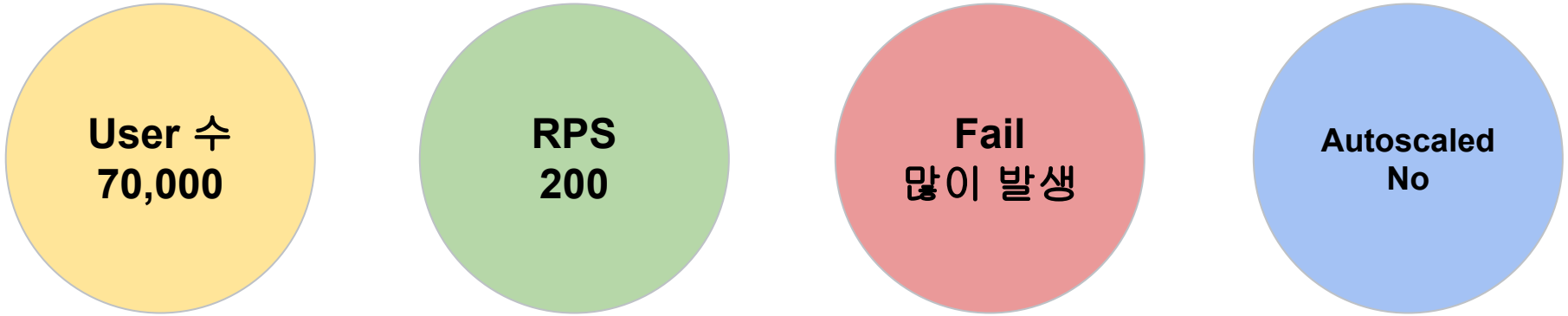


Autoscale이 일어나고, RPS목표는 달성하였으나 그전에 Fail이 많이 발생했다면,
좀더 빠른 타이밍에 Autoscale이 일어날 수 있도록 임계값을 설정해 줍니다.

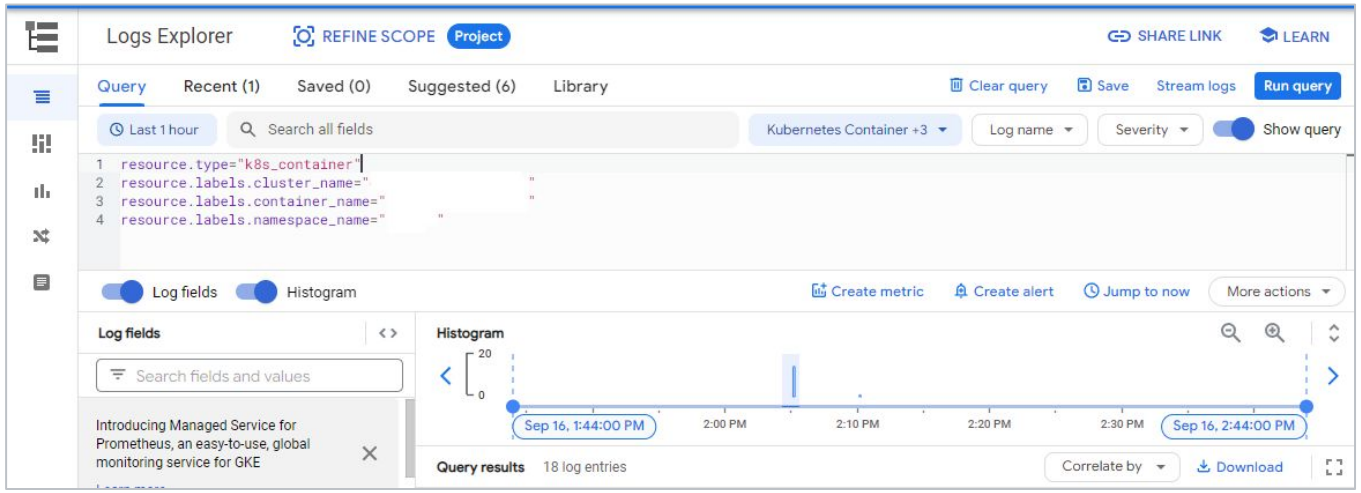
<data-collect-hpa.yaml>

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: data-collector
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
   name: data-collector
  minReplicas: 10
  maxReplicas: 100
  targetCPUUtilizationPercentage: 50
```

Case 3. 서비스 GKE자원에 문제가 없음에도 Fail이 많이 발생할 경우



Application 문제로 부하를 받고 있지 못할 수 있으므로, Container Log에 오류메시지가 없는지 확인해 봅니다.

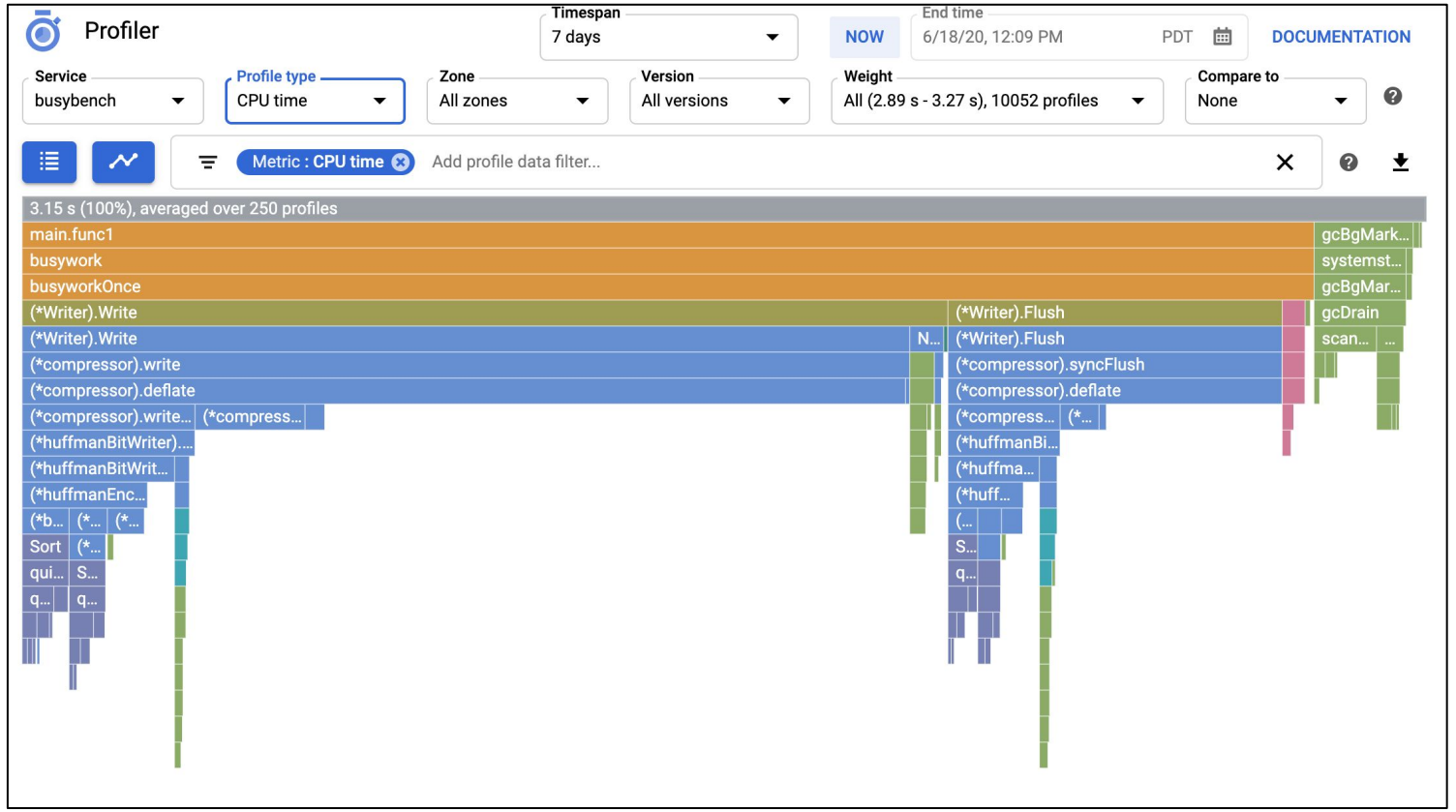


BESPIN GLOBAL Google Cloud

Google Cloud Profiler

Cloud Profiler는 오버헤드가 낮은 통계 프로파일러로, 프로덕션 애플리케이션에서 CPU 사용량과 메모리 할당 정보를 지속적으로 수집합니다. 해당 정보의 출처를 애플리케이션의 소스 코드라고 밝히고, 애플리케이션에서 가장 많은 리소스를 소비하고 있는 부분을 식별하고, 코드의 성능 특성을 알려줍니다.

Case 4. Application에서 병목현상이 의심되어 병목 지점을 확인하고 싶을 경우(계속)



설정방법

1. Application이 올라가는 GKE의 서비스 계정에 Role 추가 : Stackdriver Profiler Agent
2. Application 빌드 시 사용하는 Docker 파일에 아래 명령어 추가

```
RUN mkdir -p /opt/cprof && wget -q -O- https://storage.googleapis.com/cloud-profiler/java/latest/profiler_java_agent.tar.gz | tar xzv -C /opt/체갤
ENTRYPOINT
["java","-agentpath:/opt/cprof/profiler_java_agent.so=-cprof_service=nettyserver,-cprof_project_id=[project-name],-cprof_service_version=1.0.0,-logtostderr,-mi
```

Q&A

BESPIN GLOBAL  Google Cloud

끝, 감사합니다:)

BESPIN GLOBAL  Google Cloud