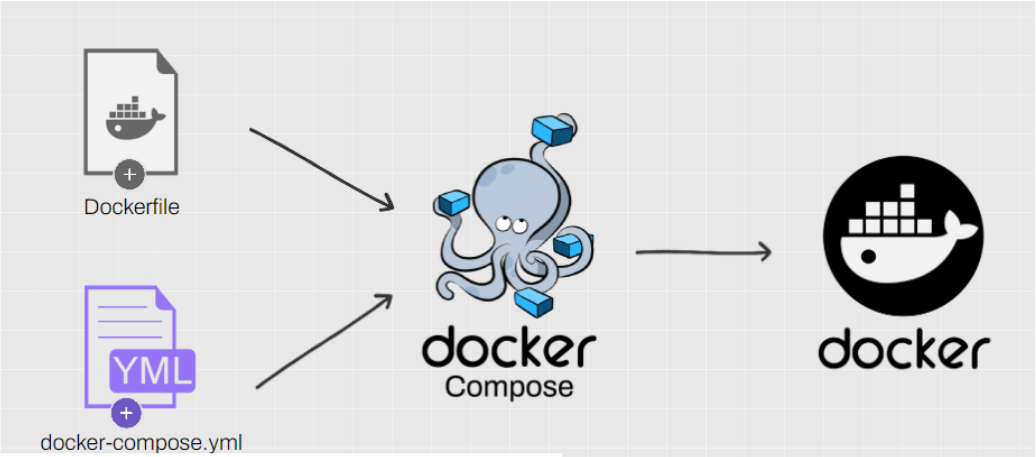
Spring Cloud Gateway com o Docker Compose

Recapitulando

Já vimos como usar o Docker conjuntamente com o Docker Compose para executar em containers nossos microsserviços desenvolvidos com Java, e também com o Eureka Server. Agora, vamos continuar trabalhando com o Docker para executar mais componentes de nosso ecossistema de microsserviços.

Como já sabemos, a primeira coisa a ser feita é criar o arquivo Dockerfile. Nele, iremos indicar a localização do arquivo **.jar** e a porta em que nossa aplicação será executada (a mesma porta configurada no **application.properties**). Por exemplo:

```
FROM adoptopenjdk/openjdk11:alpine-jre
ARG JAR_FILE=spring-boot-web.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","app.jar"]
EXPOSE 8085
```



Continuamos trabalhando no arquivo **docker-compose.yml**, já abordado na primeira parte da aula de Docker. Devemos adicionar um novo service, neste caso para o gateway. Ele ficaria assim:

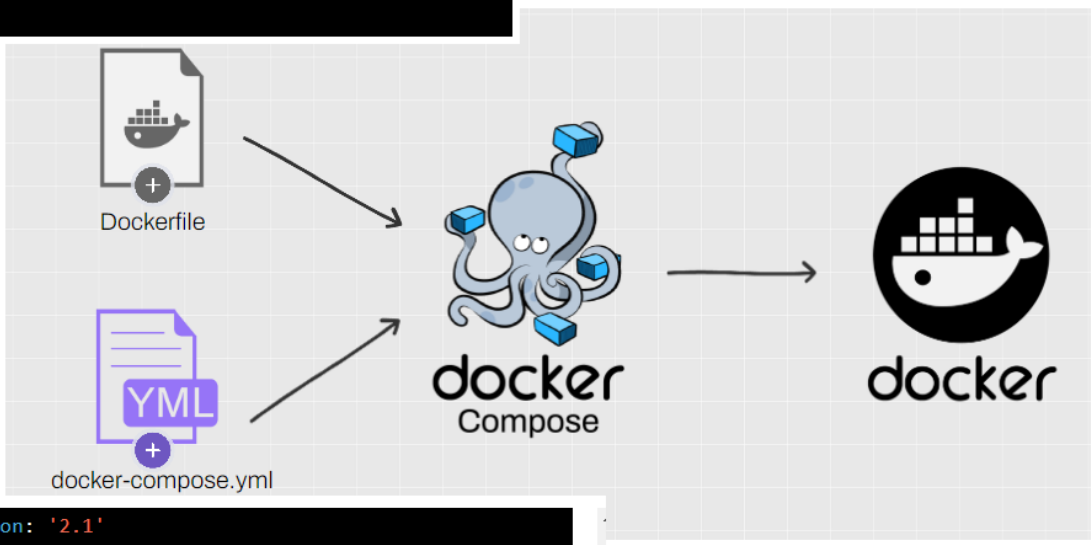
```
version: '2.1'

services:
 eureka-service:
   build: eureka-server
   mem_limit: 512m
   ports:
     - "8761:8761"
 product-service:
   build: product-service
   mem_limit: 512m
   ports:
     - "8084:8084"
 user-service:
   build: user-service
   mem_limit: 512m
   ports:
     - "8083:8083"
 gateway-service:
   build: api-gateway
   mem_limit: 512m
   ports:
     - "8085:8085"
```

Vejamos agora como realizar a configuração do Spring Cloud Config Server com o Docker Compose.

O primeiro passo é o mesmo do gateway. Devemos criar o arquivo Dockerfile em nosso projeto do server config. Em nosso caso, ele fica assim:

```
FROM adoptopenjdk/openjdk11:alpine-jre
ARG JAR_FILE=target/spring-boot-web.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","app.jar"]
EXPOSE 8086
```



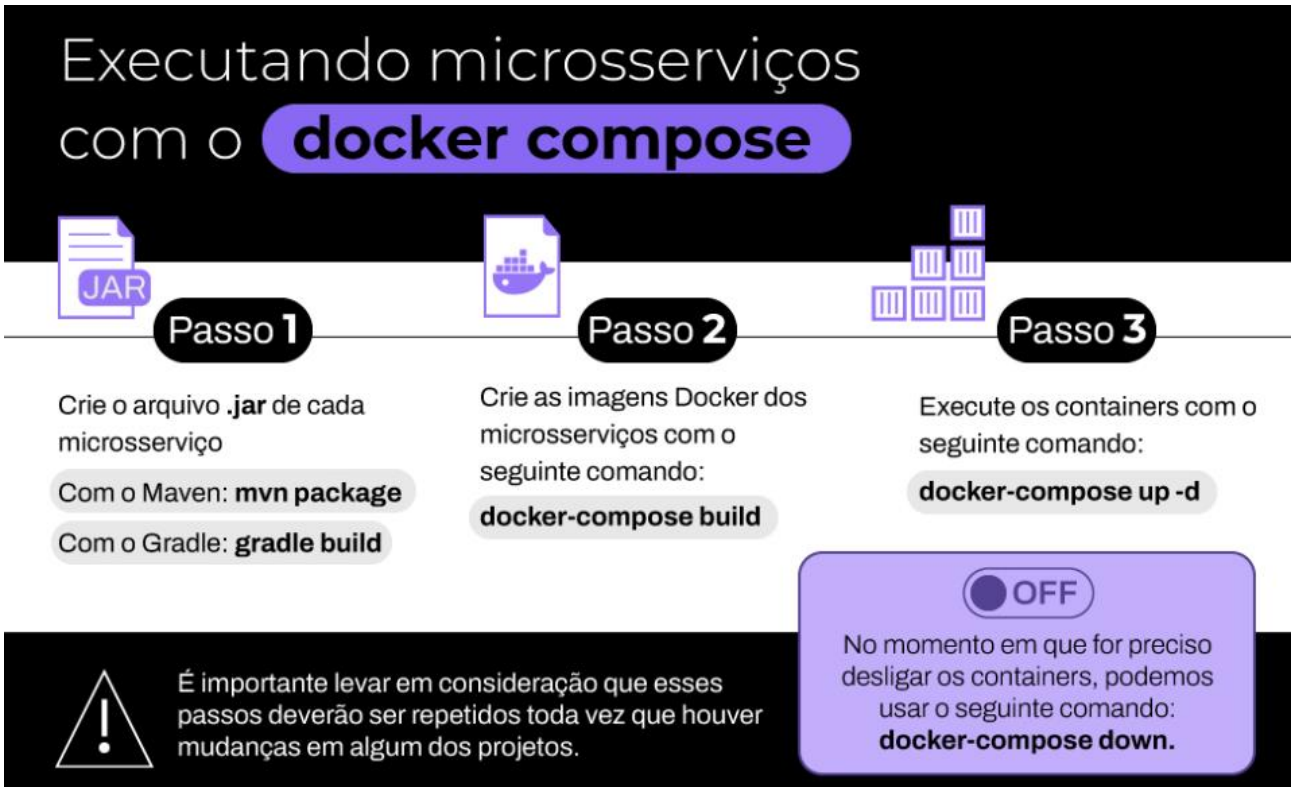

```
version: '2.1'

services:
 config-service:
   build: config-server
   mem_limit: 512m
   ports:
     - "8086:8086"
 eureka-service:
   build: eureka-server
   mem_limit: 512m
   ports:
     - "8761:8761"
 product-service:
   build: product-service
   mem_limit: 512m
   ports:
     - "8084:8084"
 user-service:
   build: user-service
   mem_limit: 512m
   ports:
     - "8083:8083"
 gateway-service:
   build: api-gateway
   mem_limit: 512m
   ports:
     - "8085:8085"
```

Agora sim, o gateway e o servidor de configuração já estão configurados para serem executados com o Docker Compose.

Passos para executar os microsserviços com o Docker Compose

De acordo com a aula anterior, mostramos a você os passos e comandos que serão necessários executar no arquivo docker-compose.yml para deixar funcionando os microsserviços incluídos no arquivo YML.

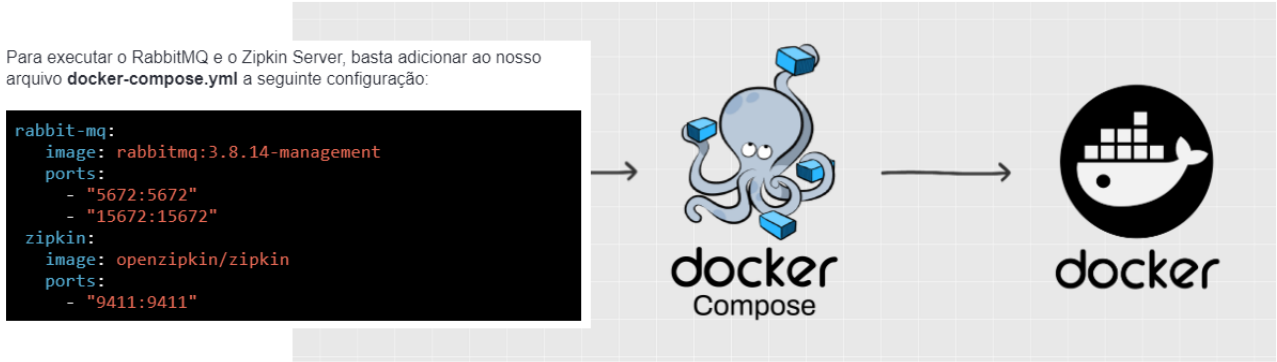


Zipkin e RabbitMQ com o Docker Compose

Por último, vejamos como configurar o Zipkin e o RabbitMQ com o Docker Compose.

No application.yml do projeto que usa o RabbitMQ e Zipkin, devemos modificar o hostname para nos conectar. Por exemplo, se eles fossem usados pelo users-service, seria preciso adicionar o seguinte dentro de sua configuração:

```
rabbitmq:
  username: guest
  password: guest
  host: rabbit-mq
  port: 5672
zipkin:
  base-url: http://zipkin:9411/
```



Conclusão

Agora que concluímos as duas aulas, conseguimos ter os componentes mais usados prontos para sua execução em containers e de forma conjunta, usando o Docker Compose. Vamos colocá-lo em prática na aula!