06

Deploy no Heroku

Transcrição

[00:00] Olá, bem-vindos de volta ao curso de Spring Boot na Alura. Agora que aprendemos como funciona o Docker com uma aplicação no Spring Boot, geramos a imagem da nossa aplicação, criamos um contêiner e executamos esse contêiner localmente, ainda vamos continuar nesse assunto.

[00:17] Só que veremos também agora outra abordagem. É provável que você não vá rodar o contêiner local, no próprio computador. Na prática você vai jogar isso em algum servidor, que provavelmente também pode ser um servidor na nuvem, utilizando algum provedor Cloud, como Amazon, Heroku, DigitalOcean, Azure, dentre outros.

[00:37] E nesse vídeo vamos aprender justamente isso, vamos simular a utilização de um servidor na nuvem para rodar o contêiner na nuvem. No caso, nós utilizaremos o Heroku.

[00:49] O Heroku é um dos provedores de Cloud que existem e ele dá suporte para aplicações Java e para aplicações utilizando o Docker. E ele também tem a opção de você criar uma conta gratuita e rodar um contêiner Docker com alguns recursos limitados, mas sem que você tenha que pagar nada. Então vai ser a nossa escolha.

[01:11] Já estou assumindo que você tem uma conta no Heroku. Caso você não tenha, você vai precisar criar uma conta no site do Heroku, “heroku.com”. Tem o processo para você criar a sua conta, é bem tranquilo. E também tem o processo para você adicionar o CLI, a ferramenta de linha de comandos para você rodar os comandos do Heroku diretamente na sua máquina.

[01:34] Considerando que você já fez isso, vamos então ver como fazer o Deploy e rodar esse contêiner Docker no Heroku. Eu já deixei aberto, no próprio site do Heroku tem um artigo onde ele ensina como fazer o Deploy de contêiners Docker.

[01:53] Tem todo o passo a passo com os comandos que teremos que executar. Vamos seguir exatamente esses comandos. O primeiro passo é que ele pede para fazer *login*. Precisamos rodar esse comando no nosso prompt. Estou no prompt, já estou no diretório da nossa aplicação. Vou rodar esse comando `heroku *login*` .

[02:13] Então ele fala que vai precisar abrir o browser para nós nos autenticarmos. Então você dá um "Enter" ou digita qualquer tecla e ele vai abrir uma aba no navegador.

[02:23] E ele vai entrar numa tela de *login* do Heroku. Você clica no botão "Log In", ele se autentica, devolve uma mensagem e fala que já podemos fechar essa aba e voltar para o terminal. E ponto, ele já está autenticado, no caso usando a minha conta.

[02:37] E o próximo passo é que precisamos também fazer *login* usando esse módulo do contêiner: `heroku contêiner:*login*` . É para fazer *login* na parte de contêiners do registro que o próprio Heroku mantém. Então ele vai se autenticar. E autenticou com sucesso.

[02:55] O próximo passo que ele fala é que você precisa clonar uma aplicação, com a aplicação que você vai usar de exemplo. Nós vamos pular esse passo porque já estamos no repositório, já temos a aplicação, não vamos clonar. Ela já é o próprio diretório que temos, é um repositório do Git, no caso.

[03:11] Então o próximo passo seria rodar o comando para criar uma aplicação no Heroku. É o comando `heroku create` . Se você der um "Enter" agora ele gera um nome aleatório para a sua aplicação. Então eu vou rodar o comando

`heroku create forum`, eu quero que ele crie a aplicação no próprio Heroku com o nome de "forum".

[03:29] Vou criar essa aplicação, vamos ver se vai dar certo. E ele fala que já existe o nome "forum", então não conseguimos utilizar esse nome. Então vou rodar o comando e pedir para ele criar com o nome "aluna-forum": `heroku create alura-forum`. Talvez dê conflito porque tem que ser um nome único. Agora deu certo, ele criou a aplicação "alura-forum".

[03:51] E o próximo passo seria para fazer o *push*, para subir o contêiner e tudo mais. Porém, precisamos fazer mais uma configuração. Como nós não fizemos o passo do clone, não clonamos um repositório, então o Heroku não conhece o nosso repositório. Precisamos rodar outro comando para configurar justamente isso.

[04:10] Vou só limpar o console. E precisamos rodar o comando `heroku git:remote –a alura-forum`. Com isso vamos ensinar para o Heroku a adicionar, configurar a nossa aplicação no Git.

[04:31] Feito isso, já estamos prontos. Só que antes de rodar o próximo passo, que é para fazer o *push*, precisamos fazer algumas configurações na nossa aplicação.

[04:41] Como nossa conta no Heroku é gratuita, ela tem uma limitação de recursos, principalmente de memória. Então se não passarmos uma configuração de memória do nosso contêiner, ele pode extrapolar esse limite.

[04:54] Então vamos no Docker file. Na linha do Entrypoint depois de `"java"` vamos passar mais um parâmetro, que é `"-Xmx512m"`. Esse é um parâmetro da JVM para limitar o tamanho da memória RAM.

[05:09] Com isso eu estou falando para o Java que quando ele for executar, ele deve rodar com no máximo 512 MB de memória. Esse é o limite que tem disponibilizado gratuito no Heroku.

[05:17] Então se não passarmos esse parâmetro, é provável que na hora em que o Java for rodar nossa aplicação com o Spring Boot ele extrapole esse valor em alguns momentos. Então eu estou limitando o tamanho máximo para 512 MB.

[05:29] Além disso, no `application-prod.properties` precisamos adicionar mais uma propriedade: `server.port=${PORT}` . Porque é o Heroku que vai passar como uma variável de ambiente. Teremos que configurar a porta que vai ser utilizada no Heroku, e ela vai ser passada com esse comando para a nossa aplicação no Spring Boot.

[05:49] Se não configuramos isso pode dar problema com a porta que o Heroku vai atribuir à nossa aplicação.

[05:55] São essas duas configurações que precisamos fazer, passar a porta no `application-prod.properties` e no Docker file passar um limite de memória RAM.

[06:04] Feito isso já poderíamos voltar para o Heroku e rodar comando do *push*. Porém, lembra que estamos simulando um ambiente de produção? Então precisamos passar as variáveis de ambiente. Passar o *profile* de produção, passar aquelas variáveis de ambiente, como fizemos no último vídeo.

[06:18] Só que no último vídeo nós rodamos diretamente pelo prompt, com `docker run -e` e as variáveis. Só que agora vamos rodar pelo Heroku, então não é pela linha de comando. Essa configuração tem que ser feita diretamente no Heroku.

[06:35] Então podemos entrar no site do Heroku, "dashboard.heroku.com/apps/alura-forum", com o nome da aplicação. Eu já estou autenticado, então ele não vai me pedir autenticação. Ele vai abrir a página de configurações da aplicação para o Heroku.

[06:58] Vamos clicar na última aba, chamada "Settings". Nessa aba tem um campo chamado "Config Vars". Se você clicar em "Reveal Config Vars" ele

revela as variáveis, e não tem nenhuma variável. Então precisamos passar todas as variáveis de ambiente aqui.

[07:12] Uma delas é a porta. Então tem a variável chamada “PORT”, que vai no campo “KEY”, e o valor vai ser “8080”. Eu quero que ele rode na porta 8080. Então o Heroku vai passar isso para nossa aplicação.

[07:23] A outra é “SPRING_PROFILES_ACTIVE” e o valor é “prod”, para passarmos que o *profile* ativo para o Spring é o de produção.

[07:33] E tem aquelas outras variáveis. Tem a URL do Datasource: “FORUM_DATASOURCE_URL”. E eu vou só copiar o valor do `application.properties`. No exemplo do curso eu estou usando o mesmo Datasource do ambiente de desenvolvimento. No caso de vocês, o ambiente de produção vai ser diferente. Mas só para agilizar.

[07:59] Outra variável de ambiente que precisamos é “FORUM_DATASOURCE_USERNAME”, e o valor no nosso caso é aquele “sa”, do H2.

[08:10] Outra variável: “FORUM_DATASOURCE_PASSWORD” e o valor no nosso caso vai ficar vazio.

[08:17] E aquela propriedade “FORUM_JWT_SECRET”, que é a senha do JSON Web Token. E no exemplo eu estava usando “123456”, então eu vou até copiar a mesma senha do `application.properties` de desenvolvimento. Mas no ambiente de produção você vai utilizar um outro valor para essa variável de ambiente.

[08:38] Essas são as nossas variáveis de ambiente. Então no caso do Heroku nós configuramos direto pela interface gráfica essas variáveis de ambiente, e não pela linha de comando, não pelo Docker file. Isso fica no próprio Heroku.

[08:51] Feito isso, já configurei as variáveis de ambiente, já fiz os ajustes no `application.properties` de produção, no Docker file para limitar a memória, está tudo pronto.

[09:03] Agora eu posso voltar no prompt de comandos, voltar naquele tutorial do Heroku e rodar o comando para ele fazer o *push* da nossa imagem para o Heroku.

[09:14] Então vou colar o comando `heroku contêiner:push web`. "Web" é o módulo Web que tem no Heroku. Ele vai subir a aplicação como uma aplicação Web.

[09:26] Ele vai fazer o Build da aplicação local, vai gerar nossa imagem, parecido com o último vídeo que geramos uma imagem localmente. Ele vai fazer a mesma coisa, vai fazer o *push* e vai enviar isso para o servidor do Heroku. Ele terminou, deu certo.

[09:42] Na sequência, o comando que vamos rodar é `heroku contêiner:release web`. E quando eu rodar esse comando ele vai fazer o *release*, vai criar e inicializar o contêiner.

[10:03] Terminando, eu já posso abrir a aplicação. E tem até o comando `heroku open` para facilitar a vida. Com esse comando ele vai abrir uma aba no navegador e vai entrar no endereço da nossa aplicação, que é "alura-forum.herokuapp.com".

[10:19] Inclusive, quando vocês forem criar a aplicação de vocês, vocês vão ter que usar outro nome, porque o "alura-forum" já vai estar em uso, e esse nome é único. É único globalmente, não só na sua conta, em todas as aplicações do Heroku inteiro não pode ter dois nomes repetidos. Então escolha um nome aleatório, diferente.

[10:37] Então ele vai inicializar. Demora um pouco para inicializar. Ele pode dar uma tela de erro. E para testarmos nós entramos no endereço "/topicos".

Lembra que temos esse endereço na nossa aplicação, que devolve o JSON? Se der algum problema ele até sugere rodar o comando `heroku logs --tail`.

[10:58] Vou rodar esse comando. E ele vai ficar acompanhando o log do servidor. Vamos ver se vai dar algum problema. Parece que deu algum erro. Ele disse que não achou a variável "FORUM_DATABASE_URL". Vou ver se ele rodou para um ambiente de produção.

[11:30] Vou dar uma olhada na interface. Vou conferir no `application.properties` se não peguei errado o nome dessas variáveis. Eu copiei errado. Copiei como "DATASOURCE" e o correto é "DATABASE". Vou até copiar do `application.properties` para não ter problema.

[11:52] Vou clicar no ícone do lápis. E ele só deixa alterar o valor. Então eu vou apagar as variáveis de URL, username e password e adicionar de novo.

[12:20] Então a "FORUM_DATABASE_USERNAME" passo com o valor "sa"; a "FORUM_DATABASE_PASSWORD" deixo com o valor vazio; e a "FORUM_DATABASE_URL" passo com o valor "jdbc:h2:mem:alura-forum". Então eu tinha digitado errado as variáveis.

[12:44] E quando você muda essas variáveis o Heroku detecta e ele reinicia a aplicação automaticamente. Agora ele vai reiniciar, vai tentar fazer o Deploy novamente.

[12:56] Já imprimiu que leu o *profile* de produção. Então ele leu corretamente a variável "SPRING_PROFILES_ACTIVE" com o valor de "prod".

[13:04] E ele vai começar o processo de Deploy da aplicação. Então ele vai carregar, vai inicializar. Pode ser que demore um pouco. Se der certo ele vai ler também as variáveis de ambiente, vai substituir no `application.properties`. Agora ele pegou certo a variável da Database URL, então acho que vai funcionar.

[13:25] E para testar nós entramos no navegador na URL/topicos, que é aquele *endpoint* público, que não precisa estar autenticado e devolve o JSON com todos os tópicos cadastrados no banco de dados. Lembra que no nosso projeto tem um arquivo `data.sql` que tem alguns tópicos pré-definidos, que sempre são inseridos na base de dados.

[13:44] Vamos só esperar ele finalizar. Deu problema de novo, dessa vez com a porta, o problema da porta que eu tinha comentado que pode acontecer.

[13:55] Vou conferir, tem a variável "PORT" com o valor "8080". Vamos ver no `application-prod.properties`. Se não me engano eu me esqueci de passar também no Docker file.

[14:16] Então no Docker file, junto com o "-Xmx", passamos mais um parâmetro: `"-Dserver.port=${PORT}"`. Tem que passar esse parâmetro da porta.

[14:33] E como eu mexi no Docker file, vou dar um "Ctrl + C" para ele parar o log. Tem que rodar aquele comando do *push* de novo. Ele vai gerar uma nova imagem da nossa aplicação, já que teve alteração no código fonte.

[14:47] Ele vai gerar uma nova imagem, vai fazer o *push* para o servidor. E assim que ele finalizar daremos o comando do *release* para ele recarregar.

[14:55] Vamos só aguardar alguns segundos. Então assim que ele terminar rodamos o comando `heroku contêiner:release web`.

[15:08] É bom acontecer esses problemas, porque é provável que você vá passar por esses problemas, e assim já vemos ao vivo como resolver cada um deles.

[15:16] Essa parte é um pouco chata, porque isso é totalmente específico do Heroku. Cada provedor de Cloud vai ter um funcionamento distinto.

[15:26] Vou rodar de novo o comando de *release*. E enquanto ele vai subindo eu vou rodar o comando `heroku logs --tail` para irmos acompanhando o log. Então ele está inicializando, vamos acompanhar o log enquanto ele inicializa o servidor.

[15:42] Ele pegou agora o server PORT, vai pegar da variável de ambiente. Pegou a variável de ambiente prod. Vai ler aquelas outras variáveis de ambiente, com a URL do Datasource, o usuário e senha. A URL já pegou.

[16:03] Vai pegar todas as outras variáveis corretamente. Está inicializando, vai inicializar os repositories, disse que inicializou a aplicação normalmente.

[16:15] Vou ver se ele imprimiu a parte da porta. Ele inicializou em outra porta, 54048. Ele não inicializou naquela nossa porta.

[16:31] Mas vamos ver se vai dar problema no navegador. Ele pegou outra porta, mas a que ele pegou, a porta que foi gerada pelo Heroku é a porta que passamos para o servidor. Por isso tem que ter a propriedade `"-Dserver.port=${PORT}"`, porque às vezes sai outra porta distinta, então pode dar problema. E carregou. Entrou certo, trouxe o nosso JSON.

[16:59] É um pouco chato porque tem que configurar essas variáveis de ambiente específicas do Heroku. Tem o passo a passo de como você configura para criar o contêiner, subir o contêiner no Heroku; tem as modificações que tem que fazer no Docker file, no `application-prod`. É a parte chata, cada provider funciona de um jeito. Mas funcionou corretamente.

[17:20] E esse foi um exemplo do que você pode fazer em vez de rodar na sua máquina, rodar no Heroku. Então só relembrando que você precisa ter uma conta no Heroku e precisa ter instalado o CLI para você conseguir digitar esses comandos Heroku no prompt do seu terminal.

[17:36] Mas tem tudo certo na documentação do Heroku como fazer a conta e a instalação do CLI, é bem tranquilo.

[17:42] Espero que você tenha gostado do vídeo, tenham aprendido como fazer o Deploy utilizando o Heroku, com o Docker mesmo. Nos vemos no próximo vídeo. Um abraço e até lá.