

Construção de infraestrutura com AWS CDK

Esse é o tutorial a ser utilizado no workshop do AWS User Group São Paulo sobre a criação da infraestrutura de um cluster com o AWS ECS e Fargate, utilizando o AWS Cloud Development Kit.

O objetivo desse tutorial é guiar o participante durante o workshop, fornecendo os passos necessários para serem executados, como forma de concretização dos conceitos a serem ensinados em cada sessão. Dito isso, é importante ressaltar que as explicações teóricas serão dadas no início de cada sessão, bem como detalhes do quê e como será construída cada parte da infraestrutura. Por isso, atente-se à explicação do instrutor no início de cada sessão, para que você consiga executar os passos aqui descritos.

1) Sessão 1

Essa sessão trata-se de uma preparação adicional do ambiente de desenvolvimento, bem como o início da construção da infraestrutura proposta, com a criação da VPC e do cluster no ECS.

1.1) Criação do usuário no IAM

Para fazer o deployment da infraestrutura criada com o projeto do AWS CDK, que é o objetivo desse tutorial, é necessário primeiro criar um usuário no AWS IAM com permissões específicas. As credenciais desse usuário serão utilizadas no tópico seguinte. Para começar, abra o console da AWS e vá até o serviço IAM.

Dentro desse console, clique no menu lateral esquerdo, na opção `Access Management -> Users`, como na figura a seguir:

Identity and Access Management (IAM)

Dashboard

▼ Access management

User groups

Users

Roles

Add user

Delete user

Find users by username or

☐

User name ▼

☐

aws-cdk

Nessa tela, clique no botão **Add user**.

Na primeira tela de criação do usuário, digite um nome que você deseja e marque a opção **Programmatic access**, como na figura a seguir:

Add user

1

2

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

workshop_awsugsp

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type*



Programmatic access

Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.



AWS Management Console access

Enables a **password** that allows users to sign-in to the AWS Management Console.

Em seguida clique em **Next**. Nessa tela, selecione a opção **Attach existing policies directly** e escolha a política de nome **Administrator Access**:

Add user

1

2

3

Set permissions



Add user to group



Copy permissions from existing user



Attach existing policies directly

Create policy

Filter policies ▾

Q Search

Show all

	Policy name ▾	Type	Used as
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	Permissions policy (1)

Continue clicando no botão **Next**, até a última tela para a criação efetiva do usuário. Depois que o usuário for criado, copie os dados de sua credencial de acesso, como marcado na figura a seguir:

User	Access key ID	Secret access key
workshop_awsugsp	AKIA5Y46KVR5O4YZHVAD	***** Show

- Created user workshop_awsugsp
- Attached policy AdministratorAccess to user workshop_awsugsp
- Created access key for user workshop_awsugsp

Essa credencial será utilizada no tópico seguinte

1.2) Configuração do AWS CLI com as credenciais do usuário criado no IAM

Para configura o AWS CLI da sua máquina de desenvolvimento, é necessário fornecer as credenciais do usuário criado no IAM no tópico anterior. Para isso, abra um terminal e digite o seguinte comando:

```
1 | aws configure
```

No primeiro parâmetro solicitado, informe a `Access Key ID` do usuário criado no IAM. Em seguida, forneça a `Secret Access Key`. O terceiro parâmetro é a região desejada, que deve ser `us-east-1`. O último parâmetro é o formato da saída dos comandos, que pode ser configurado como `JSON`.


1.3) Criação do projeto CDK

Depois de tudo estar configurado, abra um terminal em uma pasta de sua preferência e crie um projeto com o AWS CDK, executando os seguintes comandos:

```
1 | mkdir aws_ecs_fargate_cdk
2 | cd aws_ecs_fargate_cdk/
3 | cdk init app --language java
```

Os dois primeiros comandos são para criar uma pasta e navegar para dentro dela, respectivamente. O terceiro comando cria efetivamente o projeto com o AWS CDK. O resultado esperado deve ser semelhante ao trecho a seguir:

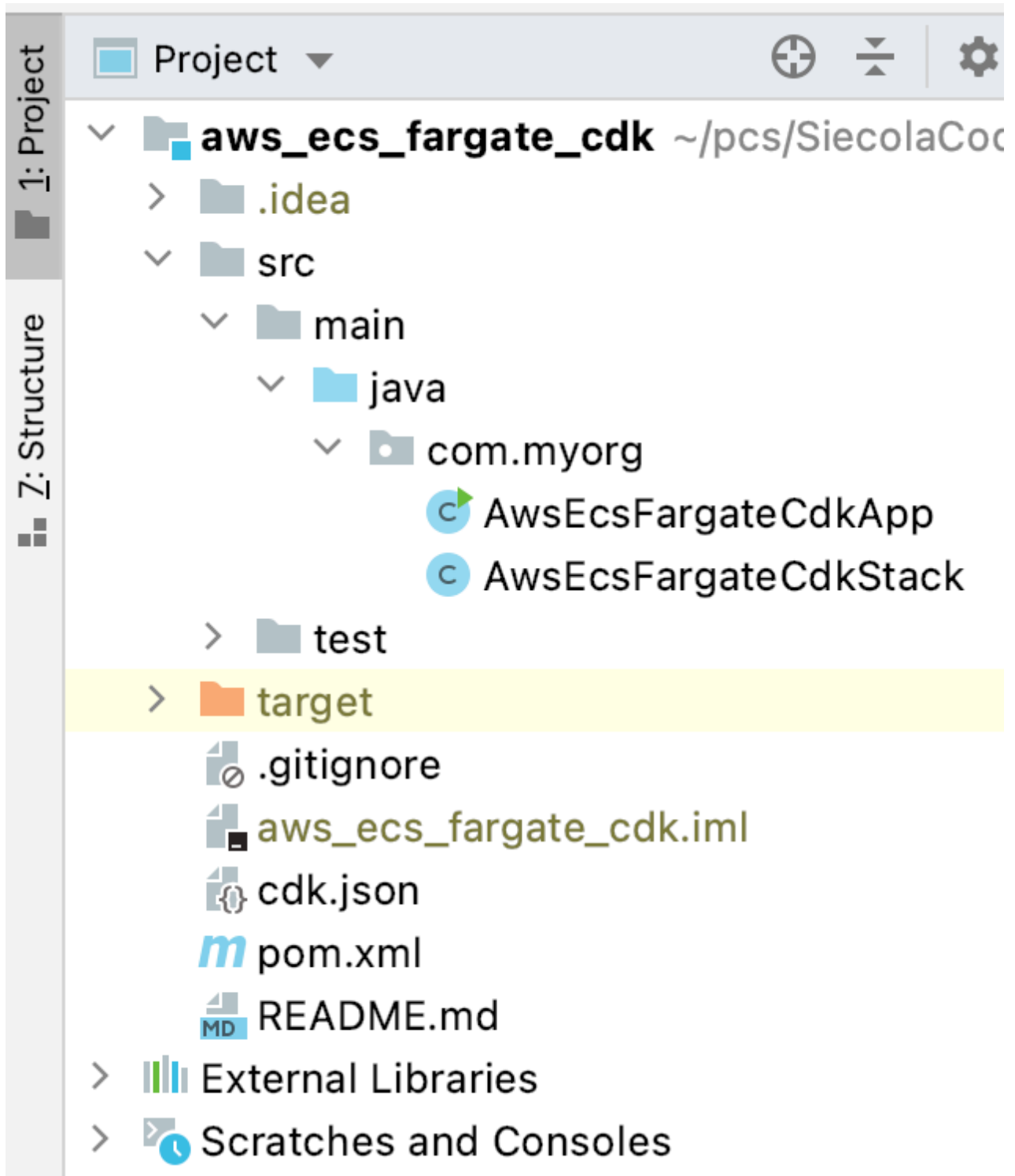
```
1 | ## Useful commands
2 |
3 | * `mvn package`      compile and run tests
4 | * `cdk ls`          list all stacks in the app
5 | * `cdk synth`       emits the synthesized CloudFormation template
```

```
6  * `cdk deploy`      deploy this stack to your default AWS account/region
7  * `cdk diff`        compare deployed stack with current state
8  * `cdk docs`        open CDK documentation
9
10 Enjoy!
11
12 Initializing a new git repository...
13 Executing 'mvn package'
14  All done!
```

Agora o projeto **Maven** que foi criado com esse comando pode ser aberto no IntelliJ IDEA.

Dentro do IntelliJ IDEA, basta abrir o projeto, no mesmo local onde está o arquivo `pom.xml`.

A estrutura do projeto no IntelliJ IDEA deve ficar semelhante à figura a seguir:



Dentro desse projeto há um arquivo de nome `pom.xml`, que gerencia as bibliotecas utilizadas por ele. Abra esse arquivo e vá até a sessão `dependencies` e adicione as seguintes bibliotecas:

```
1 <dependency>
2   <groupId>software.amazon.awscdk</groupId>
3   <artifactId>ecs</artifactId>
4   <version>${cdk.version}</version>
5 </dependency>
6
7 <dependency>
8   <groupId>software.amazon.awscdk</groupId>
```

```

9      <artifactId>ecs-patterns</artifactId>
10     <version>${cdk.version}</version>
11 </dependency>
12
13 <dependency>
14     <groupId>software.amazon.awscdk</groupId>
15     <artifactId>dynamodb</artifactId>
16     <version>${cdk.version}</version>
17 </dependency>

```

1.4) Criação da VPC

1.4.1) Construção da classe para criar a VPC

Para começar com a criação da VPC, dentro do projeto do CDK, crie uma nova classe, com o mesmo *template* da classe `AwsEcsFargateCdkStack` do projeto que foi criado. Para facilitar, você pode copiar essa classe no IntelliJ e criar uma outra específica para a VPC, de nome `VpcStack`, como trecho a seguir:

```

1  import software.amazon.awscdk.core.Construct;
2  import software.amazon.awscdk.core.Stack;
3  import software.amazon.awscdk.core.StackProps;
4
5  public class VpcStack extends Stack {
6      public VpcStack(final Construct scope, final String id) {
7          this(scope, id, null);
8      }
9
10     public VpcStack(final Construct scope, final String id, final
StackProps props) {
11         super(scope, id, props);
12
13         // The code that defines your stack goes here
14     }
15 }

```

Nessa classe, comece criando um atributo privado do tipo `Vpc` e um *getter* pra ele, logo após a declaração da classe, como no trecho a seguir:

```

1  public class VpcStack extends Stack {
2      private final Vpc vpc;
3
4      public Vpc getVpc() {
5          return vpc;
6      }
7

```

Agora dentro do construtor da classe, logo após a instrução `super(scope, id, props);`, crie de fato a Vpc, atribuindo a esse atributo privado:

```
1  this.vpc = Vpc.Builder.create(this, "Vpc01")
2      .maxAzs(3)
3      .build();
```

Veja como deve ficar a classe completa:

```
1  package com.myorg;
2
3  import software.amazon.awscdk.core.Construct;
4  import software.amazon.awscdk.core.Stack;
5  import software.amazon.awscdk.core.StackProps;
6  import software.amazon.awscdk.services.ec2.Vpc;
7
8  public class VpcStack extends Stack {
9      private final Vpc vpc;
10
11     public Vpc getVpc() {
12         return vpc;
13     }
14
15     public VpcStack(final Construct scope, final String id) {
16         this(scope, id, null);
17     }
18
19     public VpcStack(final Construct scope, final String id,
20                     final StackProps props) {
21         super(scope, id, props);
22
23         this.vpc = Vpc.Builder.create(this, "Vpc01")
24             .maxAzs(3)
25             .build();
26     }
27 }
```

1.4.2) Deployment da VPC

Para que a Vpc seja de fato criada, é necessário criar uma instância de `VpcStack` na classe `AwsEcsFargateCdkApp`, como no trecho a seguir:

```

1 public class AwsEcsFargateCdkApp {
2     public static void main(final String[] args) {
3         App app = new App();
4
5         VpcStack vpcStack = new VpcStack(app, "Vpc");
6
7         app.synth();
8     }
9 }

```

Agora abra o terminal dentro da pasta desse projeto, no mesmo local onde está o arquivo `pom.xml`, como no **exemplo** a seguir:

```

1 $ ls -la
2 total 48
3 drwxr-xr-x  11 paulosiecola  staff   352 May  7 09:21 .
4 drwxr-xr-x   3 paulosiecola  staff    96 May  5 15:18 ..
5 drwxr-xr-x  12 paulosiecola  staff   384 May  6 14:47 .git
6 -rw-r--r--   1 paulosiecola  staff   125 May  5 15:18 .gitignore
7 drwxr-xr-x  12 paulosiecola  staff   384 May  7 09:16 .idea
8 -rw-r--r--   1 paulosiecola  staff   675 May  5 15:18 README.md
9 -rw-r--r--   1 paulosiecola  staff  7915 May  7 09:21
   aws_ecs_fargate_cdk.iml
10 -rw-r--r--   1 paulosiecola  staff   490 May  5 15:18 cdk.json
11 -rw-r--r--   1 paulosiecola  staff  2848 May  6 14:49 pom.xml
12 drwxr-xr-x   4 paulosiecola  staff   128 May  5 15:18 src
13 drwxr-xr-x  10 paulosiecola  staff   320 May  5 15:19 target

```

Dentro dessa pasta é onde todos os comandos do AWS CDK serão realizados.

Como exemplo, digite o comando `cdk list`, que irá exibir todas as stacks que tiverem disponíveis no projeto.

Para fazer o deployment dessa Vpc que foi criada na classe `vpcStack`, basta executar o comando `cdk deploy Vpc`:

```

Paulos-MacBook-Pro:aws_ecs_fargate_cdk paulosiecola$ cdk deploy Vpc
Vpc: deploying...
Vpc: creating CloudFormation changeset...
[REDACTED] .....] (9/25)

9:38:17 AM | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | Vpc
9:38:43 AM | CREATE_IN_PROGRESS | AWS::EC2::VPCGatewayAttachment | Vpc01/VPCGW
9:38:43 AM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Vpc01/PrivateSubnet2/Su
9:38:43 AM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Vpc01/PrivateSubnet1/Su
9:38:43 AM | CREATE_IN_PROGRESS | AWS::EC2::Subnet | Vpc01/PublicSubnet2/Sub

```

Veja que o comando `cdk deploy` pode receber uma lista de nomes de stacks, que por enquanto só recebe a única que existe no projeto, de nome `Vpc`.

Esse comando dá início ao processo de deployment da Vpc, que poderá ser observado nos tópicos a seguir.

1.4.3) Visualização da stack que criou a VPC no console da AWS

Para visualizar a stack que criou a Vpc, abra o console do CloudFormation na AWS. Ela deverá aparecer como a figura a seguir:

CloudFormation > Stacks			
Stacks (1)			
Q Filter by stack name			
	Stack name	Status	Created time
<input type="radio"/>	Vpc	✔ CREATE_COMPLETE	2021-05-07 09:38:11 UTC-0300

1.4.4) Visualização da VPC no console da AWS

E para visualizar a VPC de fato, abra o console de VPCs na AWS. Ela deverá aparecer com o nome `Vpc/Vpc01`, como na figura a seguir:

Your VPCs (2) Info				
Q Filter VPCs				
<input type="checkbox"/>	Name ▼	VPC ID ▼	State ▼	IPv4 CIDR
<input type="checkbox"/>	Vpc/Vpc01	vpc-0630fb23b8e7334...	✔ Available	10.0.0.0/16

1.5) Criação do cluster no ECS

1.5.1) Construção da classe para criar o cluster

Para a construção da classe para criar o cluster, execute o mesmo procedimento descrito no item 1.4.1 desse tutorial, ou seja, crie uma nova classe, com o mesmo *template* da classe `AwsEcsFargateCdkStack` do projeto que foi criado, agora de nome `ClusterStack`:

```
1 import software.amazon.awscdk.core.Construct;
2 import software.amazon.awscdk.core.Stack;
3 import software.amazon.awscdk.core.StackProps;
4
5 public class ClusterStack extends Stack {
6     public ClusterStack(final Construct scope, final String id) {
```

```

7         this(scope, id, null);
8     }
9
10    public ClusterStack(final Construct scope, final String id, final
StackProps props) {
11        super(scope, id, props);
12
13        // The code that defines your stack goes here
14    }
15 }

```

Agora crie um atributo privado do tipo `cluster`, assim como seu *getter*, para expor esse objeto que será criado, como no trecho a seguir:

```

1 public class ClusterStack extends Stack {
2     private final Cluster cluster;
3
4     public Cluster getCluster() {
5         return cluster;
6     }

```

Ajuste os dois métodos/construtores para ambos receberem um novo parâmetro, que é a Vpc onde o cluster será criado:

```

1 public ClusterStack(final Construct scope, final String id, Vpc vpc) {
2     this(scope, id, null, vpc);
3 }
4
5 public ClusterStack(final Construct scope, final String id,
6                     final StackProps props, Vpc vpc) {
7     super(scope, id, props);

```

E finalmente, crie o cluster dentro da Vpc, no segundo construtor, como no trecho a seguir:

```

1 this.cluster = Cluster.Builder.create(this, id)
2     .clusterName("cluster-01")
3     .vpc(vpc)
4     .build();

```

1.5.2) Deployment do cluster ECS

Agora é necessário criar uma instância de `ClusterStack`, logo após o que foi feito com `VpcStack` na classe `AwsEcsFargateCdkApp`, porém agora passando a Vpc como parâmetro, como no trecho a seguir:

```
1 ClusterStack clusterStack = new ClusterStack(app,  
2     "Cluster", vpcStack.getVpc());  
3 clusterStack.addDependency(vpcStack);
```

Veja que na última linha, também foi adicionada a dependência da stack `vpcStack` em `clusterStack`. Isso significa que uma árvore de dependência entre as stacks começa a ser criada.

De volta ao terminal, digite o comando `cdk list` para listar todas as stacks. A stack `clusterStack` deve agora aparecer na lista.

Um comando interessante também de ser utilizado é o `cdk diff`. Ele mostra as diferenças entre o que está na sua conta da AWS e no seu projeto:

```
Paulos-MacBook-Pro:aws_ecs_fargate_cdk paulosiecola$ cdk diff
Stack Vpc
There were no differences
Stack Cluster
Conditions
[+] Condition CDKMetadata/Condition CDKMetadataAvailable: {"Fn::Equals": [{"Ref": "AWS::Region"}, "ap-northeast-2"]}, {"Fn::Equals": [{"Ref": "AWS::Region"}, "ap-southeast-2"]}, {"Fn::Equals": [{"Ref": "AWS::Region"}, "eu-central-1"]}], {"Fn::Or": [{"Ref": "AWS::Region"}, "eu-west-2"]}, {"Fn::Equals": [{"Ref": "AWS::Region"}, "us-east-1"]}, {"Fn::Equals": [{"Ref": "AWS::Region"}, "us-west-2"]}]]}
```

```
[+] AWS::ECS::Cluster Cluster ClusterEB0386A7
```

Veja que nesse caso, não há nenhuma diferença a ser aplicada na stack `vpc`, pois foi feito o deployment dela há pouco. Mas na stack `cluster` existe, obviamente, pois ainda não foi feito o deployment dela.

E para fazer o deployment da nova stack do cluster, basta digitar o comando `cdk deploy`

`Cluster`:

```
[Paulos-MacBook-Pro:aws_ecs_fargate_cdk paulosiecola$ cdk deploy Cluster
Including dependency stacks: Vpc
Vpc
Vpc: deploying...
```

☒ Vpc (no changes)

Stack ARN:

```
arn:aws:cloudformation:us-east-1:946835467386:stack/Vpc/159a3ca0-af31-11
```

```
Cluster: deploying...
Cluster: creating CloudFormation changeset...
```

[REDACTED] (3/3)

Veja que, mesmo sem comandar o deployment da VPC, o CDK verifica se algo deve ser feito com a stack de nome `vpc`, pois ela é uma dependência da stack `cluster`.

1.5.3) Visualização da stack que criou o cluster no console da AWS

Da mesma forma, é possível observar a stack de nome `cluster` no console do CloudFormation:

CloudFormation > Stacks			
Stacks (2)			
<input type="text" value="Filter by stack name"/>			
	Stack name	Status	Created time
<input type="radio"/>	Cluster	✔ CREATE_COMPLETE	2021-05-07 11:25:11 UTC-0300
<input type="radio"/>	Vpc	✔ CREATE_COMPLETE	2021-05-07 09:38:11 UTC-0300

1.5.4) Visualização do cluster no console da AWS

E para visualizar de fato o cluster que foi criado, abra o console do ECS:

☐ New ECS Experience
[Tell us what you think](#)

Amazon ECS

Clusters

Task Definitions

Account Settings

Amazon EKS

Clusters

Amazon ECR

Repositories

AWS Marketplace

Discover software

Subscriptions ↗

Clusters

An Amazon ECS cluster is a regional grouping of or instance type.

For more information, see the [ECS documentation](#).

Create Cluster

Get Started

View ☒ list

☐ card

cluster-01 >

CloudWatch monitoring

✔ Default Monitoring

2) Sessão 2

Essa sessão se dedica à criação do serviço para hospedar a aplicação no ECS utilizando o AWS Fargate. Na verdade, muitos outros recursos serão criados na AWS, como será explicado pelo instrutor ao final dessa sessão.

2.1) Criação do serviço no ECS com Fargate

2.1.1) Construção da classe para criar o serviço no ECS

Para começar, crie uma nova classe de nome `ServiceStack`, da mesma forma como foi feito na sessão anterior. Essa classe terá que receber o cluster que foi criado, como parâmetro, por isso já adicione ele nos construtores:

```
1  import software.amazon.awscdk.core.Construct;
2  import software.amazon.awscdk.core.Stack;
3  import software.amazon.awscdk.core.StackProps;
4  import software.amazon.awscdk.services.ecs.Cluster;
5
6  public class ServiceStack extends Stack {
7      public ServiceStack(final Construct scope, final String id, Cluster
cluster) {
8          this(scope, id, null, cluster);
9      }
10
11     public ServiceStack(final Construct scope, final String id,
12                         final StackProps props, Cluster cluster) {
13         super(scope, id, props);
14
15         // The code that defines your stack goes here
16     }
17 }
```

O restante do código a ser construído nesse tópico deverá ser colocado no lugar do comentário do trecho anterior.

A aplicação a ser hospedada requer a passagem da região da AWS como parâmetro. E para tornar o exemplo mais interessante, será mostrado como isso pode ser feito através de um atributo de entrada da stack e como isso pode ser feito com o CDK.

Para que a stack então receba um parâmetro, pode-se utilizar o `CfnParameter`, como no trecho a seguir:

```

1 CfnParameter awsRegion = CfnParameter.Builder.create(this, "awsRegion")
2     .type("String")
3     .description("The AWS Region")
4     .build();

```

Esse parâmetro então poderá ser passado para a aplicação, como uma variável de ambiente, como no trecho a seguir:

```

1 Map<String, String> envVariables = new HashMap<>();
2 envVariables.put("AWS_REGION", awsRegion.getValueAsString());

```

Para que os logs da aplicação apareçam no CloudWatch dentro de um grupo específico, crie esta configuração a seguir:

```

1 LogDriver logDriver = LogDriver.awsLogs(AwsLogDriverProps.builder()
2     .logGroup(LogGroup.Builder.create(this, "Service01LogGroup")
3         .logGroupName("Service01")
4         .removalPolicy(RemovalPolicy.DESTROY)
5         .build())
6     .streamPrefix("Service01")
7     .build());

```

Agora pode ser criada a definição da tarefa que será executada pelo serviço no ECS, utilizando como parâmetros a imagem Docker da aplicação, as configurações de logs e a variável de ambiente criadas anteriormente:

```

1 ApplicationLoadBalancedTaskImageOptions task =
2     ApplicationLoadBalancedTaskImageOptions.builder()
3     .containerName("ddb_sqs_demo")
4     .image(ContainerImage.fromRegistry("siecola/aws-ddb-sqs-java-
5         demo:1.0.0"))
6     .containerPort(8080)
7     .logDriver(logDriver)
8     .environment(envVariables)
9     .build();

```

Com isso em mãos, pode ser criar o serviço do tipo Fargate para utilizar o Application Load Balancer, com algumas definições adicionais de CPU, RAM e quantidade de instâncias da aplicação a serem executadas:

```

1 ApplicationLoadBalancedFargateService service01 =
2     ApplicationLoadBalancedFargateService.Builder
3     .create(this, "ALB01")
4     .serviceName("service-01")
5     .cluster(cluster)
6     .cpu(512)
7     .memoryLimitMiB(1024)
8     .desiredCount(2)
9     .listenerPort(8080)
10    .taskImageOptions(task)
11    .publicLoadBalancer(true)
12    .build();

```

Cada instância da aplicação deve ser monitorada pelo *target group*, para saber se ela está saudável ou não. Felizmente o projeto dessa aplicação já foi criado com um *endpoint* específico para isso, que fica em `/actuator/health`. Isso então pode ser configurado, como no trecho a seguir:

```

1 service01.getTargetGroup().configureHealthCheck(new HealthCheck.Builder()
2     .path("/actuator/health")
3     .port("8080")
4     .healthyHttpCodes("200")
5     .build());

```

Para finalizar, pode ser adicionada as configurações de *auto scaling* do serviço do ECS, garantindo que sempre existam entre 2 a 4 instâncias da aplicação, dependendo do consumo de CPU que todo o serviço tiver:

```

1 ScalableTaskCount scalableTaskCount = service01.getService()
2     .autoScaleTaskCount(EnableScalingProps.builder()
3     .minCapacity(2)
4     .maxCapacity(4)
5     .build());
6
7 scalableTaskCount.scaleOnCpuUtilization("Service01AutoScaling",
8     CpuUtilizationScalingProps.builder()
9     .targetUtilizationPercent(50)
10    .scaleInCooldown(Duration.seconds(60))
11    .scaleOutCooldown(Duration.seconds(60))
12    .build());

```

2.1.2) Deployment do serviço no ECS

Na classe `AwsEcsFargateCdkApp`, a instância de `ServiceStack` pode ser criada, logo após a do cluster, afinal, ela depende da criação de `clusterStack`, como pode ser visto no trecho a seguir:

```
1 ServiceStack serviceStack = new ServiceStack(app, "Service",
2     clusterStack.getCluster());
3 serviceStack.addDependency(clusterStack);
```

Para fazer o deployment desse serviço, volte ao terminal e digite o seguinte comando:

```
cdk deploy Service --parameters Service:awsRegion=us-east-1 --require-approval
never:
```

Esse comando irá iniciar o processo de deployment da stack `ServiceStack`, passando a região `us-east-1` como parâmetro para ela. Além disso, o atributo `require-approval` com o valor `never` é útil para aceitar todas as perguntas que o CDK poderia fazer, como por exemplo a abertura de portas no *security group* no qual a aplicação estará sujeita.

O processo de deployment dessa stack pode demorar alguns minutos.

Ao final do processo o console do CDK irá informar o endereço do DNS do ALB que foi criado, como no exemplo a seguir:

✓ Service

Outputs:

Service.ALB01LoadBalancerDNS71443EB4 = Servi-ALB01-142IG05RPZ3CE-1140578466.us-east-1.elb.amazonaws.com
Service.ALB01ServiceURL8B1A7735 = <http://Servi-ALB01-142IG05RPZ3CE-1140578466.us-east-1.elb.amazonaws.com>

Stack ARN:

arn:aws:cloudformation:us-east-1:946835467386:stack/Service/282217a0-af66-11eb-ac9d-0e35b405c3d1

O endereço que aparecer no seu terminal será utilizado para testar a aplicação, ainda nessa sessão.

2.1.3) Visualização da stack que criou o cluster no console da AWS

No console do CloudFormation, a stack criada pode ser observada. Um ponto a ser destacado aqui é a região que foi passada como parâmetro de entrada:

Service

[Stack info](#)[Events](#)[Resources](#)[Outputs](#)[Parameters](#)[Template](#)

Parameters (1)

Key**Value**

awsRegion

us-east-1

2.1.4) Visualização do serviço no ECS no console da AWS

Pra visualizar o serviço criado no ECS, abra seu console na AWS e clique no cluster que foi criado. Dentro dele está o serviço criado nessa sessão:

Service

[Stack info](#)[Events](#)[Resources](#)[Outputs](#)[Parameters](#)[Template](#)

Parameters (1)

Key**Value**

awsRegion

us-east-1

Na verdade ainda há muita informação a ser analisada e muitos recursos a serem observados, como será mostrado pelo instrutor ao final da sessão.

2.2) Teste da aplicação hospedada na AWS

Para testar a aplicação, copie o endereço do DNS que o seu console do CDK exibiu, ao final do processo de deployment do serviço que foi criado no tópico anterior.

Com esse endereço em mãos, abra o Postman, ou outro cliente REST de sua preferência, e faça uma requisição GET no seguinte endereço:

```
<endereço do DNS>:8080/actuator/health
```

A seguinte resposta deve ser obtida, com o código HTTP 200 OK:

```
1 {  
2   "status": "UP"  
3 }
```

Isso significa que a aplicação está de pé e sendo executada pelo ECS.

Acompanhe as outras informações que serão apresentadas pelo instrutor.

3) Sessão 3

O objetivo dessa sessão é criar a tabela do DynamoDB que será utilizada pela aplicação para salvar os produtos.

3.1) Criação da tabela no DynamoDB

3.1.1) Construção da classe para criar a tabela no DynamoDB

A tabela do DynamoDB será criada em sua própria stack, então para isso crie uma nova classe chamada `DdbStack`, da mesma forma como foi feito nas sessões anteriores, já com o atributo a ser exposto pelo seu *getter*, para expor essa tabela para a stack do serviço de ECS, que será necessário na próxima sessão:

```
1 import software.amazon.awscdk.core.Construct;  
2 import software.amazon.awscdk.core.Stack;  
3 import software.amazon.awscdk.core.StackProps;  
4 import software.amazon.awscdk.services.dynamodb.Table;  
5  
6 public class DdbStack extends Stack {  
7     private final Table productEventsDdb;  
8  
9     public Table getProductEventsDdb() {  
10         return productEventsDdb;  
11     }  
12 }
```

```

13     public DdbStack(final Construct scope, final String id) {
14         this(scope, id, null);
15     }
16
17     public DdbStack(final Construct scope, final String id, final
StackProps props) {
18         super(scope, id, props);
19
20         // The code that defines your stack goes here
21     }
22 }

```

Como explicado no início dessa sessão, a tabela, de nome `products`, será configurada com as seguintes informações:

- Modo de cobrança on-demand;
- Chave composta (`pk` e `sk`);
- Remoção do recurso ao se apagar a stack.

Isso tudo pode ser feito no CDK de forma muito simples, como mostra o trecho a seguir, que deve ser colocado no lugar do comentário nessa classe:

```

1  this.productEventsDdb = Table.Builder.create(this, "ProductsDb")
2      .tableName("products")
3      .billingMode(BillingMode.PAY_PER_REQUEST)
4      .partitionKey(Attribute.builder()
5          .name("pk")
6          .type(AttributeType.STRING)
7          .build())
8      .sortKey(Attribute.builder()
9          .name("sk")
10         .type(AttributeType.STRING)
11         .build())
12      .removalPolicy(RemovalPolicy.DESTROY)
13      .build();

```

Veja que cada configuração citada anteriormente aparece nesse trecho.

3.1.2) Deployment da tabela no DynamoDB

Para fazer o deployment da tabela no DynamoDB, vá até a classe `AwsEcsFargateCdkApp` e crie uma instância de `DdbStack`, antes da criação da instância de `ServiceStack`, como no trecho a seguir:

```

1  DdbStack ddbStack = new DdbStack(app, "Ddb");

```

Na sessão seguinte ficará mais claro porque ela deve ser criada antes de `ServiceStack`.

Novamente, de volta ao terminal do CDK, digite o comando `cdk deploy Ddb` para fazer o deployment da tabela.

3.1.3) Visualização da tabela do DynamoDB no console da AWS

No console do DynamoDB na AWS, é possível observar a tabela que foi criada, com todas as configurações que foram feitas:

products [Close](#)

Overview	Items	Metrics	Alarms	Capacity	Indexes	Global Tables	Backup
----------	-------	---------	--------	----------	---------	---------------	--------

Recent alerts

No CloudWatch alarms have been triggered for this table.

Kinesis data stream details

Use Amazon Kinesis Data Streams for DynamoDB to capture item-level changes in your table as a Kinesis data stream. [Learn more](#)

Stream enabled No

Stream name -

[Manage streaming to Kinesis](#)

Table details

Table name	products
Primary partition key	pk (String)
Primary sort key	sk (String)
Point-in-time recovery	DISABLED Enable
Encryption Type	DEFAULT Manage Encryption
KMS Master Key ARN	Not Applicable
Encryption Status	
CloudWatch Contributor Insights	DISABLED Manage Contributor Insights
Time to live attribute	DISABLED Manage TTL
Table status	Active
Creation date	May 7, 2021 at 5:06:27 PM UTC-3
Read/write capacity mode	On-Demand

Mas... Essa tabela não pode ser acessada pela aplicação ainda, como será explicado na sessão seguinte.

4) Sessão 4 - Desafio

Como explicado no início dessa sessão, a aplicação não possui permissão para acessar a tabela.

4.1) Atribuição da permissão do serviço para acessar a tabela do DynamoDB

Basicamente o papel assumido pela tarefa em execução no ECS não possui a política que dá acesso a essa tabela. Isso deve ser feito no projeto do CDK, no momento da criação do serviço do ECS.

4.1.1) Instruções para realização da tarefa

O que deve ser feito, dentro da classe `ServiceStack`, em suma, é:

- Buscar a definição da tarefa (*task definition*) que define a execução do serviço no ECS. Isso pode ser obtido através do objeto do tipo `ApplicationLoadBalancedFargateService` que foi criado nessa classe, através do método `getTaskDefinition`. Para maiores informações, consulte esse [link](#);
- Dentro da definição da tarefa, é possível obter o papel que a tarefa assume, através do método `getTaskRole`. Para maiores informações, consulte esse [link](#);
- Esse papel então deve ser passado como parâmetro para o método `grantReadWriteData` da tabela que foi criada. Isso fará com que o serviço, que roda sob as políticas desse papel, tenha permissão de leitura e escrita nessa tabela. Para maiores informações, consulte esse [link](#).

Esses passos devem ser feitos no final do construtor da classe `ServiceStack`, logo após a configuração de *auto scaling* do serviço.

Obviamente, a tabela criada na stack `DdbStack` deverá agora ser um parâmetro de entrada para a stack `ServiceStack`. Da mesma forma, a stack `ServiceStack` depende de `DdbStack`.

4.1.2) Visualização das alterações a serem feitas na stack

Antes de fazer o deployment das alterações, digite o comando `cdk diff` no terminal do CDK. Você deverá observar as seguintes alterações a serem feitas na stack `Service`:

Stack Service

IAM Statement Changes

	Resource	Effect	Action
+	<code>{"Fn::ImportValue":"Ddb:ExportsOutputFnGetAttProductsDb81C13EAFArn0F4B2516"}</code>	Allow	<code>dynamodb:BatchGetItem</code> <code>dynamodb:BatchWriteItem</code> <code>dynamodb:ConditionCheckItem</code> <code>dynamodb>DeleteItem</code> <code>dynamodb:GetItem</code> <code>dynamodb:GetRecords</code> <code>dynamodb:GetShardIterator</code> <code>dynamodb:PutItem</code> <code>dynamodb:Query</code> <code>dynamodb:Scan</code> <code>dynamodb:UpdateItem</code>

Veja que estão sendo adicionadas as permissões de leitura e escrita, dentre outras, ao papel da tarefa do serviço, para acesso a tabela do DynamoDB.

4.1.3) Deployment das alterações da stack do serviço

Para fazer o deployment dessa alterações, repita o comando a seguir:

```
1 cdk deploy Service --parameters Service:awsRegion=us-east-1 --require-approval never
```

Basta fazer o deployment somente da stack `Service`, pois somente ela foi alterada.

4.1.4) Visualização do papel assumido pelo serviço e suas permissões

No papel assumido pela tarefa, é possível agora ver que há uma permissão para acessar a tabela `products` do DynamoDB:

PermissionsTrust relationshipsTagsAccess AdvisorRevoke sessions

▼ Permissions policies (1 policy applied)

Attach policies

Policy name ▼

▼ ALB01TaskDefTaskRoleDefaultPolicy2F92956B

Policy summary{} JSONEdit policy

Q Filter

Service ▼	Access level	Resource
Allow (1 of 281 services) Show remaining 280		
DynamoDB	Limited: Read, Write	TableName string like products

4.1.5) Teste da aplicação acessando a tabela do DynamoDB

Para testar que tudo está funcionando, abra o Postman, ou seu cliente REST favorito, e acesse a API de produtos:

- Para listar todos os produtos:
 - Endereço: `/api/products`
 - Método: GET
- Para criar um novo produto:
 - Endereço: `/api/products/`
 - Método: POST
 - Body:

```
1 {  
2   "username": "matilde",  
3   "name": "Product1",  
4   "code": "COD1",  
5   "model": "Model1",  
6   "price": 10  
7 }
```

- Para buscar todos os produtos por um username:
 - Endereço: `/api/products/{username}`
 - Método: GET
- Para buscar um produto específico de um username pelo seu código:
 - Endereço: `/api/products/{username}/{code}`
 - Método: GET
- Para alterar um produto específico de um username pelo seu código:
 - Endereço: `/api/products/{username}/{code}`
 - Método: PUT
 - Body:

```
1 {  
2   "username": "matilde",  
3   "name": "Product1",  
4   "code": "COD1",  
5   "model": "Model1",  
6   "price": 10  
7 }
```

- Para excluir um produto específico de um username pelo seu código:
 - Endereço: `/api/products/{username}/{code}`
 - Método: DELETE

5) Limpeza de todas as stacks

Se desejar limpar todas as stacks e todos os recursos criados nesse tutorial, execute o seguinte comando no terminal do CDK:

```
1 | cdk destroy --all
```

No futuro, se desejar fazer o deployment de toda a infraestrutura criada aqui, basta executar o seguinte comando:

```
1 | cdk deploy --all --parameters Service:awsRegion=us-east-1 --require-approval  
never
```