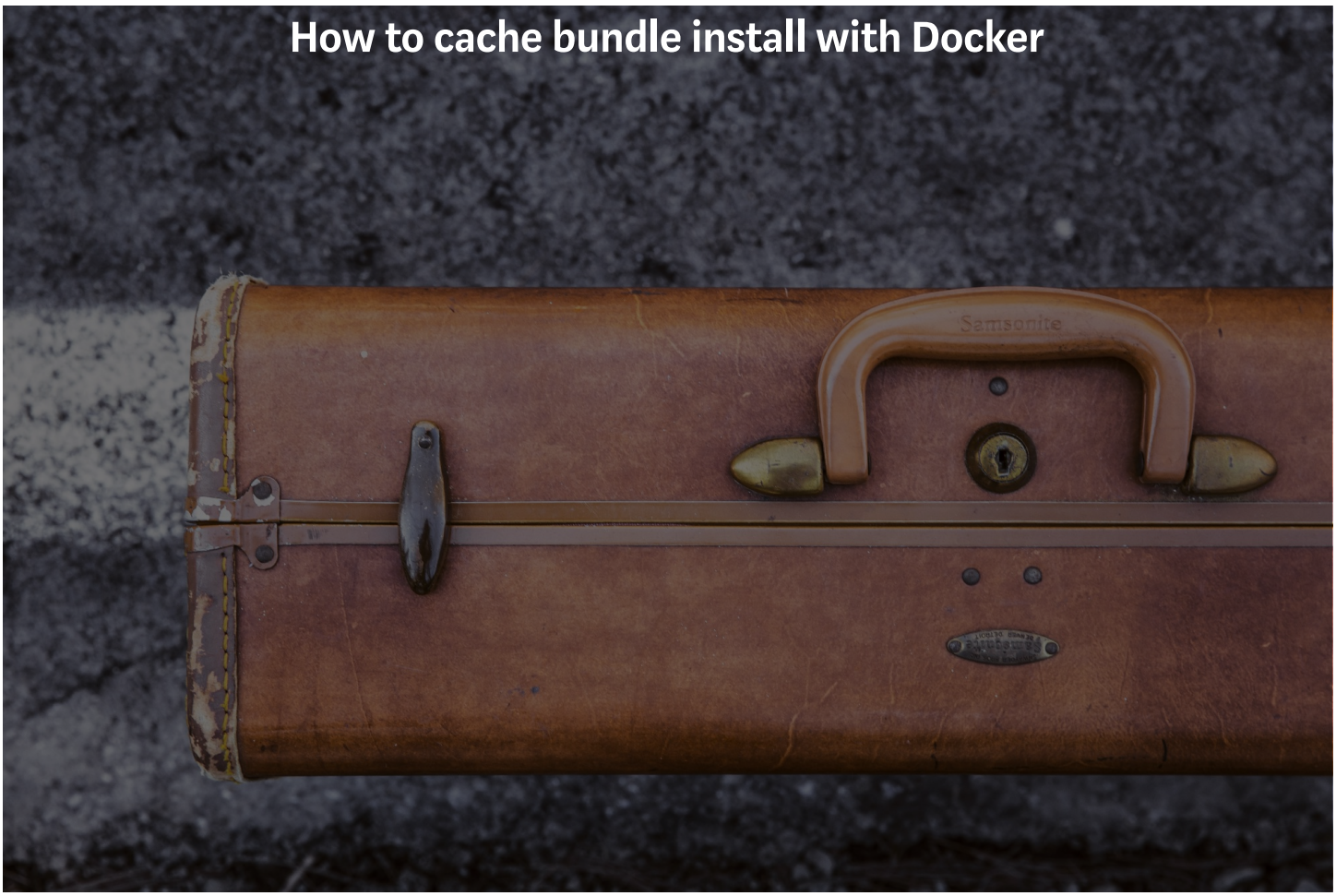
Fabiano B. Follow
CTO (@magnetis), Speaker, Ukulele player
Jul 30, 2015 · 3 min read



While dockering a Rails app, the first problem that comes out is the slow bundle install command while building the app's image.

Let's say you have this vanilla Dockerfile for a Rails app:

When you build this image with:

```
docker build -t app .
```

Docker creates a layer for each command executed. When you execute it a second time it will reuse the cache from previous execution if it did not change.

In this case there is a high probability that the ADD command will get a cold cache, because you change your app's code all the time (hopefully).

So it will stop using the cache on that line and will execute again all the next commands, including the bundle install. In summary: slow.

Bundle install before app files

So, let's use another approach:

Before the commands related to adding the app, we are copying Gemfile and Gemfile.lock to */tmp*, making it the current directory and running bundle install there.

So if nothing changed in the Gemfiles, it will hit a warm cache and just go on. This is a known technique. Made famous by this post.

An improvement compared to the first version, but what happens when we add a new gem? It will invalidate the cache and run the *entire* bundle install, installing all gems again.

Is there a way to reuse previous installed gems? Let's see.

Raise the volume

Another technique is to use a data volume to place the gems, so when bundle install is called it will reuse the gems previously installed, just

like the way it works when you install the environment directly to your host machine. So how do we do that?

The difference here is that we are overriding the $BUNDLE_PATH, this environment variable tells bundler the directory to install the gems, in this case the */box* folder. Also the bundle install was removed from the Dockerfile, soon you will understand why and where it went.

Next, we will also use docker-compose to easily mount a volume from another container:

A few things to notice here, we have some containers being declared. The first (app) will use the Dockerfile previously described to build the image. The second (box) will use busybox, a tiny image with 2.5mb in size. We will use it as a persistent ruby gems container.

The *volumes_from* directive tells compose to mount the */box* volume that exists in the box container on the app container, so in the end all gems will be installed in the box container volume.

Also the start command for the app container lives at */script/start*, it is a very simple bash script that first calls *bundle check* (kudos @sobrinho for this tip!) to check if all dependencies are already satisfied and if not calls *bundle install*; also brings up the Rails app:

One command to rule them all:

```
docker-compose up
```

This will pull the images if they do not exist locally and run the bash script above. During bundle install gems will be installed in the volume of the box container as mentioned before.

If we add another gem, all we need to do is:

```
docker-compose run app bundle install
```

Your bundle install back to life!

This will install only the new gems and reuse the previously installed on the data volume container.

Much faster!