

Title: how to use docker-compose and maven snapshot dependencies from external repos

Post Body:

I have several java components (WARs), all of them expose webservices, and they happen to use the samemessaging objects (DTOs).

This components all share a common maven dependency for the DTOs, let's call it 'messaging-dtos.jar'. This common dependency has a version number, for example messaging-dtos-1.2.3.jar, where 1.2.3 is the maven version for that artifact, which is published in a nexus repository and the like.

In the maven world, docker aside, it can get tedious to work with closed version dependencies. The solution for that is maven SNAPSHOTs. When you use for example Eclipse IDE, and you set a dependency to a SNAPSHOT version, this will cause the IDE to take the version from your current workspace instead of nexus, saving time by not having to close a version each time you make a small change.

Now, I don't know how to make this development cycle to work with docker and docker-compose. I have 'Component A' which lives in its own git repo, and messaging-dtos.jar, which lives in another git repo, and it's published in nexus.

My Dockerfile simply does a `RUN mvn clean install` at some point, bringing the closed version for this dependency (we are using Dockerfiles for the actual deployments, but for local environments we use docker-compose). This works for closed versions, but not for SNAPSHOTs (at least not for local SNAPSHOTs, I could publish the SNAPSHOT in nexus, but that creates another set of problems, with different content overwriting the same SNAPSHOT and such, been there and I would like to not come back).

I've been thinking about using docker-compose volumes at some point, maybe to mount whatever is in my local .m2 so ComponentA can find the snapshot dependency when it builds, but this doesn't feel 'clean' enough, the build would depend partially on whatever is specified in the Dockerfile and partially on things build locally. I'm not sure that'd be the correct way.

Any ideas? Thanks!

Accepted Answer: None

Highest Rated Answer:

I propose maintain two approaches: one for your local development environment (i.e. your machine) and another for building in your current CI tool.

For your local dev environment:

1. A Dockerfile that provides the system needs for your War application (i.e. Tomcat)
2. `docker-compose` to mount a volume with the built war app, from Eclipse or whatever IDE.

For CI (not your dev environment):

1. A very similar Dockerfile but one that can build your application (with maven installed)

A practical example

I use the docker feature: [multi stage build](#). A single Dockerfile for both Dev and CI envs that might be splited but I prefer to maintain only one:

```
FROM maven as build ARG LOCAL_ENV=false COPY ./src /app/ RUN mkdir /app/target/ RUN touch /app/target/app.war WORKDIR /app # R
```

The multi-stage build saves a lot of disk space discarding everything from the build, except what is being `COPY --from='ed`.

Then, `docker-compose.yml` used in Dev env:

```
version: '3' services: app: build: context: . args: LOCAL_ENV: true volumes: - ./target/ap
```

Build in CI (not your local machine):

```
# Will run `mvn clean install`, fetching whatever it needs from Nexus and so on. docker build .
```

Run in local env (your local machine):

```
# Will inject the war that should be available after a build from your IDE docker-compose up
```