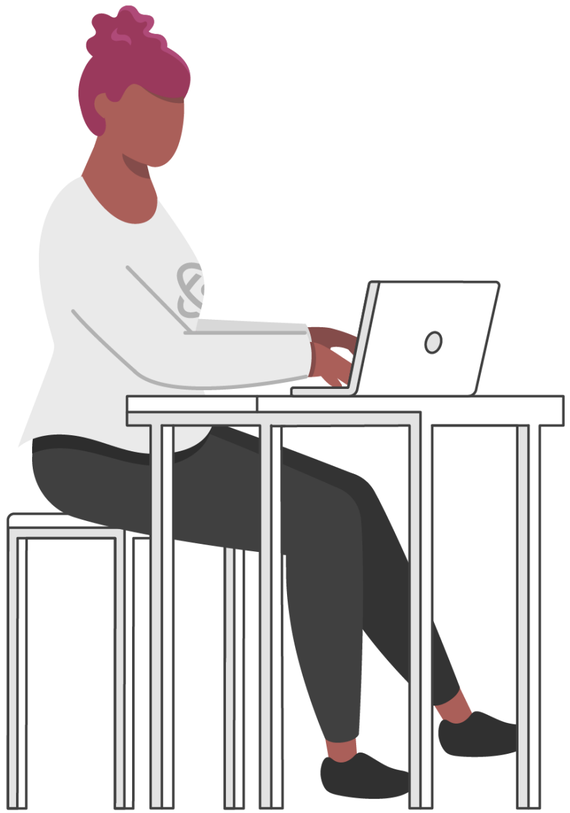
Developing Docker Apps: Core Principles

Using Volumes to Develop Applications in Containers

Nigel Brown
Freelance Technical Author

@n_brownuk | @nigelb@fosstodon.org | windsock.io



Mia is a senior software developer

- Tasked with investigating cloud native app development
- Knows a little about Docker, but not how to use it in a workflow
- She discovers core benefits: flexibility, common tooling, better productivity

Let’s join Mia on her journey of discovery!

Module Outline

Coming up:

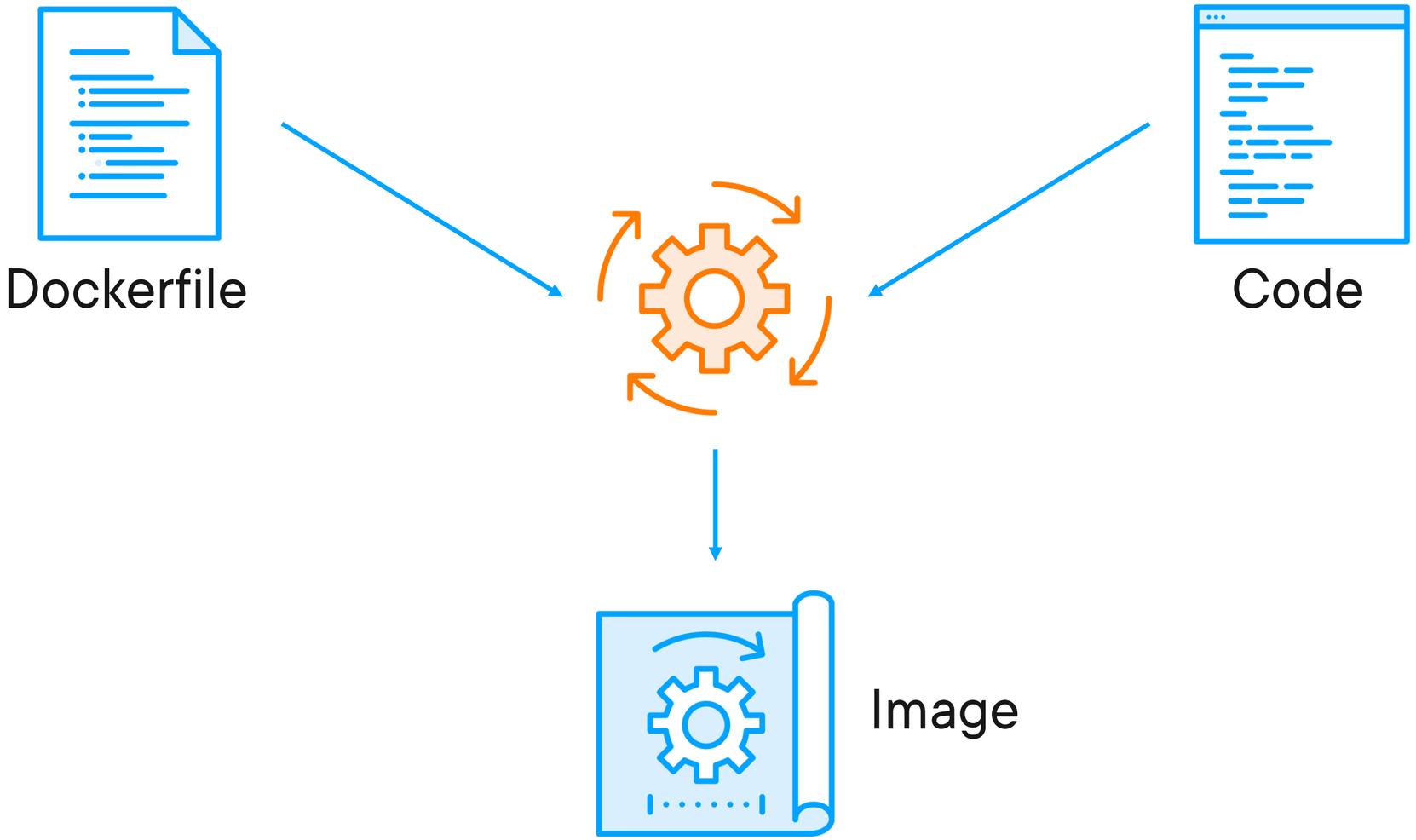
- The inner loop of software development with containers
- Data persistence with Docker volumes
- Hot reloads on source file changes
- How to handle volumes permissions
- Developing with bind mount volumes

Container Image

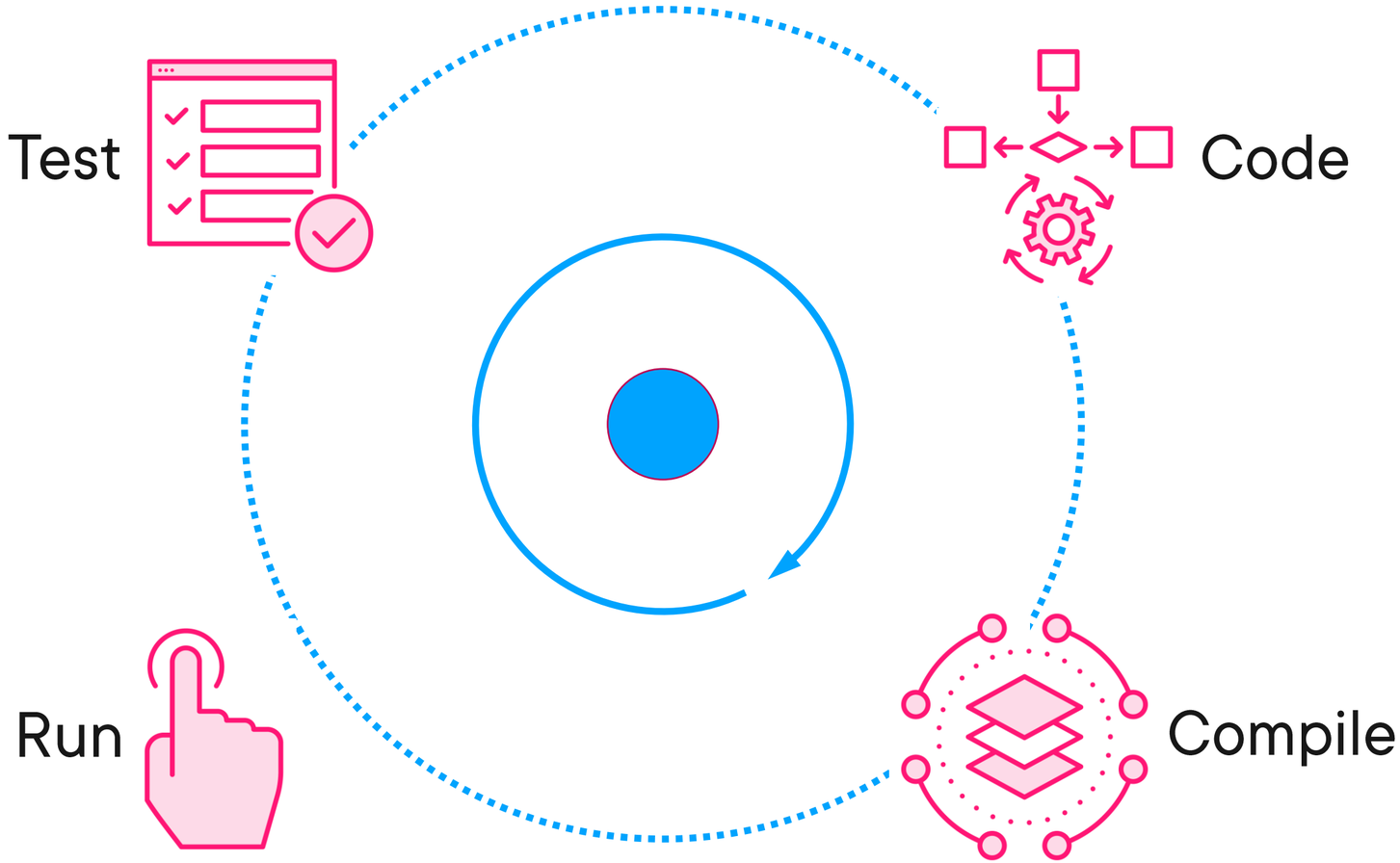
Dockerfile

```
FROM node

# Create app directory
WORKDIR /app

# Copy app source from build context
COPY . .

# Install app dependencies
RUN npm install

# Port app listens on
EXPOSE 3000

# Specify container’s default command
CMD ["node", “src/index.js"]
```

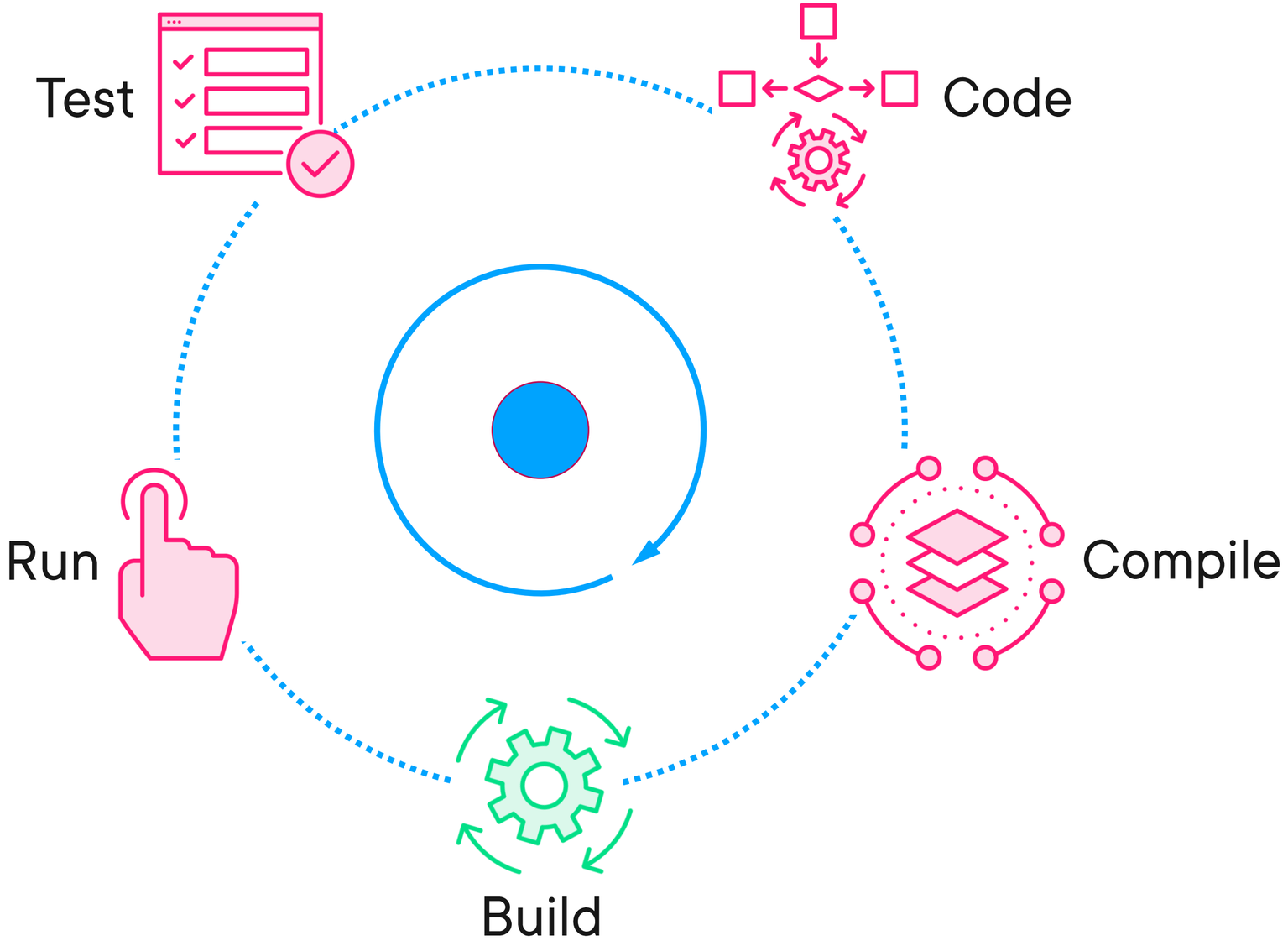
Building Images



The Inner Loop



The Inner Loop with Containers

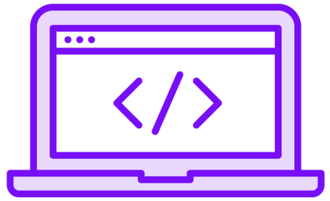


Container Image Builds

Complex image definitions can take a significant amount of time to build. This can severely impact the productivity of a software developer.

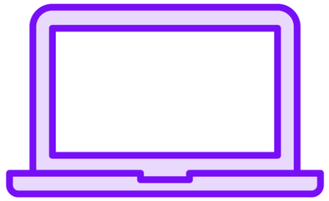
Developing Inside a Container

Iterate over the inner loop from inside the container rather than outside.



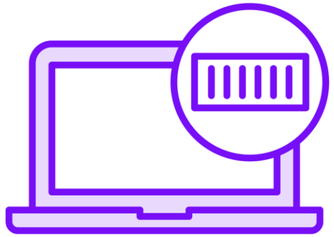
Source code

Part of the container's filesystem



Command line

Run application and tests using the CLI



Ephemerality

Changes don't persist on container deletion

We need a method for persisting changes between container invocations.

Docker Volumes

Docker uses the ‘volume’ concept to aid the persistence of data between container invocations.

Persistent storage

Area of storage located outside container’s filesystem

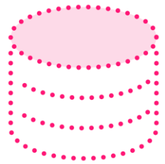
Volume plugins

Implemented using a plugin system for flexibility

Filesystem mount

Storage mounted inside container during its life

Volume Types



Tmpfs mount - used to temporarily store sensitive data in memory

Named or anonymous volume - managed by Docker using its CLI

Bind mount - arbitrary directory mounted from host to container

```
$ docker volume create code-volume
code-volume
```

Creating a Named Volume

Created explicitly with ‘volume’ sub-command

```
$ docker run --volume code-volume:/app ...
```

Creating a Named Volume

Implicit creation achieved using the ‘--volume’ flag

```
$ docker volume ls
DRIVER    VOLUME NAME
local     code-volume
```

Creating a Named Volume

Volumes can be listed (‘ls’), inspected (‘inspect’), removed (‘rm’) and so on

Merits of Named Volumes

Advantages	VS	Disadvantages
Volume is a managed object Isolated from other host activity Easy to identify and backup Better performance when using the Docker Desktop		Owned by the ‘root’ user

```
$ docker run --volume /path/on/host:/path/in/container ...
```

Using Bind Mounts

Host location mounted into the container when the container is invoked

Directory paths must be absolute paths rather than relative paths

Building an Image for an Application

Build context

```
.
├── Dockerfile
├── package.json
├── spec […]
└── src
    ├── index.js
    ├── persistence […]
    ├── routes […]
    └── static
        ├── css […]
        ├── index.html
        └── js […]
```

Dockerfile

```
FROM node

# Create app directory
WORKDIR /app

# Copy app source from build context
COPY . .

# Install app dependencies
RUN npm install

# Port app listens on
EXPOSE 3000

# Specify container’s default command
CMD ["node", “src/index.js"]
```

```
$ docker build -t myapp:1.0 .
[+] Building 22.5s (4/8)
 => [internal] load build definition from Dockerfile                    0.1s
 => => transferring dockerfile: 286B                                    0.0s
<snip>
 => [1/4] FROM docker.io/library/node@sha256:d903b23148dccca63152cb2bdf6f  21.8s
<snip>
 => [4/4] RUN npm install                                              24.2s
 => exporting to image                                                  1.3s
 => => exporting layers                                                 1.3s
 => => writing image sha256:4e6d1de1f8f419d1cdb8e1564594fc866f77c32423dd8a  0.0s
 => => naming to docker.io/library/myapp:1.0                            0.0s
```

Invoking an Image Build

A container image build is invoked with the Docker CLI

Handling Dynamic Changes

Watch for changes

Edits to source code are automatically detected inside running container

Perform hot reload

Process monitor performs a hot reload by restarting the application in the container

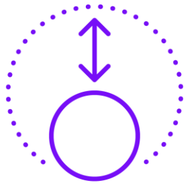
```
$ pwd
/home/nigel/myapp
$ docker run -itd -p 3000:3000 --volume $(pwd):/app myapp:1.0 nodemon src/index.js
```

Coding Inside a Container

Mount host directory with source code into container

Replace default command with hot reload utility (e.g. nodemon)

Outcomes

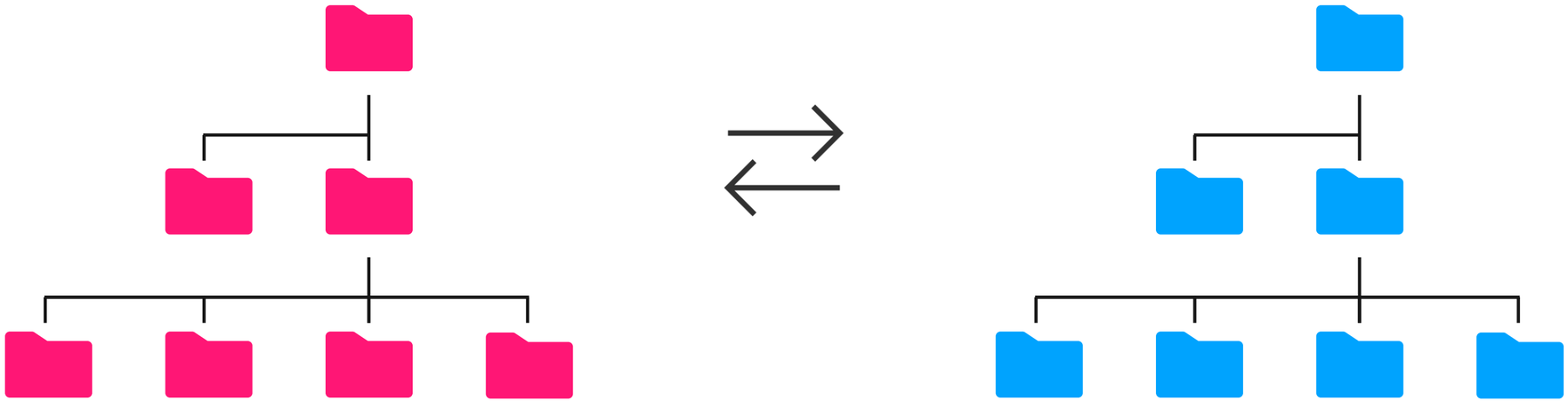


Changes made to source located on the host are reflected in the container via the bind mount volume

The hot reload utility automatically detects any changes to the source files and restarts the server

The changes can be tested to check they have implemented the desired behavior

Different User and Group IDs



Host

UID: 1000 GID: 1000

Container

UID: 0 GID: 0

```
$ touch created_on_host
$ ls -l
-rw-r--r-- 0 nigel 23 Feb 16:35 created_on_host
$ docker run --volume $(pwd):/src debian touch /src/created_in_container
$ ls -l
-rw-r--r-- 0 root  23 Feb 16:39 created_in_container
-rw-r--r-- 0 nigel 23 Feb 16:35 created_on_host
$ echo Hello >> ./created_in_container
bash: ./created_in_container: Permission denied
```

File and Directory Ownership

Mismatch in user and group IDs renders file unwritable on host

Creating a User in a Container Image

Dockerfile

```
FROM debian

# Add a group and user to match the user on the host
RUN groupadd -r --gid 1000 user \
    && useradd -r --uid 1000 -g user user
```

```
$ docker run --volume $(pwd):/src --user user myapp touch /src/created_in_container
$ ls -l
-rw-r--r-- 0 nigel  23 Feb 16:39 created_in_container
```

How Could We Improve Flexibility?

Cater for different users and IDs

Avoid rewriting of the Dockerfile

Using Build Arguments

Dockerfile

```
FROM debian

ARG UID=1000
ARG GID=1000

# Add a group and user to match the user on the host
RUN groupadd -r --gid $GID user \
    && useradd -r --uid $UID -g user user
```

```
$ docker build --build-arg UID=1001 --build-arg GID=1001 -t myapp .
```

Demo

Developing an application using a bind mount

- Create a Docker image for the app
- Bind mount the source code into a container
- Make a source code change
- Watch and test the hot reload feature

Up Next:

Separating Application Build and Execution with Multi-stage Builds

Module Summary

What we covered:

- The inner loop of development
- Developing inside containers
- Types of Docker volume
- Immediate visibility with hot reloading