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

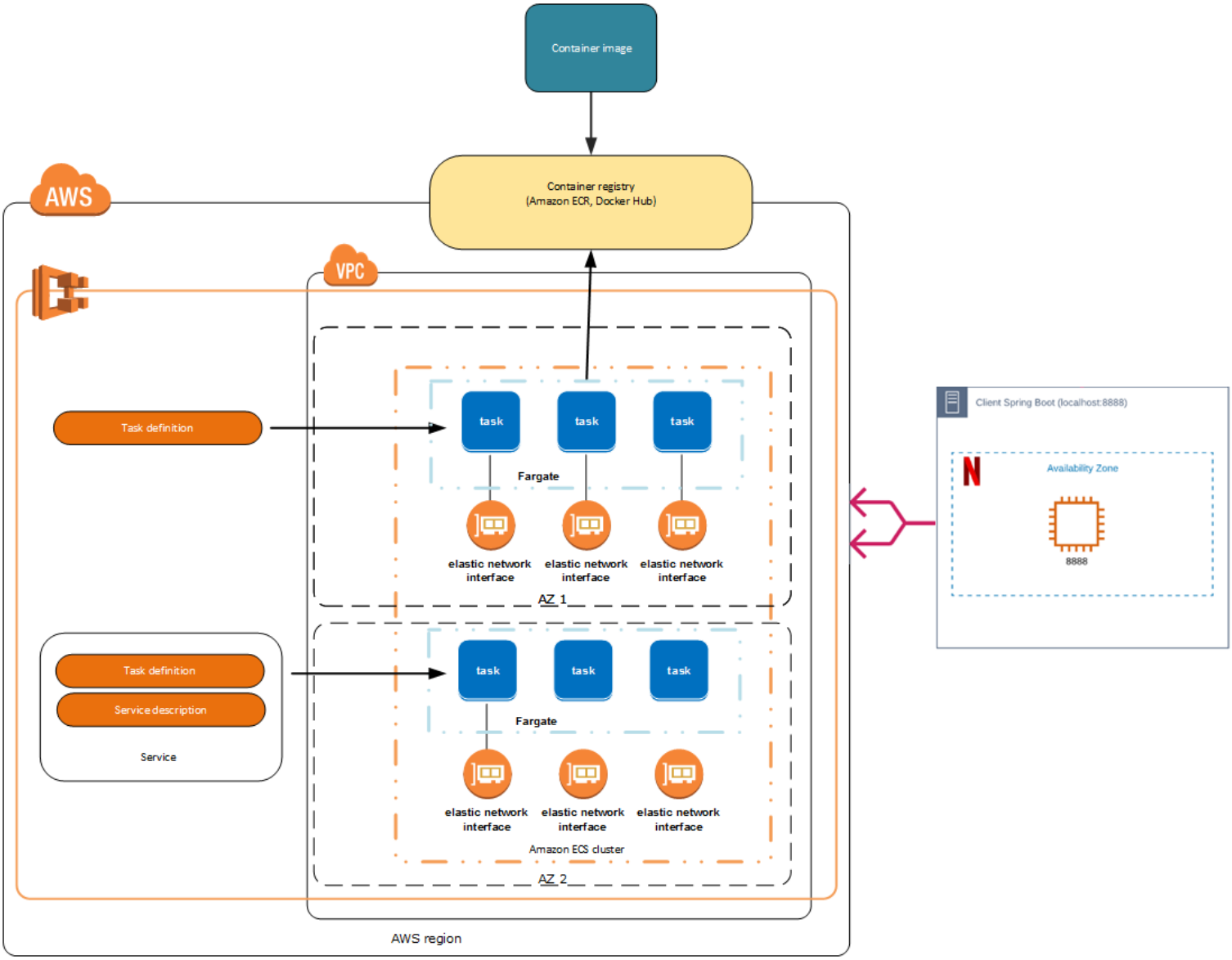
1. Introdução

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:

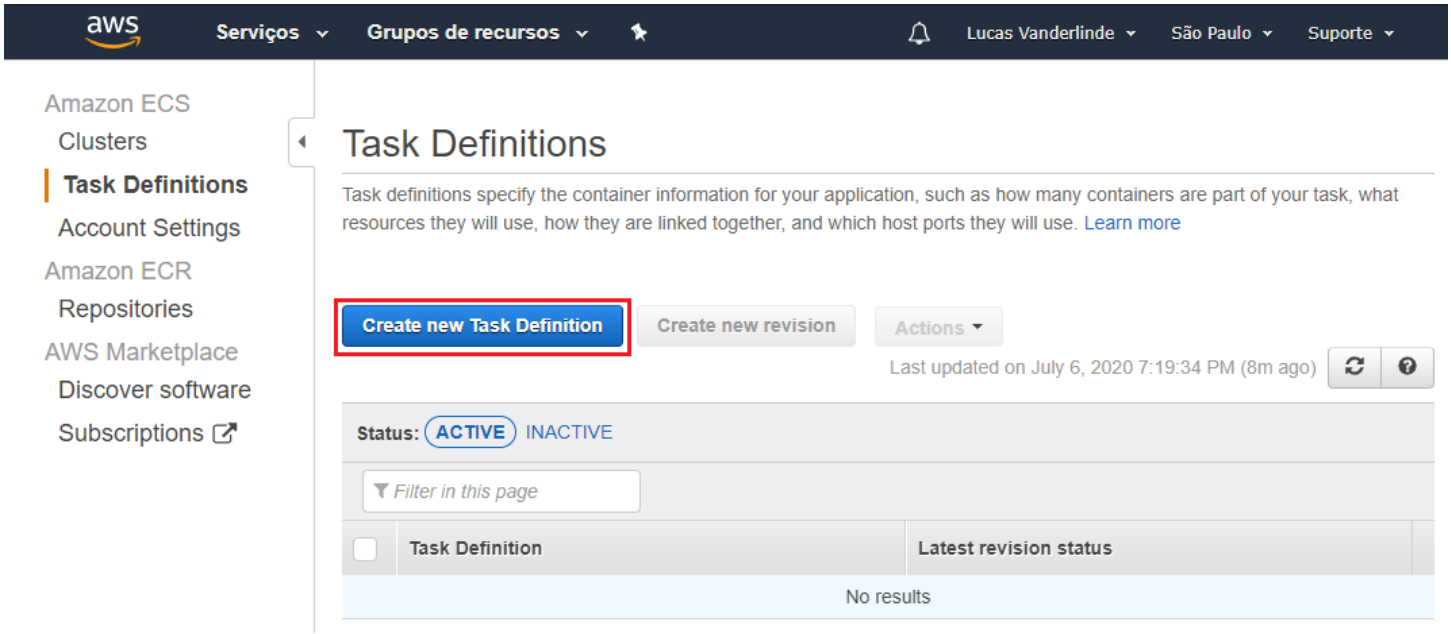
1. Antes de tudo foi preciso instalar o Docker Toolbox na máquina. Com isso podemos utilizar os serviços Docker por linha de comando. Os comandos abaixo devem ser executados na pasta */backside* juntamente com o arquivo *Dockerfile* para criar a imagem e incluí-la no repositório do Docker Hub.

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

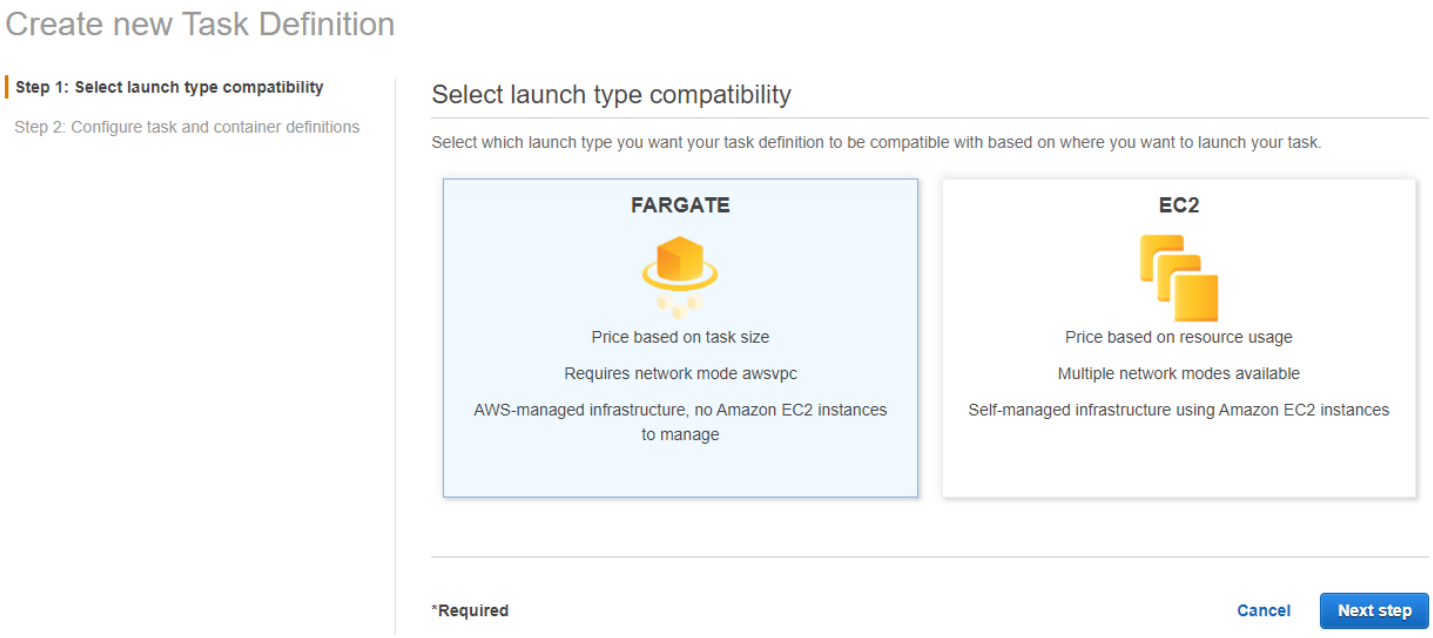
2. O *Dockerfile* da aplicação basicamente adiciona o Java e o .jar na imagem. Não é necessário adicionar nenhum servidor porque o Spring Boot contém o Tomcat embarcado.

```
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
```

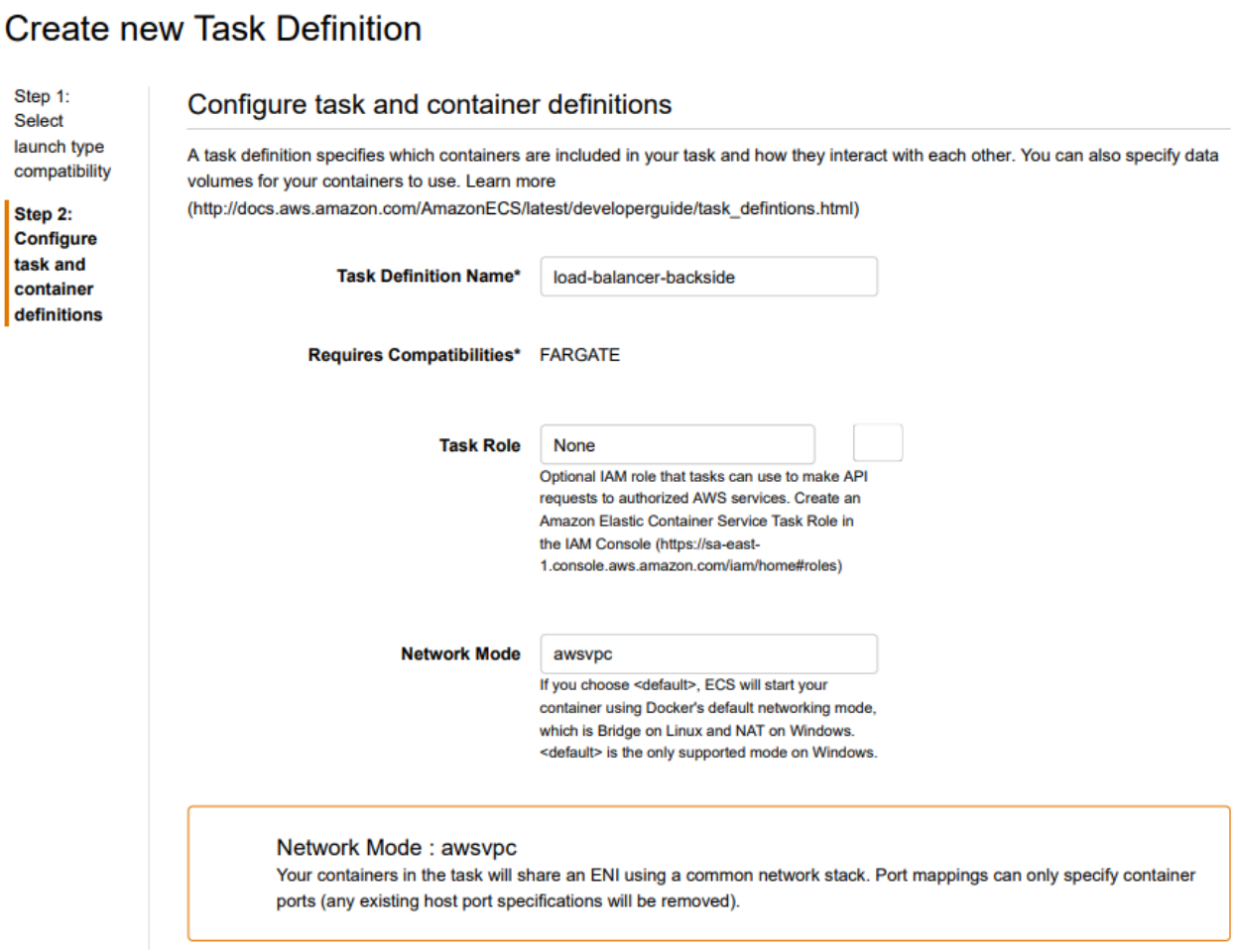
3. Com uma conta ativa na Amazon Web Services, ativar os serviços do Elastic Container Service.
4. Acessar o console do ECS e criar uma tarefa conforme os passos abaixo:



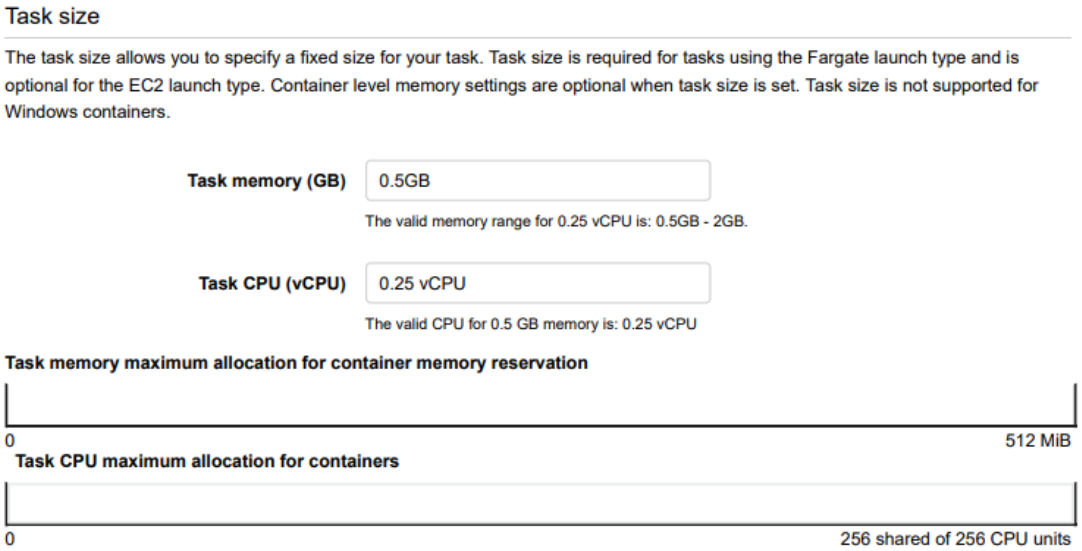
a) Criando uma tarefa no menu **Task Definitions**.



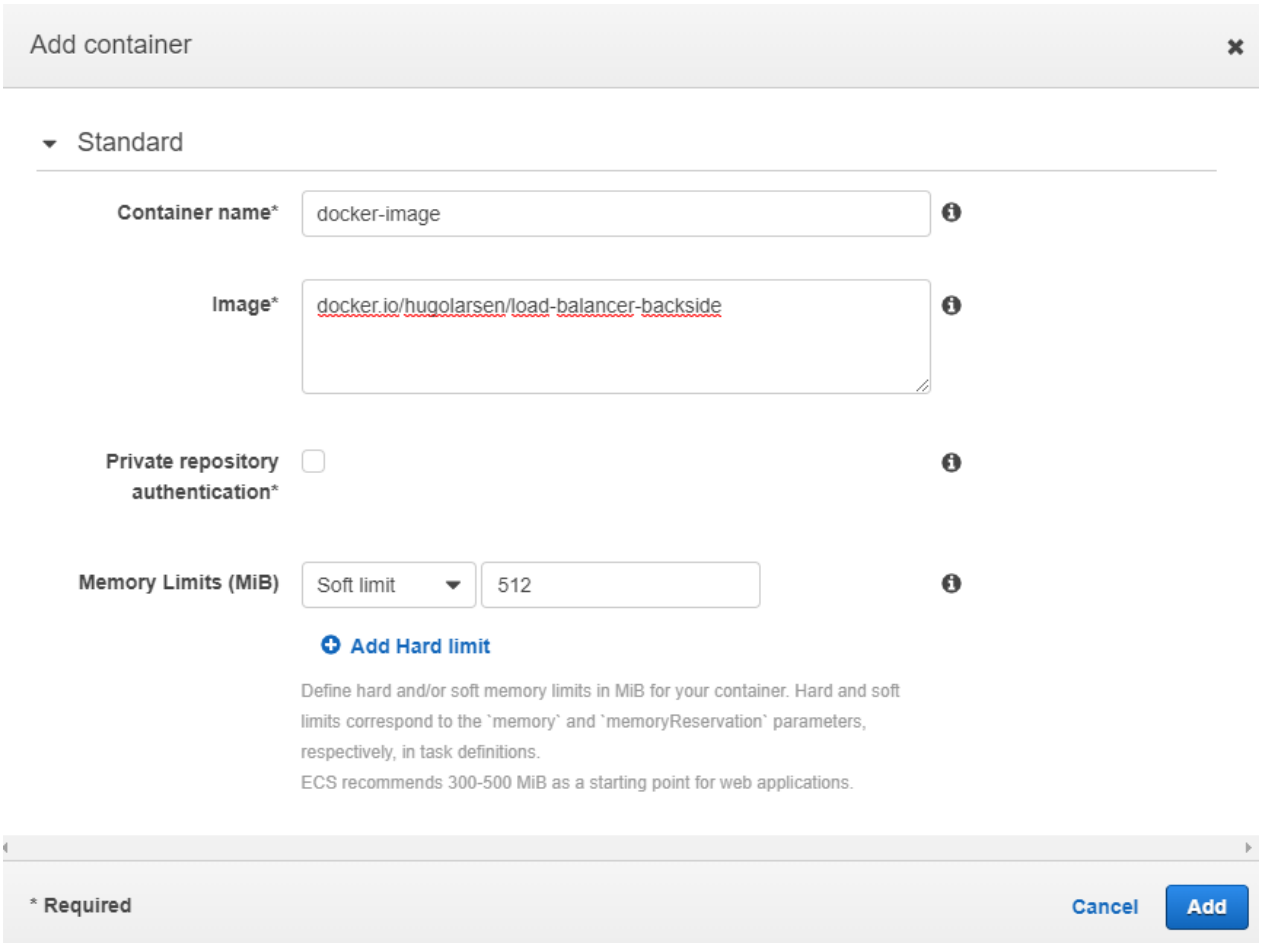
b) Selecionar o **Fargate** como sendo o launcher da tarefa.



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.

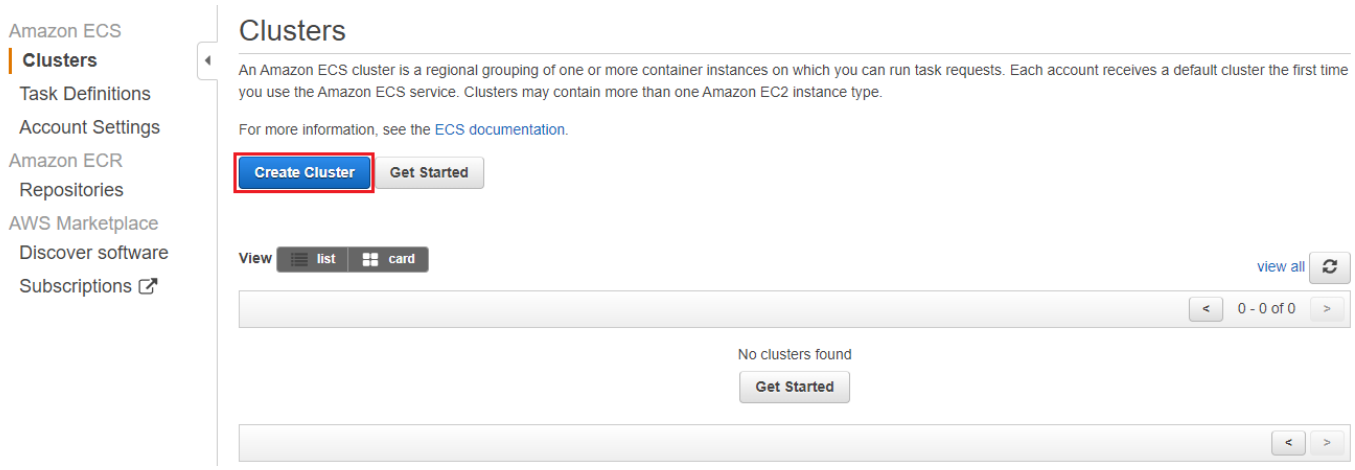


d) Definir os dados: **Task Memory (GB)** e **Task CPU (vCPU)**. Eles não precisam ser elevados porque a aplicação é simples.

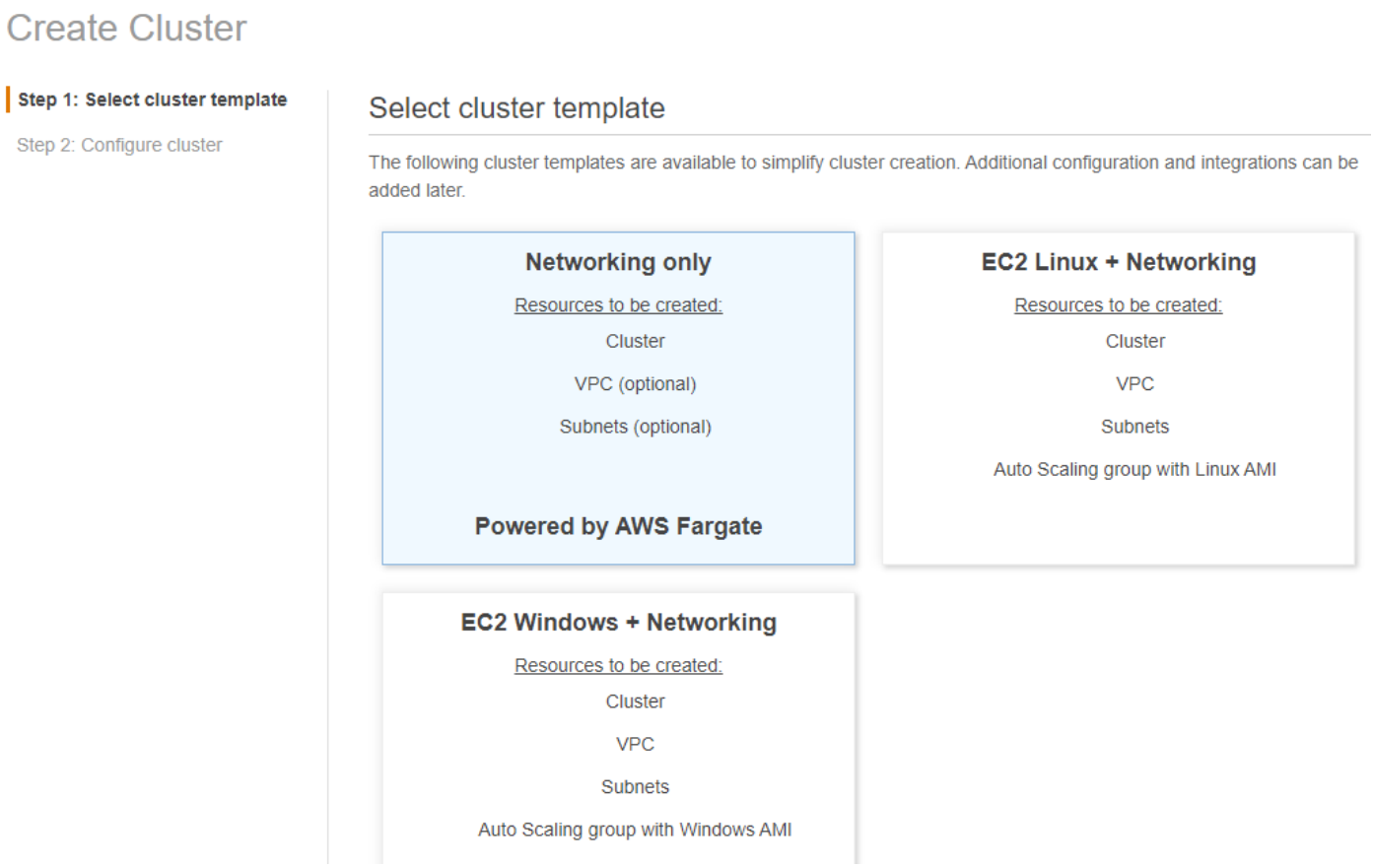


e) Clicar no botão **Add Container** e adicionar as seguintes informações da imagem Docker (que é a imagem da nossa aplicação): **Container name**, **Image**: docker.io/hugolarsen/-load-balancer-backside e **Memory Limits**: Soft limit - 512.

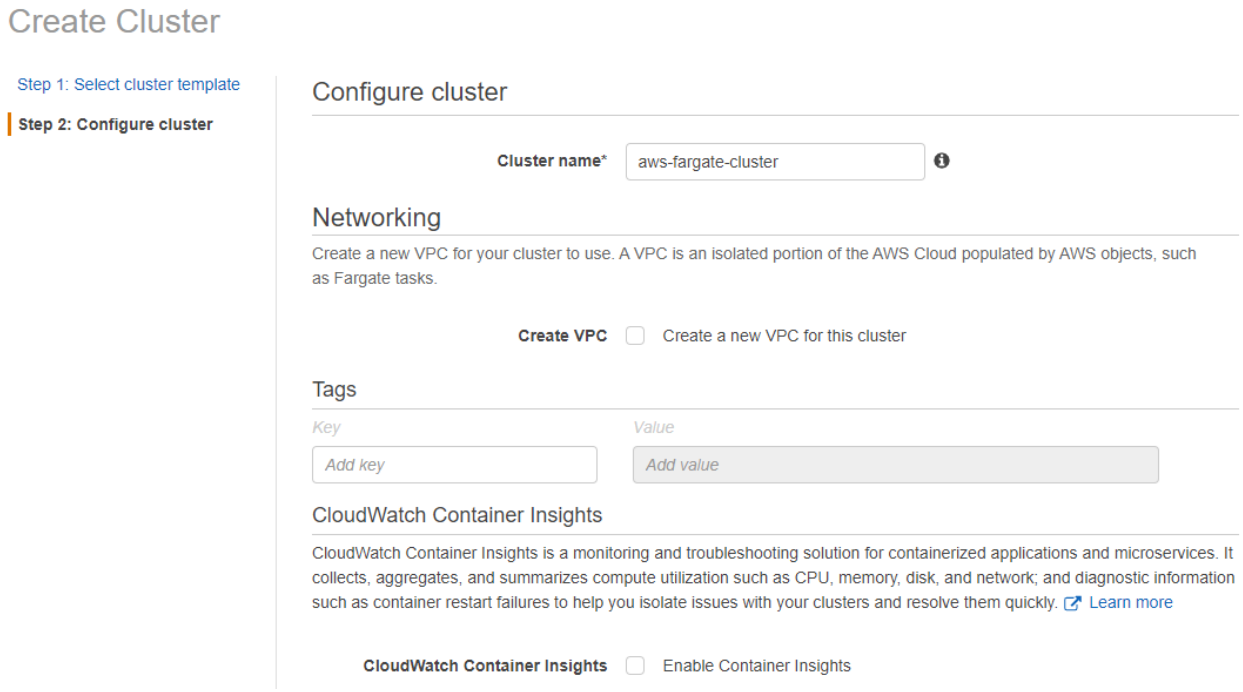
5. Com a tarefa criada com sucesso, prosseguir para a criação de um C*luster* para rodar as tarefas:



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

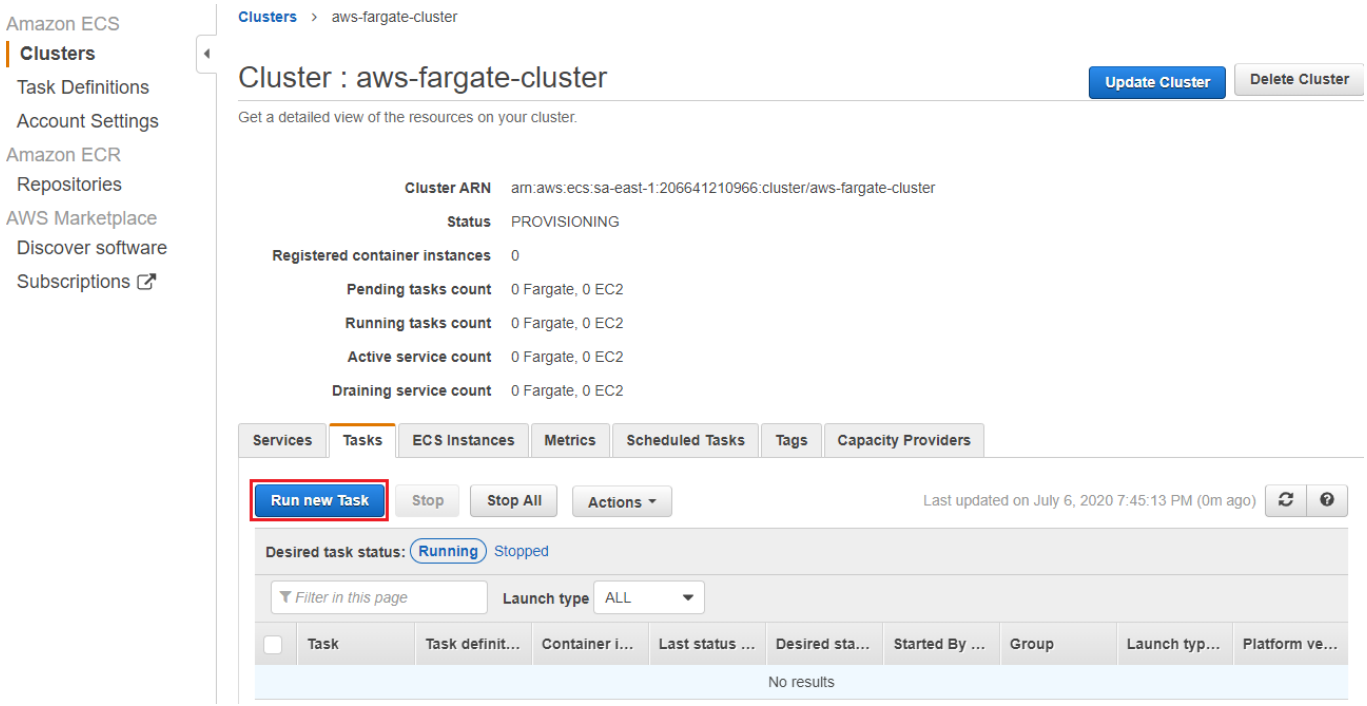


b) Selecionar a opção **Networking only**. Os recursos EC2 das outras opções não são necessários.

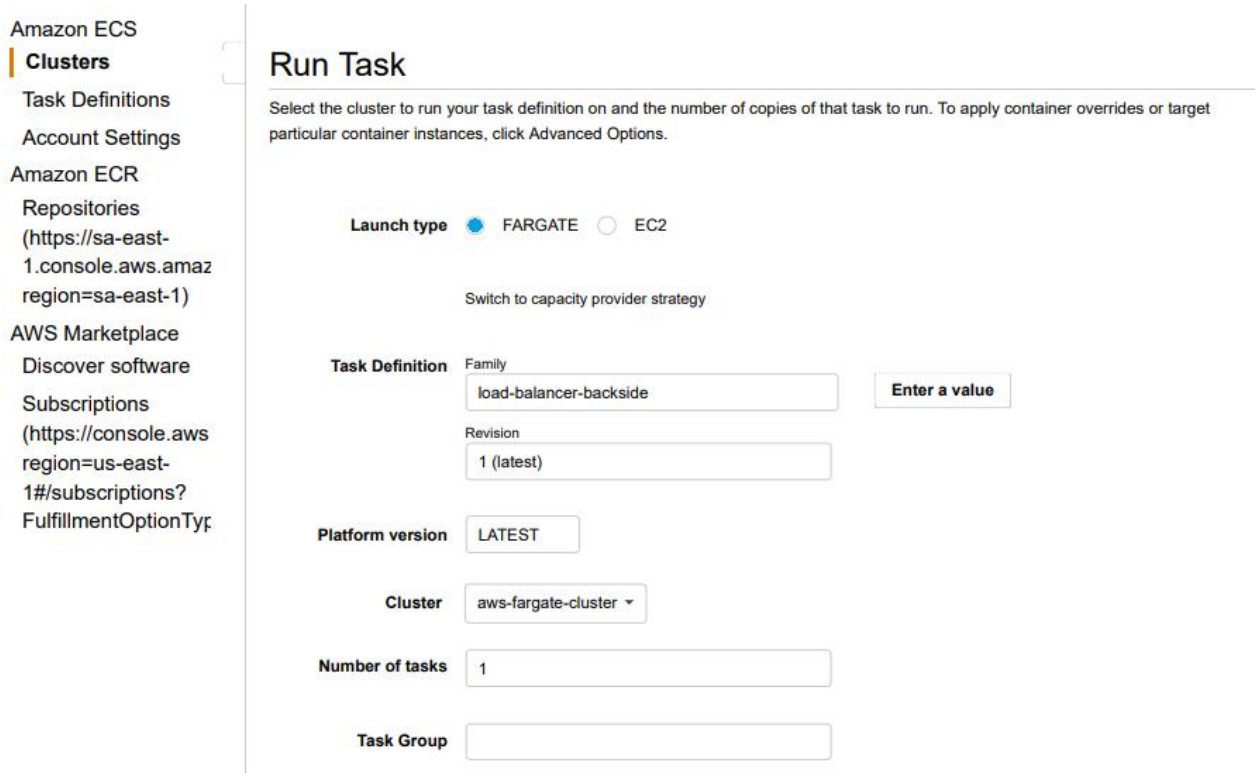


c) Finalizar a criação do *Cluster* com um nome.

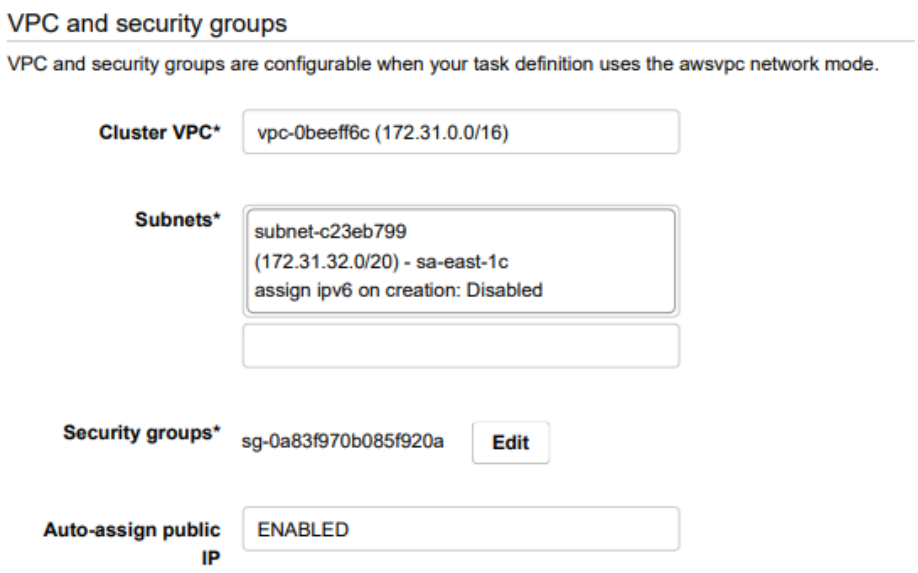
6. Abrir o detalhamento do *Cluster* criado e adicionar a tarefa criada anteriormente:



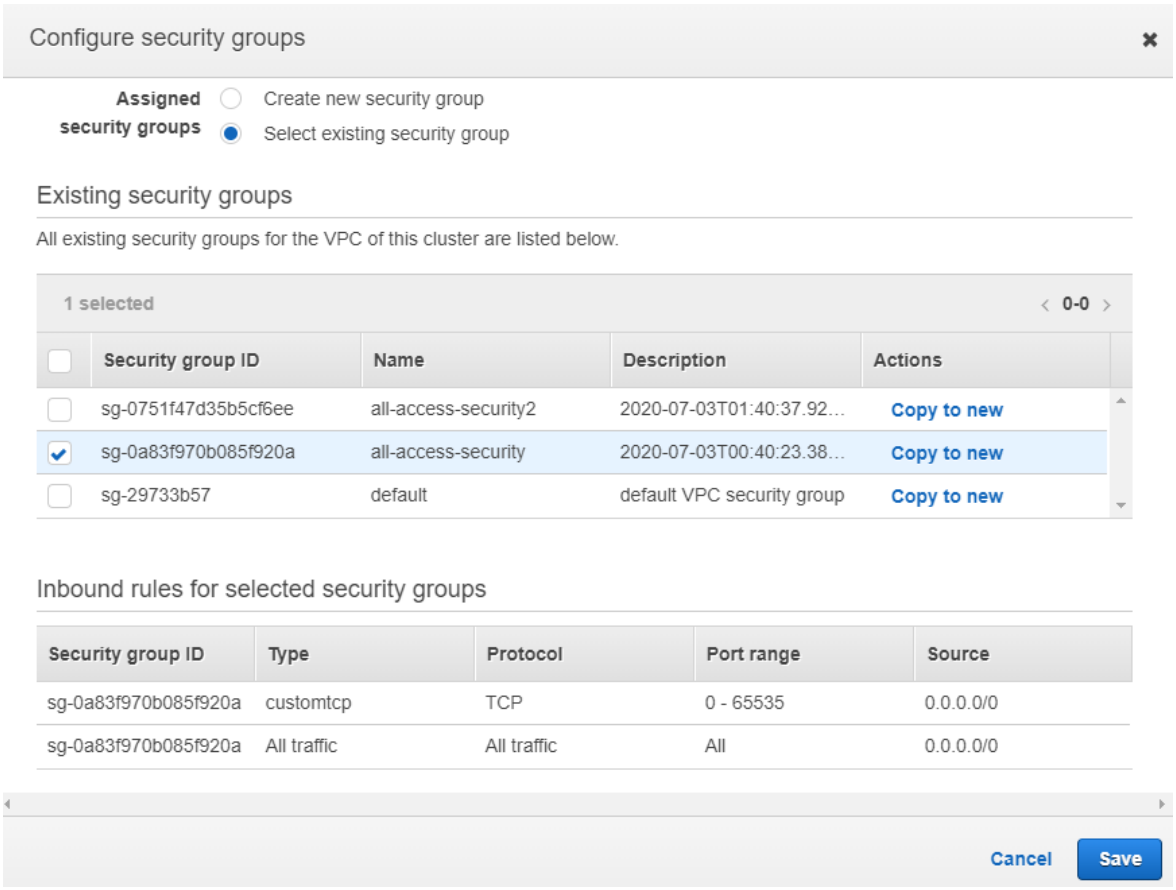
a) Rodar uma nova tarefa no *Cluster*.



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

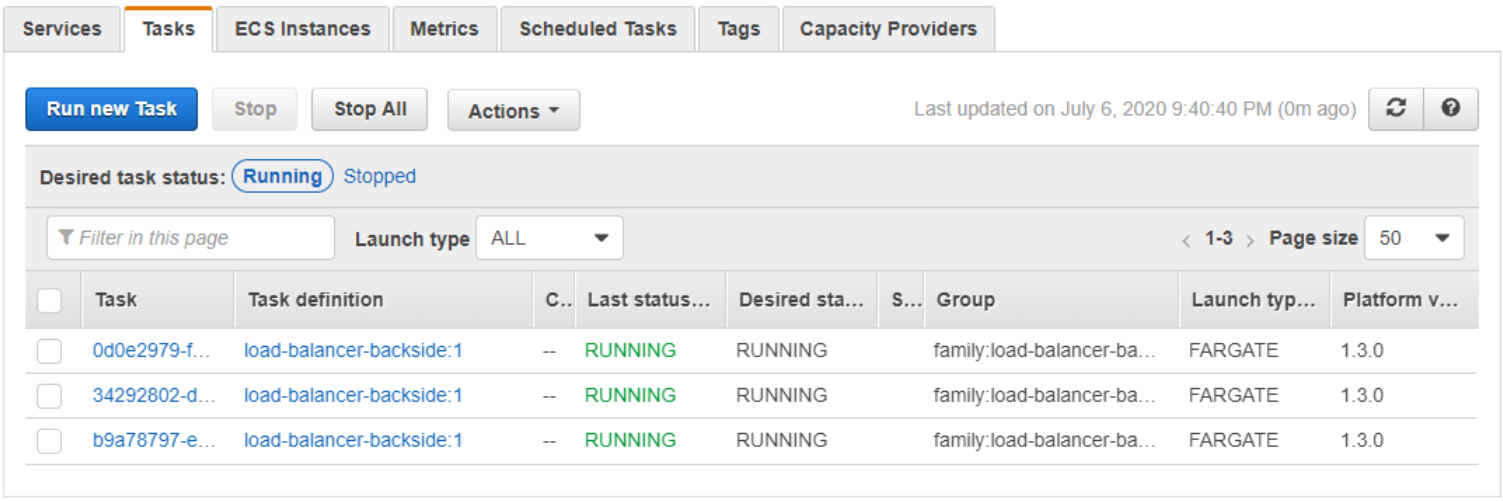


c) Definir o **Cluster VPC** e a **Subnet** carregada automaticamente por padrão do Cluster. No **Auto-assign public IP** deve ser selecionado ENABLED. Para podermos recuperar posteriormente.

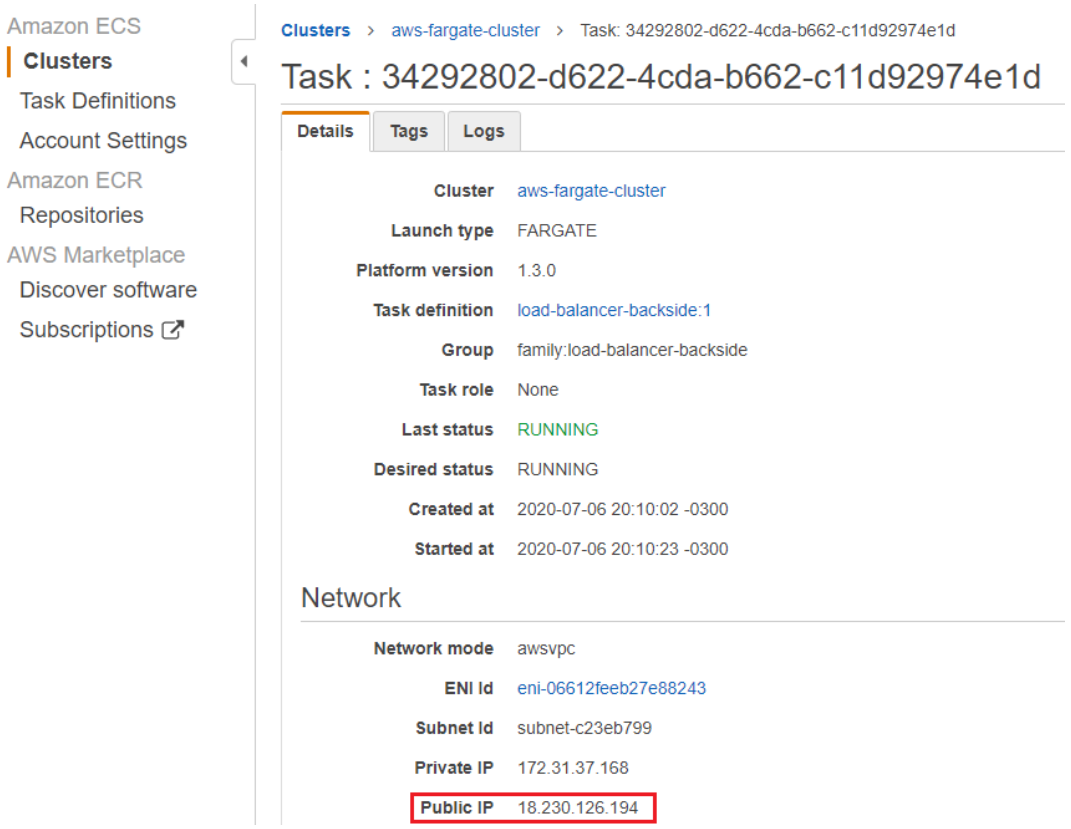


d) Para a opção **Security Groups** anterior foi criado um grupo seguro **all-access-security**. Nele foi configurado um custom TCP e All Traffic. Ou seja, todas as permissões concedidas nessa tarefa.

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



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.



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/>. Acesso em: 24 de jun. de 2020.