Understanding Docker Container Exit Codes

The most common exit codes, what they mean, and what causes them

Sandeep Madamanchi
Oct 21, 2019 · 4 min read

It’s one of the most common question that I come across: “Why is my container not running?”

Can docker container exit codes help troubleshoot this issue?

The first step in answering this question is to identify the exit code for the docker container. The exit code may give a hint as to what happened to stop the container running. This article lists the most common exit codes when working with docker containers and aims to answer two important questions:

- What does this specific exit code mean?
- What action caused this exit code?

This will ultimately help answer the original question: “Why is my container not running?”

How to Find Exit Codes

Option 1: List all containers that exited

```
docker ps --filter "status=exited"
```

Option 2: Grep by container name

```
docker ps -a grep <container-name>Example: docker ps -a | grep hello-world
```

Option 3: Inspect by container id

```
docker inspect <container-id> --format='{{.State.ExitCode}}'Example: docker
inspect ca6cbb290468 --format='{{.State.ExitCode}}'
```

Exit Codes

Common exit codes associated with docker containers are:

- **Exit Code 0**: Absence of an attached foreground process
- **Exit Code 1**: Indicates failure due to application error

- **Exit Code 137**: Indicates failure as container received SIGKILL (Manual intervention or ‘oom-killer’ [OUT-OF-MEMORY])
- **Exit Code 139**: Indicates failure as container received SIGSEGV
- **Exit Code 143**: Indicates failure as container received SIGTERM

Exit Code 0

- Exit code 0 indicates that the specific container does not have a foreground process attached.
- This exit code is the exception to all the other exit codes to follow. It does not necessarily mean something bad happened.
- Developers use this exit code if they want to automatically stop their container once it has completed its job.

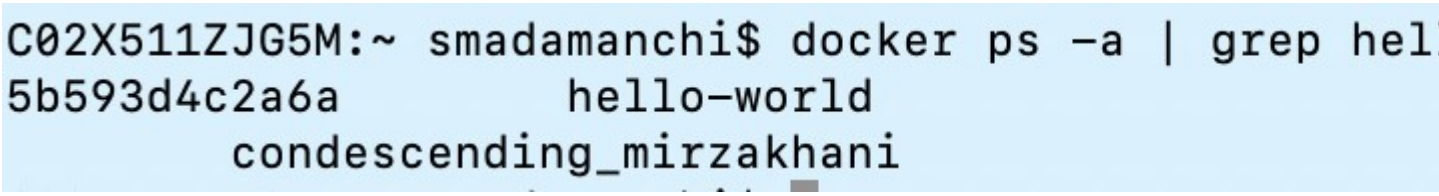
Here is an example using the public docker container — “hello-world”. If you have docker installed on your system or VM instance, run this:

```
docker run hello-world
```

You will get a message, “Hello from docker!” but try to find the container using this code:

```
docker ps -a | grep hello-world
```

You’ll notice that the container exited and the exit code is 0. This is because the container does not have any foreground process attached, such as a Java process or a shell process that runs until a SIGTERM event occurs



Exit Code 1

- Indicates that the container stopped due to either an application error or an incorrect reference in Dockerfile to a file that is not present in the container.
- An application error can be as simple as “divide by 0” or as complex as “Reference to a bean name that conflicts with existing, non-compatible bean definition of same name and class.”
- An incorrect reference in Dockerfile to a file not present in the container can be as simple as a typo (the example below has `sample.ja` instead of `sample.jar`)

Exit Code 137

- This indicates that container received SIGKILL
- A common event that initiates a SIGKILL is a docker kill. This can be initiated either manually by user or by the docker daemon:

```
docker kill <container-id>
```

- `docker kill` can be initiated manually by the user or by the host machine. If initiated by host machine, then it is generally due to being out of memory. To confirm if the container exited due to being out of memory, verify `docker inspect` against the container id for the section below and check if `OOMKilled` is true (which would indicate it is out of memory):

```
"State": {
 "Status": "exited",
 "Running": false,
 "Paused": false,
 "Restarting": false,
 "OOMKilled": true,
 "Dead": false,
 "Pid": 0,
 "ExitCode": 137,
 "Error": "",
 "StartedAt": "2019-10-21T01:13:51.7340288Z",
 "FinishedAt": "2019-10-21T01:13:51.7961614Z"
}
```

Exit Code 139

- This indicates that container received SIGSEGV
- SIGSEGV indicates a segmentation fault. This occurs when a program attempts to access a memory location that it's not allowed to access, or attempts to access a memory location in a way that's not allowed.
- From the Docker container standpoint, this either indicates an issue with the application code or sometimes an issue with the base images used by the container.

Exit Code 143

- This indicates that container received SIGTERM.
- Common events that initiate a SIGTERM are `docker stop` or `docker-compose stop`. In this case there was a manual termination that forced the container to exit:

```
docker stop <container-id>
OR
docker-compose down <container-id>
```

- **Note:** sometimes `docker stop` can also result in exit code 137. This typically happens if the application tied to the container doesn't handle SIGTERM — the docker daemon waits ten seconds then issues SIGKILL

Some uncommon exit codes with Docker containers (typically with shell script usage)

- **Exit Code 126**: Permission problem or command is not executable
- **Exit Code 127**: Possible typos in shell script with unrecognizable characters

https://medium.com/better-programming/understanding-docker-container-exit-codes-5ee79a1d58f6