Compose blog

In the previous version of the shop landscape (see tag 'document\_v2' in this repository<TODO: URL>) services were started with a shell script. Each depended on Rabbit MQ to run, so there was a URL with an IP address that depended on whatever address the host it runs on got from its DHCP server. This was brittle, so I decided to introduce docker-compose. Actually, I should say 're-introduce' because my colleague Pavel Goultiaev built a previous version using compose. In this version, I copied and finished his code.

Docker compose uses a (https://github.com/xebia/microservices-breaking-up-a-monolith/blob/docker-compose/services/messages/docker-compose.yml)docker-compose.yml file that lists all services involved. You can find detailed information about the syntax of this file (https://docs.docker.com/compose/)here.

In my case the file is located in the root of the services/messages folder in the repo mentioned above. The compose file first defines a ‘rabbit’ service, listing its image tag and port mappings.

<code>

**rabbit:  
 image:** rabbitmq:3-management  
 **ports:** - **"15672:15672"** - **"4369:4369"** - **"5672:5672"** - **"25672:25672"**

</code>

Then, it defines the other containers, like this:

<code>

**shop:  
 image:** xebia/shop\_msg\_v2  
 **ports:** - **"9002:9002"**

</code>

To access the Rabbit server from the other service, like the Shop service, I’ve updated the servie’s application.properties files, see (https://github.com/xebia/microservices-breaking-up-a-monolith/blob/docker-compose/services/messages/shop/src/main/resources/application.properties)application.properties file in resources for an example:

<code>

**rabbitmq.port**=**5672  
rabbitmq.hostname**=**rabbit**

</code>

The hostname changed from ‘localhost’ to ‘rabbit’, i.e. the label of the RabbitMQ image in the docker-compose.yml file.

In my first solution, I added a ‘depends-on’ tag:

<code>

**depends\_on:** - **"rabbit"** - **"rabbitsetup"**

</code>

because I wanted to solve a dependency problem: the shop container needs the rabbit and rabbitsetup container (see below), so I figured I should define this dependency and then docker-compose would postpone starting shop until rabbit is ready. But this is not how it works. Starting the container doesn’t mean the process is ready to accept requests from clients.

I’ve used the RabbitMQ container from DockerHub as is, but moved the (https://github.com/xebia/microservices-breaking-up-a-monolith/blob/master/services/messages/rabbit/rabbitSetup/RabbitMQSetup.sh)configuration script to a separate container. This script needs a running Rabbit server, so it waits in a loop until the server is available. So, like the Shop service, the Rabbit configuration script has a dependency on the Rabbit container. The Rabbit server in the container has to be up and running, ready for requests, for the configuration script to succeed.

This problem could be solved by adding a health check to the definition of the rabbit container in docker-compose.yml, but that would require a condition in the rabbitsetup container definition in docker-compose. Conditions are not supported anymore in version 3, see (depends\_on) <https://docs.docker.com/compose/compose-file/#depends_on)>. I decided to just wait in a loop in the Rabbit configuration script for the port to become ready, for simplicities’ sake:

<code>

**until $**(curl --output /dev/null --silent --head --fail http://rabbit:15672); **do** printf **'.'** sleep 1  
**done**

</code>

Note that the RabbitMQ client in Spring reconnects to the service when necessary. This is a nice feature, in line with the advice about container dependencies given on the Docker (<https://docs.docker.com/compose/startup-order/)website>:

<quote> The problem of waiting for a database (for example) to be ready is really just a subset of a much larger problem of distributed systems. In production, your database could become unavailable or move hosts at any time. Your application needs to be resilient to these types of failures.

</quote>

You might see RabbitMQ connection errors in the container log when starting all containers:

shop\_1 | org.springframework.amqp.AmqpIOException: java.net.SocketTimeoutException

….

</code>

That are corrected a little later by the Rabbit client library:

<code>

shop\_1 | 2017-09-06 17:14:42.241 INFO 1 --- [cTaskExecutor-1] o.s.a.r.l.SimpleMessageListenerContainer : Restarting Consumer: tags=[{}], channel=null, acknowledgeMode=AUTO local queue size=0

rabbit\_1 |

rabbit\_1 | =INFO REPORT==== 6-Sep-2017::17:14:42 ===

rabbit\_1 | accepting AMQP connection <0.817.0> (172.18.0.7:33068 -> 172.18.0.2:5672)

shop\_1 | 2017-09-06 17:14:42.357 INFO 1 --- [cTaskExecutor-2] o.s.a.r.c.CachingConnectionFactory : Created new connection: SimpleConnection@15156430 [delegate=amqp://guest@172.18.0.2:5672/]

</code>

Apparently, the Rabbit client works just like recommended by the Docker documentation. To verify, you can pause the RabbitMQ container, run the tests in scenarioTest, watch it fail, unpause the container and finally see how the test succeeds.