06

Alura Books no Docker Cloud

Transcrição

Subimos a nossa primeira aplicação, um exemplo, no Docker Cloud, então agora vamos ver como fazemos para subir a aplicação que criamos ao longo do curso, o **alura-books**.

Para tal, queremos subir diversos serviços de uma vez, e para isso poderíamos ir no Docker Cloud e criar um a um, mas seria um trabalho bem manual, então vamos ver um jeito de subir o **docker-compose.yml** para o Docker Cloud, e baseado nele os serviços são criados.

StackFile

Não existe exatamente essa opção no Docker Cloud, mas tem algo bem parecido, que são os ***Stacks***, que é a última opção que falta vermos do menu do Docker Cloud.

Stacks é como se fosse um conjunto de serviços, que criamos para trabalharem juntos, bem semelhante do **Docker Compose**. Para criar um *stack*, no menu lateral da esquerda, clicamos em ***Stacks*** e logo em seguida no botão ***Create***.

Damos um nome a ele, e nos será pedido um arquivo, um **StackFile**, que é um arquivo bem parecido ao **docker-compose.yml**, mas com algumas limitações, que não fazem muito sentido quando estamos trabalhando com o **Docker Cloud**.

Construindo o StackFile

Então, no nosso projeto, vamos criar este arquivo. Seu tipo é o mesmo que já vimos, YAML, e o chamaremos de **docker-cloud.yml**, que será o nosso **StackFile**. Para iniciar a sua configuração, vamos copiar o serviço do NGINX do **docker-compose.yml**, para vermos o que precisa e o que não precisa neste tipo de arquivo:

```
nginx:
    build:
        dockerfile: ./docker/nginx.dockerfile
        context: .
    image: douglasq/nginx
    container_name: nginx
    ports:
        - "80:80"
    networks:
        - production-network
    depends_on:
        - "node1"
        - "node2"
        - "node3"
```

Primeiramente, não precisamos colocar a versão e nem especificar que estamos declarando serviços, pois quem estiver declarado no primeiro nível (sem espaço algum) será considerado um serviço. Também não precisamos falar para ele construir a nossa imagem, pois o Docker Cloud espera que a mesma já esteja criada e já esteja no **Docker Hub**, então não precisamos mandar o arquivo para ele fazer o *build*, só precisamos dizer qual é o endereço da imagem, que é passado com a propriedade `image`:

```
nginx:
    image: douglasq/nginx
    container_name: nginx
    ports:
        - "80:80"
    networks:
        - production-network
    depends_on:
        - "node1"
        - "node2"
        - "node3"
```

Uma coisa importante é que, se estamos utilizando uma imagem própria, ela precisa estar com a *tag* `latest`, pois se não especificarmos a versão da imagem que queremos utilizar, a imagem que será utilizada é a `latest`.

Outra coisa que não faz muito sentido é o `container_name`, pois o próprio Docker Cloud batiza o *container*:

```
nginx:
    image: douglasq/nginx
    ports:
        - "80:80"
    networks:
        - production-network
    depends_on:
        - "node1"
        - "node2"
        - "node3"
```

As portas fazem sentido nós termos, pois queremos abrir uma porta externa para acessar o NGINX. E também não temos `networks`, que são criadas e administradas pelo Docker Cloud, então ele ainda não tem isso, pode ser que no futuro isso seja implementado:

```
nginx:
    image: douglasq/nginx
    ports:
        - "80:80"
    depends_on:
        - "node1"
        - "node2"
        - "node3"
```

E ele também não tem o `depends_on`, mas daqui a pouco veremos como resolver isso. Então o serviço do **nginx** ficará assim:

```
nginx:
    image: douglasq/nginx
    ports:
        - "80:80"
```

Então, para subir um serviço no Docker Cloud, basicamente dizemos qual imagem que iremos utilizar e qual porta queremos abrir, pelo menos para um serviço simples como o nosso. Então, podemos extrair os outros serviços, com o arquivo ficando

assim;

```
nginx:
    image: douglasq/nginx
    ports:
        - "80:80"

mongodb:
    image: mongo

node1:
    image: douglasq/alura-books
    ports:
        - "3000"

node2:
    image: douglasq/alura-books
    ports:
        - "3000"

node3:
    image: douglasq/alura-books
    ports:
        - "3000"
```

Especificando a ordem dos containers

Agora, como fazemos para especificar que um *container* deve subir antes de outro? Outra coisa, se eles não estão na mesma rede, como eles vão se comunicar?

Antes, fazíamos os *containers* enxergarem um ao outro colocando-os na mesma rede, e também dizíamos que um *container* dependia de outro através do `depends_on`. Resolvemos essas duas coisas com a propriedade `links`, que faz um *container* enxergar, trocar dados com outro. E não faz sentido nós fazermos o *link* com um *container* que ainda não foi construído, então quando definimos o `links`, acabamos definindo uma certa prioridade de carregamento:

```
nnginx:
    image: douglasq/nginx
    ports:
        - "80:80"
    links:
        - "node1"
        - "node2"
        - "node3"
mongodb:
    image: mongo

node1:
    image: douglasq/alura-books
    ports:
        - "3000"
    links:
        - "mongodb"

node2:
```

```
    image: douglasq/alura-books
    ports:
        - "3000"
    links:
        - "mongodb"

node3:
    image: douglasq/alura-books
    ports:
        - "3000"
    links:
        - "mongodb"
```

Subindo os serviços

Agora, voltando aos ***Stacks***, colamos o conteúdo deste arquivo e clicamos em ***Create & Deploy***. Na interface, conseguimos ver os serviços subindo, primeiro o **mongodb**, depois os três de **node** e logo após o **nginx**.

Quando o *deploy* terminar, e o estado do **ALURA-BOOKS** estiver em ***RUNNING***, vamos até a sessão ***Endpoints***, e clicamos no link gerado para acessarmos o *container* do **nginx**.

A página do **Alura Books** que já conhecemos é aberta, então nós criamos os livros adicionando a rota `/seed` e voltamos à tela inicial, assim veremos todos os livros, exatamente como estava funcionando anteriormente, então conseguimos fazer o *deploy* dos nossos cinco serviços no **Docker Cloud** de um jeito bem fácil.

Podemos conferir os *containers* criados acessando ***Containers***, no menu lateral da esquerda. Lá, temos acesso a cada um dos *logs* dos *containers* e inclusive, ao carregar diversas vezes a página da aplicação, conseguimos ver que o *load balancer* está distribuindo corretamente a carga entre os três *containers*, assim como estava na nossa máquina.

Então, trabalhando com o **Docker Cloud**, temos a vantagem de não termos que lidar com a AWS, DigitalOcean, etc, já que ele faz tudo por debaixo dos panos para nós, bastando apenas seguirmos as suas regras e configurações.

Por último, os links que acessamos são bem difíceis de se decorar, mas nós podemos configurar um DNS, que pode ser visto na sua documentação (https://docs.docker.com/docker-cloud/apps/stack-yaml-reference/), que como já falamos é muito completa, sempre podendo nos auxiliar.