Title: How does the Jib maven plugin build images without using a docker daemon?

Post Body:

For the past few months I have been experimenting with docker and have enjoyed the benefits of building and running java applications inside containers.

A few weeks ago I stumbled upon the jib maven plugin and noticed that jib can build images to docker registries **without using a docker daemon**.

After adding jib to one of my projects and running `mvn clean install jib:build`(on a VM which doesn't have docker installed), I was surprised that jib actually built and pushed an image containing my project to a remote registry.

Out of curiosity, I went online to read more about how jib builds and pushes docker images without having docker installed but found little to no information on the subject. I managed to find an article which explains a few ways of creating images without using docker and also tried to understand how the maven goal `jib:build` works by reading it's source code but neither of the two gave me any insights on what's happening behind the sceneces when you run `jib:build`.

I would greatly appreciate if someone shares more about the jib maven plugin and how it actually builds and pushes an image behind the scenes without using a docker daemon.

Accepted Answer:

(Jib dev here. I will very lightly touch on the subject at a very high level and only conceptually. Keep in mind that the following covers only one aspect of image building.)

Conceptually the anatomy of a container image is remarkably simple; it's just a collection of tarballs, plus some metadata about the image (about two JSON files). You'll get that if you untar some tarballs in an orderly fashion (union mounting, to be specific), you are left with some files and directories; these are basically the filesystem contents of an image you'll get and see at runtime. Throw in a couple small JSON files in the scene for some metadata about the image (for example, environment variables at runtime, image entrypoint, which tarballs this image is composed of, etc.), and you already have a container image in your hands. Then, you communicate with a container registry via the Docker Registry API (i.e., sending and receiving HTTP requests and responses) to upload those tarballs (after compression) and JSON files, and voila! You built and pushed an image to a registry.

So, yes, you can create those tarballs using the good old `tar` on the command-line (these tarballs are called "image layers"), create some JSON files with a text editor, and upload them to a registry using `curl`. I've done that before. Of course, in order for any container runtime to be able to actually run such an image, your tarballs may need to include some minimally required skeleton files and directories to correctly function, for example, as a Linux system (not a lot actually). But still, there's no restriction in the contents of tarballs; they don't even have to be a valid tar archive. (Yes, you can abuse a container registry to upload any garbage data. For example, this shell script uploads 40MB of random bytes to Docker Hub.) And you can still claim with the JSON metadata files that your (completely broken) "image" is composed of these garbage BLObs. (Of course, such an image will fail to run at runtime.)

Highest Rated Answer: None