Use sudo inside Dockerfile (Alpine)

Asked 7 years ago Modified 5 years, 10 months ago Viewed 134k times

38

I have this Dockerfile ...

```
FROM keymetrics/pm2:latest-alpine

RUN apk update && \
    apk upgrade && \
    apk add \
       bash

COPY . ./

EXPOSE 1886 80 443

CMD pm2-docker start --auto-exit --env ${NODE_ENV} ecosystem.config.js
```

How can I execute the `CMD` command using `sudo` ?

I need to do this because the port 443 is allowed only for sudo user.

linux docker docker-compose dockerfile pm2

Share Follow

asked Mar 11, 2018 at 22:51
ridermansb
11.1k 28 121 231

1 You are still root when CMD is executed. What makes you think that's not the case? Can you share the command line you used to start your container. – Christophe Schmitz Mar 12, 2018 at 0:01

2 Docker itself runs as root and a container will default to root unless you have a `USER` set in the image, which aren't in the base image or your Dockerfile. Can you explain a bit more about what you're trying to do and what is going wrong? Is there an error message? – Matt Mar 12, 2018 at 0:10

@ChristopheSchmitz I know that the `CMD` command is executed, my question is how to execute him with `sudo` – ridermansb Mar 12, 2018 at 10:55

@Matt I need to execute `pm2` command with `sudo` privileges because I need to run it with port 443. Locally in my machine I can run `pm2` with command `sudo pm2 sart` but to deploy my app I'm using docker and I need to run the pm2 command with sudo too – ridermansb Mar 12, 2018 at 10:57

You are already root when CMD is executed. sudo won t help there. – Christophe Schmitz Mar 12, 2018 at 10:57

2 Answers

Sorted by: Highest score (default)

54

The su-exec can be used in alpine. Do add it the package, if not already available, add the following to your Dockerfile

```
RUN apk add --no-cache su-exec
```

Inside your scripts you'd run inside docker you can use the following to become another user:

```
exec su-exec <my-user> <my command>
```

Alternatively, you could add the more familiair sudo package while building your docker-file Add the following to your Dockerfile that's FROM alpine

```
RUN set -ex && apk --no-cache add sudo
```

After that you can use sudo

```
sudo -u <my-user> <my command>
```

Share Follow edited May 14, 2019 at 11:10 answered Mar 21, 2019 at 10:01

Gerbrand
1,633 1 13 21

4 su-exec wasn't available in my alpine container. I had to modify its Dockerfile to include: RUN apk add --no-cache su-exec. sudo didn't help me as it was asking for the user's password which I didn't know.
– user674669 Mar 28, 2019 at 1:40

38

Sudo isn't shipped with Alpine images normally, and it rarely makes sense to include it inside of any container. What you need isn't sudo to bind to a low numbered port, but the root user itself, and sudo is just a common way to get root access in multi-user environments. If a container included sudo, you would need to either setup the user with a password, or allow commands to run without a password. Regardless of which you chose, you now have a privilege escalation inside the container, defeating the purpose of running the container as a normal user, so you may as well run the container as root at that point.

If the upstream image is configured to run as a non-root user (unlikely since you run `apk` commands during the build), you can specify `USER root` in your Dockerfile, and all following steps will run as root by default, including the container entrypoint/cmd.

If you start your container as a different user, e.g. `docker run -u 1000 your_image`, then to run your command as root, you'd remove the `-u 1000` option. This may be an issue if you run your container in higher security environments that restrict containers to run as non-root users.

If your application itself is dropping the root privileges, then including sudo is unlikely not help, unless the application itself has calls to sudo internally. If that's the case, update the application to drop root privileges after binding to the ports.

Most importantly, if the only reason for root inside your container is to bind to low numbered ports, then configure your application inside the container to bind to a high numbered port, e.g. 8080 and 8443. You can map this container port to any port on the host, including 80 and 443, so the outside world does not see any impact. E.g. `docker run -p 80:8080 -p 443:8443 your_image`. This simplifies your image (removing tools like sudo) and increases your security at the same time.

Share Follow

answered May 14, 2019 at 15:04

BMitch
266k 50 542 500

7 thank you ! sometimes what we are looking for is not what we actually need. This was the case for me and your explanation helped me a lot ! – TudorIftimie Aug 26, 2019 at 10:35

Fantastic answer, I was not aware that higher number ports didn't require root privileges. – Stephen Collins May 17, 2020 at 16:33

My current issue is that running a container as root causes my mounts to become root which is not really want i want since the developers go mad if their vendor/ folder is suddenly root. The only somewhat reasonable fix I found was to run the container as a user but nginx as root due to the way nginx works. – Menno van Leeuwen Aug 27, 2024 at 8:50

Nginx can drop permissions. And you can do the same for your app with tools like gosu. Going the other way is a false security, since the user has all the access as the root user with an added sudo in front of the command. You should also avoid running nginx and your app in the same container, logging and failure recovery are much more difficult. – BMitch Aug 27, 2024 at 12:19