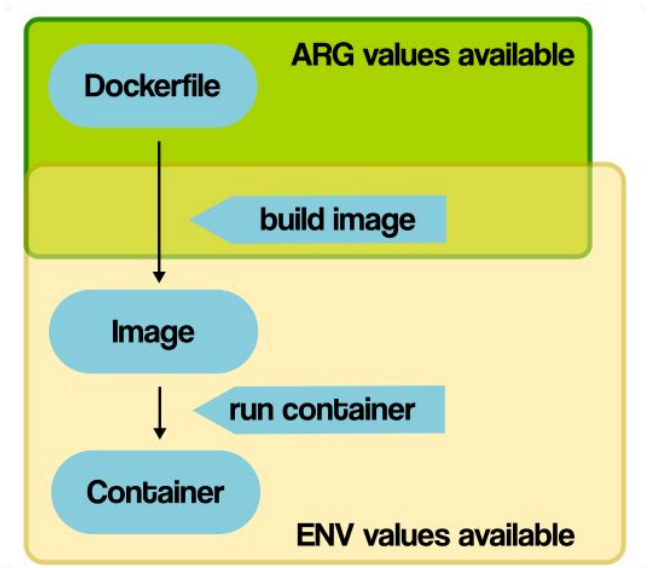
Docker ARG, ENV, env_files & .env Cheatsheet

ARG

- **built-time** arguments
- not available in containers

ENV

- **environment** variables
- available in containers



ARG and ENV in Your Dockerfile

```
ARG required_var       # expects a value on build
ARG var_name=def_value # default ARG
ENV foo=other_value    # default ENV
ENV bar=${var_name}    # dynamic ENV from ARG
```

".env" and "env_file"

.env

- has to be named `.env`, used by docker-compose
- values replace $var in docker-compose.yml
- preview with: `$ docker-compose config`

env_file

- can have any name, really
- has to be passed with `--env-file`

Same Syntax for Both:

```
env_var_name_a=another_value
env_var_name_b=yet_another_value
```

Docker CLI

Set Build Argument Value

```
$ docker build --build-arg variable_name=a_value [...]
```

Set Environment Values For Containers

Provide env var value: `$ docker run -e "foo=other_value" [...]`
Pass env var from host: `$ docker run -e foo [...]`
Pass env vars from env_file: `$ docker run --env-file=env_file_name [...]`
(Sometimes) Part of CMD: `$ docker run myimage SOME_VAR=hi python app.py`

Priority of Environment Variables (left overrides right):

`Part of CMD` > `-e` > --env-file > ENV

This docker-compose.yml snippet...

```
services:
  service_a:
    args:
      a: a_val
    environment:
      b: b_val
    env_file: file_name_a
```

... is the same as

```
services:
  service_a:
    args:
      - a=a_val
    environment:
      - b=b_val
    env_file:
      - file_name_a
```

Bonus: Docker CLI Commands to Investigate ENV And ARG

```
$ docker inspect container_id
$ docker history image_id
$ docker exec -it container_id env
```