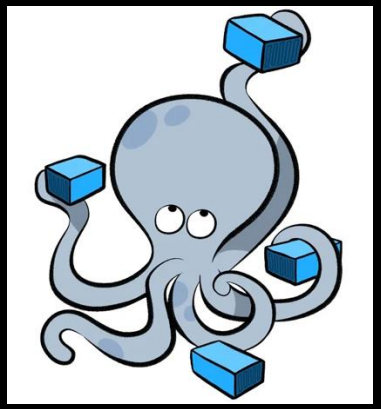
Containerization (Contd.)

Simplifying *Application Deployment* **with Containers**



Docker Compose

Multi-container Applications

Docker Compose

- Without Compose:

1. Build/Pull and run one container at a time.
2. Manually connect containers together.
3. Be careful with dependency relationships and start-up order.

- With Compose:

1. Define multi-container app in compose YAML file.
2. Single command to start entire app.
3. Handles dependencies.
4. Works with Networking, and Volumes.

The process.

1. Define your app's **environment in a Dockerfile.**
2. Define the services that make up your application in a Docker Compose YAML file.
3. Run from the CLI:

   $ docker-compose up

The building blocks.

Sample Compose file

```yaml
services:
  app:
    container_name: profileapp
    image: profile-app:1.0
    build: .
    ports:
      - 3000:3000
    environment:
      - MONGO_PASS=secret
      - MONGO_HOST=mongodb
    depends_on:
      - mongodb
  mongodb:
    container_name: mongoDB
    image: mongo:8.0-rc
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=secret
```

Environment variables in Compose.

- Two options - .env file or command line.

```yaml
1   ervices:
2    app:
6      ports:
7        - 3000:3000
8      environment:
9        - MONGO_PASS=${MONGO_INITDB_ROOT_PASSWORD}
10       - MONGO_HOST=mongodb
11     depends_on:
12       - mongodb
13   mongodb:
14     container_name: mongoDB
15     image: mongo:8.0-rc
16     ports:
17       - 27017:27017
18     environment:
19       - MONGO_INITDB_ROOT_USERNAME=${MONGO_INITDB_ROOT_USERNAME}
20       - MONGO_INITDB_ROOT_PASSWORD=${MONGO_INITDB_ROOT_PASSWORD}
21
```

.env

```
1   MONGO_INITDB_ROOT_USERNAME=admin
2   MONGO_INITDB_ROOT_PASSWORD=secret
```

Networks in Compose.

- Can reference existing networks or create them on startup (up)

```yaml
networks:
  profile-network:
    external: false
    name: 'myapp-network'
services:
  app:
    container_name: profileapp
    image: profile-app:1.0
    build: .
    . . . other configuration settings . . .
    networks:
      - profile-network
    depends_on:
      - mongodb
  mongodb:
    container_name: mongoDB
    image: mongo:8.0-rc
    . . . other configuration settings . . .
    networks:
      - profile-network
```

Volumes (General).

- Volumes map filesystems on the host computer to filesystems in a container, e.g.

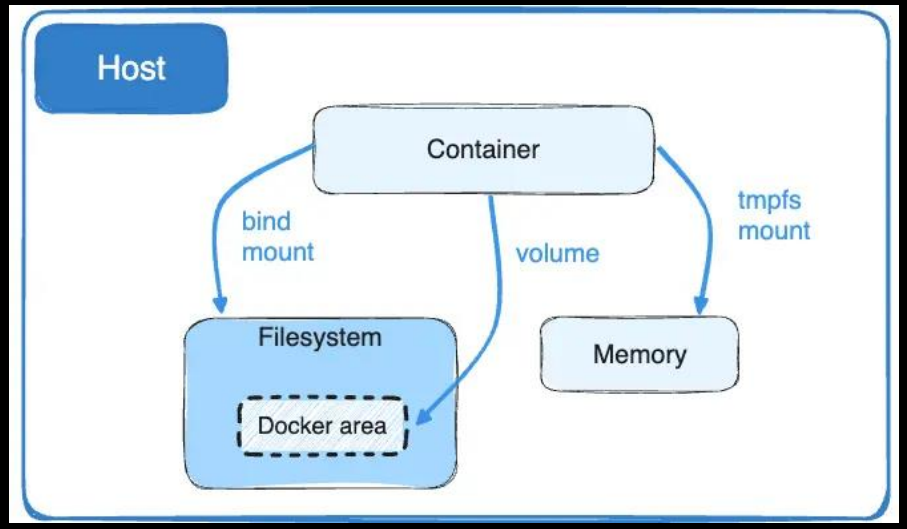
  $ docker run ... **-v ./website:/usr/share/nginx/html …**

- Allows <u>data to persist</u> longer than the lifecycle of the container, e.g. database containers.
  - When a container is removed, all data changes inside the container are lost.

- Two types:
  1. Bind mount (Original)
  2. Named volume

Bind mounts

- A file or directory on the host machine is mounted into a container.
  - It is created on demand if it does not yet exist. (host)
- Bind mounts are very performant, but they rely on the host machine's filesystem having a specific directory structure available.
- Can't use Docker CLI commands to directly manage bind mounts.



Bind mounts

- Creating bind mounts – two formats

  (1) $ docker run -d ***-v source-path:target-path:ro*** \

  --name mycontainer image-name

  – Creates the source (host) file/folder if it does not exist.

  – Target-path – container path

  – Options, e.g. ro (read only).

  (2) $ docker run -d --name mycontainer \

  --mount type=bind,source=source-path,

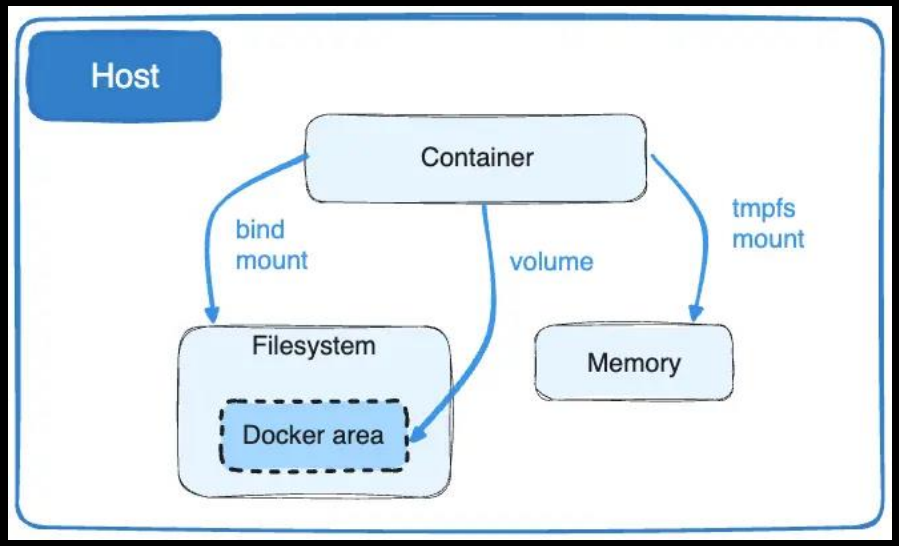
  target=target-path,readonly \

  image-name

  – **Throws an error if source path does not exist.**

Named Volumes.

- Volumes live inside Docker; not visible on host filesystem.
- Use Docker CLI commands to manage named volumes.
- Volume drivers let you store volumes on remote hosts or cloud providers, encrypt the contents of volumes.
- Volumes on Docker Desktop have much higher performance than bind mounts for Mac and Windows hosts.



Named Volumes.

- Created and managed using the CLI.

  $ docker volume create my-vol

  – Anonymous volume – name generated by Docker deamon.

  $ `docker volume ls – list all volumes.`

  $ docker volume inspect my-vol

  $ docker volume rm my-vol

  $ docker volume prune – delete all unused volumes.

- **Start a container that uses a named volume**

  **$ docker run -d --name devtest **

  -v my-vol:target-path image-name

  – **Can use the –mount form as well**

Named Volumes.

- Created and managed using the CLI.

  $ docker volume create my-vol

  – Anonymous volume – name generated by Docker deamon.

  $ `docker volume ls – list all volumes`

  $ docker volume inspect my-vol

  $ docker volume rm my-vol

  $ docker volume prune – remove volumes not used by a container

- **Start a container**

  **docker run -d --name devtest **

  -v my-vol:target-path image-name

  – **Can use the –mount form as well**

Bind mounts in Compose.

```yaml
services:
 backend:
   image: ubuntu
   volumes:
     - ./app:/app:ro
```

```yaml
volumes:
 - type: bind
   source: ./app
   target: /app
   read_only: true
```

This is within a service entry

Named Volumes in Compose.

```yaml
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
```

Creates new volume

Create new volume

```yaml
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
    external: true
```

Use an existing volume