Title: How to deploy maven multi-module spring boot to docker container

Post Body:

I'm setting up a multi-module maven project and trying to deploy it to docker a typical 3 layer project whit a separate container for a `MYSQL` DB.

I've been doing so research on the web and I came across the `docker-compose` a concept if I understand correctly with a config `YAML` file you can deploy each jar file into a different container and make them work together, ¿is this the only approach in my case?

Could I build one only `jar` file from the 3 modules and deploy it to the container? In my case i don't need to treat each project as a separate micro-service. On the other hand, I need docker compose because besides my `spring boot` project you need to run a container with a `MYSQL DB` and connect it to the `spring-boot`.

I'm not sure how to approach this scenario, thanks for any advice in advance.

Accepted Answer:

Don't confuse having n-tiers in your application with a micro-service architecture.

Micro-services are separate units of deployment and communicate among themselves over some sort of protocol (http, a message bus, something). Calling a function in process (from one jar to another) is not that, and so your individual maven modules aren't candidates for being put in containers and orchestrated by docker-compose.

If your web app communicated with the Domain Layer via, say, a restful api, then you could use docker compose to spin up two separate containers.

But if, as is likely, your 'api' is handled by controllers in the web app making in-process calls to your domain layer, you only have a single container to create and deploy.

There might still be good reasons to containerize your monolithic application. Perhaps it has complex OS dependencies and/or you want to deploy it to a PAAS like AWS Fargate or gCloud.

You might still want to use docker-compose as well, to deploy mysql in a container along side your web app.

Highest Rated Answer:

Agree with Robert, each jar/module does *not* equal a single Docker image (and thus container) in most cases.

I wanted to add that each boot application (i.e. each class with a main method) will likely end up in its own container (not always but it's a good rule of thumb). This is because each Docker image can have a single entrypoint, which is really just the command that executes to start the container. Thus each main method you create is a good candidate to put in its own container, because the `java` command can only execute one main method on startup.

Docker Compose is great for automating the startup of containers together, but it's just one of many tools that can do this (docker swarm, rancher, and kubernetes being other tools you can look at). Keep in mind, however, that these are others are heavy duty, production quality services and likely overkill for simple dev or test setups. Docker Compose works great for cases where you don't need automated scaling, fault tolerance, complex security policies, secrets management, etc.