

# Criação de microsserviços e um pipeline CI/CD com a AWS

---

## Sumário

---

[Visão geral e objetivos do projeto](#)

[O ambiente do laboratório e monitoramento do seu orçamento](#)

[Restrições de serviços da AWS](#)

[Cenário](#)

[Requisitos da solução](#)

[Dicas do projeto do laboratório](#)

[Abordagem](#)

[Fase 1: planejamento do design e estimativa de custo](#)

[Tarefa 1.1: criar um diagrama de arquitetura](#)

[Tarefa 1.2: desenvolver uma estimativa de custo](#)

[Fase 2: análise da infraestrutura da aplicação monolítica](#)

[Tarefa 2.1: verificar a disponibilidade da aplicação monolítica.](#)

[Tarefa 2.2: testar o aplicativo web monolítico](#)

[Tarefa 2.3: analisar a forma de execução da aplicação monolítica](#)

[Fase 3: criar um ambiente de desenvolvimento e inserir o código em um repositório Git](#)

[Tarefa 3.1: criar um IDE do AWS Cloud9 como seu ambiente de trabalho](#)

[Tarefa 3.2: copiar o código da aplicação para seu IDE](#)

[Tarefa 3.3: criar diretórios de trabalho com código de iniciação para dois microsserviços](#)

[Tarefa 3.4: criar um repositório Git para o código de microsserviços e enviar o código para o CodeCommit](#)

[Fase 4: configuração da aplicação como dois microsserviços e como testá-los nos contêineres do Docker](#)

[Tarefa 4.1: ajustar as configurações do grupo de segurança da instância do AWS Cloud9;](#)

[Tarefa 4.2: modificar o código fonte do microsserviço \*customer\*](#)

[Tarefa 4.3: criar o Dockerfile do microsserviço \*customer\* e iniciar um contêiner de teste](#)

[Tarefa 4.4: modificar o código fonte do microsserviço \*employee\*](#)

[Tarefa 4.5: criar o Dockerfile do microsserviço \*employee\* e iniciar um contêiner de teste](#)

[Tarefa 4.6: ajustar a porta do microsserviço \*employee\* e reconstruir a imagem](#)

[Tarefa 4.7: inserir o código no CodeCommit](#)

[Fase 5: criar repositórios do ECR, um cluster do ECS, definições de tarefa e arquivos do AppSpec](#)

[Tarefa 5.1: criar repositórios do ECR e fazer upload das imagens do Docker](#)

[Tarefa 5.2: criar um cluster do ECS](#)

[Tarefa 5.3: criar um repositório no CodeCommit para armazenar arquivos de implantação](#)

[Tarefa 5.4: criar arquivos de definição de tarefa para cada microsserviço e registrá-los no Amazon ECS](#)

[Tarefa 5.5: criar arquivos AppSpec para CodeDeploy para cada microsserviço](#)

[Tarefa 5.6: atualizar arquivos e inseri-los no CodeCommit](#)

[Fase 6: criar grupos de destino e um Application Load Balancer](#)

[Tarefa 6.1: criar quatro grupos de destino](#)

[Tarefa 6.2: criar um grupo de segurança e um Application Load Balancer e configurar regras para roteamento do tráfego](#)

[Fase 7: criar dois serviços do Amazon ECS](#)

[Tarefa 7.1: criar o serviço do ECS para o microsserviço \*customer\*](#)

[Tarefa 7.2: criar o serviço do Amazon ECS para o microsserviço \*employee\*](#)

[Fase 8: configurar o CodeDeploy e o CodePipeline](#)

[Tarefa 8.1: criar uma aplicação no CodeDeploy e grupos de implantação](#)

[Tarefa 8.2: criar um pipeline para o microsserviço \*customer\*](#)

[Tarefa 8.3: testar um pipeline CI/CD para o microsserviço \*customer\*](#)

[Tarefa 8.4: criar um pipeline para o microsserviço \*employee\*](#)

[Tarefa 8.5: testar o pipeline CI/CD para o microsserviço \*employee\*](#)

[Tarefa 8.6: observar como o CodeDeploy modificou as regras do listener do balanceador de carga](#)

[Fase 9: ajustar o código do microsserviço para fazer com que um pipeline execute novamente](#)

[Tarefa 9.1: limitar o acesso ao microsserviço \*employee\*](#)

[Tarefa 9.2: ajustar o UI para o microsserviço \*employee\* e enviar a imagem atualizada para o Amazon ECR](#)

[Tarefa 9.3: confirmar se o pipeline \*employee\* foi executado e o microsserviço atualizado](#)

[Tarefa 9.4: testar o acesso ao microsserviço \*employee\*](#)

[Tarefa 9.5: dimensionar o microsserviço \*customer\*](#)

[Apêndice](#)

[Atualização de um contêiner de teste em execução no Cloud9](#)

[Reassociar grupos de destino com o balanceador de carga](#)

[Dicas de solução de problemas](#)

## Visão geral e objetivos do projeto

---

[Voltar ao sumário](#)

Neste projeto, o desafio proposto é usar ao menos 11 ofertas da AWS, incluindo algumas que podem ser novas para você, para criar microsserviços e uma solução de integração contínua e desenvolvimento contínuo (CI/CD). Em vários cursos da AWS Academy, os alunos concluíram laboratórios práticos. Você usou diferentes serviços e recursos da AWS para criar várias soluções.

Em muitas partes do projeto, orientações detalhadas *não* são fornecidas. Isso é intencional. Essas seções específicas do projeto têm o objetivo de desafiar você a praticar habilidades adquiridas em suas experiências de aprendizado anteriores a este projeto. Em alguns casos, você pode ser desafiado a usar recursos para aprender novas habilidades de forma independente.

Ao final deste projeto, você deverá ser capaz de fazer o seguinte:

- Reconhecer como um aplicativo web Node.js é codificado e implantado para executar e conectar a um banco de dados relacional onde os dados da aplicação estão armazenados.
- Criar um ambiente de desenvolvimento integrado ao AWS Cloud9 (IDE) e um repositório (repo) de código no qual armazenar o código da aplicação.
- Dividir a funcionalidade de uma aplicação monolítica em microsserviços com contêineres.
- Usar um registro de contêiner para armazenar e controlar as versões de imagens do Docker do microsserviço em contêineres.
- Criar repositórios de código para armazenar código fonte e ativos de implantação (CI/CD).
- Criar um cluster sem servidor para alcançar a otimização de custos e os requisitos de solução de dimensionamento.
- Configurar um Application Load Balancer e diversos grupos de destino para rotear o tráfego entre os microsserviços.
- Criar um pipeline de código para implantar contêineres de microsserviços a uma implantação de cluster azul/verde.
- Usar o pipeline do código e o repositório do código para CI/CD usando a iteração no design da aplicação.

## O ambiente do laboratório e monitoramento do seu orçamento

---

[Voltar ao sumário](#)

**Este ambiente é de longa duração.** Quando o cronômetro da sessão for zerado, a sessão terminará, mas todos os dados e os recursos criados na conta AWS serão retidos. Se iniciar uma nova sessão (por exemplo, no dia seguinte), você verá que seu trabalho ainda estará no ambiente de laboratório. Além disso, a qualquer momento antes de o cronômetro da sessão zerar, você poderá escolher **Iniciar laboratório** novamente para estender o tempo de sessão atual.

❗ **Importante:** monitore seu orçamento do laboratório na interface do laboratório. Quando uma sessão do laboratório está ativa, as informações mais recentes conhecidas do orçamento restante são exibidas na parte superior desta tela. O orçamento restante observado pode não refletir a atividade mais recente da conta. Se você exceder o orçamento do laboratório, sua conta será desativada e todo o andamento e os recursos serão perdidos. Se você exceder o orçamento, entre em contato com o instrutor para obter assistência.

Se você quiser redefinir o ambiente de laboratório para o estado original, antes de iniciar o laboratório, use a opção **Redefinir** acima destas instruções. Essa opção não redefinirá seu orçamento. ⚠ **Aviso:** se você redefinir o ambiente, excluirá *de forma permanente* tudo o que criou ou armazenou na conta da AWS.

## Restrições de serviços da AWS

Neste ambiente de laboratório, o acesso aos serviços e às ações dos serviços da AWS é restrito aos necessários para concluir o projeto. Você poderá encontrar erros se tentar acessar outros serviços ou executar ações além das descritas neste laboratório.

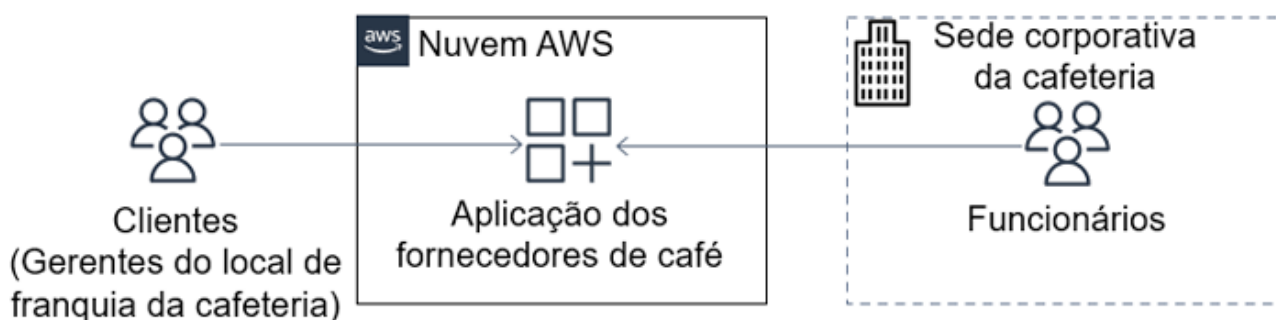
## Cenário

[Voltar ao sumário](#)

Os proprietários de uma cafeteria com muitas franquias notaram como as ofertas de café gourmet ficaram populares.

*Clientes* (os gerentes do local da franquia do café) não conseguem obter grãos de café o suficiente necessários para criar cappuccinos e lattes incríveis em suas respectivas cafeterias.

Enquanto isso, os *funcionários* da sede corporativa da cafeteria vêm enfrentando dificuldades para encontrar, de forma consistente, fornecedores de grãos de café da mais alta qualidade. Recentemente, os proprietários souberam que uma de suas fornecedoras de café favoritas quer vender a empresa. Os gerentes da sede corporativa da cafeteria aproveitaram a oportunidade de comprar a empresa. O fornecedor adquirido de café executa uma aplicação com listas de fornecedores em uma conta da AWS, conforme exibido na imagem a seguir.



A aplicação dos fornecedores de café atualmente é executada como aplicação *monolítica*. Há problemas de confiabilidade e desempenho. Essa é uma das razões pelas quais a sede corporativa da cafeteria contratou você. Neste projeto, você realizará tarefas associadas às funções de engenheiro de desenvolvimento de software (SDE), desenvolvedor de aplicação e engenheiro de suporte à nuvem.

Sua tarefa é dividir a aplicação monolítica em *microserviços*, para que você possa dimensionar os serviços de forma independente e alocar mais recursos de computação aos serviços que tiverem as maiores demandas, com o objetivo de evitar gargalos. Um design de microserviços também ajudará a evitar pontos únicos de falha, o que poderia derrubar a aplicação inteira em um design monolítico. Com serviços isolados uns dos

outros, se um microserviço ficar temporariamente indisponível, os outros microserviços podem continuar disponíveis.

Você também tem o desafio de desenvolver um pipeline CI/CD para implantar automaticamente atualizações ao cluster de produção que executa contêineres, usando uma estratégia de implantação azul/verde.

## Requisitos da solução

[Voltar ao sumário](#)

A solução deve atender aos seguintes requisitos:

- 1. Design: a solução precisa ter um diagrama de arquitetura.
- 2. Otimização de custos: a solução precisa incluir uma estimativa de custo.
- 3. Arquitetura baseada em microserviços: a solução tem de ser funcional e implantar uma arquitetura baseada em microserviço.
- 4. Portabilidade: a solução precisar ser portátil para que o código da aplicação não esteja ligado à execução em uma máquina host específica.
- 5. Dimensionamento/resiliência: a solução precisa fornecer a habilidade de aumentar a quantidade de recursos de computação dedicados a atender às solicitações à medida que os padrões de uso mudam. A solução também tem de usar uma lógica de roteamento confiável e dimensionável.
- 6. CI/CD automatizado: a solução precisa fornecer um pipeline CI/CD que possa ser invocado automaticamente quando o código for atualizado e enviado a um repositório de códigos com controle de versão.

## Dicas do projeto do laboratório

[Voltar ao sumário](#)

É útil, mas não obrigatório, ter conhecimento dos comandos bash do Linux.

**💡 Dica:** uma das referências on-line do Linux Bash podem ser encontradas nas [páginas Linux Man em linux.die.net](#). Você pode buscar ou navegar pelos comandos para saber mais sobre eles.

Por fim, use a engenhosidade ao concluir este projeto. Por exemplo, consulte a [Documentação da AWS](#) ou um mecanismo de pesquisa se precisar de uma resposta a uma pergunta técnica.

Ao trabalhar no projeto, você encontrará outras dicas ajudar você a concluir fases ou tarefas específicas.

## Abordagem

[Voltar ao sumário](#)

A seguinte tabela descreve as fases do projeto:

Fase	Detalhes	Requisito da solução
1	Crie um diagrama da arquitetura e o custo estimado da solução.	R1, R2
2	Analise o design da aplicação monolítica e teste a aplicação.	R3
3	Crie um ambiente de desenvolvimento no AWS Cloud9 e insira o código fonte monolítico no CodeCommit.	R3
4	Divida o design monolítico em microserviços e inicie o teste em contêineres do Docker.	R3, R4

Fase	Detalhes	Requisito da solução
5	Crie repositórios do ECR para armazenar imagens do Docker. Crie um cluster do ECS, definições de tarefa do ECS e arquivos de especificação da aplicação no CodeDeploy.	R3
6	Crie grupos de destino e um Application Load Balancer que roteie o tráfego na web a eles.	R5
7	Criar serviços do ECS.	R5
8	Configure aplicações e grupos de implantação no CodeDeploy e crie dois pipelines CI/CD usando o CodePipeline.	R5, R6
9	Modifique os microsserviços e a capacidade de dimensionamento e use os pipelines que você criou para implantar melhorias iterativas à produção, usando uma estratégia de implantação azul/verde.	R5, R6

## Fase 1: planejamento do design e estimativa de custo

Nesta fase, você planejará o design de sua arquitetura. Primeiro, você criará um diagrama da arquitetura.

Você também estimará o custo da solução proposta e apresentará a estimativa ao instrutor. O planejamento do design e a estimativa de custo são as primeiras etapas importantes de qualquer solução. Analise os vários componentes da arquitetura para ajustar a estimativa de custo. O custo, bem como os recursos e as limitações de serviços da AWS específicos, são fatores importantes ao criar uma solução.

### Tarefa 1.1: criar um diagrama de arquitetura

[Voltar ao sumário](#)

1. Crie um diagrama da arquitetura para ilustrar o que você planeja criar. Considere a forma como atenderá cada requisito da solução. Leia as fases neste documento para saber quais serviços e recursos da AWS foi solicitado que você use. Inclua os seguintes serviços ou recursos em seu diagrama:

- Amazon Virtual Private Cloud (Amazon VPC)
- Amazon EC2: instâncias, Application Load Balancer, grupos de destino
- AWS CodeCommit: repositório
- AWS CodeDeploy
- AWS CodePipeline: pipeline
- Amazon Elastic Container Service (Amazon ECS): serviços, contêineres, tarefas
- Amazon Elastic Container Registry (Amazon ECR): repositório
- Ambiente do AWS Cloud9
- AWS Identity and Access Management (IAM): funções
- Amazon Relational Database Service (Amazon RDS)
- Amazon CloudWatch: logs

#### Referências

- [AWS Architecture Icons](#): este site fornece ferramentas para desenhar diagramas de arquitetura da AWS.
- [AWS Reference Architecture Diagrams](#) (Diagramas de arquitetura de referência da AWS): este site fornece uma lista de diagramas de arquitetura da AWS para diversos casos de uso. É possível usar esses diagramas como referências.

## Tarefa 1.2: desenvolver uma estimativa de custo

[Voltar ao sumário](#)

Desenvolva uma estimativa de custo que exiba o custo para executar a solução para a qual você criou um diagrama de arquitetura. Suponha que a solução será executada na Região us-east-1 por 12 meses. Use a [Calculadora de Preços da AWS](#) para essa estimativa.

Se for instruído a fazer isso, conclua os requisitos adicionais a seguir:

- Adicione o diagrama da arquitetura e a estimativa de custo nos slides da apresentação. O instrutor pode querer verificar essas informações como parte da avaliação de seu trabalho neste projeto. Um modelo de apresentação é fornecido.
- Faça capturas de tela do seu trabalho no final de cada tarefa ou fase para incluir na apresentação ou no documento. O instrutor pode usar a apresentação ou o documento para ajudar a avaliar o desempenho da sua conclusão dos requisitos do projeto.

### Referência

- Um modelo de apresentação em PowerPoint está disponível no material do curso.

## Fase 2: análise da infraestrutura da aplicação monolítica

Nesta tarefa, você analisará a infraestrutura da aplicação atual e testar o aplicativo web.

### Tarefa 2.1: verificar a disponibilidade da aplicação monolítica

[Voltar ao sumário](#)

1. Verifique se o aplicativo web monolítico pode ser acessado pela internet.

- Navegue até o console do Amazon EC2.
- Copie o endereço IPv4 público da instância *MonolithicAppServer* e carregue-o em uma nova guia do navegador.

O site dos fornecedores de café é exibido.

■ **Observação:** a página está disponível em `http://`, não em `https://`. Seu navegador poderá indicar que o site não é seguro por não ter um certificado SSL/TLS válido. Você pode ignorar o aviso neste ambiente de desenvolvimento.

### Tarefa 2.2: testar o aplicativo web monolítico

[Voltar ao sumário](#)

Nesta tarefa, você adicionará dados ao aplicativo web, testará a funcionalidade e observará os diferentes caminhos de URL usados para exibir as diferentes páginas. É importante entender esses caminhos de URL para quando você dividir a aplicação em microsserviços mais adiante.

1. Selecione **List of suppliers** (Lista de fornecedores).

Observe que o caminho do URL inclui `/suppliers`.

2. Adicione um novo fornecedor.



- Na página em que você adiciona um novo fornecedor, observe que o caminho do URL inclui `/supplier-add`.
- Preencha todos os campos para criar uma entrada de fornecedor.

### 3. Edite uma entrada.

- Na página em que você edita uma entrada de fornecedor, observe que o caminho do URL inclui `supplier-update/1`.
- Modifique o registro de alguma maneira e salve a alteração.  
Observe que a alteração foi salva no registro.

## Tarefa 2.3: analisar a forma de execução da aplicação monolítica

[Voltar ao sumário](#)

1. Use o EC2 Instance Connect para se conectar à instância *MonolithicAppServer*.
2. Analise como a aplicação está executando.

- Na sessão do terminal, execute o comando a seguir:

```
sudo lsof -i :80
```

O que você notou na saída do comando? Que porta e protocolo o daemon *node* está usando?

- Depois, execute os comandos a seguir:

```
ps -ef | head -1; ps -ef | grep node
```

O que você notou na saída do comando? Que usuário nesta instância do EC2 está executando um processo *node*? O ID do processo (PID) *node* corresponde a algum dos PIDs da saída do comando que você executou antes desse último?

3. Para analisar a estrutura da aplicação, execute os seguintes comandos:

```
cd ~/resources/codebase_partner  
ls
```

É aqui que fica o arquivo `index.js`. Ele contém a lógica base da aplicação, que você vai ver em detalhes daqui a pouco.

**Para pensar:** com base no que você observou, o que você pode determinar sobre como e onde essa aplicação de nó está executando? Como você acha que ela foi instalada? Se alguma biblioteca era um pré-requisito, qual era necessária para a execução? Onde essa aplicação armazena os dados?

4. Conecte um cliente MySQL ao banco de dados RDS em que a aplicação do nó armazena os dados.

- Encontre e copie o endpoint do banco de dados RDS em execução no ambiente do laboratório.
- Para verificar se o banco de dados pode ser alcançado da instância *MonolithicAppServer* no número de porta MySQL padrão, use o `nmmap -Pn` comando com o endpoint do banco de dados do RDS que você copiou.
- Para conectar-se ao banco de dados, use o cliente MySQL já instalado na instância *MonolithicAppServer*. Use os seguintes valores:

- **Username:** `admin`

- **Password:** `lab-password`

5. Observe os dados no banco de dados.

- Do prompt `mysql>`, execute comandos SQL conforme apropriado para ver que um banco de dados denominado *COFFEE* contém uma tabela chamada *suppliers*.

Esta tabela contém a entrada do fornecedor ou entradas que você adicionou anteriormente, ao testar o aplicativo web.

- Saia do cliente MySQL e feche a guia do EC2 Instance Connect. Feche também a guia do aplicativo web dos fornecedores de café.

## Referências

- [Connect to Your Linux Instance with EC2 Instance Connect \(Conectar à sua instância do Linux usando o EC2 Instance Connect\)](#)
- [Connecting from the MySQL Command-Line Client \(Unencrypted\) \(Conectar a partir do cliente de linha de comando MySQL \[não criptografado\]\)](#)
- Para obter informações sobre os comandos `ls`, `ps`, `grep` e `netmap`, consulte as [Páginas principais do Linux em linux.die.net](#).

## Fase 3: criar um ambiente de desenvolvimento e inserir o código em um repositório Git

Nesta fase, você criará um ambiente de desenvolvimento usando o AWS Cloud9. Você também colocará o código de sua aplicação no AWS CodeCommit, que é um repositório compatível com o Git.

### Tarefa 3.1: criar um IDE do AWS Cloud9 como seu ambiente de trabalho

[Voltar ao sumário](#)

Nesta tarefa, você criará um ambiente de desenvolvimento. Se você não está familiarizado com o AWS Cloud9, as referências a seguir podem ser úteis:

- [Creating an Environment in AWS Cloud9 \(Criar um ambiente no AWS Cloud9\)](#)
- [Tour of the AWS Cloud9 IDE \(Tour do IDE do AWS Cloud9\)](#)

1. Crie uma instância do AWS Cloud9 denominada `MicroservicesIDE` e abra o IDE.

Ela deverá executar como uma nova instância do EC2 de tamanho `t3.small` e executar o Amazon Linux 2. A instância deve ser compatível com conexões SSH e executar na `LabVPC` na `Public Subnet1`.

### Tarefa 3.2: copiar o código da aplicação para seu IDE

[Voltar ao sumário](#)

Nesta tarefa, você copiará o código de origem da aplicação monolítica para seu ambiente de desenvolvimento.

1. No painel **Detalhes da AWS** nas instruções deste laboratório, baixe o arquivo **labsuser.pem** no seu computador local.
2. Faça upload do arquivo `.pem` ao seu IDE do AWS Cloud9 e use o comando do Linux `chmod` para definir as [permissões adequadas](#) no arquivo. Assim, você poderá usá-lo para se conectar a uma instância do EC2.
3. Crie um diretório temporário na instância do AWS Cloud9 em `/home/ec2-user/environment/temp`.



4. No console do Amazon EC2, pegue o endereço IPv4 privado da instância *MonolithicAppServer*.
5. Use o comando do Linux `scp` no terminal Bash na instância do AWS Cloud9 para copiar o código fonte da aplicação do nó da instância *MonolithicAppServer* para o diretório temporário que você criou na instância do AWS Cloud9.

O trecho a seguir fornece um exemplo do comando scp:

```
scp -r -i ~/environment/labuser.pem  
ubuntu@$appServerPrivIp:/home/ubuntu/resources/codebase_partner/* ~/environment/temp/
```

6. No navegador de arquivos do IDE, verifique se os arquivos de origem da aplicação foram copiados para o diretório temporário da instância do AWS Cloud9.

## Tarefa 3.3: criar diretórios de trabalho com código de iniciação para dois microsserviços

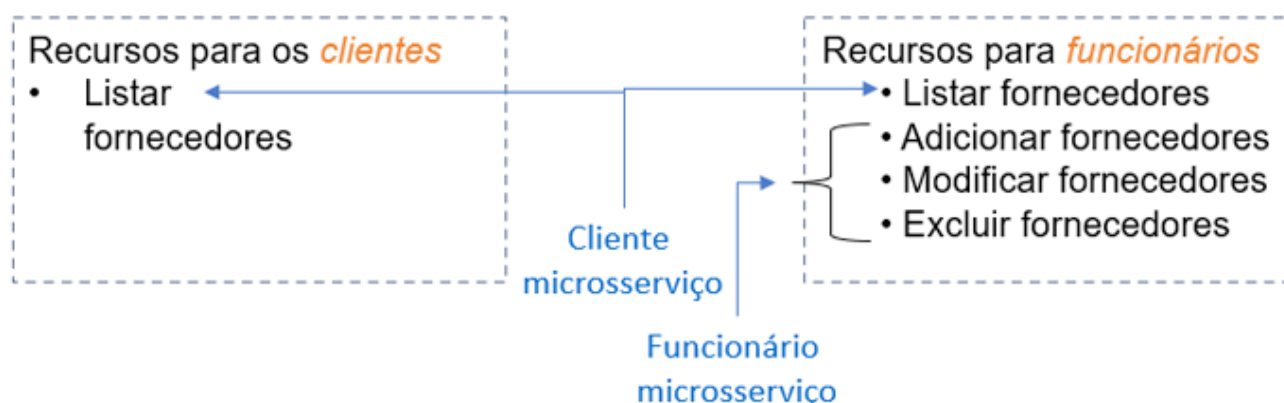
[Voltar ao sumário](#)

Nesta tarefa, você criará áreas em seu ambiente de desenvolvimento que darão suporte à separação da lógica da aplicação em dois microsserviços diferentes.

Com base nos requisitos da solução deste projeto, faz sentido dividir a aplicação monolítica em dois microsserviços. Dê o nome a esses microsserviços de *customer* e *employee*.

A tabela a seguir explica a funcionalidade necessária para cada microsserviço.

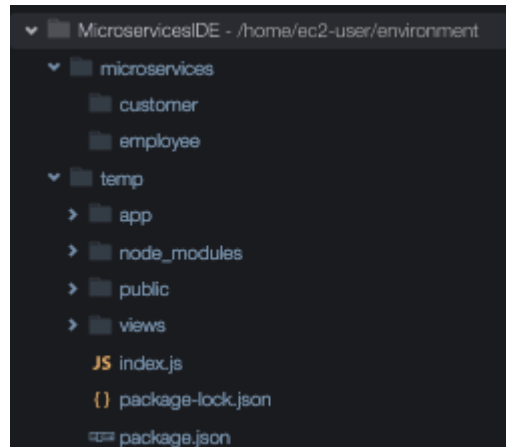
Usuário principal	Funcionalidade do microsserviço	Nível de acesso
Cliente	O microsserviço <i>customer</i> fornecerá a funcionalidade de que os clientes (os gerentes da franquia da cafeteria que querem comprar os grãos de café) precisam. Os clientes precisam de uma visualização somente leitura das informações de contato dos fornecedores para conseguir comprar os grãos de café. Você vai pensar nos gerentes da franquia da cafeteria como <i>customers</i> (clientes) da aplicação.	Somente leitura
Funcionário	O microsserviço <i>employee</i> fornecerá a funcionalidade de que os funcionários (os funcionários da sede corporativa da cafeteria) precisam. Os funcionários precisam adicionar, modificar e excluir fornecedores que estão listados na aplicação. Os funcionários são responsáveis por manter as listas corretas e atualizadas.	Leitura/gravação



O microserviço `employee` será, em algum momento, disponibilizado apenas para os funcionários. Você fará isso primeiramente encapsulando os dois em microserviços separados (nas fases 3 e 4 do projeto) e, depois, na fase 9 do projeto, você limitará quem poderá acessar o microserviço `employee`.

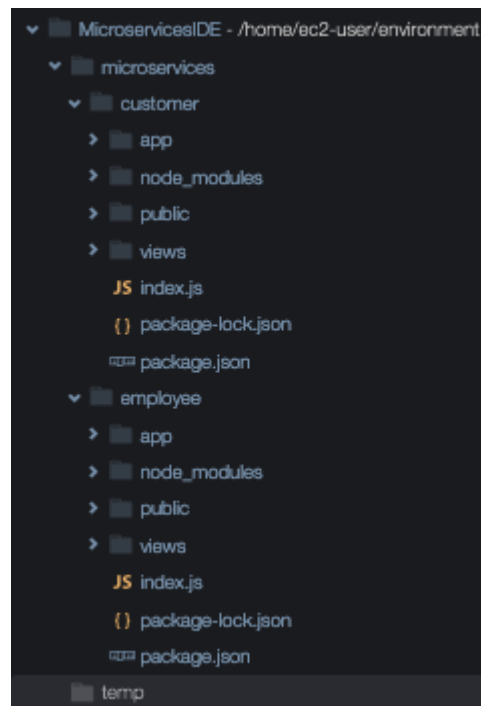
1. No diretório **microservices**, crie dois novos diretórios denominados `customer` e `employee`.

Verifique se a estrutura do diretório corresponde à imagem a seguir:



2. Insira uma cópia do código fonte para a aplicação monolítica em cada diretório novo e remova os arquivos do diretório **temporário**.

Verifique se a estrutura do diretório corresponde à imagem a seguir:



3. Exclua o diretório **temporário** vazio.

## Tarefa 3.4: criar um repositório Git para o código de microserviços e enviar o código para o CodeCommit

[Voltar ao sumário](#)

Agora, você tem diretórios chamados `customer` e `employee`, e cada um terá um microserviço. Os dois microserviços replicarão a lógica em sua aplicação monolítica. Porém, você também poderá evoluir a funcionalidade da aplicação e cronometrar a implantação das melhorias de recursos para esses microserviços separadamente.

Colocar esse código fonte em um repositório Git trará benefícios; Neste projeto, você usará o CodeCommit como repositório (repo) Git.

1. Crie um repositório CodeCommit chamado `microservices`.
2. Para colocar o código inalterado da aplicação no repositório `microservices` do CodeCommit, execute os seguintes comandos:

```
cd ~/environment/microservices
git init
git branch -m dev
git add .
git commit -m 'two unmodified copies of the application code'
git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
git push -u origin dev
```

💡 **Dica:** para obter informações sobre comandos do Git, consulte a [Documentação do Git](#).

**Análise:** ao executar esses comandos, você primeiramente *inicializou* o diretório de microsserviços para ser um repositório Git. Depois, você criou uma *ramificação* no repositório de nome *dev*. Você *adicionou* todos os arquivos do diretório de microsserviços ao repositório Git e *confirmou*. Depois, você definiu o repositório de microsserviços que você criou no CodeCommit como a *origem remota* desta área do repositório Git em seu IDE. Por fim, você *enviou* as alterações que foram confirmadas na ramificação *dev* para a origem remota.

3. Configure seu cliente Git para que ele saiba seu usuário e endereço de e-mail.

💡 **Dica:** para obter informações sobre os comandos que você precisa usar, consulte [Getting Started - First-Time Git Setup \(Introdução: primeira configuração do Git\)](#).

📌 **Observação:** você não precisa usar seu nome real ou endereço de e-mail. No entanto, concluir esta etapa é uma parte importante da configuração de um cliente Git.

4. Em uma nova aba do navegador, vá até o console do CodeCommit e observe se o código já está no repositório dos microsserviços.

## Fase 4: configuração da aplicação como dois microsserviços e como testá-los nos contêineres do Docker

Nesta fase, você modificará as duas cópias do código de inicialização da aplicação monolítica para que a funcionalidade da aplicação seja implementada como dois microsserviços separados. Para fins de testes iniciais, você executará os contêineres na mesma instância do EC2 que hospeda o IDE do AWS Cloud9 que você está usando. Você usará esse IDE para criar as imagens do Docker e iniciar os contêineres do Docker.

### Tarefa 4.1: ajustar as configurações do grupo de segurança da instância do AWS Cloud9

[Voltar ao sumário](#)

Nesta fase do projeto, você usará a instância do AWS Cloud9 como seu ambiente de teste. Você executará os contêineres do Docker na instância para testar os microsserviços que você criará. Para poder acessar os contêineres de um navegador pela internet, é necessário abrir as portas em que você vai executar os contêineres.

1. Ajuste o grupo de segurança da instância do EC2 do AWS Cloud9 para permitir a entrada do tráfego de rede nas portas 8080 e 8081.

## Tarefa 4.2: modificar o código fonte do microserviço *customer*

[Voltar ao sumário](#)

Nesta tarefa, você editará o código fonte do microserviço *customer*. Lembre-se de que o código fonte com o qual você está começando é uma cópia exata da aplicação monolítica. Portanto, ele ainda tem recursos que serão gerenciados pelo microserviço *employee* e que você não quer usar como parte do microserviço *customer*. Especificamente, os clientes não devem poder adicionar, editar ou excluir fornecedores; portanto, as alterações feitas por você removerão essa funcionalidade do código fonte. Idealmente, o código fonte deverá conter apenas o código necessário.

1. No painel do AWS Cloud9, *recolha* o diretório **employee**, caso esteja expandido, e *expanda* o diretório **customer**.
2. Edite o arquivo `customer/app/controller/supplier.controller.js` para que as funções restantes forneçam ações somente leitura que você quer que os clientes possam realizar.

💡 **Dica:** depois de editar o arquivo, ele terá apenas as linhas a seguir:

```
const Supplier = require("../models/supplier.model.js");
const {body, validationResult} = require("express-validator");

exports.findAll = (req, res) => {
  Supplier.getAll((err, data) => {
    if (err)
      res.render("500", {message: "The was a problem retrieving the list of suppliers"});
    else res.render("supplier-list-all", {suppliers: data});
  });
};

exports.findOne = (req, res) => {
  Supplier.findById(req.params.id, (err, data) => {
    if (err) {
      if (err.kind === "not_found") {
        res.status(404).send({
          message: `Not found Supplier with id ${req.params.id}.`
        });
      } else {
        res.render("500", {message: `Error retrieving Supplier with id ${req.params.id}`});
      }
    } else res.render("supplier-update", {supplier: data});
  });
};
```

3. Edite o arquivo `customer/app/models/supplier.model.js`. Exclua as funções desnecessárias para que o que sobre sejam funções somente leitura.

❗ **Importante:** *MANTENHA* a última linha do arquivo: `module.exports = Supplier;`

📌 **Observação:** o modelo ainda conterà duas funções: `Supplier.getAll` e `Supplier.findById`.

4. Mais adiante no projeto, ao implantar microserviços atrás de um Application Load Balancer, você terá de fazer os *employees* poderem navegar da página principal do *customer* para a área do aplicativo web em que eles podem adicionar, editar ou excluir entradas de fornecedores. Para poder fazer isso, edite o arquivo `customer/views/nav.html`:

- Na linha 3, altere `Monolithic Coffee suppliers` para `Coffee suppliers`
- Na linha 7, altere `Home` para `Customer home`
- Adicione uma nova linha *após* a linha 8 que contenha o seguinte HTML:

❗ **Importante:** *NÃO* exclua ou substitua qualquer linha existente no arquivo.

```
<a class="nav-link" href="/admin/suppliers">Administrator link</a>
```

**Análise:** adicionar este link fornecerá um caminho de navegação para essas páginas que serão hospedadas no caminho de UR `/admin/`.

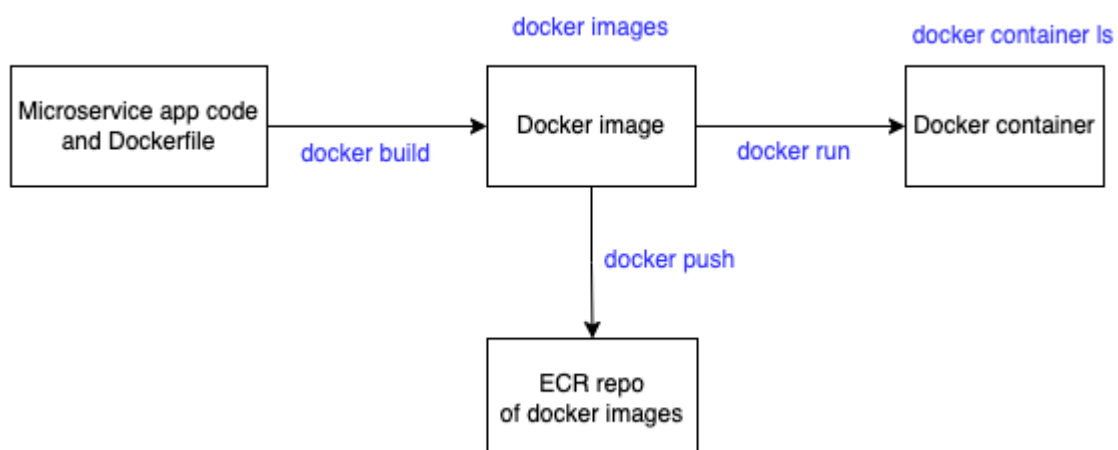
5. Você não quer que os clientes vejam o botão **Add a new supplier** (Adicionar novo fornecedor) ou qualquer outro botão de **edição** ao lado das linhas dos fornecedores. Para implementar essas alterações, edite o arquivo `customer/views/supplier-list-all.html`:
  - Remova a linha 32, que contém `Add a new supplier`.
  - Remova as linhas 26 e 27, que contém `badge badge-info` e `supplier-update`.
6. Como o microserviço `customer` não precisa de suporte a ações de leitura/gravação, **EXCLUA** os seguintes arquivos `.html` do diretório `customer/views`:
  - `supplier-add.html`
  - `supplier-form-fields.html`
  - `supplier-update.html`
7. Edite o arquivo `customer/index.js` conforme necessário para levar em consideração o fato de que a aplicação do nó agora executará em contêineres do Docker:
  - Comente nas linhas de 27 a 37 (todas as linhas devem começar com `//`).
  - Na linha 45, altere o número da porta para `8080`

💡 **Dica:** lembre-se de que, quando essa aplicação executou na instância *MonolithicAppServer*, isso ocorreu na porta 80. No entanto, quando ela executa em um contêiner do Docker, o contêiner deverá executar na porta 8080.

## Tarefa 4.3: criar o Dockerfile do microserviço *customer* e iniciar um contêiner de teste

[Voltar ao sumário](#)

O diagrama a seguir fornece uma visão geral de como você usará o [Docker](#). Os ativos que você usará ou criará são representados por retângulos. Os comandos que você usará são exibidos em azul entre e acima dos retângulos.



Com a base do código da aplicação e um Dockerfile, que você criará, você criará a *imagem do Docker*. Uma imagem do Docker é um modelo que tem instruções para criar um *contêiner* do Docker. Pense em uma *imagem* do Docker mais ou menos como um equivalente a uma imagem de máquina da Amazon (AMI), de onde você pode iniciar uma instância do EC2. Um *contêiner* do Docker é mais ou menos como uma instância do EC2. No entanto, as imagens do Docker e os contêineres são muito menores.

1. No diretório **customer**, crie um novo arquivo chamado `Dockerfile` que contenha o seguinte código:

```
FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

**Análise:** este código do Dockerfile especifica que uma distribuição do Alpine Linux com requisitos de runtime do Node.js deve ser usada para criar uma imagem do Docker. O código também especifica que o contêiner deve permitir o tráfego de rede na porta TCP 8080 e que a aplicação deve ser executada e iniciada quando um contêiner criado da imagem for iniciado.

2. Crie uma imagem a partir do Dockerfile customer.
  - No terminal do AWS Cloud9, altere o diretório **customer**.
  - Execute o seguinte comando:

```
docker build --tag customer .
```

📌 **Observação:** na saída, ignore o aviso do npm sobre o campo repositório.

Observe que a compilação baixou a imagem inicial Alpine do nó e concluiu as outras instruções, como especificado no Dockerfile.

3. Verifique se a imagem do Docker denominada customer foi criada.

- Execute um comando do Docker para listar as imagens do Docker das quais seu cliente do Docker está ciente.

💡 **Dica:** para encontrar o comando que você precisa executar, consulte [Use the Docker Command Line](#) (Uso da linha de comando do Docker) na documentação do Docker.

💡 **Dica:** a saída do comando deve ser semelhante ao seguinte:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
customer	latest	cdc593c9bf51	59 seconds ago	82.7MB
node	11-alpine	f18da2f58c3d	3 years ago	75.5MB

📌 **Observação:** a imagem do *nó* é a Alpine Linux que você identificou no conteúdo do Dockerfile para ser baixada e usada como imagem de início. O seu cliente Docker baixou-a do docker.io. A imagem *customer* é a que você criou.

4. Inicie um *contêiner* do Docker que execute o microsserviço *customer* na porta 8080. Como parte do comando, transmita uma variável de ambiente para informar à aplicação de nós o local correto do banco de dados.
  - Para definir uma variável dbEndpoint em sua sessão terminal, execute os seguintes comandos:



```
dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep
'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint
```

🔍 **Observação:** você pode encontrar o endpoint do banco de dados manualmente no console do Amazon RDS e defini-lo como uma variável de ambiente executando `dbEndpoint=<actual-db-endpoint>` em vez de usar um comando `cat`.

❗ **Importante:** se você fechar o terminal do AWS Cloud9 ou interromper e reiniciar o ambiente de laboratório do projeto e, depois, precisar executar um comando que usa a variável `$dbEndpoint`, pode ser necessário criar a variável novamente. Para testar se a variável está definida, execute `echo $dbEndpoint`

- O código a seguir fornece um exemplo do comando que você deve executar para iniciar um contêiner a partir da imagem:

```
docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer
```

**Análise:** esse comando iniciou um contêiner com o nome `customer_1` usando a imagem `customer` que você criou como o modelo. O parâmetro `-d` no comando especificou que ele deve executar em segundo plano. Após o parâmetro `-p`, o *primeiro número* especificou a porta na instância do AWS Cloud9 para publicar o contêiner. O *segundo número* indicou a porta em que o contêiner está sendo executado no *namespace do docker* (também é a porta 8080). O parâmetro `-e` transmite a localização do host do banco de dados como uma variável de ambiente para o Docker, que fornece à aplicação do nó a informação de que precisa estabelecer conectividade ao banco de dados.

5. Verifique quais contêineres do Docker estão executando na instância do AWS Cloud9.

💡 **Dica:** para encontrar o comando que você precisa executar, consulte [Use the Docker Command Line](#) (Uso da linha de comando do Docker) na documentação do Docker.

6. Verifique se o microsserviço `customer` está executando no contêiner e funcionando conforme o esperado.

- Carregue a seguinte página em uma nova guia do navegador. Substitua o placeholder do endereço IP pelo endereço IPv4 público da instância do AWS Cloud9 que você está usando: `http://<cloud-9-public-IPv4-address>:8080`

O microsserviço do aplicativo web carregará no navegador.

❗ **Importante:** se você interromper o ambiente do laboratório e iniciá-lo novamente, o endereço IPv4 público da instância do AWS Cloud9 mudará.

- Selecione **List of suppliers** (Lista de fornecedores).
- Confirme se as entradas de fornecedores que você inseriu anteriormente são exibidas.

🔍 **Observação:** apesar de você ter modificado o local em que a aplicação executa, ela ainda se conecta ao mesmo banco de dados do RDS em que os registros do fornecedor são armazenados.

💡 **Dica:** o link **Administrator** (Administrador) não funciona porque você ainda não criou o microsserviço `employee`.

- Confirme se a página dos fornecedores não tem botões para **Adicionar novo fornecedor** e **editar**.

💡 **Dica de solução de problemas:** caso alguma funcionalidade esteja faltando, siga as etapas a seguir. (1) interrompa e exclua o contêiner em execução; (2) modifique o código fonte do microsserviço conforme apropriado; (3) crie uma imagem do Docker atualizada a partir do código fonte; (4) inicie um novo contêiner de teste; e (5) verifique se a funcionalidade agora está disponível. Para obter uma lista de comandos a serem executados para realizar essas etapas, consulte [Atualização de um contêiner de teste em execução no Cloud9](#) no apêndice deste arquivo.

7. Confirme e envie as alterações do código fonte para o CodeCommit.

💡 **Dica:** você pode usar o [Painel de controle da origem do Git](#) no IDE do AWS Cloud9 ou usar os comandos `git commit` e `git push` no terminal.

📌 **Observação:** se seu instrutor pediu que você colete informações sobre a sua solução, registre os comandos que executar nesta etapa e a saída retornada.

8. Opcional: observe os detalhes de confirmação no console do CodeCommit.

Na área **Commits** (Confirmações) do repositório, selecione o ID da confirmação mais recente. Role para baixo para ver informações sobre o que mudou nos arquivos desde a confirmação anterior.

Observe que as linhas excluídas são exibidas em vermelho e as linhas adicionadas são exibidas em verde para que você possa ver todos os detalhes de todas as mudanças de todos os arquivos que foram modificados.

## Tarefa 4.4: modificar o código fonte do microserviço *employee*

[Voltar ao sumário](#)

Nesta tarefa, você modificará o código fonte do microserviço *employee*, de uma maneira parecida com a que você modificou o código fonte do microserviço *customer*. Os clientes (gerentes da franquia da cafeteria) devem ter acesso somente leitura aos dados da aplicação, mas os funcionários da sede corporativa da cafeteria devem poder adicionar novas entradas ou modificar entradas existentes na lista de fornecedores de café.

Como você verá mais à frente neste projeto, você implantará os microserviços atrás de um Application Load Balancer e roteará o tráfego para os microserviços com base no caminho contido no URL da solicitação. Desta forma, se o caminho do URL incluir `/admin/`, o balanceador de carga roteará o tráfego para o microserviço *employee*. Caso contrário, se o caminho do URL não incluir `/admin/`, o balanceador de carga roteará o tráfego para o microserviço *customer*.

Por conta da necessidade de rotear o tráfego, muito do trabalho desta tarefa é configurar o microserviço *employee* para adicionar `/admin/` ao caminho das páginas a que serve.

1. No IDE do AWS Cloud9, volte para a visualização de arquivo (visualização toggletree).
2. *Recolha* o diretório **customer** e *expand* o diretório **employee**.
3. No arquivo `employee/app/controller/supplier.controller.js`, em todas as chamadas de redirecionamento, adicione `/admin` ao caminho.

💡 **Dica:** para encontrar as três linhas que precisam ser atualizadas, execute os seguintes comandos no terminal:

```
cd ~/environment/microservices/employee
grep -n 'redirect' app/controller/supplier.controller.js
```

4. No arquivo `employee/index.js`, atualize as chamadas `app.get`, `app.post` e o número da porta.
  - Para todas as chamadas `app.get` e `app.post`, adicione `/admin` ao primeiro parâmetro.

💡 **Dica:** para encontrar as sete linhas que precisam ser atualizadas, execute os seguintes comandos no terminal:

```
grep -n 'app.get|app.post' index.js
```

❗ **Importante:** depois de editar a linha 22, o caminho deverá ser `/admin`, não `/admin/`

- Na linha 45, altere o número da porta para `8081`

🔑 **Observação:** ao executar tanto o contêiner do microserviço *customer* quanto o do *employee* na instância do AWS Cloud9 para testar, eles precisarão usar diferentes números de porta para que não entrem em conflito um com o outro.

5. Nos arquivos `employee/views/supplier-add.html` e `employee/views/supplier-update.html`, nos caminhos de ação do formulário, adicione `/admin` ao caminho.

💡 **Dica:** para encontrar as três linhas que precisam ser atualizadas nos dois arquivos, execute o seguinte comando no terminal:

```
grep -n 'action' views/
```

6. Nos arquivos `employee/views/supplier-list-all.html` e `employee/views/home.html`, nos caminhos de HTML, adicione `/admin` ao caminho.

💡 **Dica:** para encontrar as três linhas que precisam ser atualizadas nos dois arquivos, execute o seguinte comando no terminal:

```
grep -n 'href' views/supplier-list-all.html views/home.html
```

7. No arquivo `employee/views/header.html`, modifique o título para `Manage coffee suppliers`

8. Edite o arquivo `employee/views/nav.html`.

- Na linha 3, altere `Monolithic Coffee suppliers` para `Manage coffee suppliers`
- Na linha 7, substitua a linha existente do código pelo seguinte:

```
<a class="nav-link" href="/admin/suppliers">Administrator home</a>
```

🔑 **Observação:** tanto o valor `href` quanto o nome do link são modificados na nova linha.

- Adicione uma nova linha *após* a linha 8 que contenha o seguinte HTML:

❗ **Importante:** NÃO exclua ou substitua qualquer linha existente no arquivo.

```
<a class="nav-link" href="/">Customer home</a>
```

**Análise:** mais adiante no projeto, ao implantar microserviços atrás de um Application Load Balancer, você terá de fazer os *employees* poderem navegar da página de administrador de volta para a área do cliente do aplicativo web. Adicionar este link fornecerá um caminho de navegação para os funcionários às páginas que serão hospedadas pelo microserviço *customer* no caminho do URL `/`.

- Salve as alterações.

## Tarefa 4.5: criar o Dockerfile do microserviço *employee* e iniciar um contêiner de teste

[Voltar ao sumário](#)

Nesta tarefa, você criará uma imagem do Docker para o microserviço *employee*. Depois, você iniciará um contêiner de teste a partir da imagem e verificará se o microserviço funciona conforme esperado.

1. Crie um Dockerfile para o microserviço *employee*.

- Duplica o Dockerfile do microserviço *customer* para a área de microserviço *employee*.

- Edite `employee/Dockerfile` para alterar o número da porta na linha `EXPOSE` para `8081`.
- 2. Crie a imagem do Docker para o microserviço `employee`. Especifique `employee` como tag.
- 3. Execute um contêiner denominado `employee_1` com base na imagem do `employee`. Execute-o na porta 8081 e certifique-se de transmitir o endpoint do banco de dados.
- 4. Verifique se o microserviço `employee` está executando no contêiner e se as funções do microserviço estão de acordo com o esperado.
  - Carregue a página da web do microserviço em uma nova guia do navegador em `http://<cloud9-public-ip-address>:8081/admin/suppliers`  
A aplicação deve ser carregada.
  - Verifique se essa visualização mostra os botões de edição de fornecedores existentes e de adição de um novo fornecedor.  
  
**📌 Observação:** links que devem levar você ao microserviço `customer` não funcionarão. Por exemplo, se você escolher **Customer home** (Página inicial do cliente) ou **Suppliers list** (Lista de fornecedores), as páginas não serão encontradas porque o link presume que o microserviço `customer` também executa na porta 8081 (apesar de não ser o caso). Você pode ignorar este problema, pois os links deverão funcionar conforme o esperado ao implantar os microserviços ao Amazon ECS mais adiante.
  - Faça um teste de adicionar um novo fornecedor.  
Verifique se o novo fornecedor aparece na página de fornecedores.
  - Faça um teste de edição de um fornecedor existente.  
Verifique se o fornecedor editado aparece na página de fornecedores.
  - Faça um teste de exclusão de um fornecedor existente.
    - Na linha de entrada de um determinado fornecedor, escolha **edit** (editar).
    - Na parte inferior da página, selecione **Delete this supplier** (Excluir este fornecedor) e, depois, **Delete this supplier** novamente.  
Verifique se o fornecedor não aparece mais na página de fornecedores.
- 5. Para ver detalhes sobre os dois contêineres de teste em execução, execute o seguinte comando:

```
docker ps
```

## Tarefa 4.6: ajustar a porta do microserviço `employee` e reconstruir a imagem

[Voltar ao sumário](#)

Ao testar o microserviço `employee` na instância do AWS Cloud9, você o executou na porta 8081. No entanto, ao implantá-lo no Amazon ECS, ele precisará executar na porta 8080. Para ajustar isso, você precisará modificar dois arquivos.

1. Edite os arquivos `employee/index.js` e `employee/Dockerfile` para alterar a porta de `8081` para `8080`.
2. Crie novamente a imagem do Docker para o microserviço `employee`.
  - Para interromper e excluir o contêiner existente (suponha que o nome do contêiner seja `employee_1`), execute o seguinte comando:

```
docker rm -f employee_1
```

- Verifique se o seu terminal está no diretório **employee**.
- Use o comando `docker build` para criar uma imagem nova a partir dos arquivos de origem que você editou mais recentemente. Use a tag `employee`.
- 💡 **Dica:** se você criar uma imagem com o nome de uma imagem já existente, a imagem existente será substituída.
- 📌 **Observação:** você não precisa executar um novo contêiner de teste, então não é necessário executar `docker run`.

## Tarefa 4.7: inserir o código no CodeCommit

[Voltar ao sumário](#)

Nesta tarefa, você confirmará e enviará as alterações feitas ao microserviço `employee` para o CodeCommit.

1. Reveja as atualizações que você fez ao código fonte. Para alcançar esse objetivo:
  - Selecione o ícone de controle da origem no IDE do AWS Cloud9.

Observe a lista de alterações, que indica quais arquivos foram alterados desde a última vez que você inseriu arquivos no repositório remoto do Git (CodeCommit).
  - Escolha um dos arquivos que foi modificado, como `index.js`, para comparar a versão da última confirmação no Git com a versão mais recente. As alterações estão em destaque.

Isso demonstra o benefício de usar um sistema de controle de origem e um IDE compatível com Git, como o AWS Cloud9. Você pode revisar as alterações ao seu código antes de confirmar.
2. Coloque suas alterações no CodeCommit.
  - 💡 **Dica:** você realizou esse mesmo tipo de ação na tarefa 4.3. Você pode realizar esta etapa usando o painel de controle da origem do Git ou usar os comandos `git commit` e `git push` no terminal.

## Fase 5: criar repositórios do ECR, um cluster do ECS, definições de tarefa e arquivos do AppSpec

Neste momento, você já implantou com êxito diversos requisitos de solução. Você dividiu a aplicação monolítica em dois microserviços que podem executar como contêineres do Docker. Você também verificou que os contêineres suportam as ações de aplicações necessárias, como adicionar, editar e excluir entradas do banco de dados. A arquitetura dos microserviços ainda usa o Amazon RDS para armazenar as entradas dos fornecedores de café.

Porém, seu trabalho ainda não acabou. Há mais requisitos de solução a serem implementados. Os contêineres podem executar na instância do AWS Cloud9, mas isso não é uma arquitetura de implantação dimensionável. Você precisa poder dimensionar o número de contêineres que executam em cada microserviço de acordo com a necessidade. Além disso, você precisa ter um balanceador de carga para rotear o tráfego para o microserviço apropriado. Por fim, você precisa conseguir atualizar com facilidade a base de código do microserviço de cada aplicação de forma independente e implementar essas alterações na produção. Nas fases restantes do projeto, você trabalhará para realizar esses requisitos de solução.

# Tarefa 5.1: criar repositórios do ECR e fazer upload das imagens do Docker

[Voltar ao sumário](#)

Nesta fase, você fará upload das imagens do Docker mais recentes de dois microsserviços para separar os repositórios do Amazon ECR.

1. Para autorizar seu cliente Docker a se conectar ao serviço do Amazon ECR, execute os seguintes comandos:

```
account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
echo $account_id
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
```

Uma mensagem na saída do comando indica que o login foi bem-sucedido.

2. Crie um repositório separado privado no ECR para cada microsserviço.

- Nomeie o primeiro repositório como `customer`
- Nomeie o segundo repositório como `employee`

3. Defina as permissões no repositório do ECR *customer*.

- Para obter informações sobre a edição de uma política JSON existente, consulte [Setting a Private Repository Statement](#) (Definição de uma declaração de repositório privado) no guia do usuário do Amazon ECR.
- Substitua as linhas existentes na política pelo seguinte:

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "ecr:*"
    }
  ]
}
```

4. Use a mesma abordagem para definir as mesmas permissões no repositório do ECR *employee*.
5. Marque as imagens do Docker com seu valor *registryId* exclusivo (ID da conta) para facilitar o gerenciamento e o monitoramento dessas imagens.
  - No IDE do AWS Cloud9, execute os seguintes comandos:



```

account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)

# Verify that the account_id value is assigned to the $account_id variable
echo $account_id

# Tag the customer image
docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest

# Tag the employee image
docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest

```

📌 **Observação:** os comandos não retornam a saída.

- Execute o comando `docker` apropriado para verificar se as imagens existem e se as tags foram aplicadas.

💡 **Dica:** para encontrar o comando que você precisa executar, consulte [Use the Docker Command Line](#) (Uso da linha de comando do Docker) na documentação do Docker.

💡 **Dica:** a saída do comando deve ser semelhante à imagem a seguir. Observe que a tag *mais recente* foi aplicada e o nome da imagem inclui o nome do repositório remoto onde você pretende armazená-la:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
720494114539.dkr.ecr.us-east-1.amazonaws.com/employee	latest	35657d6eadf3	About an hour ago	82.7MB
employee	latest	35657d6eadf3	About an hour ago	82.7MB
<none>	<none>	50f4bc0e6210	2 hours ago	82.7MB
720494114539.dkr.ecr.us-east-1.amazonaws.com/customer	latest	a0ef8c8ac2f9	2 hours ago	82.7MB
customer	latest	a0ef8c8ac2f9	2 hours ago	82.7MB
node	11-alpine	f18da2f58c3d	3 years ago	75.5MB

6. Execute o comando `docker` apropriado para enviar as imagens do Docker para o Amazon ECR.

💡 **Dica:** para encontrar o comando que você precisa executar, consulte [Use the Docker Command Line](#) (Uso da linha de comando do Docker) na documentação do Docker.

💡 **Dica:** antes de executar os comandos do Docker, execute o comando a seguir para definir `account_id` como uma variável no terminal. Depois, ao executar os comandos do Docker, você pode usar a referência do ID da conta como `$account_id`.

```

account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)

```

💡 **Dica adicional:** os comandos executados devem ser semelhantes aos comandos a seguir, mas com **REPLACE\_ME** substituído pelo comando correto:

```

docker REPLACE_ME $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
docker REPLACE_ME $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest

```

A saída para *cada* comando do Docker que você executa para enviar as imagens para o Amazon ECR deve ser semelhante ao seguinte:

```

The push refers to repository [642015801240.dkr.ecr.us-east-1.amazonaws.com/node-app]
006e0ec54dba: Pushed
59762f95cb06: Pushed
22736f780b31: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:f75b60adddb8d6343b9dff690533a1cd1fbb34ccce6f861e84c857ba7a27b77d
size: 1783

```

7. Confirme se as duas imagens agora estão armazenadas no Amazon ECR e que cada uma tem o rótulo *mais recente* aplicado.

## Tarefa 5.2: criar um cluster do ECS

[Voltar ao sumário](#)

Nesta etapa, você criará um cluster do Amazon ECS.

1. Crie um cluster sem servidor do AWS Fargate com o nome de `microservices-serverlesscluster`

Ele deve estar configurado para usar `LabVPC`, `PublicSubnet1` e `PublicSubnet2` (remova qualquer outra sub-rede). *NÃO* selecione instâncias do Amazon EC2 ou do ECS Anywhere.

❗ **Importante:** após escolher o botão para criar o cluster, no banner que aparecer no topo da página, selecione **View in CloudFormation** (Ver no CloudFormation). Aguarde até que a pilha que cria o cluster obtenha o status `CREATE_COMPLETE` antes de prosseguir para a próxima tarefa. *Se a pilha não criar, por qualquer motivo e, portanto, reverter*, repita essas etapas e tente novamente. Deve dar certo da segunda vez.

## Tarefa 5.3: criar um repositório no CodeCommit para armazenar arquivos de implantação

[Voltar ao sumário](#)

Nesta tarefa, você criará outro repositório no CodeCommit. Este repositório armazenará os arquivos de especificação de configuração da tarefa que o Amazon ECS usará para cada microsserviço. O repositório também armazenará arquivos de especificação AppSpec que o CodeDeploy usará para cada microsserviço.

1. Crie um novo repositório no CodeCommit com o nome de `deployment` para armazenar os arquivos de configuração de implantação.
2. No AWS Cloud9, no diretório **environment** (ambiente), crie um novo diretório denominado `deployment`. Inicialize o diretório como repositório Git com uma ramificação denominada `dev`.

## Tarefa 5.4: criar arquivos de definição de tarefa para cada microsserviço e registrá-los no Amazon ECS

[Voltar ao sumário](#)

Nesta tarefa, você criará um arquivo de definição da tarefa para cada microsserviço e registrar as definições de tarefa no Amazon ECS.

1. No novo diretório **deployment**, crie um arquivo vazio chamado `taskdef-customer.json`
2. Edite o arquivo **taskdef-customer.json**.
  - Copie o código JSON a seguir e cole-o no arquivo:

```
{
  "containerDefinitions": [
    {
```

```

    "name": "customer",
    "image": "customer",
    "environment": [
      {
        "name": "APP_DB_HOST",
        "value": "<RDS-ENDPOINT>"
      }
    ],
    "essential": true,
    "portMappings": [
      {
        "hostPort": 8080,
        "protocol": "tcp",
        "containerPort": 8080
      }
    ],
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-create-group": "true",
        "awslogs-group": "awslogs-capstone",
        "awslogs-region": "us-east-1",
        "awslogs-stream-prefix": "awslogs-capstone"
      }
    }
  },
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024",
  "executionRoleArn": "arn:aws:iam::<ACCOUNT-ID>:role/PipelineRole",
  "family": "customer-microservice"
}

```

- Substitua alguns valores no arquivo:
  - Na linha 37, substitua `<ACCOUNT-ID>` pelo real ID da conta.
  - Na linha 9, substitua `<RDS-ENDPOINT>` pelo endpoint real do RDS.
- Salve as alterações.

3. Para registrar a definição da tarefa *customer* no Amazon ECS, execute o comando a seguir:

```
aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-customer.json"
```

4. No console do Amazon ECS, verifique se a definição de tarefa *customer-microservice* agora aparece no painel **Definições de tarefa**. Observe também que o número de revisão é exibido após o nome da definição de tarefa.

💡 **Dica:** consulte a [documentação do ECS](#) se achar útil.

5. No diretório *deployment*, crie um arquivo de especificação `taskdef-employee.json`.

- Adicione o mesmo código JSON que atualmente está no arquivo *taskdef-customer.json* (onde você já definiu o ID da conta e os endpoints do RDS).
- Depois de colar o código, altere as três ocorrências de `customer` para `employee`

6. Para registrar a definição da tarefa *employee* no Amazon ECS, execute um comando da AWS CLI.

7. No console do Amazon ECS, verifique se a definição de tarefa *employee-microservice* agora aparece no painel **Definições de tarefa**. Observe também que o número de revisão é exibido após o nome da definição de tarefa.

## Tarefa 5.5: criar arquivos AppSpec para CodeDeploy para cada microsserviço

[Voltar ao sumário](#)

Nesta tarefa, você continuará a concluir tarefas para oferecer suporte à implantação do aplicativo web com base em microsserviços para executar em um cluster do ECS, onde a implantação tem o suporte de um pipeline CI/CD. Nesta tarefa específica, você criará dois [arquivos de especificação da aplicação \(AppSpec\)](#), um para cada microsserviço. Esses arquivos fornecerão instruções para que o CodeDeploy implante os microsserviços no Amazon ECS na infraestrutura do Fargate.

1. Crie um arquivo AppSpec para o microsserviço *customer*.
  - No diretório *deployment*, crie um arquivo chamado `appspec-customer.yaml`.
  - Copie o código YAML a seguir e cole-o no arquivo:
    - ❗ **Importante:** NÃO modifique `<TASK_DEFINITION>`. Essa configuração será atualizada automaticamente quando o pipeline for executado.

```
version: 0.0
Resources:
  - TargetService:
    Type: AWS::ECS::Service
    Properties:
      TaskDefinition: <TASK_DEFINITION>
      LoadBalancerInfo:
        ContainerName: "customer"
        ContainerPort: 8080
```

📌 **Observação:** esse arquivo está no formato YAML. Em YAML, recuo é importante. Verifique se o código no seu arquivo mantém os níveis de recuo, conforme mostrado no bloco de código anterior.

- Salve as alterações.
2. No mesmo diretório, crie um arquivo AppSpec para o microsserviço *employee*.
    - Nomeie o arquivo `appspec-employee.yaml`.
    - O conteúdo do arquivo deve ser o mesmo que o do arquivo *appspec-customer.yaml*. No entanto, altere `customer` on the `containerName` line to be `employee`

## Tarefa 5.6: atualizar arquivos e inseri-los no CodeCommit

[Voltar ao sumário](#)

Nesta tarefa, você atualizará os dois arquivos de definição de tarefa. Depois, você enviará os quatro arquivos que você criou nas duas últimas tarefas para o repositório *deployment*.

1. Edite o arquivo `taskdef-customer.json`.

- Modifique a linha 5 para corresponder ao seguinte:

```
"image": "<IMAGE1_NAME>",
```

- Salve a alteração.

**Análise:** `<IMAGE1_NAME>` não é um nome de imagem válido, e é por isso que você originalmente definiu o nome da imagem como *customer* antes de executar o comando da AWS CLI para registrar a primeira revisão do arquivo no Amazon ECS. No entanto, neste momento do projeto, é importante definir o valor da imagem como um valor de texto de espaço reservado. Mais adiante no projeto, quando você configurar um pipeline, você identificará `IMAGE1_NAME` como texto de espaço reservado que pode ser atualizado dinamicamente. Resumindo, o CodePipeline definirá o nome correto da imagem dinamicamente no runtime.

## 2. Edite o arquivo **taskdef-employee.json**.

- Modifique a linha 5 para corresponder ao seguinte:

```
"image": "<IMAGE1_NAME>",
```

- Salve a alteração.

## 3. Envie os quatro arquivos para o CodeCommit.

■ **Observação:** enviar os arquivos mais recentes para o CodeCommit é essencial. Mais adiante, quando você criar o pipeline CI/CD, o pipeline pegará esses arquivos do CodeCommit e usará os detalhes contidos neles como instruções para implantar atualizações dos microsserviços no cluster do Amazon ECS.

# Fase 6: criar grupos de destino e um Application Load Balancer

Nesta fase, você criará um Application Load Balancer, que fornecerá um URL de endpoint. Esse URL atuará como o ponto de entrada HTTPS para que os clientes e funcionários acessem sua aplicação por um navegador da web. O balanceador de carga terá regras de roteamento e acesso que determinarão para qual grupo de destino dos contêineres em execução a solicitação do usuário deve ser direcionada.

## Tarefa 6.1: criar quatro grupos de destino

[Voltar ao sumário](#)

Nesta tarefa, você criará quatro grupos de destino, dois para cada microsserviço. Como você configurará uma implantação azul/verde, o CodeDeploy requer dois grupos de destino para cada grupo de implantação.

■ **Observação:** azul/verde é uma estratégia de implantação em que você cria dois ambientes separados, porém idênticos. Um ambiente (azul) executa a versão atual da aplicação, e o outro (verde) executa a nova versão da aplicação. Para obter mais informações, consulte [Blue/Green Deployments](#) (Implantações azul/verde) no whitepaper *Overview of Deployment Options on AWS* (Visão geral das opções de implantação na AWS).

### 1. Crie o primeiro grupo de destino para o microsserviço *customer*.

- Navegue até o console do Amazon EC2.
- No painel de navegação, escolha **Grupos de destino**.
- Escolha **Criar grupo de logs** e configure o seguinte:
  - **Escolher um tipo de destino:** selecione **endereços IP**.

- **Nome do grupo de destino:** insira: `customer-tg-one`.
  - **Protocolo:** selecione **HTTP**.
  - **Porta:** insira `8080`.
  - **VPC:** escolha **LabVPC**.
  - **Caminho da verificação de integridade:** insira `/`.
- Selecione **Próximo**.
  - Na página **Registrar destinos**, aceite todos os padrões (não registre nenhum destino) e selecione **Criar grupo de destino**.
2. Crie um segundo grupo de destino para o microserviço *customer*. Use as mesmas configurações do primeiro grupo de destino, mas use `customer-tg-two` como nome do grupo de destino.
  3. Crie um grupo de destino para o microserviço *employee*. Use as mesmas configurações dos outros grupos de destino, mas com as seguintes exceções:
    - **Nome do grupo de destino:** insira: `employee-tg-one`.
    - **Caminho da verificação de integridade:** insira `/admin/suppliers`.
  4. Crie um segundo grupo de destino para o microserviço *employee*. Use as mesmas configurações dos outros grupos de destino, mas com as seguintes exceções:
    - **Nome do grupo de destino:** insira: `employee-tg-two`
    - **Caminho da verificação de integridade:** insira `/admin/suppliers`.
- ❗ **Importante:** confirme atentamente o nome e o número da porta de cada grupo de destino. A imagem a seguir fornece um exemplo:

	Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
<input type="checkbox"/>	customer-tg-one	arn:aws:elasticloadbalanci...	8080	HTTP	IP	None associated	vpc-0baa92e22c5c5022a
<input type="checkbox"/>	customer-tg-two	arn:aws:elasticloadbalanci...	8080	HTTP	IP	None associated	vpc-0baa92e22c5c5022a
<input type="checkbox"/>	employee-tg-one	arn:aws:elasticloadbalanci...	8080	HTTP	IP	None associated	vpc-0baa92e22c5c5022a
<input type="checkbox"/>	employee-tg-two	arn:aws:elasticloadbalanci...	8080	HTTP	IP	None associated	vpc-0baa92e22c5c5022a

## Tarefa 6.2: criar um grupo de segurança e um Application Load Balancer e configurar regras para roteamento do tráfego

[Voltar ao sumário](#)

Nesta tarefa, você criará um Application Load Balancer. Você também definirá dois listeners para o balanceador de carga: um na porta 80 e outro na porta 8080. Para cada listener, você definirá regras de roteamento com base no caminho, para que o tráfego seja roteado ao grupo de destino correto dependendo do URL que um usuário tentar carregar.

1. Crie um novo grupo de segurança do EC2 denominado `microservices-sg` para usar em *LabVPC*. Adicione regras de entrada que permitam tráfego TCP de qualquer endereço IPv4 nas portas 80 e 8080.
2. No console do Amazon EC2, crie um Application Load Balancer denominado `microservicesLB`.
  - Faça-o voltado para a internet para endereços IPv4.
  - Use *LabVPC*, *Public Subnet1*, *Public Subnet2* e o grupo de segurança *microservices-sg*.



- Configure dois listeners nele. O primeiro deve ouvir o `HTTP:80` e encaminhar o tráfego para `customer-tg-two` por padrão. O segundo deve ouvir o `HTTP:8080` e encaminhar o tráfego para `customer-tg-one` por padrão.

3. E uma segunda regra para o listener do `HTTP:80`. Defina a seguinte lógica para esta nova regra:

- SE o **Caminho** for `/admin/*`
- ENTÃO **Encaminhe para...** o grupo de destino **employee-tg-two**.

As configurações devem ser as que estão na imagem a seguir:

Name tag	Priority	Conditions (If)	Actions (Then) <a href="#">↗</a>
-	1	Path Pattern is /admin/*	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <b>employee-tg-two</b>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>
<i>Default</i>	Last (default)	<i>If no other rule applies</i>	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <b>customer-tg-two</b>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>

4. E uma segunda regra para o listener do `HTTP:8080`. Defina a seguinte lógica para esta nova regra:

- SE o **Caminho** for `/admin/*`
- ENTÃO **Encaminhe para...** o grupo de destino **employee-tg-one**.

As configurações devem ser as que estão na imagem a seguir:

Name tag	Priority	Conditions (If)	Actions (Then) <a href="#">↗</a>
-	1	Path Pattern is /admin/*	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <b>employee-tg-one</b>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>
<i>Default</i>	Last (default)	<i>If no other rule applies</i>	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <b>customer-tg-one</b>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>

## Fase 7: criar dois serviços do Amazon ECS

Nesta fase, você criará um serviço no Amazon ECS para cada microserviço. Apesar de ser possível implantar os dois microserviços em um único serviço do ECS, para este projeto será mais fácil gerenciar os microserviços de forma independente se cada um for implantado em seu próprio serviço do ECS.

### Tarefa 7.1: criar o serviço do ECS para o microserviço *customer*

[Voltar ao sumário](#)

1. No AWS Cloud9, crie um novo arquivo denominado `create-customer-microservice-tg-two.json` no diretório **deployment**.
2. Copie o código JSON a seguir e cole-o no arquivo:

```
{
  "taskDefinition": "customer-microservice:REVISION-NUMBER",
  "cluster": "microservices-serverlesscluster",
  "loadBalancers": [
    {
      "targetGroupArn": "MICROSERVICE-TG-TWO-ARN",
      "containerName": "customer",
      "containerPort": 8080
    }
  ]
}
```

```

    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
      "type": "CODE_DEPLOY"
    },
  },
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "PUBLIC-SUBNET-1-ID",
        "PUBLIC-SUBNET-2-ID"
      ],
      "securityGroups": [
        "SECURITY-GROUP-ID"
      ],
      "assignPublicIp": "ENABLED"
    }
  }
}

```

### 3. Edite o arquivo **create-customer-microservice-tg-two.json**:

- Substitua **REVISION-NUMBER** pelo número da revisão mais recente da definição de tarefa do *customer-microservice* registrado no Amazon ECS.
  - Se for a primeira vez que está realizando esta etapa, o número de revisão deve ser **1**.
  - Se está repetindo a etapa, encontre o número da revisão mais recente no console do Amazon ECS escolhendo **Definições de tarefa** e selecionando **customer-microservice**.
- Substitua **MICROSERVICE-TG-TWO-ARN** pelo ARN real do grupo de destino do *customer-tg-two*.
- Substitua **PUBLIC-SUBNET-1-ID** pelo ID real da sub-rede: *Public Subnet1*.
- Substitua **PUBLIC-SUBNET-2-ID** pelo ID real da sub-rede: *Public Subnet2*.
- Substitua **SECURITY-GROUP-ID** pelo ID real do grupo de segurança de *microservices-sg*.
- Salve as alterações.

### 4. Para criar o serviço do Amazon ECS para o microserviço *customer*, execute os seguintes comandos:

```

cd ~/environment/deployment
aws ecs create-service --service-name customer-microservice --cli-input-json
file://create-customer-microservice-tg-two.json

```

**💡 Dica de solução de problemas:** se você está repetindo esta etapa e criou o serviço do ECS anteriormente, pode aparecer um erro sobre a criação do serviço não ser idempotente. Para resolver este erro, force a exclusão do serviço do console do Amazon ECS, espere que esvazie e execute os comandos novamente.

## Tarefa 7.2: criar o serviço do Amazon ECS para o microserviço *employee*

[Voltar ao sumário](#)

1. Criar um serviço do Amazon ECS para o microserviço *employee*.

- Copie o arquivo JSON que você criou para o microsserviço `customer` e denomine-o `create-employee-microservice-tg-two.json`. Salve-o no mesmo diretório.
- Modifique o arquivo **create-employee-microservice-tg-two.json**:
  - Na linha 2, altere `customer-microservice` para `employee-microservice` e atualize também o **número da revisão**.
  - Na linha 6, insira o ARN do grupo de destino **employee-tg-two**.
    - 💡 **Dica:** não apenas altere `customer` para `employee` nesta linha. O ARN é exclusivo de outras maneiras.
  - Na linha 7, altere `customer` para `employee`
- Salve as alterações.

2. Execute o comando da AWS CLI apropriado para criar o serviço no Amazon ECS.

📌 **Observação:** se você for ao console do Amazon ECS e olhar para os serviços no cluster, poderá ver *0/1 Tarefa em execução*, conforme exibido na imagem a seguir. Isso é esperado no momento, pois você ainda não iniciou conjuntos de tarefa para esses serviços.

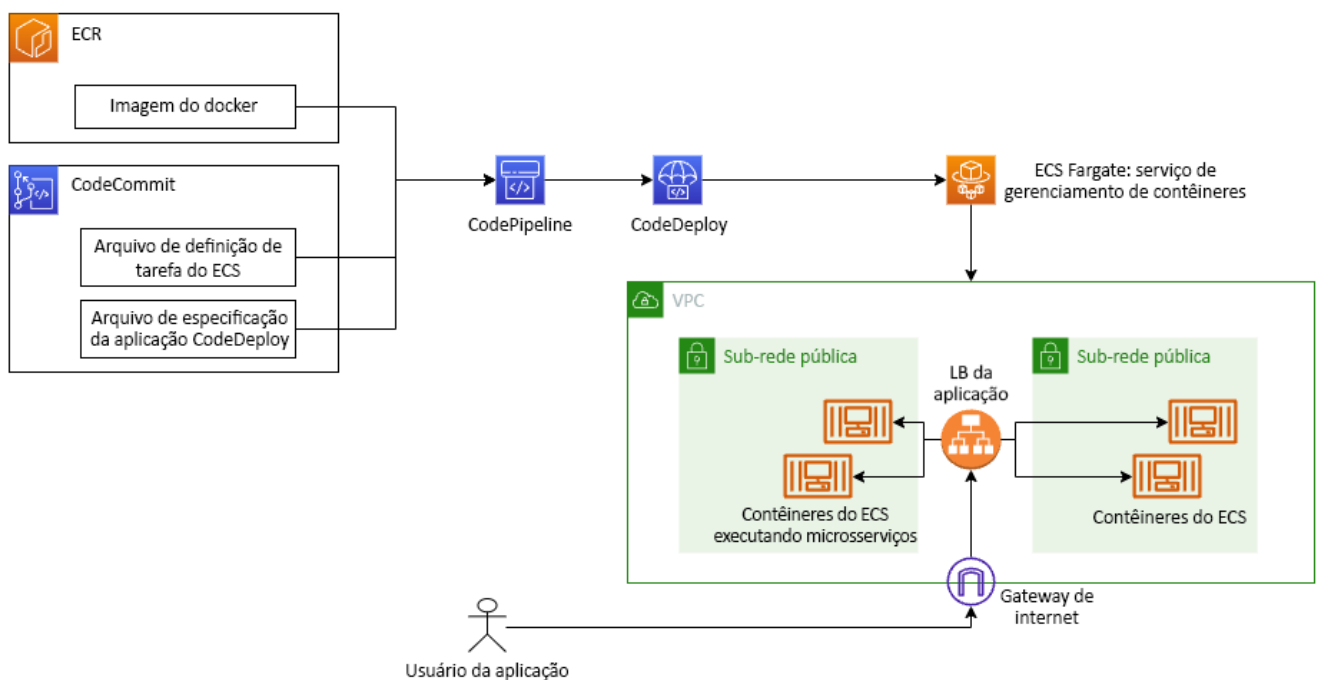
The screenshot shows the Amazon ECS console interface. At the top, there's a header for 'microservices-serverlesscluster' with buttons for 'Update cluster' and 'Delete cluster'. Below this is a 'Cluster overview' section with a table showing cluster details: ARN (microservices-serverlesscluster), Status (Active), CloudWatch monitoring (Default), and Registered container instances (-). Below the overview is a 'Services' section with a table showing service details: Draining (-), Active (2), Pending (-), and Running (-). At the bottom, there's a 'Services (2)' section with a table listing the services: 'employee-microservice' and 'customer-microservice'. Both services are in 'Active' status, have an ARN starting with 'arn:aws:ecs:us-...', and are of type 'REPLICA'. The 'Deployments and tasks' column shows '0/1 Tasks run...' for both services. The 'Task definition' column shows 'employee-microservice' and 'customer-microservice' respectively. The 'Revision' column shows '1' for both services.

Service name	Status	ARN	Service type	Deployments and tasks	Last deploy...	Task definition	Revision
employee-microservice	Active	arn:aws:ecs:us-...	REPLICA	0/1 Tasks run...	-	employee-microservice	1
customer-microservice	Active	arn:aws:ecs:us-...	REPLICA	0/1 Tasks run...	-	customer-microservice	1

## Fase 8: configurar o CodeDeploy e o CodePipeline

Agora que você definiu o Application Load Balancer, os grupos de destino e os serviços do Amazon ECS que abrangem a infraestrutura na qual você implantará seus microsserviços, a próxima etapa é definir o pipeline CI/CD para implantar a aplicação.

O diagrama a seguir ilustra o papel do pipeline na solução que você está criando.



*Descrição do diagrama:* o pipeline será invocado por atualizações do CodeCommit, onde você armazenou os arquivos de definição de tarefa do ECS e os arquivos AppSpec do CodeDeploy. O pipeline também pode ser invocado por atualizações a um dos arquivos de imagem do Docker que você armazenou no Amazon ECR. Ao ser invocado, o pipeline chamará o serviço do CodeDeploy para implantar as atualizações. O CodeDeploy tomará as ações necessárias para implantar as atualizações ao ambiente verde. Presumindo que não ocorram erros, o novo conjunto de tarefas substituirá o conjunto de tarefas atual.

## Tarefa 8.1: criar uma aplicação no CodeDeploy e grupos de implantação

[Voltar ao sumário](#)

Uma aplicação do CodeDeploy é uma coleção de grupos de implantação e revisões. Um grupo de implantação especifica o serviço do Amazon ECS, o balanceador de carga, o listener de teste opcional e dois grupos de destino. Um grupo também especifica quando redirecionar o tráfego para o conjunto de tarefas de substituição e quando terminar tanto o conjunto de tarefas original quanto a aplicação do Amazon ECS após uma implantação bem-sucedida.

1. Use o console do CodeDeploy para criar uma aplicação do CodeDeploy com o nome `microservices` que usa o Amazon ECS como plataforma de computação.

💡 **Dica:** consulte [Create an Application for an Amazon ECS Service Deployment \(Console\)](#) (Criar uma aplicação para uma implantação de serviço do Amazon ECS (Console) no *Guia do usuário do AWS CodeDeploy*).

❗ **Importante:** NÃO crie um grupo de implantação ainda. Você fará isso na próxima etapa.

2. Crie um grupo de implantação no CodeDeploy para o microserviço `customer`.
  - Na página de detalhes da aplicação `microservices`, selecione a guia **Deployment groups** (Grupos de implantação).
  - Escolha **Create deployment group** (Criar grupo de implantação) e configure o seguinte:
    - **Deployment group name** (Nome do grupo de implantação): insira `microservices-customer`.
    - **Service role** (Perfil de serviço): coloque o cursor na caixa de pesquisa e selecione o ARN de **DeployRole**.
    - Na seção **Environment configuration** (Configuração de ambiente):
      - **ECS cluster name** (Nome do cluster do ECS): escolha **microservices-serverlesscluster**.

- **ECS service name** (Nome do serviço do ECS): escolha **customer-microservice**.
  - Na seção **Load balancers** (Balanceadores de carga):
    - **Load balancer** (Balanceador de carga): escolha **microservicesLB**.
    - **Production listener port** (Porta do listener de produção): escolha **HTTP :80**.
    - **Test listener port** (Porta do listener de teste): escolha **HTTP:8080**.
    - **Target group 1 name** (Nome do grupo de destino 1): escolha **customer-tg-two**.
    - **Target group 2 name** (Nome do grupo de destino 2): escolha **customer-tg-one**.
  - Na seção **Deployment settings** (Configurações de implantação):
    - **Traffic rerouting** (Redirecionamento de tráfego): escolha **Reroute traffic immediately** (Redirecionar tráfego imediatamente).
    - **Deployment configuration** (Configuração de implantação): escolha **CodeDeployDefault.ECSAllAtOnce**.
    - **Original revision termination** (Término da revisão original): dias: **0**, horas: **0**, minutos: **5**
  - Clique em **Create deployment group** (Criar grupo de implantação).
3. Crie um grupo de implantação no CodeDeploy para o microserviço *employee*: Especifique as mesmas configurações que na etapa anterior, exceto o seguinte:
- **Deployment group name** (Nome do grupo de implantação): insira `microservices-employee`
  - **ECS service name** (Nome do serviço do ECS): escolha **employee-microservice**.
  - **Target group 1 name** (Nome do grupo de destino 1): escolha **employee-tg-two**.
  - **Target group 2 name** (Nome do grupo de destino 2): escolha **employee-tg-one**.

## Tarefa 8.2: criar um pipeline para o microserviço *customer*

[Voltar ao sumário](#)

Nesta tarefa, você criará um pipeline para atualizar o microserviço *customer*. Quando você definir o pipeline pela primeira vez, você configurará o CodeCommit como a origem e o CodeDeploy como o serviço responsável pela implantação. Depois, você editará o pipeline para adicionar o serviço do Amazon ECR como segunda origem.

Com uma implantação azul/verde do Amazon ECS, que você especificará nesta tarefa, você provisiona um novo conjunto de contêineres, no qual o CodeDeploy instala a versão mais recente de sua aplicação. O CodeDeploy, então, redireciona o tráfego do balanceador de carga de um conjunto de contêineres existente, que executa a versão anterior da sua aplicação, para o novo conjunto de contêineres, que executa a versão mais recente. Depois que o tráfego for redirecionado para os novos contêineres, os contêineres existentes poderão ser terminados. Com uma implantação azul/verde, você pode testar a nova versão da aplicação antes de enviar tráfego de produção a ela.

### Referências

- Os cursos AWS Academy Cloud Architecting e AWS Academy Cloud Developing incluem laboratórios práticos que exploram recursos do CodePipeline.
- [Guia do usuário do AWS CodePipeline](#)

1. No console do CodePipeline, crie um pipeline *customer* com as seguintes configurações:
  - **Pipeline name** (Nome do pipeline): insira `update-customer-microservice`.
  - **Service role** (Perfil de serviço): selecione o ARN para o **PipelineRole**.

- **Source provider** (Provedor de origem): escolha **AWS CodeCommit**.

- **Repository name** (Nome do repositório): escolha **deployment** (implantação).

📌 **Observação:** você definiu dois repositórios do CodeCommit. O repositório *deployment* (implantação) contém os arquivos de definição de tarefa do Amazon ECS e os arquivos AppSpec do CodeDeploy dos quais o pipeline precisará e que você vai escolher aqui.

- **Branch name** (Nome da ramificação): escolha **dev**.

📌 **Observação:** *pule* a etapa de criação.

- **Deploy provider** (Fornecedor de implantação): **Amazon ECS (Blue/Green)** [Amazon ECS (azul/verde)]

- **Region** (Região): US East (N. Virginia) [(Leste dos EUA (Virgínia do Norte))].

- **AWS CodeDeploy application name** (Nome da aplicação no AWS CodeDeploy): **microservices**

- **AWS CodeDeploy deployment group** (Grupo de implantação do AWS CodeDeploy): **microservices-customer**.

- Em **Definição de tarefa do Amazon ECS**:

- Defina um **SourceArtifact** com valor de `taskdef-customer.json`

- Em **AWS CodeDeploy AppSpec file** (Arquivo AppSpec do AWS CodeDeploy):

- Defina um **SourceArtifact** com valor de `appspec-customer.yaml`

📌 **Observação:** deixe os campos *Dynamically update task definition image* (Atualizar dinamicamente a imagem de definição da tarefa) *em branco* por enquanto.

📌 **Observação:** depois de criar o pipeline, ele começará a execução imediatamente e, por fim, falhará no estágio *Deploy* (Implantar). Ignore isso por enquanto e vá para a próxima etapa.

2. Edite o pipeline *update-customer-microservice* para adicionar outra *origem*.

- Na seção **Edit: Source** (Editar: origem), escolha **Edit stage** (Editar estágio) e adicione uma ação com os detalhes a seguir:

- **Action name** (Nome da ação): `Image`

- **Action provider** (Provedor da ação): Amazon ECR

- **Repository name** (Nome do repositório): **customer**

- **Image tag** (Tag da imagem): **mais recente**

- **Output artifacts** (Artefatos de saída): `image-customer`

3. Edite a ação *deploy* do pipeline *update-customer-microservice*.

- Edite o pipeline **update-customer-microservice**

- Na seção **Edit: Deploy** (Editar: implantar), escolha **Edit stage** (Editar estágio), adicione um artefato de entrada como descrito abaixo:

- No cartão **Deploy Amazon ECS (Blue/Green)** [Implantar o Amazon ECS (Azul/verde)], selecione o ícone de edição (lápis).

- Em **Input artifacts** (Inserir artefatos), escolha **Add** (Adicionar) e, depois, **image-customer**.

📌 **Observação:** agora *SourceArtifact* e *image-customer* estão listados como artefatos de entrada.

- Em **Atualizar dinamicamente a imagem de definição de tarefa**, em **Artefato de entrada com detalhes em imagem**, selecione **image-customer**.

- Em **Texto de espaço reservado na definição de tarefa**, insira `IMAGE1_NAME`.

**Análise:** lembre-se de que, em uma fase anterior, você inseriu um texto de espaço reservado *IMAGE1\_NAME* no arquivo *taskdef-customer.json* antes de enviá-lo ao CodeCommit. Na tarefa atual, você configurou a lógica que substituirá o texto de espaço reservado pelo nome real da imagem que a fase de origem do CopePipeline retornará.



## Tarefa 8.3: testar um pipeline CI/CD para o microsserviço *customer*

[Voltar ao sumário](#)

Nesta tarefa, você testará se o pipeline CI/CD para o microsserviço *customer* funciona conforme o esperado.

❗ **Importante:** se você está repetindo esta tarefa, confirme se todos os grupos de destino ainda estão associados ao Application Load Balancer. A seção [Reassociar grupos de destino com o balanceador de carga](#) do apêndice fornece mais detalhes.

1. Inicie a implantação do microsserviço *customer* do Amazon ECS no Fargate.

- Navegue até o console do CodePipeline.
- Na página **Pipelines**, clique no link do pipeline denominado **update-customer-microservice**.
- Para forçar um teste das configurações atuais do pipeline, selecione **Release change** (Lançar alteração) e, depois, **Release** (Lançar).

📌 **Observação:** ao invocar o pipeline, você criou uma nova revisão da definição da tarefa.

Espere até que as duas tarefas *Source* (Origem) exibam o status de *Succeeded - just now* (Êxito: agora mesmo).

- Na seção **Deploy** (Implantar), espere até o link **Details** (Detalhes) aparecer e clique nele.

Uma página do CodeDeploy é exibida em uma nova guia do navegador.

2. Observe o progresso no CodeDeploy.

- Role até a parte inferior da página e observe a seção **Deployment lifecycle events** (Eventos do ciclo de vida da implantação).

💡 **Dica:** se você vir um erro "Primary task group must be behind listener" (Grupo de tarefas principal deve estar atrás do listener), consulte a seção [Reassociar grupos de destino com o balanceador de carga](#) no apêndice.

Dentro de alguns minutos, se tudo foi configurado corretamente, todos os eventos do ciclo de vida de implantação terão êxito. Não espere que isso aconteça; vá para a próxima etapa. Deixe esta página aberta.

3. Carregue o microsserviço *customer* em uma guia do navegador e teste-o.

- Localize o valor **DNS name** do balanceador de carga **microservicesLB** e cole-o em uma nova guia do navegador.
- O microsserviço *customer* é carregado. Se não carregar, tente adicionar **:8080** ao fim do URL e tente novamente.

**Análise:** lembre-se de que o balanceador de carga tem dois listeners: um na porta 80 e outro na porta 8080. A porta 8080 é onde a tarefa de substituição definida executará nos primeiros cinco minutos. Portanto, se você carregar o URL :80 dentro dos primeiros cinco minutos, a página do microsserviço *customer* pode não carregar, mas você já verá a página em 8080. Após cinco minutos, o microsserviço deverá ser visível e estar disponível nas duas portas.

- No aplicativo web da cafeteria, selecione **List of suppliers** ou **Suppliers list** (Lista de fornecedores). A página de fornecedores carrega. Ela *não* deverá ter os botões de editar ou adicionar fornecedores, pois é uma página do cliente.

4. Observe as tarefas em execução no console do Amazon ECS.

- Navegue até o console do Amazon ECS.
- Na lista de clusters, clique no link para **microservices-serverlesscluster**.

Na guia **Serviços**, observe que o serviço *employee-microservice* será exibido. O status **Deployments and tasks** (Implantação e tarefas) mudará à medida que a implantação azul/verde avançar nos eventos do ciclo de vida.

- Selecione a guia **Tarefas**.

Aqui, você pode ver as tarefas que estão em execução. Pode haver mais de uma tarefa em execução por serviço que você definiu.

- Clique no link para uma das tarefas listadas. Pode ser que haja apenas uma.

Aqui, é possível ver os detalhes do contêiner e as informações de configuração, como endereços IP associados ao contêiner em execução.

5. Volte à página do CodeDeploy que está aberta em outra guia do navegador.

As cinco etapas da implantação terão tido êxito e a tarefa de substituição agora está atendendo ao tráfego.

6. Observe as configurações do balanceador de carga e do grupo de destino.

- No console do Amazon EC2, escolha **Grupos de destino**.

Você pode notar que o grupo de destino *customer-tg-two* não está mais associado ao balanceador de carga. Isso acontece porque o CodeDeploy está gerenciando as regras do listener do balanceador de carga e pode ter determinado que alguns dos grupos de destino não são mais necessários.

- Observe as regras do listener do HTTP:80.

A regra padrão mudou aqui. A regra padrão "se nenhuma outra regra se aplica", anteriormente apontada para *customer-tg-two*, agora aponta para *customer-tg-one*. Isso acontece porque o CodeDeploy gerenciou ativamente seu Application Load Balancer.

- Observe as regras do listener do HTTP:8080.

As duas regras ainda encaminham para os grupos de destino "um".

Parabéns! Você implantou com êxito um dos dois microsserviços ao Amazon ECS no Fargate usando um pipeline CI/CD.

## Tarefa 8.4: criar um pipeline para o microsserviço *employee*

[Voltar ao sumário](#)

Nesta tarefa, você criará um pipeline para o microsserviço *employee*.

1. Crie um pipeline para o microsserviço *employee* com as seguintes especificações:

- Nome do pipeline: `update-employee-microservice`
- ARN da função: **PipelineRole**
  - Provedor de origem: **AWS CodeCommit**
    - Nome do repositório: **deployment**
    - Nome da ramificação: **dev**
- Fornecedor de implantação: **Amazon ECS (Blue/Green)** [Amazon ECS (azul/verde)].
- Nome da aplicação no AWS CodeDeploy: **microservices**
- Grupo de implantação do AWS CodeDeploy: **microservices-employee**
- Definição de tarefa do Amazon ECS: **SourceArtifact**
  - Caminho: `taskdef-employee.json`
  - Arquivo AppSpec do AWS CodeDeploy: **SourceArtifact**

■ Caminho: `appspec-employee.yaml`

2. Adicione outra *origem* para o pipeline do microsserviço *employee*. Adicione uma ação com os seguintes detalhes:
  - Nome da ação: `Image`
  - Provedor da ação: **Amazon ECR**
  - Nome do repositório: **employee**
  - Tag da imagem: **mais recente**
  - Artefatos de saída: `image-employee`
3. Edite a ação do Amazon ECS (azul/verde) no estágio de implantação:
  - Adicione outro *artefato de entrada* e escolha **image-employee**.
  - Em **Atualizar dinamicamente a imagem de definição de tarefa**, em **Artefato de entrada com detalhes em imagem**, selecione **image-employee**.
  - Em **Texto de espaço reservado na definição de tarefa**, insira `IMAGE1_NAME`.

## Tarefa 8.5: testar o pipeline CI/CD para o microsserviço *employee*

[Voltar ao sumário](#)

Nesta tarefa, você testará um pipeline que você acabou de definir para o microsserviço *employee*.

❗ **Importante:** se você está repetindo esta tarefa, confirme se todos os grupos de destino ainda estão associados ao Application Load Balancer. A seção [Reassociar grupos de destino com o balanceador de carga](#) do apêndice fornece mais detalhes, caso necessário.

1. Inicie a implantação do microsserviço *employee* do Amazon ECS no Fargate.
  - Use o recurso *release change* para forçar um teste do pipeline.
  - Siga o progresso no CodeDeploy. Dentro de alguns minutos, se tudo foi configurado corretamente, todos os *eventos do ciclo de vida de implantação* terão êxito.
2. Carregue o microsserviço *employee* em uma guia do navegador.
  - Na guia do navegador onde o microsserviço está em execução, escolha **Administrator link** (Link de administrador).

Você será direcionado a uma página no formato <http://microserviceslb-UNIQUE-NUMBER.us-east-1.elb.amazonaws.com/admin/suppliers>.

O microsserviço *employee* carrega. Se não carregar, tente adicionar `:8080` após `amazonaws.com` no URL.

  - Selecione **List of suppliers** ou **Suppliers list** (Lista de fornecedores).

A página de fornecedores será exibida. Esta versão da página *não* terá os botões de editar ou adicionar fornecedores. Todos os links no aplicativo web da cafeteria agora funcionarão, pois você já implantou ambos os microsserviços.
3. Observe as tarefas em execução no console do Amazon ECS.

O status **Deployments and tasks** (Implantação e tarefas) mudará à medida que a implantação azul/verde avançar nos eventos do ciclo de vida.
4. Volte à página CodeDeploy para confirmar as cinco etapas da implantação terão tido êxito e a tarefa de substituição agora está atendendo ao tráfego.

Parabéns! Você implantou com êxito o microsserviço *employee* ao Amazon ECS no Fargate usando um pipeline CI/CD.

## Tarefa 8.6: observar como o CodeDeploy modificou as regras do listener do balanceador de carga

[Voltar ao sumário](#)

1. Observe as configurações do balanceador de carga e do grupo de destino.

- No console do Amazon EC2, escolha **Grupos de destino**.

■ **Observação:** se você já estava com a página aberta, atualize-a.

Observe que o grupo de destino *customer-tg-two* não está mais associado ao balanceador de carga. Isso acontece porque o CodeDeploy está gerenciando as regras do listener do balanceador de carga.

■ **Observação:** se você está repetindo esta etapa, os grupos de destino atualmente anexados e não anexados podem ser diferentes.

- Observe as regras do listener do HTTP:80.

A regra padrão mudou aqui. Para a regra padrão "se nenhuma outra regra se aplica", "encaminhar para o grupo de destino" anteriormente apontava para *customer-tg-two*, mas agora aponta para *customer-tg-one*.

- Observe as regras do listener do HTTP:8080.

As duas regras ainda encaminham para os grupos de destino "um".

## Fase 9: ajustar o código do microsserviço para fazer com que um pipeline execute novamente

Nesta fase, você verá os benefícios da arquitetura de microsserviços e o pipeline CI/CD que você criou. Você começará ajustando as regras do listener do balanceador de carga que estão relacionadas ao microsserviço *employee*. Você também atualizará o código fonte do microsserviço *employee*, gerará uma nova imagem do Docker e enviará essa imagem para o Amazon ECR, fazendo com que o pipeline execute e atualize a implantação da produção. Você também aumentará o número de contêineres que oferece suporte ao microsserviço *customer*.

### Tarefa 9.1: limitar o acesso ao microsserviço *employee*

[Voltar ao sumário](#)

Nesta tarefa, você limitará acesso ao microsserviço *employee* apenas para pessoas que tentarem conectar-se a ele a partir de um endereço IP específico. Ao limitar o IP de origem a um endereço IP específico, apenas usuários com acesso à aplicação desse IP podem acessar as páginas e editar ou excluir entradas de fornecedores.

1. Confirme se todos os grupos de destino ainda estão associados ao Application Load Balancer.

No console do Amazon EC2, verifique se os quatro grupos de destino ainda estão associados ao balanceador de carga. Reassocie grupos de destino conforme o necessário indo para a próxima etapa.

💡 **Dica:** para detalhes, consulte [Reassociar grupos de destino com o balanceador de carga](#) no apêndice.

2. Descubra seu endereço IPv4 público.

💡 **Dica:** um recurso que você pode usar para isso é o <https://www.whatismyip.com>.

3. Edite as regras para o listener do **HTTP:80**.

Para a regra que atualmente tem "IF Path is /admin/\*" nos detalhes, adicione uma segunda condição para direcionar o usuário para os grupos de destino apenas se o IP de origem da solicitação é seu endereço IP.

💡 **Dica:** para o IP de origem, cole seu endereço IPv4 público e adicione `/32`. A imagem a seguir mostra um exemplo:

Name tag	Priority	Conditions (If)
-	1	<ul style="list-style-type: none"><li>• Source IP is 172.125.76.41/32, AND</li><li>• Path Pattern is /admin/*</li></ul>

4. Edite as regras para o listener do **HTTP:8080**.

Edite as regras da mesma forma que você editou as regras para o listener do **HTTP:80**. Você quer que o acesso aos grupos de destino `employee` sejam limitados ao seu endereço IP.

## Tarefa 9.2: ajustar o UI para o microserviço *employee* e enviar a imagem atualizada para o Amazon ECR

[Voltar ao sumário](#)

Nesta tarefa, você ajustará os microserviços implantados.

1. Edite o arquivo `employee/views/nav.html`.

- Na linha 1, altere `navbar-dark bg-dark` para `navbar-light bg-light`
- Salve a alteração.

2. Para gerar uma nova imagem do Docker dos arquivos de origem microserviço *employee* que você modificou e para rotular a imagem, execute os comandos a seguir:

```
docker rm -f employee_1
cd ~/environment/microservices/employee
docker build --tag employee .
dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep
'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint
account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
echo $account_id
docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
```

3. Envie uma imagem atualizada para o Amazon ECR para que o pipeline *update-employee-microservice* seja invocado.

- No console do CodePipeline, navegue a página de detalhes para o pipeline *update-employee-microservice*. Deixe esta página aberta.
- Para enviar a imagem do novo microserviço *employee* para o Amazon ECR, execute os seguintes comandos no seu IDE do AWS Cloud9 IDE:

```
#refresh credentials in case needed
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
#push the image
docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
```

- Confirme se a saída é semelhante ao seguinte.

```
b20e9725db21: Pushed
715b0f96c609: Pushed
44141cf9260f: Layer already exists
d81d715330b7: Layer already exists
1dc7f3bb09a4: Layer already exists
dcaceb729824: Layer already exists
f1b5933fe4b5: Layer already exists
```

Ao menos uma camada deve indicar que foi enviada, o que indica que a imagem foi modificada desde que foi enviada ao Amazon ECR. Você também pode ver o repositório do Amazon ECR para confirmar o último carimbo de data/hora da imagem marcado como o mais recente.

## Tarefa 9.3: confirmar se o pipeline *employee* foi executado e o microsserviço atualizado

[Voltar ao sumário](#)

1. Observe os detalhes do pipeline *update-employee-microservice* no console do CodePipeline.

Observe que, quando você atualizou uma nova imagem do Docker no Amazon ECR, o pipeline foi invocado e executado. Observe que o pipeline pode levar um minuto ou dois para ver que a imagem do Docker foi atualizada antes da invocação do pipeline.

2. Observe os detalhes no console do CodeDeploy.

## Tarefa 9.4: testar o acesso ao microsserviço *employee*

[Voltar ao sumário](#)

Nesta tarefa, você testará o acesso ao microsserviço *employee*.

1. Teste o acesso às páginas do microsserviço *employee* em `http://<alb-endpoint>/admin/suppliers` e `http://<alb-endpoint>:8080/admin/suppliers` do mesmo dispositivo que você usou até agora no projeto. Substitua `<alb-endpoint>` pelo nome do DNS do balanceador de carga *microservicesLB*.

Ao menos uma das duas páginas deve carregar com êxito.

Observe que o banner com o título da página é de uma cor clara agora por conta da alteração que você fez ao arquivo `nav.html`. Páginas hospedadas pelo microsserviço *customer* ainda terão um banner escuro. Isso demonstra que, ao usar uma arquitetura de microsserviços, você pode modificar, de forma independente, a UI ou recursos de cada microsserviço sem afetar outros.

2. Teste o acesso às mesmas páginas de microsserviço *employee* de um dispositivo diferente.

Por exemplo, você pode usar seu telefone para conectar-se a partir da rede celular e não da mesma rede Wi-Fi usada pelo seu computador. Você quer que o dispositivo use um endereço IP diferente do seu computador para conectar-se à internet.

Você verá um erro 404 em qualquer página que carregar, e a página exibirá "Coffee suppliers" (Fornecedores de café) em vez de "Manage coffee suppliers" (Gerenciar fornecedores de café). Essa é a evidência de que você não conseguiu se conectar ao microsserviço *employee* de outro endereço IP.



💡 **Dica:** se você não tem outra rede disponível, execute o seguinte comando no terminal do AWS Cloud9: `curl http://<alb-endpoint>/admin/suppliers`. O endereço IP de origem da instância do AWS Cloud9 é diferente do IP de origem do navegador. O resultado incluirá `<p class="lead">Sorry, we don't seem to have that page in stock</p>`.

Isso prova que as regras atualizadas no listener do balanceador de carga estão funcionando conforme o esperado.

💡 **Dica:** se a atualização não funcionar conforme esperado, você pode ir até à pagina **Deployments** (Implantações) no console do CodeDeploy dentro de cinco minutos para interromper a implantação e reverter para a versão anterior. Então, você escolheria **Stop and roll back deployment** (Interromper e reverter implantação) e, depois, **Stop and rollback** (Interromper e reverter). Essa ação redirecionaria o tráfego de produção para o conjunto de tarefas original e excluiria o conjunto de tarefas de substituição. Você definiu a configuração de cinco minutos na fase 8, tarefa 1.

## Tarefa 9.5: dimensionar o microserviço *customer*

[Voltar ao sumário](#)

Nesta tarefa, você aumentará o número de contêineres que oferece suporte ao microserviço *customer*. Você pode fazer essa alteração sem que o pipeline *update-customer-microservice* execute.

1. Atualize o serviço *customer* no Amazon ECS.

- Execute o seguinte comando:

```
aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3
```

Uma resposta grande em formato JSON é retornada.

- Acesse a [Visualização dos serviços do Amazon ECS](#).

Deve ser semelhante à imagem a seguir.

Services	Tasks	Infrastructure	Metrics	Scheduled tasks	Tags																		
<div>Services (2) <a href="#">Info</a></div> <div><input type="text" value="Filter services by value"/> <span>All launch types</span> <span>All service types</span></div> <table><thead><tr><th><input type="checkbox"/></th><th>Service name</th><th>Status</th><th>ARN</th><th>Service type</th><th>Deployments and tasks</th></tr></thead><tbody><tr><td><input type="checkbox"/></td><td>employee-microservice</td><td>Active</td><td>arn:aws:ecs:us-...</td><td>REPLICA</td><td><div></div> 2/1 Tasks run...</td></tr><tr><td><input type="checkbox"/></td><td>customer-microservice</td><td>Active</td><td>arn:aws:ecs:us-...</td><td>REPLICA</td><td><div></div> 1/3 Tasks run...</td></tr></tbody></table>						<input type="checkbox"/>	Service name	Status	ARN	Service type	Deployments and tasks	<input type="checkbox"/>	employee-microservice	Active	arn:aws:ecs:us-...	REPLICA	<div></div> 2/1 Tasks run...	<input type="checkbox"/>	customer-microservice	Active	arn:aws:ecs:us-...	REPLICA	<div></div> 1/3 Tasks run...
<input type="checkbox"/>	Service name	Status	ARN	Service type	Deployments and tasks																		
<input type="checkbox"/>	employee-microservice	Active	arn:aws:ecs:us-...	REPLICA	<div></div> 2/1 Tasks run...																		
<input type="checkbox"/>	customer-microservice	Active	arn:aws:ecs:us-...	REPLICA	<div></div> 1/3 Tasks run...																		

Neste exemplo, *customer-microservice* exibe 1/3 das tarefas em execução porque você aumentou a contagem desejada de 1 para 3, mas os dois novos contêineres ainda estão sendo iniciados. Se aguardar por tempo suficiente, ao olhar para a guia **Tarefas**, você verá que os três contêineres executam para dar suporte ao microserviço *customer*. Isso demonstra como você pode dimensionar os microserviços de forma independente um do outro.

■ **Observação:** O *employee-microservice* pode exibir 2/1 tarefas em execução. Isso pode acontecer porque um conjunto de tarefas de substituição (que tem um contêiner) foi criado, mas o conjunto da tarefa original continua ativo pelos primeiros cinco minutos para o caso de você decidir reverter.


# Finalização da sessão

**Lembrete:** este é um ambiente de laboratório de longa duração. Os dados são retidos até que você use o orçamento alocado ou que a data de término do curso seja atingida (o que ocorrer primeiro).

Para preservar o orçamento quando encerrar o dia ou quando acabar de trabalhar ativamente na atribuição temporariamente, faça o seguinte:

1. Na parte superior da página, escolha  **Encerrar laboratório** e selecione  para confirmar que deseja encerrar o laboratório.

Uma painel de mensagem indica que o laboratório está terminando.

 **Observação:** a escolha de **Encerrar laboratório** neste ambiente *não* excluirá os recursos que você criou. Ele estará lá na próxima vez que você escolher Iniciar laboratório (por exemplo, em outro dia).

2. Para fechar o painel, escolha **Fechar** no canto superior direito.

## Apêndice

### Atualização de um contêiner de teste em execução no AWS Cloud9

[Voltar ao sumário](#)

Se precisar atualizar o código fonte de um microserviço *após* a execução do contêiner, você precisará: (1) interromper e excluir o contêiner; (2) criar uma nova imagem; e (3) executar a nova imagem. As sessões a seguir fornecem uma referência para a conclusão dessas etapas.

### Atualização de um contêiner *customer* em execução no AWS Cloud9

```
# Stop and delete the specified container (assumes that the container name is customer_1)
docker rm -f customer_1

# Must be in the directory that has the Dockerfile before you attempt to build a new image
cd ~/environment/microservices/customer

# Build a new image from the latest source files and overwrite any existing image
docker build --tag customer .

# Ensure the dbEndpoint variable is set in your terminal session
dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep
'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint

# Create and run a new container from the image (and pass the DB location to the container)
docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer

# URL to test the microservice running on the AWS Cloud9 test container
echo "Customer microservice test container running at http://$(curl ifconfig.me):8080"
```

### Atualização de um contêiner *employee* em execução no AWS Cloud9

```
# Stop and delete the specified container (assumes that the container name is employee_1)
docker rm -f employee_1
```

```
# Must be in the directory that has the Dockerfile before you attempt to build a new image
cd /home/ec2-user/environment/microservices/employee

# Build a new image from the latest source files and overwrite any existing image
docker build --tag employee .

# Ensure the dbEndpoint variable is set in your terminal session
dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep
'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint

# Create and run a new container from the image (and pass the DB location to the container)
docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee

# URL to test the microservice running on the AWS Cloud9 test container
echo "Employee microservice test container running at http://$(curl
ifconfig.me):8081/admin/suppliers"
```

## Referência: outros comandos do Docker

Para encontrar comandos que você precisa executar, consulte [Use the Docker Command Line](#) (Uso da linha de comando do Docker) na documentação do Docker.

Os comandos selecionados a seguir podem ser úteis:

```
docker container ls # List the running containers
docker rm -f <container> # Stop and delete the specified container
docker images # List the images
docker build --tag <tag-for-image> . # Create a new image from a Dockerfile (must be in the
same directory as the Dockerfile)
docker run -d --name <name-to-give-container> -p <port>:<port> <image-tag> # Run a
container from an image
docker ps # List the running Docker containers
```

## Referência: análise de rede e visualização de logs do Docker

```
# Helpful commands to see what is happening
sudo lsof -i :8080
sudo lsof -i :8081
ps -ef | head -1; ps -ef | grep docker

# Docker log viewing - returns container numbers
sudo ls /var/lib/docker/containers

# View the log for a container where you know the container number
sudo less /var/lib/docker/containers/<container-number>/*.log

# See the IP addresses that Docker containers use
docker inspect network bridge
```


## Reassociar grupos de destino com o balanceador de carga

[Voltar ao sumário](#)

Como o CodeDeploy gerencia as regras do Application Load Balancer, às vezes você verá que alguns grupos de destino não estão mais associados ao balanceador de carga. As etapas a seguir explicam como reassociar os grupos de destino.

1. Observe a configuração original que você fez para o listener do **HTTP:80**.

A imagem a seguir exibe a configuração original para este listener:

Name tag	Priority	Conditions (If)	Actions (Then) 
-	1	Path Pattern is /admin/*	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <a href="#">employee-tg-two</a>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>
<i>Default</i>	Last (default)	<i>If no other rule applies</i>	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <a href="#">customer-tg-two</a>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>


Se você implantou os microsserviços anteriormente com êxito e as configurações exibidas na imagem estão sendo usadas atualmente para a implantação (em que os grupos de destino em uso pelo listener do HTTP:80 têm “two” no nome), então, desta vez, você definirá o listener do HTTP:80 para usar grupos de destino que tenham “one” no nome. Efetivamente, o conjunto de tarefas verde se tornou o conjunto de tarefas azul e vice-versa.

2. Edite as regras do balanceador de carga do **HTTP:80** para garantir que ainda correspondam à imagem anterior. Se não corresponderem, ajuste-as:

- Navegue até o console do Amazon EC2.
- No painel de navegação, selecione **Balanceadores de carga**.
- Clique no link do balanceador de carga **microservicesLB**.
- Na guia **Listeners e regras**, clique no link **HTTP:80**.
- No painel **Regras do listener**, verifique se a regra **Padrão** encaminha para **customer-tg-one**. Se não acontecer dessa forma:
  - Selecione a regra.
  - No menu **Ações**, selecione **Editar regra**.
  - Defina **Encaminhar para o grupo de destino** como **customer-tg-one** e escolha **Salvar alterações**.
- Ainda na página do **HTTP:80**, no painel **Regras do listener**, verifique se a regra com **Path Pattern is /admin/** encaminha para **employee-tg-one**. Se não acontecer dessa forma:
  - Selecione a regra.
  - No menu **Ações**, selecione **Editar regra** e escolha **Próximo**.
  - No painel **Ações**, defina **Encaminhar para o grupo de destino** como **employee-tg-one**.
  - Selecione **Próximo** e, depois, **Salvar alterações**.

3. Observe a configuração original que você fez para o listener do **HTTP:8080**.

A imagem a seguir exibe a configuração original para este listener:

Name tag	Priority	Conditions (If)	Actions (Then) 
-	1	Path Pattern is /admin/*	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <a href="#">employee-tg-one</a>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>
<i>Default</i>	Last (default)	<i>If no other rule applies</i>	<b>Forward to target group</b> <ul style="list-style-type: none"><li>• <a href="#">customer-tg-one</a>: 1 (100%)</li><li>• Group-level stickiness: Off</li></ul>

Se você implantou os microsserviços anteriormente com êxito e as configurações exibidas na imagem anterior estão sendo usadas atualmente para a implantação (em que os grupos de destino em uso pelo listener do HTTP:8080 têm “two” no nome), então, desta vez, você definirá o listener do HTTP:8080 para usar grupos de destino que tenham “two” no nome.

4. Edite as regras do balanceador de carga do **HTTP:8080** para garantir que ainda correspondam à imagem anterior. Se não corresponderem, ajuste-as:
  - Navegue até o console do Amazon EC2.
  - No painel de navegação, selecione **Balanceadores de carga**.
  - Clique no link do balanceador de carga **microservicesLB**.
  - Na guia **Listeners e regras**, clique no link **HTTP:8080**.
  - No painel **Regras do listener**, verifique se a regra **Padrão** encaminha para **customer-tg-two**. Se não acontecer dessa forma:
    - Selecione a regra.
    - No menu **Ações**, selecione **Editar regra**.
    - Defina **Encaminhar para o grupo de destino** como **customer-tg-two** e escolha **Salvar alterações**.
  - Ainda na página do **HTTP:8080**, no painel **Regras do listener**, verifique se a regra com **Path Pattern is /admin/** encaminha para **employee-tg-two**. Se não acontecer dessa forma:
    - Selecione a regra.
    - No menu **Ações**, selecione **Editar regra** e escolha **Próximo**.
    - No painel **Ações**, defina **Encaminhar para o grupo de destino** como **employee-tg-two**.
    - Selecione **Próximo** e, depois, **Salvar alterações**.
5. Confirme se todos os grupos de destino estão reassociados.
  - No painel de navegação do console do Amazon EC2, escolha **Grupos de destino**.
  - Confirme se os quatro grupos de destino agora estão associados ao balanceador de carga *microservicesLB*.

❗ **Importante:** cada vez que você executar os pipelines, comece por confirmar que os quatro grupos de destino estão associados aos listeners do balanceador de carga e ajuste conforme necessário.

Em um ambiente de produção, você pode escolher automatizar as etapas que acabou de concluir nos locais em que deseja verificar e ajustar a configuração do Application Load Balancer antes de cada execução de um pipeline. O CodeDeploy fornece um recurso chamado AppSpec *hooks*, que você pode usar para criar uma função do AWS Lambda com código para realizar tarefas como essa. Depois, você pode especificar que o hook seja executado antes, durante ou após um evento de ciclo de vida específico ao referenciá-lo na definição do AppSpec. Como criar uma função Lambda personalizada e como usar esse recurso está além do escopo deste projeto. Para obter mais informações, consulte a [seção AppSpec 'hooks'](#) no *Guia do usuário do AWS CodeDeploy*.

## Dicas de solução de problemas

[Voltar ao sumário](#)

**Problema:** registros do banco de dados não são exibidos no aplicativo web ou microserviço.

Verifique o seguinte:

- Se isso acontecer quando você implantar no Amazon ECS, procure os detalhes do erro no Amazon CloudWatch Logs. Tente reiniciar o banco de dados do Amazon RDS.
- Se isso acontecer quando você estiver executando um contêiner na instância do AWS Cloud9, verifique se sua instância do AWS Cloud9 está executando na *Public Subnet1*.

**Problema:** configuração de ação inválida

Verifique o seguinte:

- O artefato não pode ser maior que 3 MB. Veja o bucket **codepipeline-xxxx** no Amazon S3. Navegue até o prefixo (pasta) **update-customer-micr/SourceArti** ou **update-employee-micr/SourceArti**. Garanta que nenhum objeto seja maior de 3 MB. Apenas envie arquivos para o repositório do CodeCommit se eles foram mencionados nessas instruções. Evite publicar arquivos além dos de definição de tarefa e AppSpec neste repositório.

**Problema:** uma mensagem em **Implantações e tarefas** no console do Amazon ECS diz *service....-microservice is unable to consistently start tasks successfully* (serviço... - o microsserviço não pôde iniciar as tarefas consistentemente).

Verifique o seguinte:

- Confirme se todos os grupos estão associados ao balanceador de carga.

**Problema:** o status de implantação do CodeDeploy está preso na etapa 1 ou a aplicação não funciona quando implantada no Amazon ECS.

Verifique o seguinte:

- No console do Amazon ECS, clique no link de **microservices-serverlesscluster**. Escolha o nome do serviço (**employee-microservice** ou **customer-microservice**) que está travado. Depois, escolha a guia **Implantações e eventos**. Essa visualização mostra o progresso do CodeDeploy ao implantar o conjunto de tarefas. A guia **Configuração e tarefas** também pode fornecer informações úteis por mostrar se alguma tarefa (contêineres) realmente foi iniciada.
- No console do AWS CloudTrail, no painel de navegação, selecione **Histórico de eventos**. Procure eventos que possam ter resultado em erros. Para observar detalhes de um evento, selecione o link com o **Nome do evento**.
- No console do CloudWatch, no painel de navegação, selecione **Logs** e, depois, **Grupos de logs**. Clique no link para o grupo de logs **awslogs-capstone**. Para ver os logs e detalhes, clique no link **Fluxo de logs** para ver o horário do evento mais recente.