Debugging DockerContainers

Rajesh Kumar

DevOps Architect

@RajeshKumarIN | www.RajeshKumar.xyz

There are two ways to use the container.

1. Either you reuse the same container over and over, or
2. You remove your container once it exits and recreate a new one every time.

Reusing the same container has the benefit of being faster to boot up and also have access to the previous data.

Recreating a new container every time means scrubbing all your previous data.

Log File

/var/log/containers on your host, and the files are named in the following convention: <service>_start_errors.log. These logs files will contain any output created by your start command, and are a good starting point in understanding why your container won't start.

docker ps command

- Check for any running containers with docker ps - if the container isn't running there, there may be an issue preventing it from starting.

docker ps –a command

- In this case, we can run docker ps -a to check for old containers that have failed to start (or are no longer running). Take note of the container ID for your latest deployment, as we'll need it in the next step.

View stdout history with the logs command

- Anything that gets written to stdout for the process that is pid 1 inside the container will get captured to a history file on the host, where it can be viewed with the **logs** command.

docker logs <container-id>

- Now let's check the log of that failed container by running docker logs <container-id>, which will output any error messages for why the container couldn't start.

IMPORTANT

Make sure to have the -ti flags in your docker run command

Example

```
$ docker run -d --name=logtest alpine /bin/sh -c "while true; do
sleep 2; df -h; done"
35f6353d2e47ab1f6c34073475014f4ab5e0b131043dca4454f67be9d8ef1253
$ docker logs logtest
Filesystem Size Used Available Use% Mounted on
none 93.7G 2.0G 87.0G 2% /
tmpfs 3.8G 0 3.8G 0% /dev
tmpfs 3.8G 0 3.8G 0% /sys/fs/cgroup
... etc ...
```

Stream stdout with the attach command

- If you want to see what is written to stdout in real time then the **attach**command is your friend.

Example

```
$ docker run -d --name=logtest alpine /bin/sh -c "while true; do
sleep 2; df -h; done"
26a329f1e7074f0c0f89caf266ad145ab427b1bcb35f82557e78bafe053faf44
$ docker attach logtest
Filesystem Size Used Available Use% Mounted on
none 93.7G 2.0G 87.0G 2% /
tmpfs 3.8G 0 3.8G 0% /dev
tmpfs 3.8G 0 3.8G 0% /sys/fs/cgroup
… etc …
Filesystem Size Used Available Use% Mounted on
none 93.7G 2.0G 87.0G 2% /
tmpfs 3.8G 0 3.8G 0% /dev
tmpfs 3.8G 0 3.8G 0% /sys/fs/cgroup
… etc …
```

Get process stats with the top command

- The docker **top** command is exactly what it sounds like: top that runs in the container.

```
$ docker run -d — name=toptest alpine:3.1 watch "echo 'Testing top'"
fc54369116fe993ae45620415fb5a6376a3069cdab7c206ac5ce3b57006d4241
$ docker top toptest
UID PID … TIME CMD
root 26339 … 00:00:00 watch "echo 'Testing top'"
root 26370 … 00:00:00 sleep 2
```

View container details with the inspect command

- The **inspect** command returns information about a container or an image. Here's an example of running it on the toptest container from the last example above.

Example

- $ docker inspect image_name

Some of the more valuable bits of intelligence you can get are:

1. Current state of the container. (In the “State” property.)
2. Path to the log history file. (In the “LogPath” field.)
3. Values of set environment vars. (In the “Config.Env” field.)
4. Mapped ports. (In the “NetworkSettings.Ports” field.)

View image layers with the history command

- $ docker history image_name

THANKS