Multi-Container Application

Basic concepts

Roberto Valentini (INFN Bari)
roberto.valentini@ba.infn.it

Outline

- Multi-Container Application
  - Why
  - Communication
  - Docker CLI
- Docker-compose
  - Services
  - Volume, Network and Dependencies
  - CLI
- References

Multi-Container Application: Why

At some time we need to use multiple service for our application
Example add a database to a website.

The correct choice is to containerize every service individually.

This because you want for example:

- Update incompatible library
- Update one of the service
- Scale your service independently

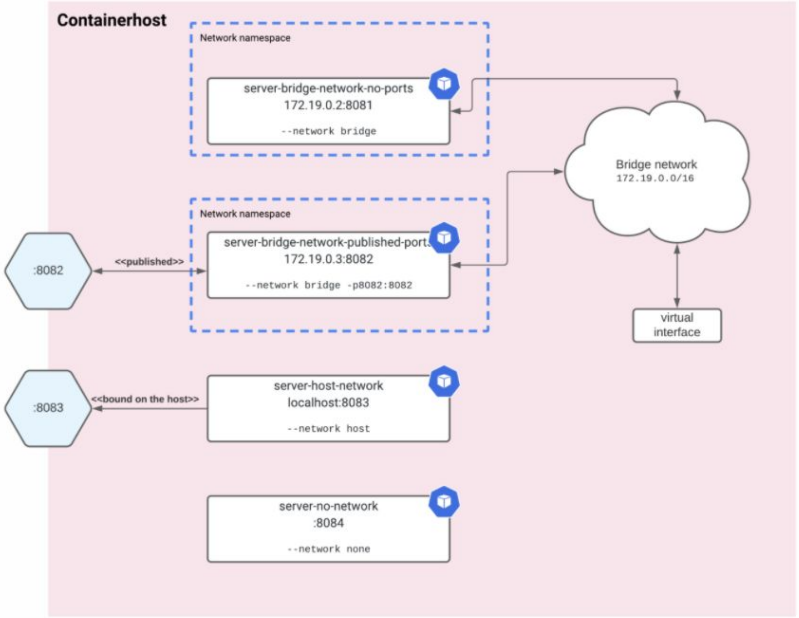
Remember that containers, by default, run in isolation and don’t know anything about other processes or containers.

Multi-Container Application: Comunication

If two containers are on the same network, they can talk to each other.

If they aren't, they can't.

Bridge network can be used when multiple containers are running in standard mode and need to communicate with each other.
We can use dedicated network per Multi-Container Application.



Multi-Container Application: Docker CLI

For running a Multi-container application like a phpmyadmin we can use for example this command in docker cli.

To start, stop and restart the application we need to act on both container.

In a complex application this need more effort.

```
docker run -d \
  --network phpmyadmin_net --network-alias db \
  -v db-data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=s3cret \
  -e MYSQL_USER=phpma \
  -e MYSQL_PASSWORD=s3cret \
  mariadb:10.3
```

```
docker run -d \
  --network phpmyadmin_net --network-alias phpmyadmin \
  -p 8080:80 \
  phpmyadmin/phpmyadmin
```

Docker-Compose

Docker Compose is a tool to help define and share multi-container applications.
With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.

These YAML rules, both human-readable and machine-optimized, provide us an effective way to snapshot the whole project in a few lines.

Almost every rule replaces a specific Docker command so that in the end we just need to run: `docker-compose up`

```yaml
version: "3.7"
services:
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - 8080:80
    network:
      - phpmyadmin_net
  db:
    image: mariadb:10.3
    expose:
      - 3306
    environment:
      "MYSQL_ROOT_PASSWORD": "s3cret"
      "MYSQL_USER": "phpma"
      "MYSQL_PASSWORD": "s3cret"
    volumes:
      - db-data:/var/lib/mysql
    networks:
      - phpmyadmin_net

volumes:
  db-data:

networks:
  phpmyadmin_net: {}
```

Phpmyadmin compose example

Docker-Compose: Services

First of all we need to define our Multi-Container Application Services. Every services was a component of our application and can be based on different image.
Alternatively to use a registry image we can build image from Dockerfile.

```
services:
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
  db:
    image: mariadb:10.3
```

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  ports:
    - 8080:80
db:
  image: mariadb:10.3
  expose:
    - 3306
  environment:
    "MYSQL_ROOT_PASSWORD": "s3cret"
    "MYSQL_USER": "phpma"
    "MYSQL_PASSWORD": "s3cret"
```

Now we can customize the container like we do on cli

- set environment variable or file
- expose port
- set restart policy
- map volume
- map port
- ecc

Docker-Compose: Volume, Network and Dependencies

```
    image: mariadb:10.3
    volumes:
      - db-data:/var/lib/mysql

volumes:
  db-data:
```

For persistent data we can define a volume and map it into the container.
Alternatively we can map local filesystem path to container path.

By default Compose sets up a single network for your app.
You can explicitly define the network name.
Each container for a service is both *reachable* by other containers on that network, and *discoverable* by them at a hostname identical to the container name.

```
    image: mariadb:10.3
    networks:
      - phpmyadmin_net

volumes:
  db-data:

networks:
  phpmyadmin_net: {}
```

```
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    depends_on:
      - db
```

In a environment where one service depends on another, we can define this dependencies to force docker-compose to initialize the environment with the correct sequence.

Docker-Compose: CLI

Now the YAML describe out application.

We can use docker-compose command line interface to manage the application.

With `docker-compose up` we can initialize and start our environment

With `docker-compose start` we can start our already created environment

With `docker-compose restart` we can start our already created environment

With `docker-compose stop` we can stop our already created environment

With `docker-compose down` we can stop and destroy our environment

Handson link

https://baltig.infn.it/rvalentini/handson-docker-comopse

References

- https://docs.docker.com/compose/
- https://en.wikipedia.org/wiki/YAML
- https://docs.docker.com/compose/reference/
- https://docs.docker.com/get-started/08_using_compose/