Introduction to Docker Compose

INTERMEDIATE DOCKER

Mike Metzger
Data Engineering Consultant

What is Docker Compose?

- Additional command-line tool for Docker
- Define and manage multi-container applications
- Specify containers, networking, and storage volumes in a single file
  - `compose.yml` or `compose.yaml`
  - Older compose files may be named `docker-compose.yaml`
- Easy to share / demo applications

Example compose.yaml

```yaml
# Define the services
services:
  # Define the container(s), by name
  webapp:
    image: "webapp"
    # Optionally, define the port forwarding
    ports:
      - "8000:5000"
  # Define any other containers required
  redis:
    image: "redis:alpine"
```

Starting an application

- `docker compose up`
  - On older systems, `docker-compose up`
  - `docker compose -f <yaml> up`
  - `docker compose up -d`

```
$ docker compose up
[+] Running 2/0
 ? Network composetest_default    Created
 ? Container composetest-redis-1  Created
 ? Container composetest-web-1    Created
Attaching to redis-1, web-1
redis-1  | 1:C 11 Mar 2024 04:09:51.754 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
web-1    |  * Serving Flask app 'app.py'
web-1    |  * Running on http://127.0.0.1:5000
```

Checking status of applications

- `docker compose ls`

```
$ docker compose ls
NAME                STATUS              CONFIG FILES
webapp              running(2)          /webapp/docker-compose.yml
```

Stopping an application

- `docker compose down`
  - `docker-compose down`
  - `docker compose -f <yaml> down`

```
$ docker compose down
[+] Running 3/3
 ? Container composetest-redis-1  Removed
 ? Container composetest-web-1    Removed
 ? Network composetest_default    Removed
```

Let's practice!

INTERMEDIATE DOCKER

Creating compose.yaml files

INTERMEDIATE DOCKER

Mike Metzger
Data Engineering Consultant

YAML

- Yet Another Markup Language
  - YAML Ain't Markup Language
- Text file, but spacing matters (like Python)
- Used in many development scenarios for configuration
- Rules can be tricky, mainly keep entries lined up as in examples

```yaml
services:
  postgres:
    container_name: postgres
    image: postgres:latest
    ports:
      - "5432:5432"
    restart: always
  pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4:latest
    ports:
      - "5050:80"
    restart: always
```

Main sections

- Different sections handle different components
- `services:` list the containers to load
- `networks:` handles networking definitions
- `volumes:` controls any volume mounting
- `configs:` handles configuration options without custom images
- `secrets:` Provides options to handle passwords, tokens, API keys, etc
- Refer to the Docker Compose Documentation for more information

```
services:
  ... # Define containers
networks:
  ... # Define any networking details
volumes:
  ... # Define storage requirements
configs:
  ... # Define special config details
secrets:
  ... # Define passwords / etc
```

[1] https://docs.docker.com/compose/compose-file/

Services section

- Defines all required resources for the application
- Primarily specifies the containers and images to be used
- Extensive options available, but only apply to the individual container(s)
- Indention is applied as needed
- First subsection is the name of each component, followed by the settings

Services example

```yaml
services:
  # Resource name
  postgres:
    # Container name, otherwise random
    container_name: postgres
    # Container image to use
    image: postgres:latest
    # Any port mapping required
    ports:
      # Network details
      - "5432:5432"
  # Next resource
  pgadmin:
      ...
```

- Resource name
- `container_name:` , the assigned name of the container otherwise it's random
- `image:` , which container image to use
- `ports:` , contains a list of any port mapping required
- Followed by next resources required

Additional comments

- `config.yaml` syntax is extensive
  - Covering very small portion of `compose.yaml` options
  - Review the documentation!
- It's typically not required to build a `compose.yaml` file from scratch

[1] https://docs.docker.com/compose/compose-file/

Let's practice!

INTERMEDIATE DOCKER

Dependencies and troubleshooting in Docker Compose

INTERMEDIATE DOCKER

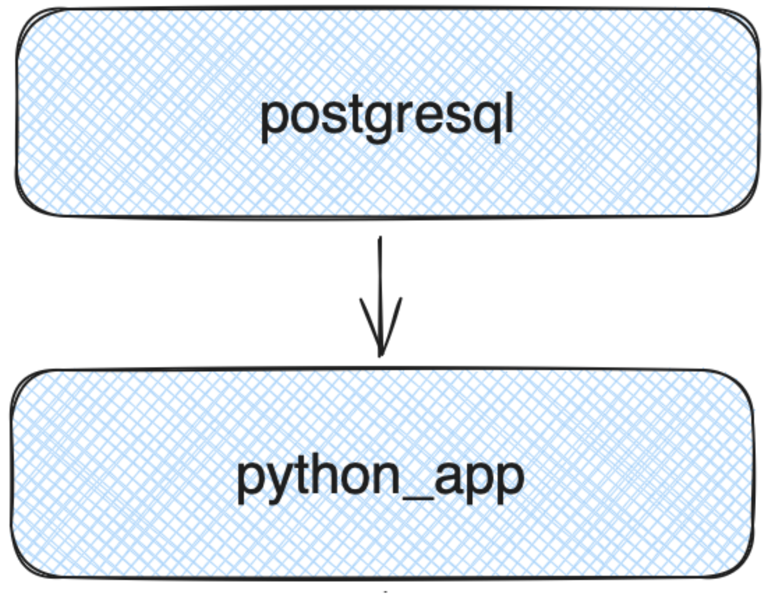
Mike Metzger
Data Engineering Consultant

What are dependencies?

- Dependencies define the order of resource startup
- Resources (containers) may require other resources
- Example web application
  - Database container `postgresql` must start first

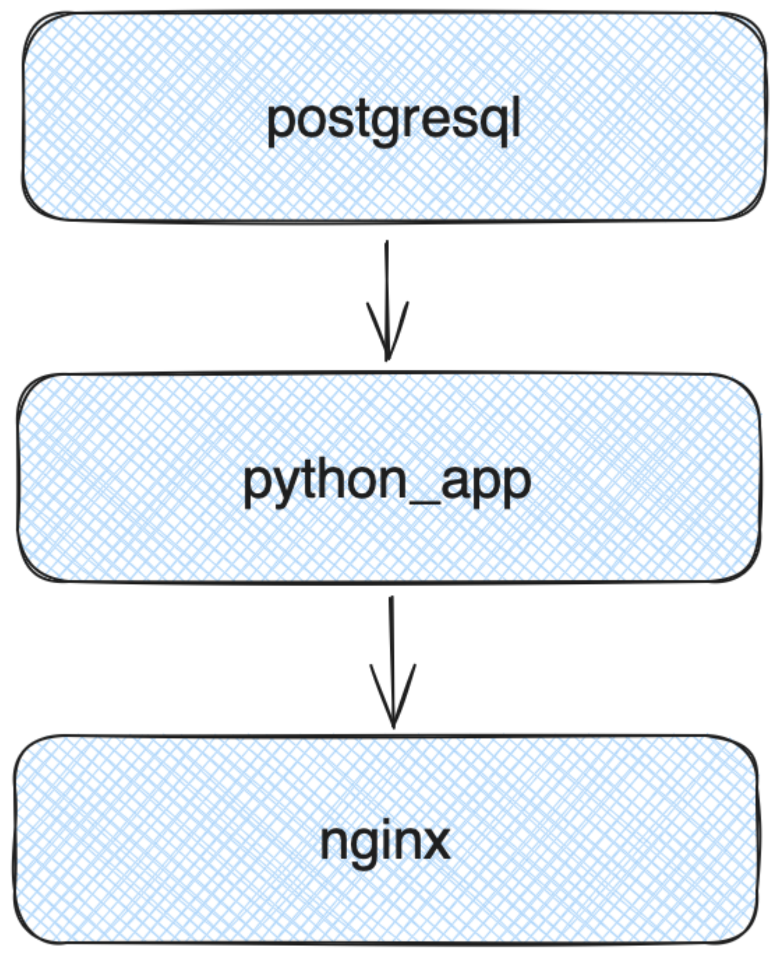
What are dependencies?

- Dependencies define the order of resources
- Resources (containers) may require other resources
- Example web application
  - Database container `postgresql` must start first
  - Then the `python_app`



What are dependencies?

- Dependencies define the order of resources
- Resources (containers) may require other resources
- Example web application
  - Database container `postgresql` must start first
  - Then the `python_app`
  - Finally, the `nginx` web server



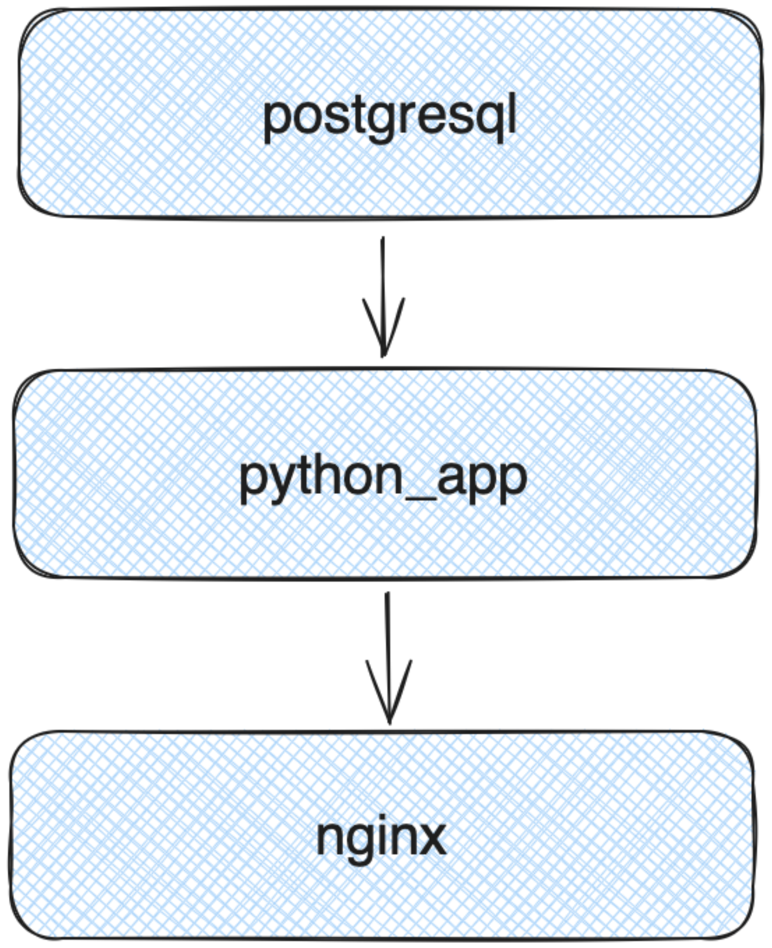
depends_on

- Dependencies defined using the `depends_on` attribute
- Can chain dependencies as per example
- Or, can have multiple dependencies per resource if required
- Order of the `compose.yaml` file does not matter

```
services:
  postgresql:
    image: postgresql:latest

  python_app:
    image: custom_app
    depends_on:
      - postgresql

  nginx:
    image: nginx/latest
    depends_on:
      - python_app
```

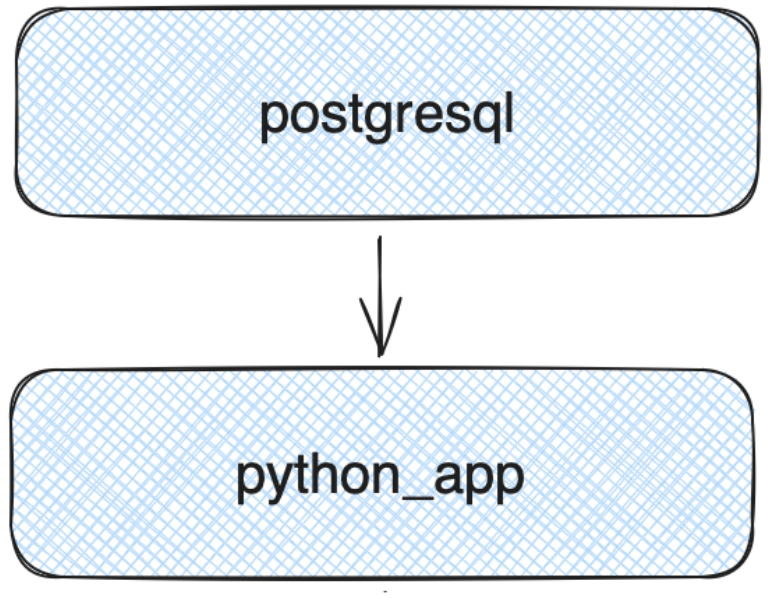
Shutting down applications

- Shutting down an application occurs in reverse order
- Stops `nginx` resource



Shutting down applications

- Shutting down an application occurs in reverse order
- Stops `nginx` resource
- Then stops the `python_app` resource



Shutting down applications

- Shutting down an application occurs in reverse order
- Stops `nginx` resource
- Then stops the `python_app` resource
- And finally the `postgresql` resource

Other options

- Docker Compose provides other options for dependencies
- `condition:` defines how to decide when resource is ready.
  - `service_started` - Resource has started normally
    - Default behavior
  - `service_completed_successfully` - Resource ran to completion, such as a initial configuration / etc
  - `service_healthy` - Resource meets a criteria defined by `healthcheck`

```
services:
  nginx:
    image: nginx/latest
    depends_on:
      python_app:
        condition: service_started

python_app:
    image: custom_app
    depends_on:
      postgresql:
        condition: service_healthy
```

Docker Compose troubleshooting tools

- Docker Compose has additional troubleshooting tools
- `docker compose logs` - Gathers output from all resources in application

```
redis-1  | * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis-1  | * Running mode=standalone, port=6379.
redis-1  | * Server initialized
redis-1  | * Ready to accept connections tcp
web-1    | * Serving Flask app 'app.py'
web-1    |  * Running on all addresses (0.0.0.0)
web-1    |  * Running on http://172.20.0.2:5000
web-1    | Press CTRL+C to quit
```

- `docker compose logs <resourcename>`

docker compose top

- `docker compose top` shows status of resources within an application

```
composetest-redis-1
UID    PID     PPID    C     STIME    TTY    TIME         CMD
999    2767    2726    0     01:16    ?      00:03:27     redis-server *:6379

composetest-web-1
UID     PID     PPID    C     STIME    TTY    TIME         CMD
root    2768    2740    0     01:16    ?      00:00:23     /usr/local/bin/python /usr/local/
```

Let's practice!

INTERMEDIATE DOCKER

Creating a data service within Docker

INTERMEDIATE DOCKER

Mike Metzger
Data Engineering Consultant

Data sharing

- `docker run -v <host directory>:<container directory>`
  - `-v ~/hostdata:/containerdata`

Data sharing in compose.yaml

- Also present in `compose.yaml` files

```
services:
  resource:
    name: resource1

    # Section named volumes
    volumes:
      - <host directory>:<container directory>
      # Such as:
      - ~/hostdata:/containerdata
```

Networks

- `docker run --network <networkname>`
  - `docker run --network net1`
- In `compose.yaml` resources

```
services:
  resource:
    name: resource1

    networks:
      network_name:
      # Such as:
      net1:
```

Port mapping

- `docker run -p hostport:containerport`
  - `-p 8000:8000`
- Available in `compose.yaml` resources

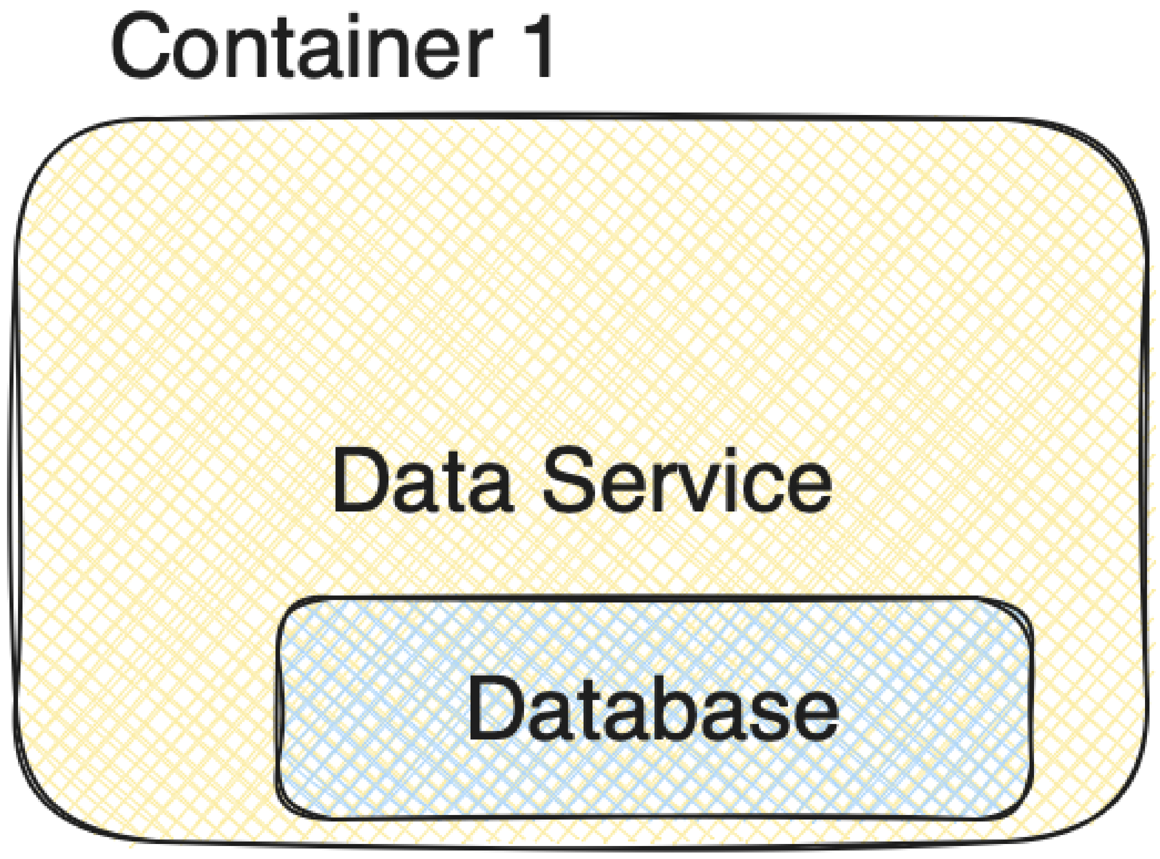
```
services:
  resource:
    name: resource1

    ports:
      - hostport:containerport
      # Such as:
      - 8000:8000
```

docker inspect

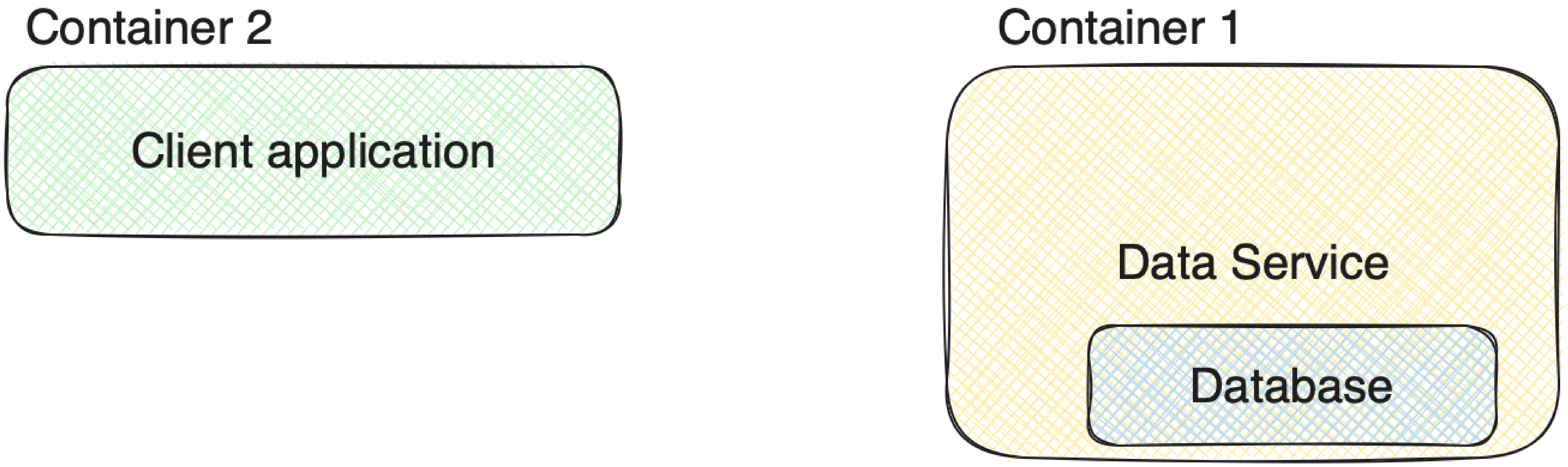
- Determine information about provisioned containers
  - `docker inspect <id / name>`
- Provides various levels of information
  - `Mounts` : Provides mounted data information
  - `NetworkSettings` : Network information
    - `NetworkSettings:Networks` : Shows the Docker network(s) connection details

```
    "Config": {
       "Mounts": [...]
       ...
       "Networks": {
                "network1": {
        ...
```

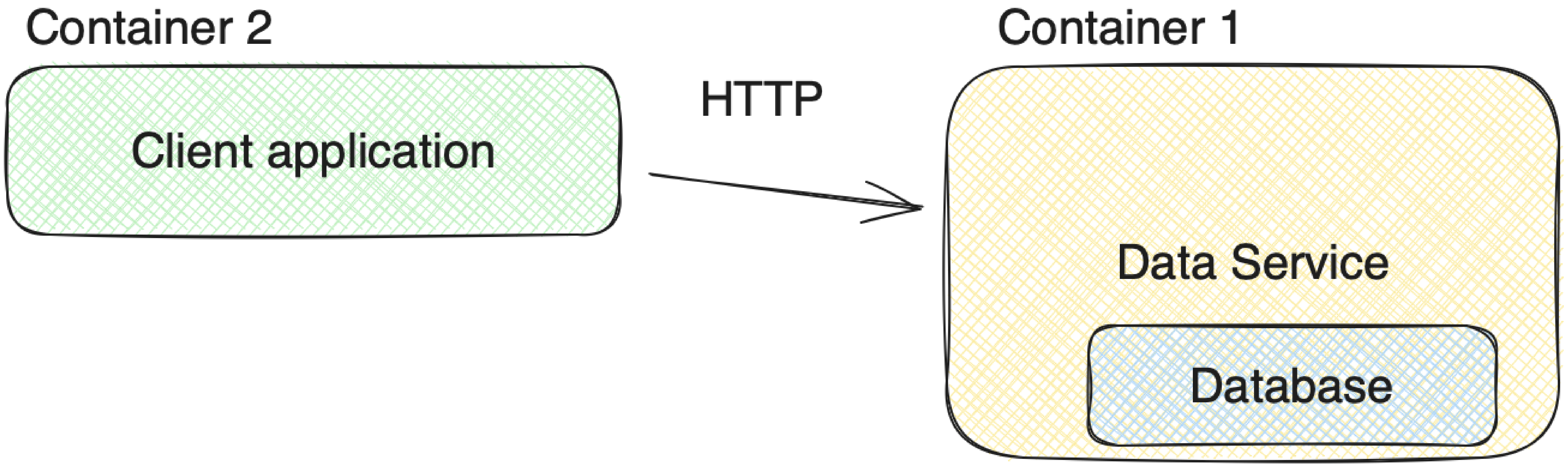
Data service



Data service



Data service



Let's practice!

INTERMEDIATE DOCKER

Course review

INTERMEDIATE DOCKER

Mike Metzger
Data Engineering Consultant

Next steps

- Review Docker documentation **docs.docker.com**
- Containerize more applications
- Create custom repositories
- Docker Swarm
- Kubernetes
- CI/CD
- Mapping to host GPU hardware

Congratulations!

INTERMEDIATE DOCKER