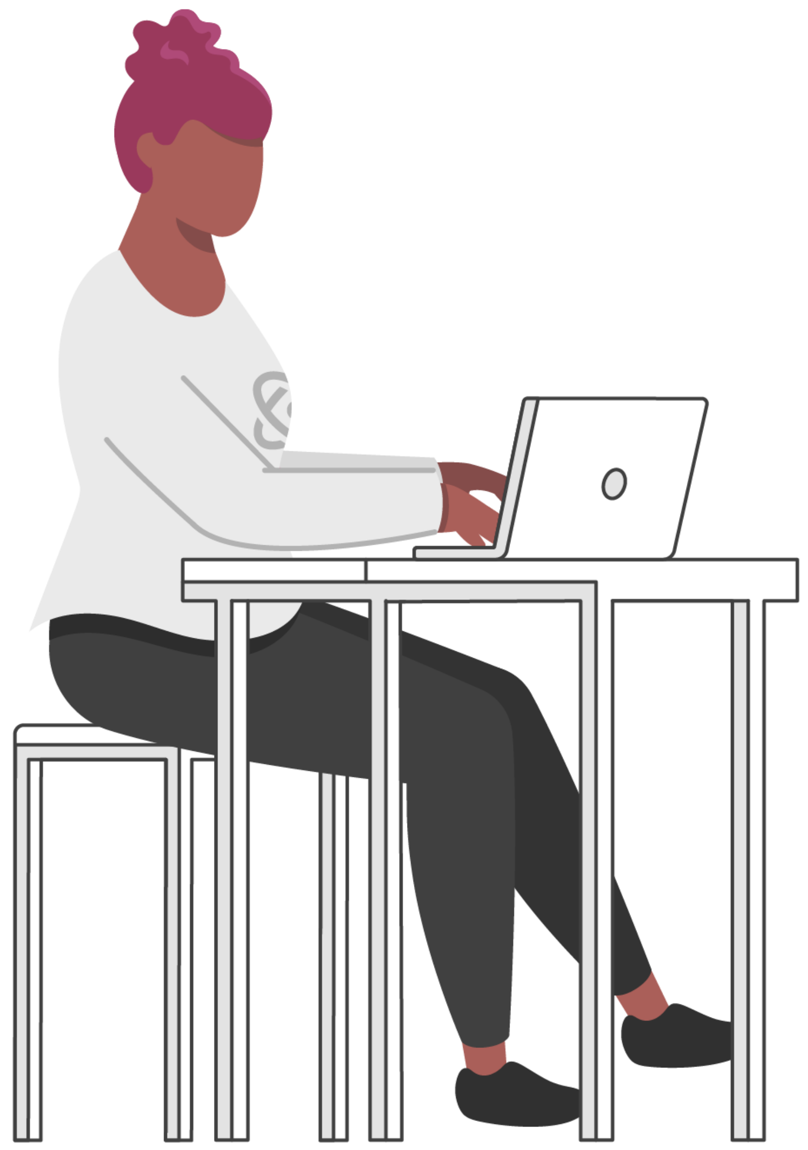
Making Configuration Data Available to Containerized Applications

Nigel Brown

@n_brownuk www.windsock.io



Mia is on a journey to Docker adoption

- Learned how to use containers in the software development lifecycle
- Leverages Docker's features for the situation in hand
- Applies best practice when authoring Dockerfiles
- How does Docker cope with multiple target environments?

Let's see what Mia learns about configuration

Module Outline

Coming up:

- Providing configuration data to software applications
- Defining configuration in Dockerfiles
- Using Docker’s flexibility to make configuration available when it’s needed
- Using environment variables to control image builds and container execution

Application Configuration

An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc).

The Twelve-Factor App, https://12factor.net/config

Examples of Configuration

Database server URL for an application's back end

Port the application listens on for client requests

An application's logging level for debugging purposes

The version of the app or API served by the environment

Twelve Factor App Recommendations

Separate code and config

Don't define configuration as constants in the application's source code

Store config in env vars

Do use environment variables to define the application's configuration

Benefits of Configuration in the Environment

No leakage of sensitive information hard-coded in software applications

Straightforward onboarding of new environments to host software applications

No need to re-test software applications due to changes to the configuration

Configuration defined in env vars is agnostic concerning languages and operating systems

Environment Variables

```
func main() {

    <snip>

    if len(*url) == 0 {
        *url = os.Getenv("REDIS_URL")
        if len(*url) == 0 {
            fmt.Println("a URL must be specified")
            flag.Usage()
            os.Exit(1)
        }
    }

    <snip>

}
```

ENV Dockerfile Instruction

Variable definition
Used to define configuration items in the environment

Available for builds
Variables can be used by other Dockerfile instructions

Persists to container
Variables are present in containers derived from image

Using the ENV Instruction

Method #1

```
<snip>

ENV REDIS_HOST “redis_server”

<snip>
```

Method #2

```
<snip>

ENV REDIS_HOST=“redis_server” \
    REDIS_PORT=6379

<snip>
```

Method #2 is recommended over method #1

```
$ docker build -t redis .
$ docker run --rm redis printenv REDIS_HOST
redis_server
```

Environment Variables in Containers

A containerized app can read the value of the variable from its environment

Using Variables in Image Builds

Dockerfile

```
<snip>

ENV URL="https://nginx.org/download" \
    VERSION="1.18.0"

RUN wget -q "${URL}/nginx-${VERSION}.tar.gz"      && \
    wget -q "${URL}/nginx-${VERSION}.tar.gz.asc“

<snip>
```

Defining Variables at Point of Build

The values held by environment variables can be assigned when image builds are initiated.

Build Arguments

Arguments are provided on the command line when an image build is initiated with a Dockerfile

A corresponding variable is defined in the image's Dockerfile using the ARG instruction

The value provided for the variable on the command line is substituted into the Dockerfile

Using the ARG Instruction

Dockerfile

```
<snip>

ARG VERSION="1.18.0"

ENV URL="https://nginx.org/download"

RUN wget -q "${URL}/nginx-${VERSION}.tar.gz"     && \
    wget -q "${URL}/nginx-${VERSION}.tar.gz.asc"

<snip>
```

```
$ docker build --build-arg VERSION="1.19.5" ...
```

Building Images with Build Arguments

Value of variable in image is provided by build argument on the command line

Different image variants can be built without altering the Dockerfile

ARG Instruction or ENV Instruction?

ARG Instruction	ENV Instruction
For values only known at build time	Generally used for defining variables
Useful for variables required for builds	Useful for persisting variables in image
Scoped from line in which it is defined	ENV variables trump ARG variables
Not visible when inspecting image	Visible in image’s configuration

Persisting a Build Argument Variable

Dockerfile

```
<snip>

ARG VERSION

ENV URL="https://nginx.org/download" \
    ENV="${VERSION:-1.18.0}"

RUN wget -q "${URL}/nginx-${VERSION}.tar.gz"      && \
    wget -q "${URL}/nginx-${VERSION}.tar.gz.asc"

<snip>
```

Defining Configuration

Authoring
Dockerfile
ENV instruction

Building
Dockerfile & CLI options
ARG instruction, --build-arg

Running
CLI options
--env, --env-file

Defining Configuration

Authoring
Dockerfile
ENV instruction

Building
Dockerfile & CLI options
ARG instruction, --build-arg

Running
CLI options
--env, --env-file

Defining Configuration

Authoring
Dockerfile
ENV instruction

Building
Dockerfile & CLI options
ARG instruction, --build-arg

Running
CLI options
--env, --env-file

```
$ docker run --rm --env REDIS_HOST=redis_server --env REDIS_PORT=6379 redis
```

Setting Variables at Runtime

Configuration as environment variables can be set using the -e, --env CLI option

If already defined in the Dockerfile, the CLI definition overrides the set values

```
$ export REDIS_HOST=redis_server REDIS_PORT=6379
$ docker run --rm --env REDIS_HOST --env REDIS_PORT redis
```

Using Exported Variables at Runtime

Values for variables exported in the environment can also be provided to a container

Reading Environment Variables from a File

redis.env

```
REDIS_HOST=redis_server
REDIS_PORT=6379
```

```
$ docker run --rm --env-file $(pwd)/redis.env redis printenv REDIS_HOST REDIS_PORT
REDIS_HOST=redis_server
REDIS_PORT=6379
```

Don’t be tempted to expose secrets as environment variables. They leak.

Demo

Consuming configuration from within a container

- Amend Dockerfile to include ARG instruction for NODE_ENV variable
- Persist variable using combination of ARG and ENV instructions
- Configure alternative for app’s port using configuration in the environment

Up Next:

Configuring Logging for Containerized Applications

Module Summary

What we covered:

- Config in the environment for apps
- Defining variables in Dockerfiles
- Using the CLI to define variables
- Methods for making variables available at the preferred point in time
- Config consumption in containers