Title: Caching Maven parent in offline build

Post Body:

TL/DR: How can I cache the `pom.xml` file's `<parent>` so my build can be run in offline mode?

I'm using Docker to build a maven project. My goal is to add two steps to the build: one to download all of the dependencies, and another to build the project. Here's what my Dockerfile looks like so far:

```
FROM maven:3.3-jdk-8  # Download the project dependencies (so they can be cached by Docker) ADD pom.xml /runtime/ WORKDIR /run
```

~~This seems to work fine for the plugins. I checked the /root/.m2/repository and everything seems to be in order.~~

~~Edit: When double checking for the /root/.m2/repository directory, it's no longer there. For some reason, Maven isn't saving any of the dependencies to this location.~~

Edit 2: After building the Docker image, there's no `/root/.m2/repository` directory. However, if I run `mvn dependency:go-offline` from within a shell inside the Docker container, the directory is created without a problem.

When I attempt build my application, I get the following error:

```
[ERROR] [ERROR] Some problems were encountered while processing the POMs: [FATAL] Non-resolvable parent POM for com.example:se
```

The problem seems to be that `mvn dependency:go-offline` isn't resolving the parent. When I run the build in offline mode, it breaks.

Here are the relevant portions of my `pom.xml` file:

```
<?xml version='1.0' encoding='UTF-8'?>  <project xmlns='http://maven.apache.org/POM/4.0.0' xmlns:xsi='http://www.w3.org/2001/X
```

Accepted Answer:

If you customize maven's `settings.xml` file you can save your repository files on image.

Create a custom version of `settings.xml`, with the `localRepository` setting modified, like this:

```
<settings xmlns='http://maven.apache.org/SETTINGS/1.0.0'                 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
```

Then override the default configuration when building your image:

```
FROM maven:3-jdk-8  COPY settings.xml /usr/share/maven/conf/settings.xml  RUN mkdir -p /usr/src/app WORKDIR /usr/src/app  ADD
```

Now your repository is stored in `/usr/share/maven/repo`.

Using onbuild

You can also create a base image using [ONBUILD](#), this will allow to have custom configured images for every maven project.

Like this:

```
FROM maven:3-jdk-8  COPY settings.xml /usr/share/maven/conf/settings.xml  RUN mkdir -p /usr/src/app WORKDIR /usr/src/app  ONBU
```

Then build the image:

```
docker build -t mvn_bldr .
```

This will create a *template* for other maven images. Then you can create your custom *downstream* image with:

```
FROM mvn_bldr
```

If you want to customize your image further you can add more instructions, every instruction of the template will be triggered after the `FROM mvn_bldr` command, as in the docs:

> The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile.

Highest Rated Answer:

It may seem like using `RUN` with Docker is like executing commands in a shell script, but it's not. Every instance of `RUN` gets applied to a new container that results from the changes created by the previous command. So each command is executing inside of a new container context.

Dockerifles can contain a `VOLUME` reference, which mounts an external directory inside the Docker container. If there were any files inside the volume, those files are wiped out. If you don't explicitly specify a volume, Docker is happy to instead create an empty folder.

While my Dockerfile doesn't contain an explicit reference to a `VOLUME`, its parent does. So, even though my `mvn dependency:go-offline` command was running, those files were being wiped out in the next step by the `VOLUME` specified in the [docker-maven Dockerfile](#).

In the end, I couldn't find a good way to make the `maven` Docker image work, so I switched to the `openjdk` image and installed `maven` via `apt-get` instead.