

ECS+Locust로 부하 테스트 진행하기

박윤곤

아이스크림에듀

목차

- 발표자 소개
- Locust로 부하 테스트 작성하기
- Docker 이미지 만들고 ECR에 올리기
- ECS에서 테스트 하기
 - 하나의 컨테이너에서 테스트하기
 - 여러 컨테이너에서 테스트하기
- Q&A

발표자 소개

- 아이스크림에듀 (2018.01~)
 - '아이스크림 홈런' 내 'AI 생활기록부' 서비스 인프라 운영



문제의 시작

- 데이터를 API Gateway(+Firehose)로 보내도록 구성했는데...
 - 어디까지 버틸 수 있는 지 궁금했음
 - 여러 대의 EC2로 부하 테스트: 불편한 점?
 - 그래서 컨테이너 기반으로 테스트 해 보려고 함

Locust로
부하 테스트 작성하기

Locust?

- Python 기반의 부하 테스트 도구
- 설치 방법

```
$ pip install locustio
```

- 상세한 설치 방법은 [링크](#) 참조

Locust 파일 만들기

Example code

A fundamental feature of Locust is that you describe all your test in Python code. No need for clunky UIs or bloated XML, just plain code.

Select example ▶

- Simple
- With HTML parsing
- Nested TaskSets

출처: <https://locust.io>

```
locustfile.py

from locust import HttpLocust, TaskSet, task, between

class WebsiteTasks(TaskSet):
    def on_start(self):
        self.client.post("/login", {
            "username": "test_user",
            "password": ""
        })

    @task
    def index(self):
        self.client.get("/")

    @task
    def about(self):
        self.client.get("/about/")

class WebsiteUser(HttpLocust):
    task_set = WebsiteTasks
    wait_time = between(5, 15)

$ locust -f locustfile.py
```

HttpLocust 클래스

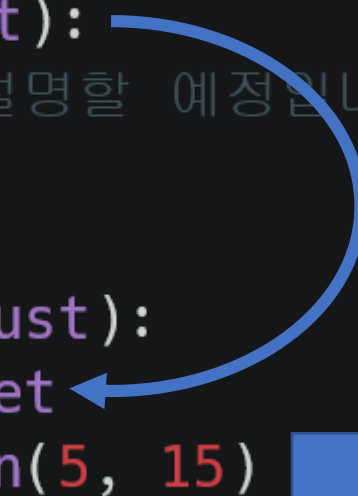
- Locust 클래스: 하나의 사용자를 나타냄
 - task_set 속성: TaskSet 클래스 → 사용자가 실행할 것들을 나타냄
 - wait_time 속성: Task 실행 사이의 간격
 - weight 속성: 여러 Locust 클래스가 존재할 때 가중치
 - host 속성: 테스트 할 사이트의 URL prefix
- HttpLocust 클래스: HTTP를 사용하는 사용자를 나타냄
 - client 속성: HttpSession을 갖고 있음. 쿠키를 지원하며 세션 유지 가능

HttpLocust 클래스 - 예제

```
from locust import HttpLocust, TaskSet, between

class MyTaskSet(TaskSet):
    # 다음 슬라이드에서 설명할 예정입니다.
    pass

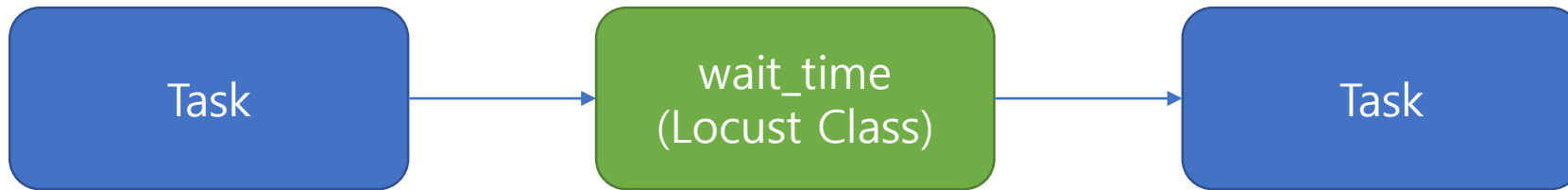
class MyLocust(HttpLocust):
    task_set = MyTaskSet
    wait_time = between(5, 15)
```



between
constant
constant_pacing

TaskSet 클래스

- 하나의 사용자(Locust/HttpLocust 클래스)가 실행할 작업
 - Locust 클래스가 생성되면, task_set 속성에 있는 task를 실행함



- TaskSet 클래스에도 wait_time을 설정할 수 있음

TaskSet 클래스

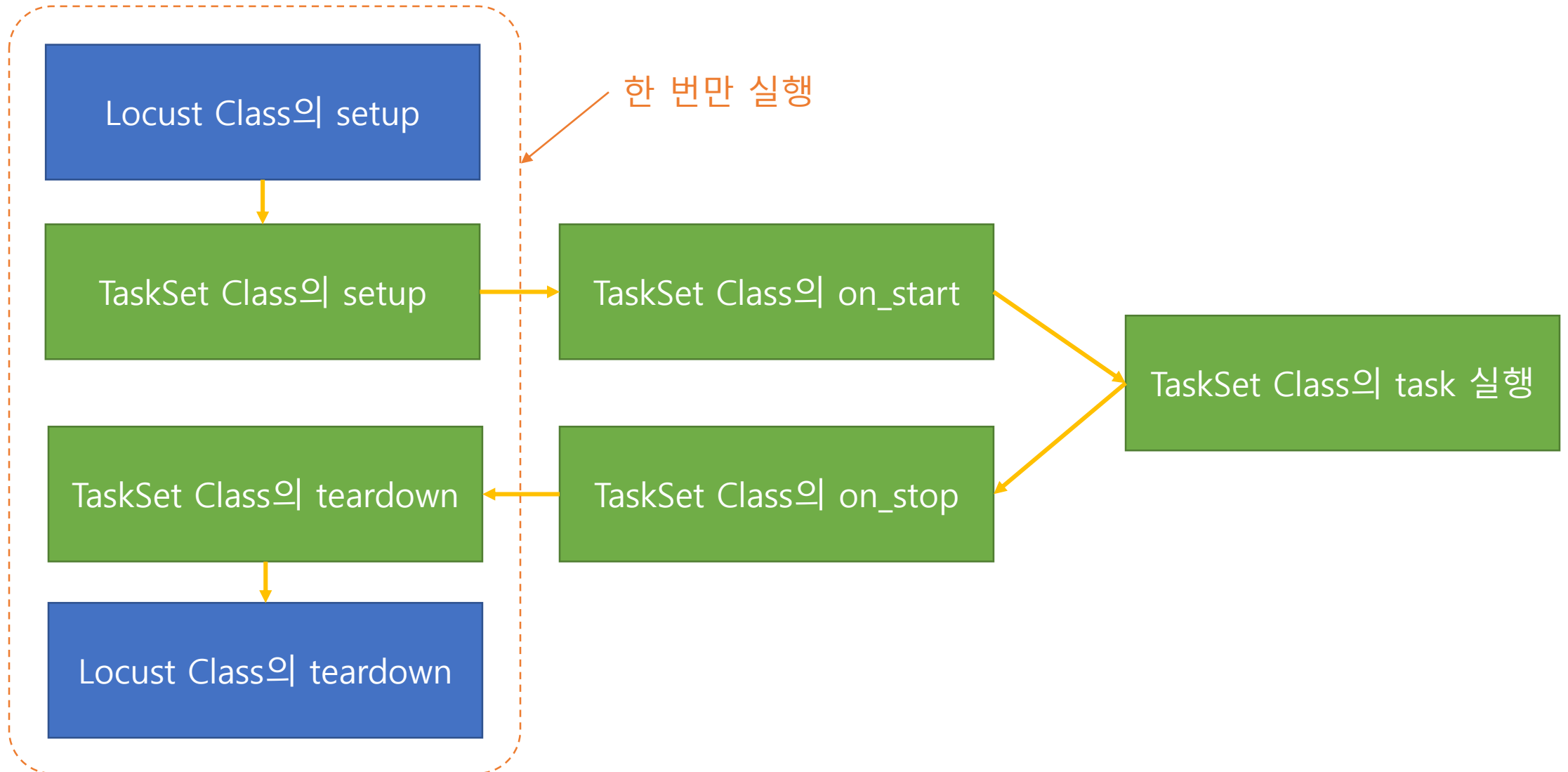
- 기본적으로 @task Decorator를 이용함
 - @task(숫자) 와 같이 사용하면 실행 비율을 조정할 수 있음
 - @task(1), @task(2)이 있으면 @task(2)로 설정된 Task가 2배 더 많이 실행됨

```
class ForumPage(TaskSet):  
    @task(2)  
    def read_thread(self):  
        self.client.get('/about')  
  
    @task(1)  
    def read_home(self):  
        self.client.get('/')
```

이벤트 수행 순서

- setup / teardown method
 - Locust 클래스나 TaskSet 클래스에 존재
 - 모든 작업을 실행하기 전 / 종료 전 한 번 실행
- on_start / on_stop method
 - TaskSet 클래스에 정의할 수 있음
 - 사용자가 TaskSet 클래스를 실행할 때 / TaskSet이 중지될 때 호출됨

이벤트 수행 순서



Docker 이미지 만들고
ECR에 올리기

Docker 이미지 만들기

- 참고자료
 - [Running Locust with Docker](#)
- 내가 만든 스크립트를 Docker 이미지에 추가하기 (Dockerfile)

```
FROM locustio/locust  
  
ADD '스크립트 파일 이름' locustfile.py
```

- Locust 공식 이미지는 /locustfile.py 파일을 찾아 실행함
 - 다른 경로에 있다면 LOCUSTFILE_PATH 환경 변수에 지정

Docker 이미지 만들기

- ECR에 Docker 이미지 올리기 (CLI 이용)

```
aws ecr create-repository --repository-name (저장소 이름)
$(aws ecr get-login --no-include-email)
docker build -t (저장소 이름) .
docker tag (저장소 이름):latest (계정 ID).dkr.ecr.ap-northeast-2.amazonaws.com/(저장소 이름):latest
docker push (계정 ID).dkr.ecr.ap-northeast-2.amazonaws.com/(저장소 이름):latest
```

- Windows의 경우 ECR에 로그인 할 때 (PowerShell)

```
Invoke-Expression -Command (aws ecr get-login --no-include-email)
```


Docker 이미지 테스트 (로컬에서)

- 다음 명령을 입력합니다.

```
docker run -p 8089:8089 -e TARGET_URL=(테스트 할 URL) (Docker 이미지 Tag)
```

- 그리고 localhost:8089로 접속해 보면?

Start new Locust swarm

Number of users to simulate

Hatch rate (users spawned/second)

Host

Start swarming

ECS에서 테스트 하기

왜 ECS를 선택했나?

- Fargate를 쓸 수 있어서: 서버를 따로 관리할 필요가 없음!
- Kubernetes 좋긴 한데...
 - **Kubernetes에 익숙하지 않음**
 - [Fargate on EKS](#)가 나왔지만, 서울 리전에서는 못 씀

ECS 클러스터 생성하기

- CLI로 ECS 클러스터 생성하기

```
aws ecs create-cluster --cluster-name (클러스터 이름)
```

ECS Task 생성하기

- ECS가 Task를 실행할 수 있는 IAM Role 만들기
 - [링크한 매뉴얼](#)의 1단계만 진행

ECS Task 작업 정의 ([상세 설명](#))

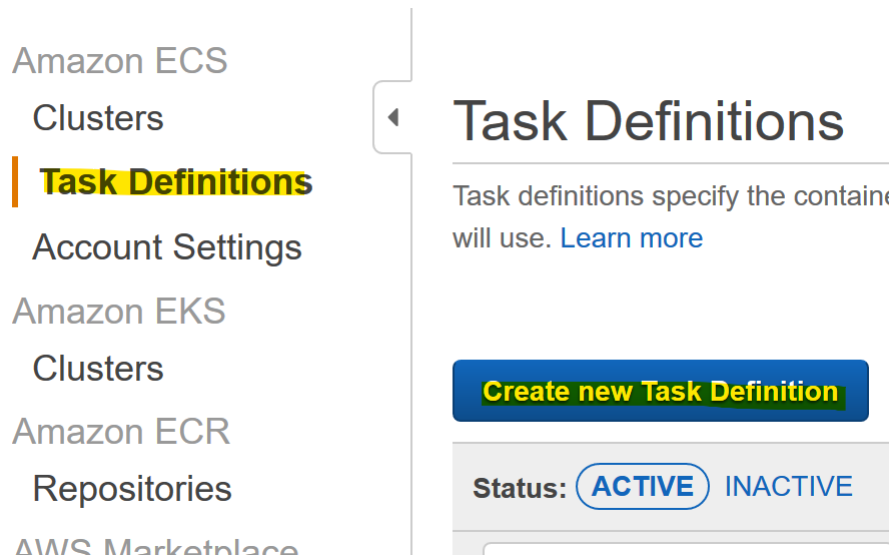
```
{
  "requiresCompatibilities": [ "FARGATE" ],
  "containerDefinitions": [{
    "name": "(Task 이름)",
    "image": "(계정 ID).dkr.ecr.ap-northeast-2.amazonaws.com/(저장소 이름)",
    "portMappings": [{ "containerPort": 8089, "protocol": "tcp" }]
  }],
  "networkMode": "awsvpc",
  "memory": "2048", → memory/cpu 조합은 링크 참조
  "cpu": "1024",
  "executionRoleArn": "(앞에서 만든 IAM Role)",
  "family": "(Task 이름)"
}
```

ECS Task 작업 정의 생성하기

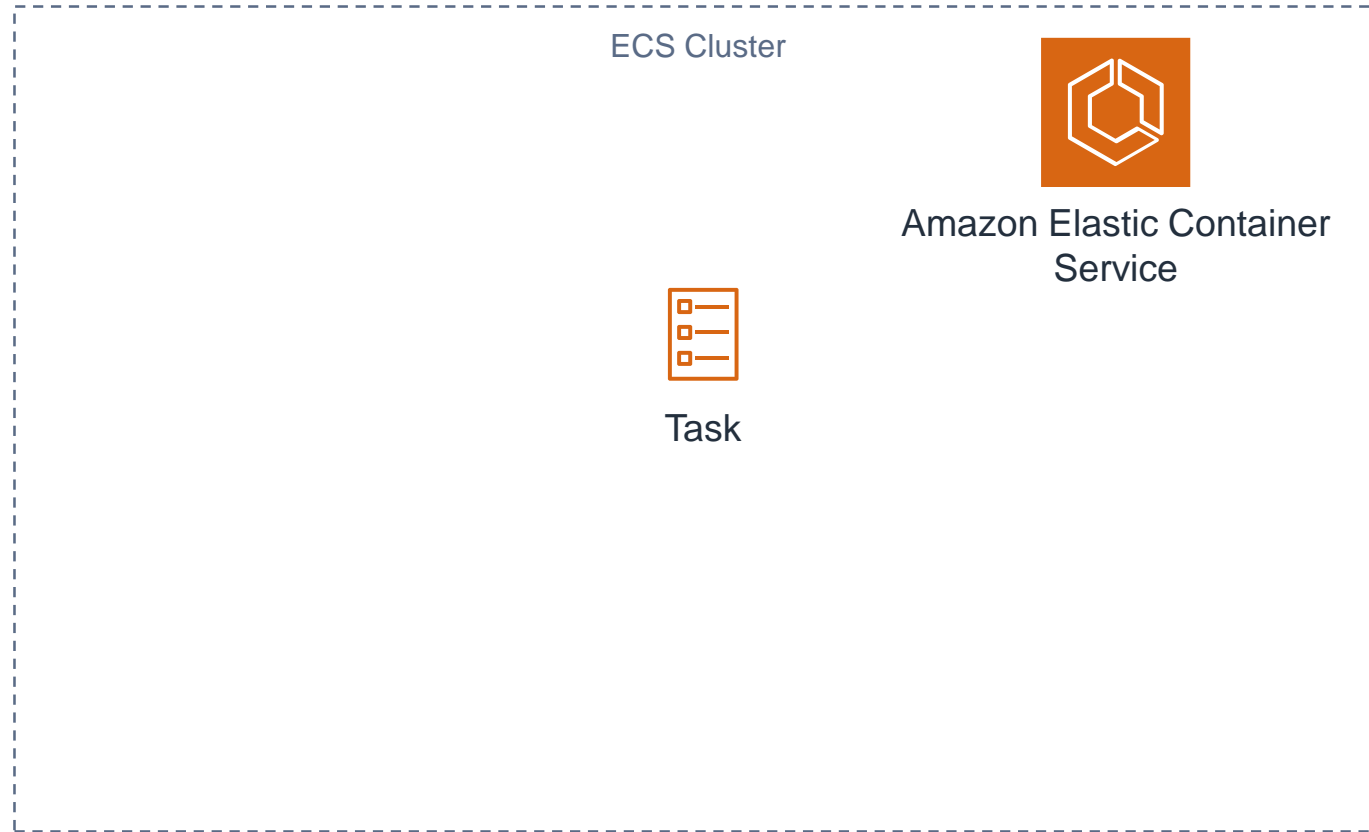
- CLI로 생성하기

```
aws ecs register-task-definition --cli-input-json file:///./task-definition.json
```

- 콘솔에서 생성하기 (이후 과정은 [매뉴얼](#) 참조)



하나의 컨테이너로 테스트하기



하나의 컨테이너로 테스트하기

- 보안 그룹을 생성합니다. (8089 포트를 허용)

```
aws ec2 create-security-group --description "Security Group for Locust Fargate Task" --group-name "Locust_Fargate_SG"
{
  "GroupId": "sg-(보안 그룹 ID)"
}

aws ec2 authorize-security-group-ingress --group-id sg-(보안 그룹 ID) --protocol tcp --port 8089 --cidr 0.0.0.0/0
```

하나의 컨테이너로 테스트하기

- Task에서 사용할 환경변수를 바꾸기 위한 JSON 파일
 - [링크 참조](#)
 - TARGET_URL 환경 변수: 테스트 할 URL 지정
- ECS Task를 실행합니다. (Output에서 taskArn 속성의 내용 메모!)

```
aws ecs run-task --launch-type FARGATE --cluster (클러스터 이름) --count 1 \
--network-configuration 'awsvpcConfiguration={subnets=["서브넷 ID","다른 서브넷 ID"],securityGroups=["앞에서 만든 보안 그룹 ID"],assignPublicIp="ENABLED"}' \
--overrides file://(앞에서 만든 JSON 파일) --task-definition (Task 이름:버전)
```

Task의 Public IP 확인하기

- Task의 Public IP를 확인한 뒤, (Public IP):8089로 접속

```
aws ecs describe-tasks --cluster (클러스터 이름) --tasks (Task ID-taskArn의 'task/' 뒷부분)
(결과 중)
```

```
...
{
  "name": "networkInterfaceId",
  "value": "eni-(ENI ID)"
}
```

```
aws ec2 describe-network-interfaces --network-interface-ids (ENI ID)
(결과 중)
```

```
...
  "PublicIp": "xxx.xx.xx.xxx",
...
```

Task 종료

- CLI에서 다음과 같이 입력

```
aws ecs stop-task --cluster (클러스터 이름) --task (Task ID)
```

- Task ID를 모르면 다음 명령으로 찾을 수 있습니다.

```
aws ecs list-tasks --cluster (클러스터 이름)
{
  "taskArns": [
    "arn:aws:ecs:ap-northeast-2:(계정 ID):task/(Task ID)"
  ]
}
```

여러 컨테이너로 테스트하기

- master/slave 각각의 IP를 알면 서로 통신 가능하지 않을까?

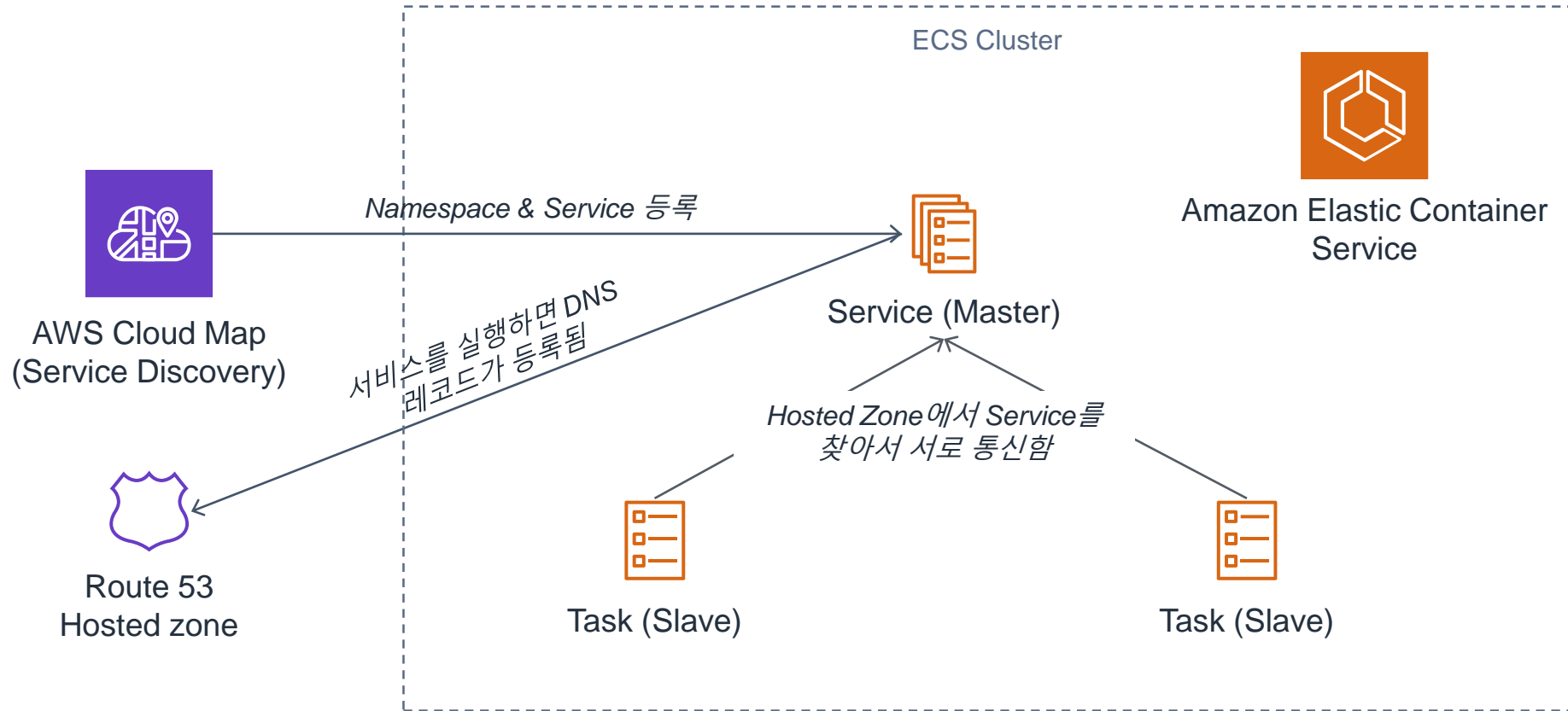
여러 컨테이너로 테스트하기

• ~~master/slave 각각의 IP를 알면 서로 통신 가능하지 않을까?~~

(안 됩니다)

- 컨테이너 간 통신
 - Service Discovery (Cloud Map)
 - App Mesh

시스템 구성도 (여러 개의 컨테이너)



Service Discovery 설정

- 네임스페이스 생성

```
aws servicediscovery create-private-dns-namespace --name (Namespace 이름) --vpc  
(VPC ID) --region ap-northeast-2
```

(Output에서 OperationId 값 복사)

```
aws servicediscovery get-operation --operation-id (앞에서 가져온 Operation ID)
```

(Output에서 Targets 안의 NAMESPACE 부분의 값을 복사)

- 서비스 생성

```
aws servicediscovery create-service --name master --dns-config  
'NamespaceId="(Namespace ID)",DnsRecords=[{Type="A",TTL="300"}]' --health-  
check-custom-config FailureThreshold=1
```

Output에서 "Service" 안의 "Id" 부분의 값을 복사

Service Discovery 설정

- 보안 그룹 설정 (Master-Slave 간 통신을 위해 5557,5558 포트 개방)

```
aws ec2 create-security-group --description "Security Group for Locust Fargate Task" --group-name "Locust_Fargate_Master_SG"
{
  "GroupId": "sg-(보안 그룹 ID)"
}
aws ec2 authorize-security-group-ingress --group-id sg-(보안 그룹 ID) --protocol tcp --port 8089 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress --group-id sg-(보안 그룹 ID) --protocol tcp --port 5557 --cidr (IP 대역/서브넷 마스크)
aws ec2 authorize-security-group-ingress --group-id sg-(보안 그룹 ID) --protocol tcp --port 5558 --cidr (IP 대역/서브넷 마스크)
```

다시 Task 정의 만들기

- 앞에서 만든 Task는 5557, 5558 포트가 개방되어 있지 않음
- 그래서 다시 Task 정의를 만들어야 함
- 변경된 내용 ([파일 참조](#))

```
"portMappings": [  
  { "containerPort": 8089, "protocol": "tcp" },  
  { "containerPort": 5557, "protocol": "tcp" },  
  { "containerPort": 5558, "protocol": "tcp" }  
],  
"environment": [  
  { "name": "TARGET_URL", "value": "(테스트할 URL)" },  
  { "name": "LOCUST_MODE", "value": "master" }  
]
```

다시 Task 정의 만들기

- CLI에서 Task 정의 생성

```
aws ecs register-task-definition --cli-input-json file://(Task 정의 파일 경로)
```

Master 역할을 할 서비스 올리기

- 서비스 실행을 위한 설정 파일

```
{
  "cluster": "(클러스터 이름)",           // ECS 클러스터 이름
  "serviceName": "(서비스 이름)",         // 서비스 이름
  "taskDefinition": "(Task 이름:버전)",    // Task 정의
  "serviceRegistries": [                  // 앞에서 만든 Service Discovery의 서비스와 연결
    {
      "registryArn": "arn:aws:servicediscovery:(Region):(계정 ID):service/(Service Discovery에 등록된 서비스 ID)"
    }
  ],
  "networkConfiguration": {               // 네트워크 설정
    "awsvpcConfiguration": {
      "assignPublicIp": "ENABLED",
      "securityGroups": [ "(보안 그룹 ID)" ],
      "subnets": [ "(서브넷 ID)", "(다른 서브넷 ID)" ]
    }
  },
  "desiredCount": 1                       // 1개의 컨테이너만 실행
}
```

Master 역할을 할 서비스 올리기

- CLI로 서비스 생성

```
aws ecs create-service --cli-input-json file://(서비스 설정 파일 경로)
```

- Master 서비스의 DNS 레코드 확인

```
aws servicediscovery get-namespace --id (Namespace ID) --region ap-northeast-2  
(HostedZoneId 속성의 값을 찾아 메모)  
aws route53 list-resource-record-sets --hosted-zone-id (Host Zone ID) --region  
ap-northeast-2  
Output 중 "ResourceRecordSets" 안의 "Name" 속성 값을 찾아 메모  
( 'Master 서비스 이름.Namespace 이름' 과 같은 방식으로 등록되어 있음)
```

Slave 역할을 할 Task 올리기

- Master/Slave 모두 동일한 Locust 파일을 갖고 있어야 하므로, Master의 작업 정의를 그대로 이용
- Task Override를 위한 환경 변수 ([참고](#))
 - LOCUST_MODE: slave
 - LOCUST_MASTER_HOST: 앞에서 얻은 Master 서비스의 DNS 정보 (Master 서비스.Namespace 이름)

Slave 역할을 할 Task 올리기

- Task 실행하기

```
aws ecs run-task --launch-type FARGATE --cluster (클러스터 이름) --count 2 \
--network-configuration 'awsvpcConfiguration={subnets=["서브넷 ID","다른 서브넷 ID"],securityGroups=["앞에서 만든 보안 그룹 ID"],assignPublicIp="ENABLED"}' \
--overrides file://(앞에서 만든 JSON 파일) --task-definition (Task 이름:버전)
```

- Master 서비스의 Public IP:8089로 접속하면?

Start new Locust swarm

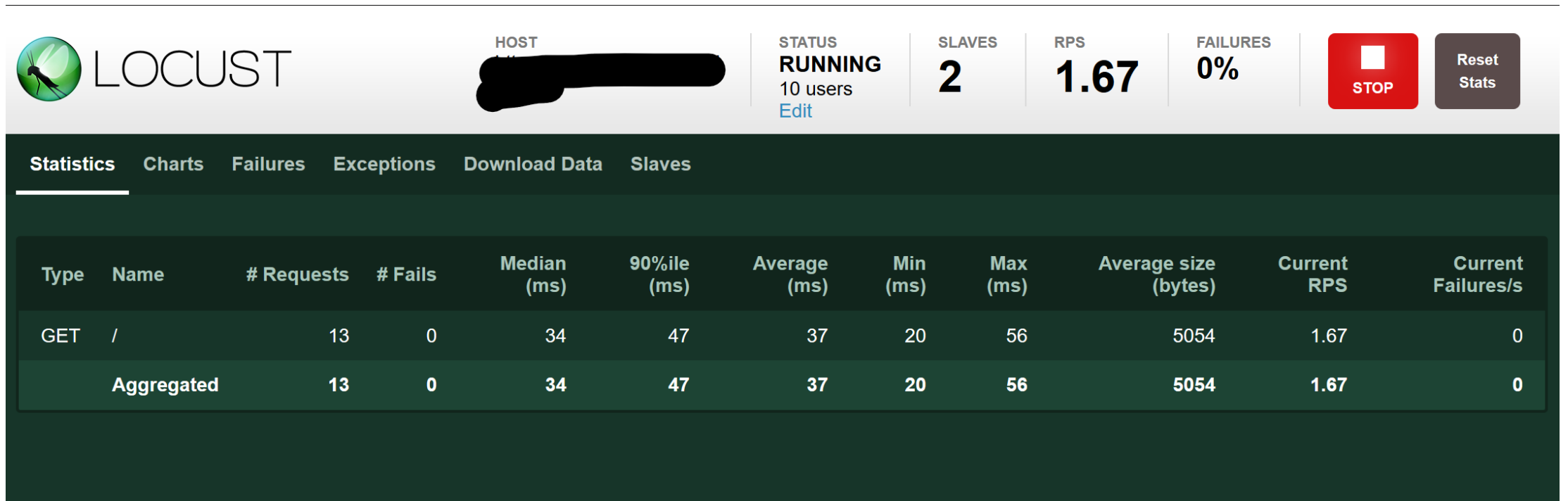
Number of users to simulate

Hatch rate (users spawned/second)

Host

Start swarming

테스트 결과 확인하기



리소스 정리하기

- 전체 과정

- ECS Cluster에서 실행 중인 서비스 종료 후 삭제
- ECS Cluster 내 Slave Task 모두 종료
- Service Discovery 내 등록된 서비스 삭제
- Service Discovery Namespace 삭제
- ECS 클러스터 삭제
- ECR 저장소 삭제

DEMO

Master – Slave 구성

개선해야 할 것들

- JavaScript를 실행하는 페이지 성능 측정이 어려움
- 테스트 내용이 달라질 때마다 Docker 이미지를 만들어야 함
- 지금까지 설명한 것들을 모아서 한 번에 만들 수 있는 방법?

Q&A

감사합니다!

<https://github.com/rubysoho07/locust-on-ecs>

hahafree12@gmail.com