

# Sistemas Distribuídos - Trabalho Final - Cluster Spring Boot com Load Balance + Docker

Eduardo Zunino Feller, Hugo Marcel Larsen, Lucas Vanderlinde

Departamento de Sistemas e Computação  
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

`efeller@furb.br`, `hlarsen@furb.br`, `lucvanderlinde@furb.br`

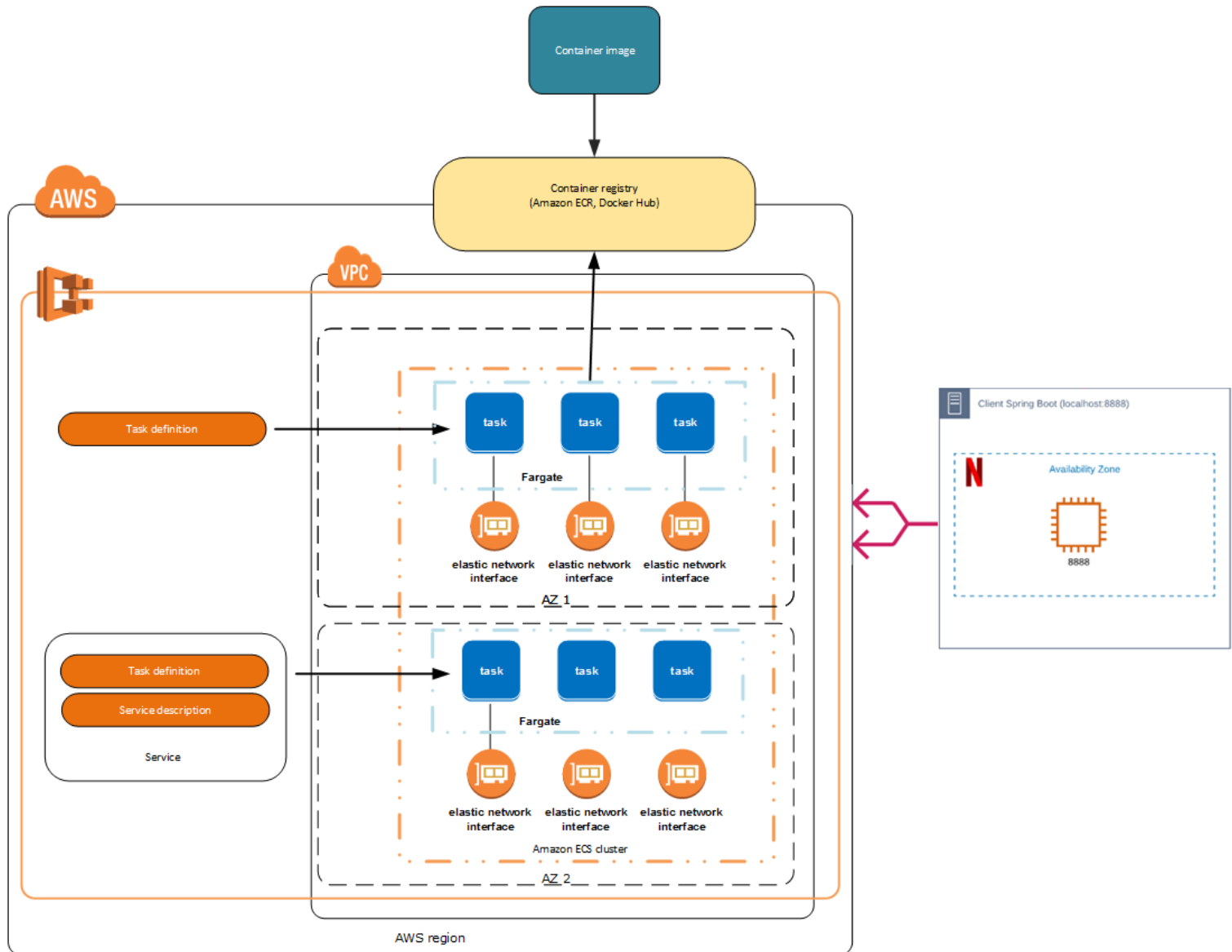
## 1. Introdução

Este relatório apresentará toda a parte de arquitetura e funcionamento de uma aplicação *Client-Server clusterizada*. Utilizando Spring Boot e Ribbon (Netflix Open Source Software) para o *load balance* entre as instâncias. Como método de balanceamento, será utilizado *Round-robin*, que no nosso caso, balanceará a carga entre as três instâncias rodando no serviço Elastic Container Service (ECS) da Amazon Web Services. Cada nova requisição recebida em nosso *cluster* seria atendida por um destes nós. A primeira seria atendida pelo nó 1, a próxima pelo nó 2, a seguinte pelo nó 3 e então voltando ao nó 1 e reiniciando o processo. Com esta técnica é possível distribuir o trabalho de forma equilibrada para cada máquina disponível sem precisar calcular métricas de balanceamento.

O propósito do trabalho foi aperfeiçoar o aprendizado em tecnologias muito importantes no mercado de trabalho e que eram desconhecidas para todos os integrantes do grupo, dentre elas, toda a parte de configuração do ECS, implementação de *Clusters* balanceados usando Spring Boot + Ribbon, e a utilização do Docker para efetuar a criação da imagem e de tarefas.

## 2. Desenvolvimento

A arquitetura do projeto pode ser descrita conforme a imagem a seguir:



O *Client* da aplicação é alocado num servidor Spring Boot. Quando acessada a URL <http://localhost:8888/> o *Client*, utilizando Ribbon, faz uma requisição HTTP para o *Server*, alocado num *cluster* no ECS. A requisição pode cair em qualquer uma das três instâncias, dependendo da ordem que estiver no *Round-Robin*.

A aplicação no *Server* irá retornar as informações: IP, Host e Porta. Além das variáveis de ambiente da máquina atual.

## 2.1. Server-Side

No lado do servidor a implementação é responsável apenas por retornar as informações de internet e as variáveis de ambiente da máquina instanciada no ECS.

Utilizadas apenas as classes *InetAddress* e *System* nativas do Java conforme abaixo:

```
try {
    InetAddress inetAddress = InetAddress.getLocalHost();
    StringBuilder result = new StringBuilder();
    result.append("<b>IP  
Address:</b>").append(inetAddress.getHostAddress()).append("<br>");
    result.append("<b>Host  
Name:</b>").append(inetAddress.getHostName()).append("<br>");
    result.append("<b>Port:</b>").append(environment.getProperty("server.port")).append("<br>");

    result.append("<b>Properties:</b>");
    result.append("<ul>");
    for (Object propertyKeyName : System.getProperties().keySet()) {
        result.append("<li>");
        result.append(propertyKeyName).append(":  
").append(System.getProperty(propertyKeyName.toString()));
        result.append("</li>");
    }
    result.append("</ul>");

    return result.toString();
} catch (UnknownHostException e) {
    return e.toString();
}
```

## 2.2. Client-Side

No lado do cliente é onde temos a maior parte das configurações, e consequentemente, a maior parte da complexidade.

Para adicionar o Ribbon ao nosso projeto é necessário apenas adicionar uma anotação a classe do *Client* conforme abaixo:

```
@RibbonClient(name = "backside", configuration = BalancerConfiguration.class)
```

Devemos adicionar também a anotação `@LoadBalanced` ao nosso `RestTemplate` conforme abaixo:

```
@LoadBalanced
@Bean
RestTemplate restTemplate(){
    return new RestTemplate();
}
```

Como vemos na anotação `@RibbonClient` acima, devemos também passar uma classe de configuração. Na nossa aplicação, a classe de configuração define como será feito o *Ping* de checagem de saúde das instâncias. E qual será o algoritmo de balanceamento de carga. Podemos ver o código abaixo:

```
@Bean
public IPing ribbonPing(IClientConfig config) {
    return new PingUrl();
}

@Bean
public IRule ribbonRule(IClientConfig config) {
    return new AvailabilityFilteringRule();
}
```

O método responsável por efetuar a chamada para o *Server* em si contém a chamada HTTP para o serviço mapeado pelo arquivo **application.yml** do Spring. Utilizando o `RestTemplate` balanceado pelo Ribbon conforme definimos acima.

```
@RequestMapping("/")
public String get() {
    return this.restTemplate.getForObject("http://backside/host",
String.class);
}
```

O Ribbon é comumente utilizado junto com o Eureka, o serviço de *Service Discovery* da Netflix. Como no nosso caso temos apenas três serviços num cluster do ECS, devemos mapear os nossos serviços de forma fixa.

No arquivo de configurações abaixo, podemos ver a configuração necessária para que tenhamos um *Client* balanceado e funcional.

```
backside:
  ribbon:
    eureka:
      enabled: false
    listOfServers:
18.230.126.194:8092,18.230.134.5:8092,15.228.3.52:8082
    ServerListRefreshInterval: 15000
```

No arquivo acima definimos primeiramente o nome do serviço que foi definido no **application.yml**, que está na aplicação *server-side*. Após isso devemos definir que não estamos utilizando o Eureka. E então a lista de servidores que o Ribbon deve checar e utilizar para efetuar o balanceamento de carga. Por último, temos a definição do intervalo de checagem da saúde das instâncias.

### 2.3. Infraestrutura

Na parte da infraestrutura temos o nosso servidor rodando em três tarefas dentro de um *Cluster* no serviço ECS. O serviço ECS da Amazon Web Services já possui um Docker instalado em suas máquinas/*clusters*, então é preciso apenas indicar a imagem que deve ser usada, sem a necessidade de fazer alguma configuração por linha de comando. Os passos abaixo descrevem como o objetivo foi alcançado:

1. Antes de tudo foi preciso instalar o Docker Toolbox na máquina. Com isso podemos utilizar os serviços Docker por linha de comando. Os comandos abaixo devem ser executados na pasta */backside* juntamente com o arquivo *Dockerfile* para criar a imagem e incluí-la no repositório do Docker Hub.

```
docker build -t backside .
docker tag backside hugolarsen/load-balancer-backside
docker login
docker push hugolarsen/load-balancer-backside
```

2. O *Dockerfile* da aplicação basicamente adiciona o Java e o .jar na imagem. Não é necessário adicionar nenhum servidor porque o Spring Boot contém o Tomcat embarcado.

```
FROM openjdk:8-jdk-alpine
WORKDIR /
ADD /target/distributed-systems-final-backside-0.1.0.jar distributed-
systems-final-backside.jar
CMD java -jar distributed-systems-final-backside.jar
```

3. Com uma conta ativa na Amazon Web Services, ativar os serviços do Elastic Container Service.
4. Acessar o console do ECS e criar uma tarefa conforme os passos abaixo:

The screenshot shows the AWS Management Console interface for Task Definitions. The left sidebar contains navigation links for Amazon ECS Clusters, Task Definitions (highlighted), Account Settings, Amazon ECR Repositories, AWS Marketplace Discover software, and Subscriptions. The main content area is titled 'Task Definitions' and includes a description: 'Task definitions specify the container information for your application, such as how many containers are part of your task, what resources they will use, how they are linked together, and which host ports they will use. [Learn more](#)'. Below the description are buttons for 'Create new Task Definition' (highlighted with a red box), 'Create new revision', and 'Actions'. A status bar shows 'Status: ACTIVE INACTIVE' and a filter input 'Filter in this page'. A table with columns 'Task Definition' and 'Latest revision status' is shown, but it contains 'No results'.

a) Criando uma tarefa no menu **Task Definitions**.

## Create new Task Definition

### Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

### Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

The image compares two launch types: FARGATE and EC2. FARGATE is highlighted with a blue border and features an icon of a cube on a stand. Its characteristics are: 'Price based on task size', 'Requires network mode awsvpc', and 'AWS-managed infrastructure, no Amazon EC2 instances to manage'. EC2 features an icon of three stacked squares and has characteristics: 'Price based on resource usage', 'Multiple network modes available', and 'Self-managed infrastructure using Amazon EC2 instances'.

\*Required

[Cancel](#)

[Next step](#)

b) Selecionar o **Fargate** como sendo o launcher da tarefa.

## Create new Task Definition

Step 1:  
Select  
launch type  
compatibility

Step 2:  
Configure  
task and  
container  
definitions

### Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. Learn more ([http://docs.aws.amazon.com/AmazonECS/latest/developerguide/task\\_definitions.html](http://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html))

Task Definition Name\* load-balancer-backside

Requires Compatibilities\* FARGATE

Task Role

None

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the IAM Console (<https://sa-east-1.console.aws.amazon.com/iam/home#roles>)

Network Mode

awsvpc

If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows.

Network Mode : awsvpc

Your containers in the task will share an ENI using a common network stack. Port mappings can only specify container ports (any existing host port specifications will be removed).

- c) Definir os dados: **Task Definition Name** e **Task Role**. Note que o **Network Mode** awsvpc não permite Port Mapping, ou seja, não podemos mudar a porta da aplicação que irá executar na tarefa.

### Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

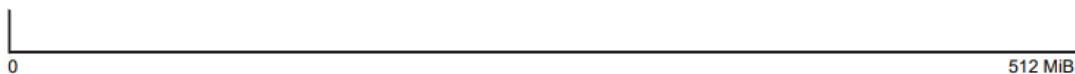
Task memory (GB) 0.5GB

The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

Task CPU (vCPU) 0.25 vCPU

The valid CPU for 0.5 GB memory is: 0.25 vCPU

Task memory maximum allocation for container memory reservation



Task CPU maximum allocation for containers



- d) Definir os dados: **Task Memory (GB)** e **Task CPU (vCPU)**. Eles não precisam ser elevados porque a aplicação é simples.

Add container

▼ Standard

Container name\*

docker-image

Image\*

docker.io/hugolarsen/load-balancer-backside

Private repository authentication\*

☐

Memory Limits (MiB)

Soft limit

512

+ Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.  
ECS recommends 300-500 MiB as a starting point for web applications.

\* Required

Cancel

Add

- e) Clicar no botão **Add Container** e adicionar as seguintes informações da imagem Docker (que é a imagem da nossa aplicação): **Container name**, **Image**: `docker.io/hugolarsen/-load-balancer-backside` e **Memory Limits**: Soft limit - 512.

5. Com a tarefa criada com sucesso, prosseguir para a criação de um *Cluster* para rodar as tarefas:

Amazon ECS

Clusters

Task Definitions

Account Settings

Amazon ECR

Repositories

AWS Marketplace

Discover software

Subscriptions

Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each account receives a default cluster the first time you use the Amazon ECS service. Clusters may contain more than one Amazon EC2 instance type.  
For more information, see the [ECS documentation](#).

Create Cluster

Get Started

View

list

card

0 - 0 of 0

view all

No clusters found

Get Started

- a) Criando um Cluster no menu **Clusters**.



## Create Cluster

### Step 1: Select cluster template

Step 2: Configure cluster

### Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

**Networking only**  
Resources to be created:  
Cluster  
VPC (optional)  
Subnets (optional)  
  
**Powered by AWS Fargate**

**EC2 Linux + Networking**  
Resources to be created:  
Cluster  
VPC  
Subnets  
Auto Scaling group with Linux AMI

**EC2 Windows + Networking**  
Resources to be created:  
Cluster  
VPC  
Subnets  
Auto Scaling group with Windows AMI

- b) Selecionar a opção **Networking only**. Os recursos EC2 das outras opções não são necessários.

## Create Cluster

Step 1: Select cluster template

### Step 2: Configure cluster

### Configure cluster

Cluster name\*  ⓘ

### Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC ☐ Create a new VPC for this cluster

### Tags

Key	Value
<input type="text" value="Add key"/>	<input type="text" value="Add value"/>

### CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. [Learn more](#)

CloudWatch Container Insights ☐ Enable Container Insights

- c) Finalizar a criação do *Cluster* com um nome.

6. Abrir o detalhamento do *Cluster* criado e adicionar a tarefa criada anteriormente:

Amazon ECS

- Clusters
- Task Definitions
- Account Settings

Amazon ECR

- Repositories

AWS Marketplace

- Discover software
- Subscriptions

Clusters > aws-fargate-cluster

### Cluster : aws-fargate-cluster

Get a detailed view of the resources on your cluster.

Cluster ARN: am:aws:ecs:sa-east-1:206641210966:cluster/aws-fargate-cluster

Status: PROVISIONING

Registered container instances: 0

Pending tasks count: 0 Fargate, 0 EC2

Running tasks count: 0 Fargate, 0 EC2

Active service count: 0 Fargate, 0 EC2

Draining service count: 0 Fargate, 0 EC2

Services | **Tasks** | ECS Instances | Metrics | Scheduled Tasks | Tags | Capacity Providers

**Run new Task** | Stop | Stop All | Actions

Last updated on July 6, 2020 7:45:13 PM (0m ago)

Desired task status: **Running** | Stopped

Filter in this page | Launch type: ALL

	Task	Task definit...	Container i...	Last status ...	Desired sta...	Started By ...	Group	Launch typ...	Platform ve...
No results									

a) Rodar uma nova tarefa no *Cluster*.

Amazon ECS

- Clusters
- Task Definitions
- Account Settings

Amazon ECR

- Repositories

AWS Marketplace

- Discover software
- Subscriptions

### Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. To apply container overrides or target particular container instances, click Advanced Options.

Launch type: ☒ FARGATE ☐ EC2

Switch to capacity provider strategy

Task Definition: Family: load-balancer-backside | Enter a value

Revision: 1 (latest)

Platform version: LATEST

Cluster: aws-fargate-cluster

Number of tasks: 1

Task Group:

b) Selecionar o *Fargate* como **Launch type** e definir a **Task Definition** como sendo a tarefa criada anteriormente.

## VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

**Cluster VPC\*** vpc-0beeff6c (172.31.0.0/16)

**Subnets\***  
subnet-c23eb799  
(172.31.32.0/20) - sa-east-1c  
assign ipv6 on creation: Disabled

**Security groups\*** sg-0a83f970b085f920a [Edit](#)

**Auto-assign public IP** ENABLED

- c) Definir o **Cluster VPC** e a **Subnet** carregada automaticamente por padrão do Cluster. No **Auto-assign public IP** deve ser selecionado **ENABLED**. Para podermos recuperar posteriormente.

### Configure security groups

**Assigned security groups** ☐ Create new security group  
☒ Select existing security group

#### Existing security groups

All existing security groups for the VPC of this cluster are listed below.

1 selected < 0-0 >				
<input type="checkbox"/>	Security group ID	Name	Description	Actions
<input type="checkbox"/>	sg-0751f47d35b5cf6ee	all-access-security2	2020-07-03T01:40:37.92...	<a href="#">Copy to new</a>
<input checked="" type="checkbox"/>	sg-0a83f970b085f920a	all-access-security	2020-07-03T00:40:23.38...	<a href="#">Copy to new</a>
<input type="checkbox"/>	sg-29733b57	default	default VPC security group	<a href="#">Copy to new</a>

#### Inbound rules for selected security groups

Security group ID	Type	Protocol	Port range	Source
sg-0a83f970b085f920a	customtcp	TCP	0 - 65535	0.0.0.0/0
sg-0a83f970b085f920a	All traffic	All traffic	All	0.0.0.0/0

[Cancel](#) [Save](#)

- d) Para a opção **Security Groups** anterior foi criado um grupo seguro **all-access-security**. Nele foi configurado um custom TCP e All Traffic. Ou seja, todas as permissões concedidas nessa tarefa.

7. O passo de adição de uma tarefa no Cluster foi repetido mais duas vezes. Dessa forma tivemos três tarefas executando corretamente conforme abaixo:

Services

Tasks

ECS Instances

Metrics

Scheduled Tasks

Tags

Capacity Providers

Run new Task

Stop

Stop All

Actions

Last updated on July 6, 2020 9:40:40 PM (0m ago)

Desired task status:

Running

Stopped

Filter in this page

Launch type

ALL

< 1-3 >

Page size

50

<input type="checkbox"/>	Task	Task definition	C..	Last status...	Desired sta...	S...	Group	Launch typ...	Platform v...
<input type="checkbox"/>	0d0e2979-f...	load-balancer-backside:1	--	RUNNING	RUNNING		family:load-balancer-ba...	FARGATE	1.3.0
<input type="checkbox"/>	34292802-d...	load-balancer-backside:1	--	RUNNING	RUNNING		family:load-balancer-ba...	FARGATE	1.3.0
<input type="checkbox"/>	b9a78797-e...	load-balancer-backside:1	--	RUNNING	RUNNING		family:load-balancer-ba...	FARGATE	1.3.0

8. Com as três tarefas executando corretamente, entrar no detalhamento de cada uma e buscar o **Public IP** para inserir no arquivo **application.yml** descrito anteriormente. Assim contendo todos os endereços a serem balanceados pelo Ribbon.

Amazon ECS

Clusters

Task Definitions

Account Settings

Amazon ECR

Repositories

AWS Marketplace

Discover software

Subscriptions

Clusters > aws-fargate-cluster > Task: 34292802-d622-4cda-b662-c11d92974e1d

Task : 34292802-d622-4cda-b662-c11d92974e1d

Details

Tags

Logs

Cluster

aws-fargate-cluster

Launch type

FARGATE

Platform version

1.3.0

Task definition

load-balancer-backside:1

Group

family:load-balancer-backside

Task role

None

Last status

RUNNING

Desired status

RUNNING

Created at

2020-07-06 20:10:02 -0300

Started at

2020-07-06 20:10:23 -0300

Network

Network mode

awsipc

ENI Id

eni-06612feeb27e88243

Subnet Id

subnet-c23eb799

Private IP

172.31.37.168

Public IP

18.230.126.194

- a) **Public IP** no detalhamento da tarefa executando no Cluster.

9. Executado o *Client* e acessado o endereço <http://localhost:8888>. Verificar as informações da máquina em que foi redirecionada a requisição.

### 3. Conclusão

Em virtude do que foi mencionado, conclui-se que o projeto foi finalizado conforme o que havia sido definido, com todos os requisitos alcançados. Quanto a dificuldade do projeto, verificamos um maior nível de complexidade na configuração do *Docker* local e no *Cluster* ECS, tivemos essa dificuldade devido ao não conhecimento dessas tecnologias pelos membros da equipe. Quanto a parte do *Ribbon* também não havíamos conhecimento, porém, encontramos grande quantidade de materiais na internet para apoio.

Como trabalho futuro, sugerimos a implementação de um *Client* com uma interface mais bonita, que interprete as informações da máquina e apresente os dados de forma mais amigável ao usuário, incluindo também mais informações. Outra sugestão seria a utilização de forma correta de um *Client* e de um *Middleware*. Na aplicação atual o *Client* faz os papéis dos dois sendo que o correto seria ter um serviço “meio de campo” para receber as requisições e fazer o balanceamento.

### 4. Referências

**What is Amazon Elastic Container Service?** Amazon Elastic Container Service - Developer Guide. Disponível em: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>. Acesso em: 24 de jun. de 2020.

**Microservices Tutorial: Ribbon as a Load Balancer.** MITRA, Shamik. Microservices Zone, 15 de ago. de 2017. Disponível em: <https://dzone.com/articles/microservices-tutorial-ribbon-as-a-load-balancer-1>. Acesso em: 24 de jun. de 2020.

**How to deploy Spring Boot microservices to Amazon ECS container service.** MCKENZIE, Cameron. Youtube, 13 de jul. de 2019. Disponível em: <https://www.youtube.com/watch?v=3yBIRmUJhio>. Acesso em: 24 de jun. de 2020.

**Install Docker Toolbox on Windows.** Docker. Disponível em: [https://docs.docker.com/toolbox/toolbox\\_install\\_windows/](https://docs.docker.com/toolbox/toolbox_install_windows/). Acesso em: 24 de jun. de 2020.