Title: Running docker as non-root user OR running jenkins on tomcat as root user

Post Body:

I am trying to build a docker image using docker-maven plugin, and plan to execute the mvn command using jenkins. I have jenkins.war deployed on a tomcat instance instead of a standalone app, which runs as a non-root user. The problem is that docker needs to be run as root user, so maven commands need to be run as root user, and hence jenkins/tomcat needs to run as root user which is not a good practice (although my non-root-user is also sudoer so I guess won't matter much).

So bottom line, I see two solutions : Either run docker as non-root user (and need help on how to do that) OR Need to run jenkins as root (And not sure how to achieve that as I changed environment variable /config and still its not switching to root).

Any advice on which solution to choose and how to implement it ?

Accepted Answer:

> The problem is that docker needs to be run as root user, so maven commands need to be run as root user,

No, a docker run can be done with a `-u` (`--user`) parameter in order to use a non-root user inside the container.

> Either run docker as non-root user

Your user (on the host) needs to be part of the `docker` group. Then you can run the docker service with that user.

As commented, this is not very secure.
See:

- '`chrisfosterelli/dockerrootplease`'
- 'Understanding how uid and gid work in Docker containers'

That last links ends with the following findings:

> - If there’s a known uid that the process inside the container is executing as, it could be as simple as restricting access to the host system so that the uid from the container has limited access.
> - **The better solution is to start containers with a known uid using the`--user`** (you can use a username also, but remember that it’s just a friendlier way of providing a uid from the host’s username system), and **then limiting access to the uid on the host that you’ve decided the container will run as**.
> - Because of how uids and usernames (and gids and group names) map from a container to the host, specifying the user that a containerized process runs as can make the process appear to be owned by different users inside vs outside the container.

Regarding that last point, you now have **user namespace (userns) remapping** (since docker 1.10, but I would advice 17.06, because of issue 33844).

Highest Rated Answer:

I am also stuck on how to setup a docker build server.

Here's where I see ground truth right now...

> Docker commands require root privileges
>
> - This is because if can run arbitrary docker commands, you have the same powers as root on the host. (You can build a container runnings as root internally, with a filesystem mount to anywhere on the host, thus allowing any root action.)
>
> The 'docker' group is a big lie IMHO. It's effectively the same as making the members root.
>
> The **only** way I can see to wrap docker with any kind of security for non-root use is to build custom bash scripts to launch very specific docker commands, then to **carefully audit** the security implications of those commands, then add those scripts to the sudoers file (granting passwordless sudo to non-root users).

In the world where we integrate docker into development pipelines (e.g. putting docker commands in Maven builds or allow developers to make arbitrary changes to build definitions for a docker build server), I have idea how you maintain any security.