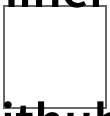
~~DOCKER-COMPOSE~~

DOCKER COMPOSE

linkedin.com/in/arthurio

minervaproject.com



github.com/arthurio/docker-compose-slides

🙏

PLEASE USE AN ALIAS

```
alias dc="docker compose"
```

DOCKER-COMPOSE.YML

🍽️

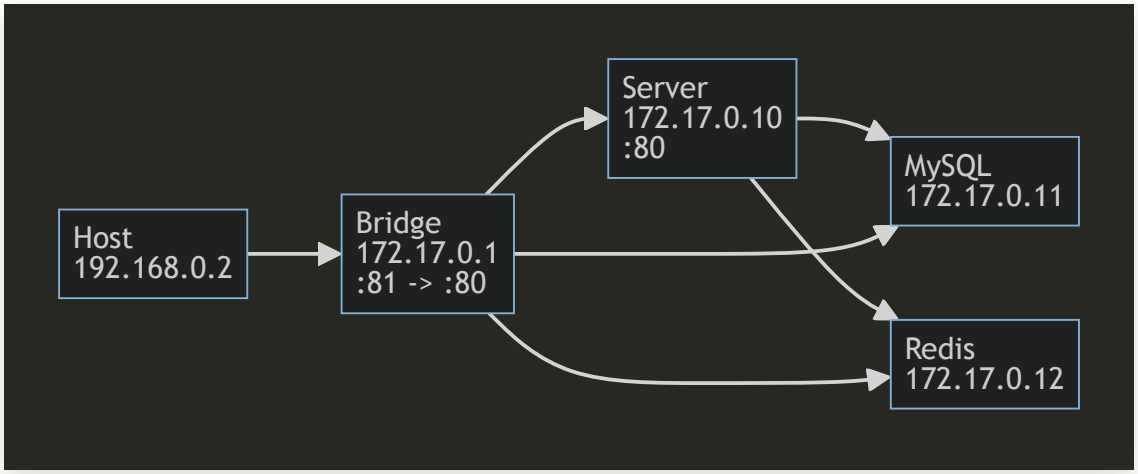
SERVICES

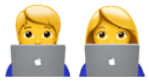
```
services:
  my-service:
    image: my-image
    build:
      ...
    entrypoint:
      - /bin/bash
    command:
      - sleep
      - infinity
    ports:
      - "80:80"
    environment:
      - FOO=foo
    volumes:
      - ./src:/src
    environment:
      FOO=foo
    links:
      - my-db
    depends_on:
      - my-setup
```

VOLUMES

```
services:
  my-service:
    volumes:
      - my-empty-volume:/data
volumes:
  my-empty-volume:
```

NETWORK


Server
172.17.0.10
:80
MySQL
172.17.0.11
Host
192.168.0.2
Bridge
172.17.0.1
:81 -> :80
Redis
172.17.0.12



MAIN COMMANDS

UP

- Run in the background

```
dc up -d
```

- Target a specific service

```
dc up -d my-service
dc up -d --no-deps my-other-service
```

DOWN

- Take down your services

```
dc down
```

- Remove orphans as well

```
dc run one-off  # ran without --rm
dc down --remove-orphans
```

RUN

- Keep the container around after completion

```
dc run tests
```

- Remove the container after completion

```
dc run --rm tests
```

RUN (CONTINUED)

- Publish service ports

```
dc run -P tests  # All ports
dc run -p 81:80  # Specific port
          ^   ^
        host   container
```

- Set environment variables

```
dc run -e FOO=foo tests
export BAR=bar
dc run -e BAR tests
```

LOGS

```
dc logs  # Print all logs
dc logs -f  # Print and follow all logs
dc logs -f my-service my-other-service #
Specific services
```

LS / PS

- Show all active projects

```
dc ls
```

- Show all active services for current project

```
dc ps
```

DEBUGGING

- Bring up your services

```
dc up -d
```

- Stop the one you want to debug

```
dc stop my-server
```

- Add a breakpoint somewhere in your code, then:

```
dc run -P --rm my-server
```

WARNING

If you are running your server with multiple threads, the debbuging console's rendering may get scrambled. To fix this, you can run the server with only one thread.

REMOTE DEBUGGING

One other option if you can't afford single threads is to use a `remote` debugger. Set the `breakpoint` in you server that you have kept running with `dc run` and then attach to it with your IDE or debugger program.

```
from pudb.remote import set_trace
set_trace()
```

Make sure that port `6889` is in your service's ports, then:

```
dc up -d
telnet 127.0.0.1 6899
```

VSCODE (FROM COPILOT)

- Install the `ms-vscode-remote.remote-containers` extension
- Open the command palette and run `Remote-Containers: Attach to Running Container...`
- Select the container you want to debug
- Add a breakpoint in your code
- Run the debugger

THINGS I LEARNED RECENTLY

PROFILES

Don't start automatically with `up` unless specified.

```
services:
  my-service:
    profiles:
      - shell
```

You don't have to specify the profile when using `run` but you do for `up`.

```
dc --profile shell up
dc --profile shell down
dc run --rm my-service
```

DEPENDENCY CONDITIONS

```
depends_on:
  my-service:
    condition: service_started
  my-setup:
    condition:
service_completed_successfully
```

NETWORK ALIASES

```
nginx:
  ...
  networks:
    default:
      aliases:
        - my-app.local.com
        - my-admin.local.com
```

PROJECTS

By default docker compose uses the name of the parent folder of the `docker-compose.yml` file. But if you have a different configuration file for something like integration tests, you can specify the project name with the `--project-name` flag. This allows you to keep the same service names in both configuration files and not add different prefixes/suffixes.

```
alias dci='docker compose --project-name
"${PWD##*/}"-test -f docker-
compose.integration-tests.yml'
```