zwischenzugs

The Most Pointless Docker Command Ever

2 Minutes

What?

This article will show you how you can undo the things Docker does for you in a Docker command. Clearer now?

OK, Docker relies on Linux namespaces to isolate effectively copy parts of the system so it ends up looking like you are on a separate machine.

For example, when you run a Docker container:

```
$ docker run -ti busybox ps -a
PID USER COMMAND
 1 root ps -a
```

it only 'sees' its own process IDs. This is because it has its own PID namespace.

Similarly, you have your own network namespace:

```
$ docker run -ti busybox netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags Type State I-Node Path
```

You also have your own view of inter-process communication and the filesystem.

Go on then

This is possibly the most pointless possible docker command ever run, but here goes:

```
docker run -ti
    --privileged
    --net=host --pid=host --ipc=host
    --volume /:/host
    busybox
    chroot /host
```

The three '=host' flags bypass the network, pid and ipc namespaces. The volume flag mounts the root filesystem of the host to the '/host' folder in the container (you can't mount to '/' in the container). The privileged flags gives the user full access to the root user's capabilities.

All we need is the chroot command, so we use a small image (busybox) to chroot to the filesystem we mounted.

What we end up with is a Docker container that is running as root with full capabilities in the host's filesystem, will full access to the network, process table and IPC constructs on the host. You can even 'su' to other users on the host.

If you can think of a legitimate use for this, please drop me a line!

Why?

Because you can!

Also, it's quite instructive. And starting from this, you can imagine scenarios where you end up with something quite useful.

Imagine you have an image – called 'filecheck' – that runs a check on the filesystem for problematic files. Then you could run a command like this (which won't work BTW – filecheck does not exist):

```
docker run --workdir /host -v /:/host:ro filecheck
```

This modified version of the pointless command dispenses with the chroot in favour of changing the workdir to '/host', and – crucially – the mount now uses the ':ro' suffix to mount the host's filesystem read-only, preventing the image from doing damage to it.

So you can check your host's filesystem relatively safely without installing anything.

You can imagine similar network or process checkers running for their namespaces.

Can you think of any other uses for modifications of this pointless command?
This post is based on material from Docker in Practice (http://manning.com/miell/?a_aid=zwischenzugs&a_bid=e0d48f62), available on Manning's Early Access Program. ***Get 39% off with the code: 39miell***

Published by zwischenzugs

View all posts by zwischenzugs

5 thoughts on "The Most Pointless Docker Command Ever"

acsandeep says:
November 6, 2015 at 12:12 pm
How about using this trick to collect stats for Collectd while still isolating it from the Main OS

↳ Reply

zwischenzugs says:
November 6, 2015 at 12:24 pm
Yes, monitoring is a good use case for this – I recently abandoned a containerized nagios implementation because it was essentially leading me down this path.

↳ Reply

mviereck says:
October 19, 2016 at 10:28 pm
Thanks for this pointless, in no way isolated docker command! It helps me to isolate the core of two problems I have with GPU acceleration for GUI apps in docker (https://github.com/mviereck/x11docker). Running my images with `–privileged –icp=host –pid=host –net=host` the problems went away, and I could break down to need only `–net` for one and `–ipc` for the other problem. Now I can look further for the core of the problems as I have a direction to look at, hopefully not needing one of these options at last.

↳ Reply

Pingback: How to execute a command directly on the host system through docker.sock in a Docker container?

Jacky Wang says:
December 23, 2022 at 5:06 pm
Thanks for your sharing! In one of my projects, I need to detect keystrokes from the underly host while the app runs on a container. We are not exposed to just the file system namespaces only. I wonder if this technique can help me to do that, or any extra commands that I will need.

↳ Reply