Database container & Volumes

Objectives

On completion of this section you will be able to:

- Explain the need for Docker volumes
- Differentiate between a named volume and a bind mount
- Create a named volume and bind mount
- Deploy your application and database to the cloud the run in containers
- Use a .env file with compose to configure your system
- Use a Docker private network and explain how Docker creates private IPs and DNS names

Inter-microservice communication

Public internet
Azure private network
Docker private network
Virtual machine
Application container
Rabbitmq combined container

80 / 443
kong
8445
public IP:
private IP:

/categories
/question/:category
/moderate
/categories
/categories
/submit
/docs

4000
question
3000
/categories

4100
moderate
3100
/categories
submitted question

4200
submit
3200
categories

4002
MongoDB
27017
mongo vol

4002
MySQL
3306
mysql vol

4001
etl
3001
moderated questions

categories

4102
rmq-admin
15672

4101
MODERATED_QUESTIONS
5672

4201
SUBMITTED_QUESTIONS
5672

4202
rmq-admin
15672

public IP:
private IP:

public IP:
private IP:

public IP:
private IP:

Container port number
Application port number

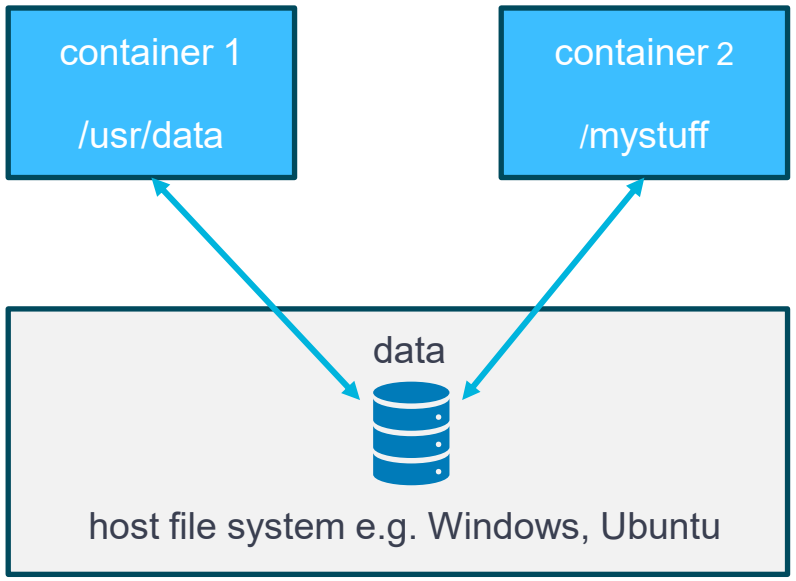
Docker volumes

- You could store data in the image and container. For example, the data files could be stored in, and accessed from, the container by copying them at build time or posting data into them using an API. However, this just makes the image and container larger, and the data is lost when the container is deleted as they are ephemeral
- There are essentially five volume types, but we will focus on the main two: **named volumes** and **bind mounts**. These are used for persisting or accessing data outside the container. These are created on the host but managed by docker. Multiple containers can share a named volume
- As a volume exist on the host, it's stored in a protected docker directory, so on container deletion, the data persists
- A database for example needs somewhere to store its data outside the ephemeral container. A volume satisfies this
- Volumes can be created in a Dockerfile but as we are creating multiple containers, we will use compose so will configure the volumes in there

Named Volume

- This will associate a name with the folder value in the name value pair. It's as if you had created a disk with the specified name. Writing to **/usr/src/data** in the container, will write to a protected docker folder on the file system called **data** (or whatever you call the volume) as if it was a logical disk if declared as follows:
-

```
volumes:
    - data:/usr/src/data
```



compose.yaml

```
services:
  cont1:
    volumes
    - data:/usr/stuff

  cont2:
    volumes
      - data:/mystuff

volumes:
  data:
```

- Compose file maps specific service folder to a volume
- Compose declares the creation of a volume called **data** on the host OS - can be elsewhere also e.g. S3

Docker bind mounts

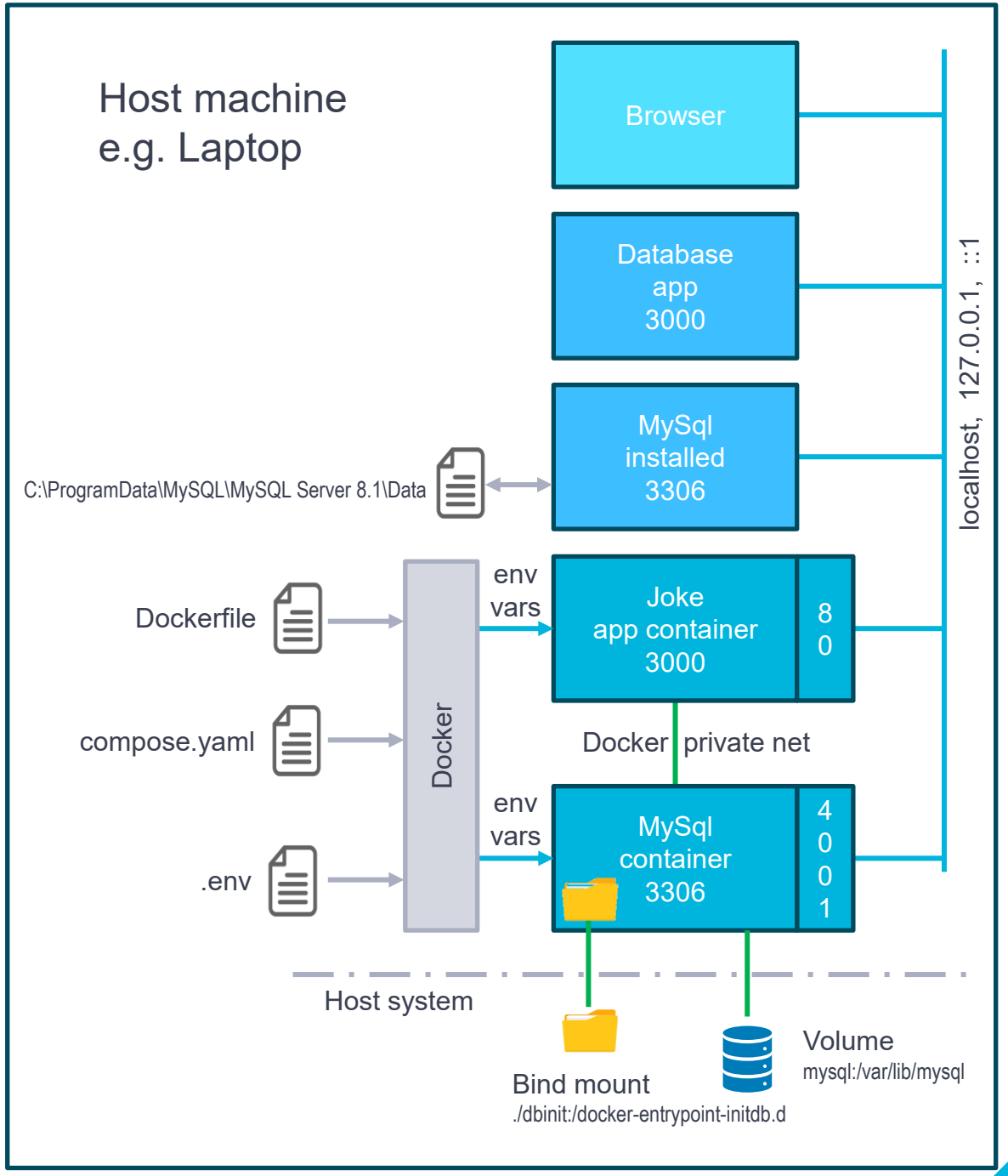
- A bind mount maps a directory in the container to a directory on the host machine
  - Unlike a volume, no space is created, write to one and data is seen in the other
- Say a file or dir outside the container is needed in the container at /usr/app/whatever
- But the data is actually on the host in say /home/username/stuff
  - A bind mount which maps these will enable the container to access /usr/app/whatever but the actual data accessed is /home/username/stuff on the host
- e.g., declare a bind mount as a volume but not to create a volume, just map two folders onto each other
- ```
  volumes:
    - /hostPath:/containerPath
    - /home/username/stuff:/usr/app/whatever
  ```
  ```

App & Database containers

- MySql data stored in files on host file system (Win)
- MySql data Container files stored on file system (Linux)
  - But that is the Alpine Linux in the container
  - Volume maps data outside the container
  - Alpine dir mapped to Ubuntu Docker dir
  - Docker dir Ubuntu: /var/lib/docker/volumes/
  - Docker references as mysql or whatever you named it
- In Windows, volumes are stored in wsl at:
  - \\wsl$\docker-desktop-data\data\docker\volumes

\Demos\data\poor joke sql to improve in lab\app.js

jokePoorMySqlContainers.mp4



Some useful compose commands

- Docker compose has a cli like Docker. However, most of the commands can be used in the desktop if you don't like typing. Details here: https://docs.docker.com/compose/reference/
- **docker compose build** will build all the images defined in the compose.yaml file
- **docker compose up** will start all the containers
- or with one command, **docker compose up --build** will do both
- **docker compose up -d** will run the containers in detached mode
- **docker compose exec -it etl sh** This will start the shell if sh is available. Some images have bash in them too
- **docker compose exec -it mysql-svr bash** The mysql image has bash and sh
- **docker compose exec -it caller sh** This only has sh as it is built from alpine. -it gives us access to a terminal
- **docker exec** does the same thing but uses the container name not the service name
  - **docker exec -it etl-cont sh**
  - **docker exec -it mysql-cont bash**
  - **docker exec -it caller-cont sh**
- **docker compose down** will stop and delete all containers, networks, volumes created with up
  - named volumes won't be deleted unless you add **--volumes or -v**
  - images are not deleted unless you add **--rmi local | all**
- **docker system prune --all -- volumes** deletes everything
- **docker compose logs -f** useful to see all console.log statements when on cloud
- **docker compose restart *service-name*** to restart a stopped container
- **docker compose config** you can check before building, what the compose file looks like after all env substitutions for checking