Run multiple processes in a container

- Use a wrapper script
- Use Bash job controls
- Use a process manager

A container's main running process is the `ENTRYPOINT` and/or `CMD` at the end of the `Dockerfile`. It's best practice to separate areas of concern by using one service per container. That service may fork into multiple processes (for example, Apache web server starts multiple worker processes). It's ok to have multiple processes, but to get the most benefit out of Docker, avoid one container being responsible for multiple aspects of your overall application. You can connect multiple containers using user-defined networks and shared volumes.

The container's main process is responsible for managing all processes that it starts. In some cases, the main process isn't well-designed, and doesn't handle "reaping" (stopping) child processes gracefully when the container exits. If your process falls into this category, you can use the `--init` option when you run the container. The `--init` flag inserts a tiny init-process into the container as the main process, and handles reaping of all processes when the container exits. Handling such processes this way is superior to using a full-fledged init process such as `sysvinit` or `systemd` to handle process lifecycle within your container.

If you need to run more than one service within a container, you can achieve this in a few different ways.

Use a wrapper script

Put all of your commands in a wrapper script, complete with testing and debugging information. Run the wrapper script as your `CMD`. The following is a naive example. First, the wrapper script:

```
#!/bin/bash


# Start the first process
```

```
./my_first_process &

# Start the second process
./my_second_process &

# Wait for any process to exit
wait -n

# Exit with status of process that exited first
exit $?
```

Next, the Dockerfile:

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
COPY my_first_process my_first_process
COPY my_second_process my_second_process
COPY --chmod=755 my_wrapper_script.sh my_wrapper_script.sh
CMD ./my_wrapper_script.sh
```

Use Bash job controls

If you have one main process that needs to start first and stay running but you temporarily need to run some other processes (perhaps to interact with the main process) then you can use bash's job control. First, the wrapper script:

```
#!/bin/bash

# turn on bash's job control
set -m

# Start the primary process and put it in the background
./my_main_process &

# Start the helper process
./my_helper_process

# the my_helper_process might need to know how to wait on the
# primary process to start before it does its work and returns
```

```
# now we bring the primary process back into the foreground
# and leave it there
fg %1
```

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
COPY my_main_process my_main_process
COPY my_helper_process my_helper_process
COPY --chmod=755 my_wrapper_script.sh my_wrapper_script.sh
CMD ./my_wrapper_script.sh
```

Use a process manager

Use a process manager like `supervisord`. This is more involved than the other options, as it requires you to bundle `supervisord` and its configuration into your image (or base your image on one that includes `supervisord`), along with the different applications it manages. Then you start `supervisord`, which manages your processes for you.

The following Dockerfile example shows this approach. The example assumes that these files exist at the root of the build context:

- `supervisord.conf`
- `my_first_process`
- `my_second_process`

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
RUN apt-get update && apt-get install -y supervisor
RUN mkdir -p /var/log/supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
COPY my_first_process my_first_process
COPY my_second_process my_second_process
CMD ["/usr/bin/supervisord"]
```

If you want to make sure both processes output their `stdout` and `stderr` to the container logs, you can add the following to the `supervisord.conf` file:

```
[supervisord]
nodaemon=true
```

```
logfile=/dev/null
logfile_maxbytes=0

[program:app]
stdout_logfile=/dev/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

□□□ #2
□ □□□ □□□ 2026-10-31 19:21:51 UTC
□ □□□ □□□ 2026-10-31 19:21:53 UTC