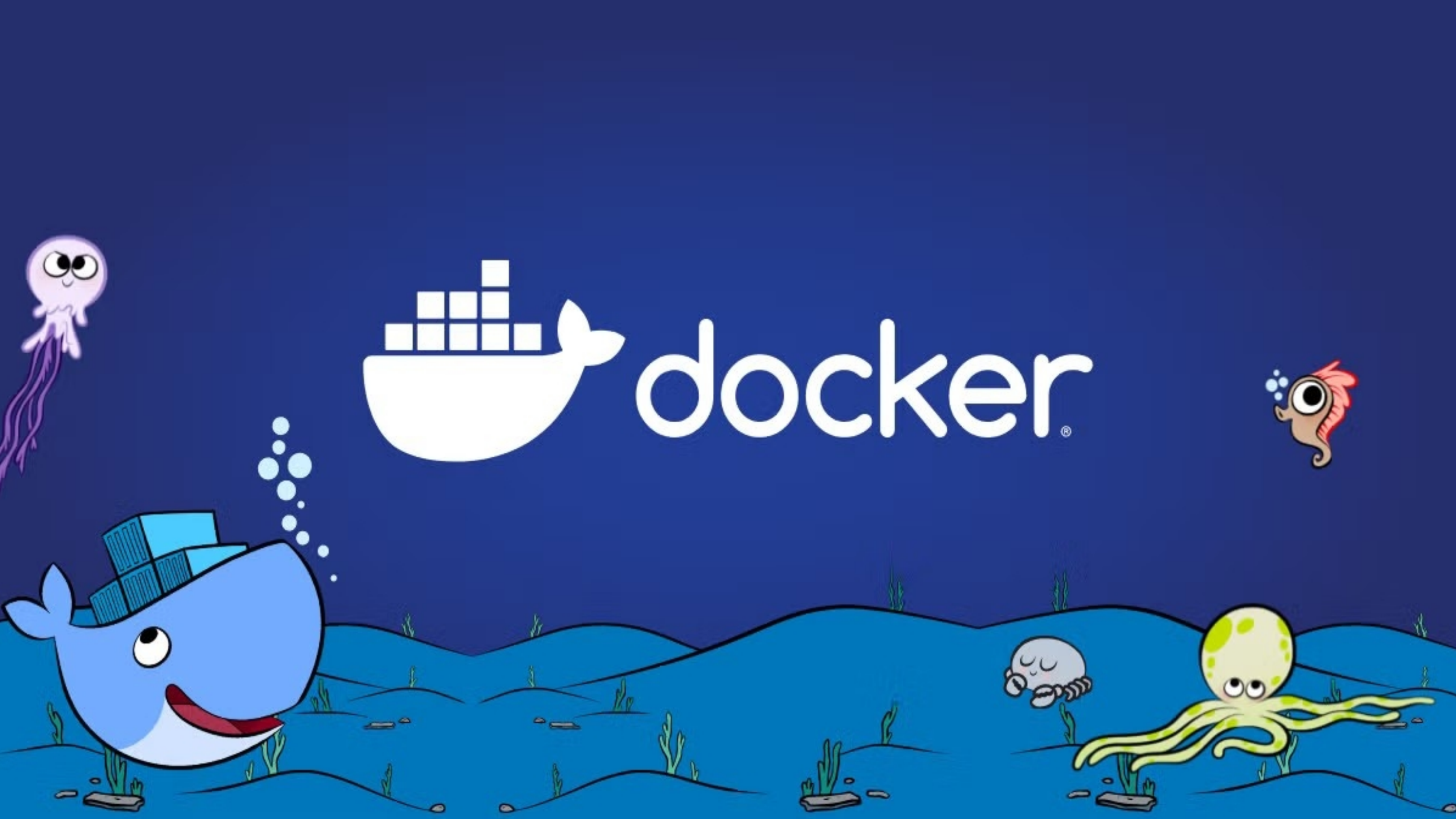
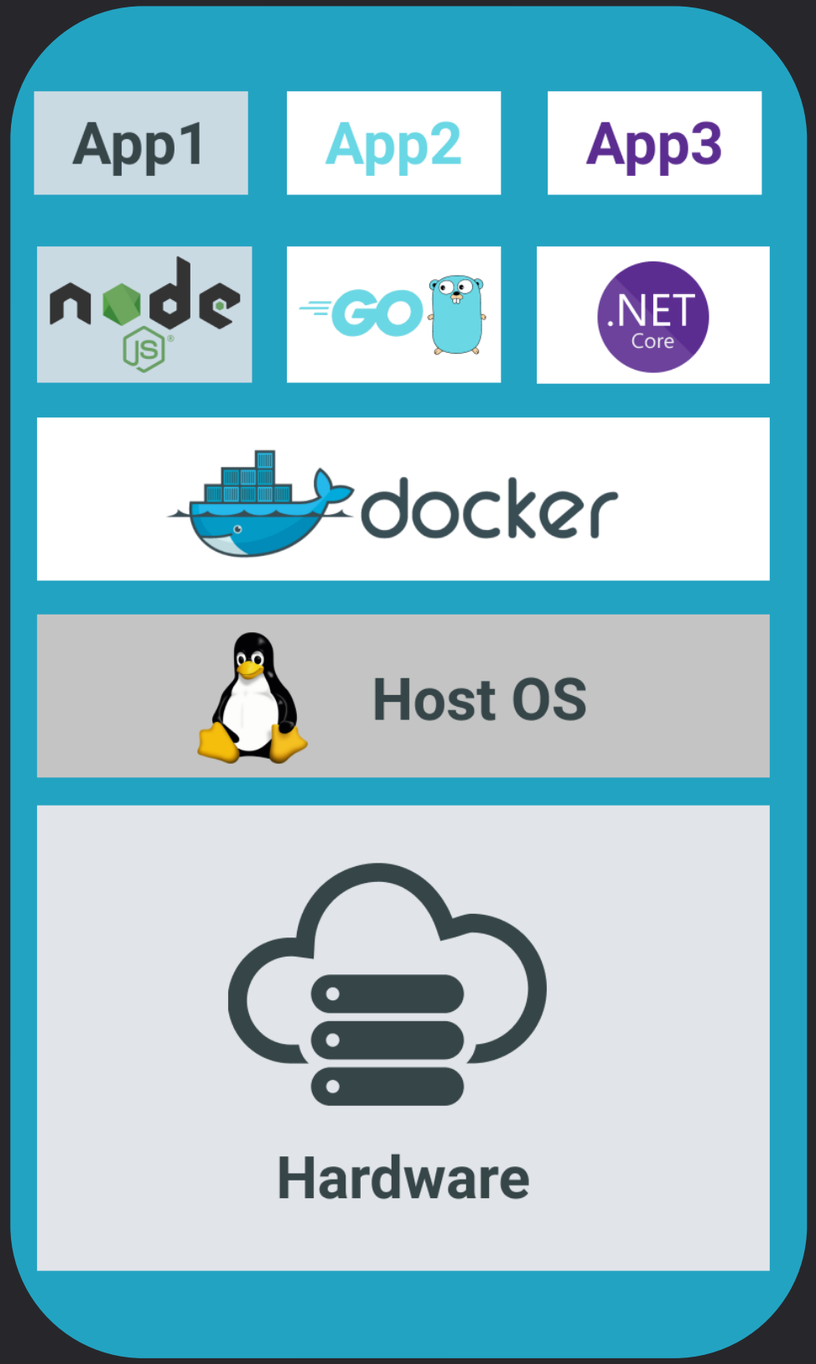

docker



Introduction to Docker

This presentation will introduce Docker. Docker is a platform for developing,
shipping,
and running
applications in containers. We will explain its benefits, architecture, and basic commands. Docker increases efficiency, consistency, and portability.

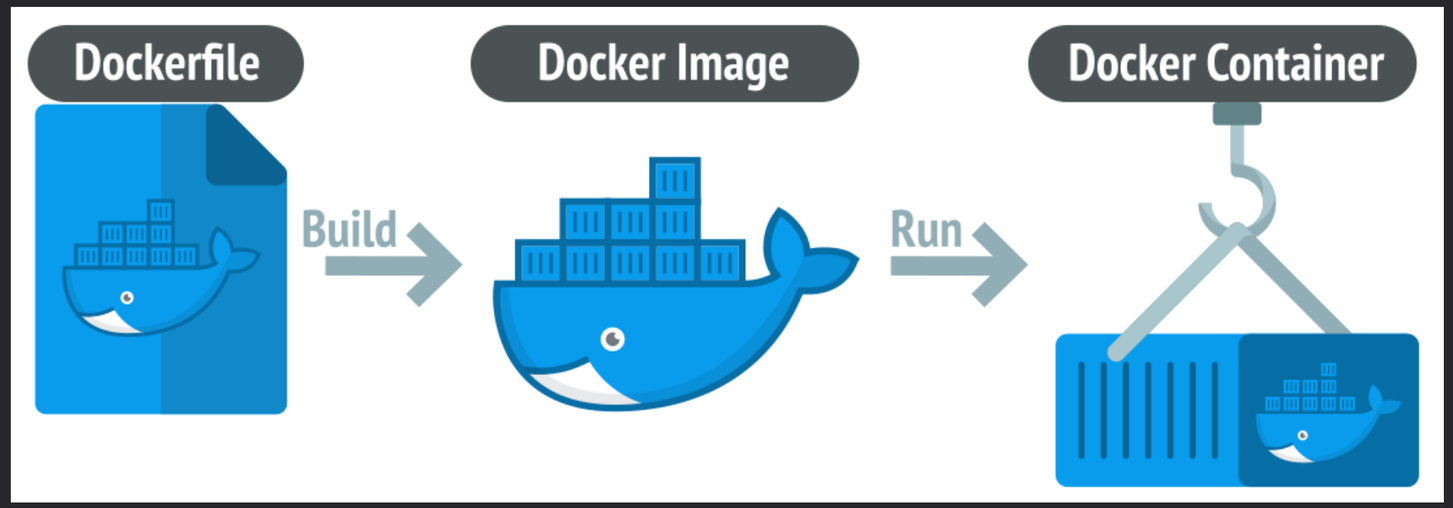
Understanding Containers

What are Containers?

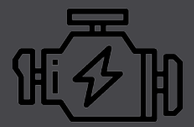
Containers are lightweight, standalone packages of software. They include application code, runtime, system tools, and settings.

Key Benefits

- Isolation from infrastructure
- Resource efficiency
- Sharing host OS kernel



Docker Architecture



Docker Engine

Builds, runs, and manages containers.

Docker Daemon

Manages images, containers, networks.

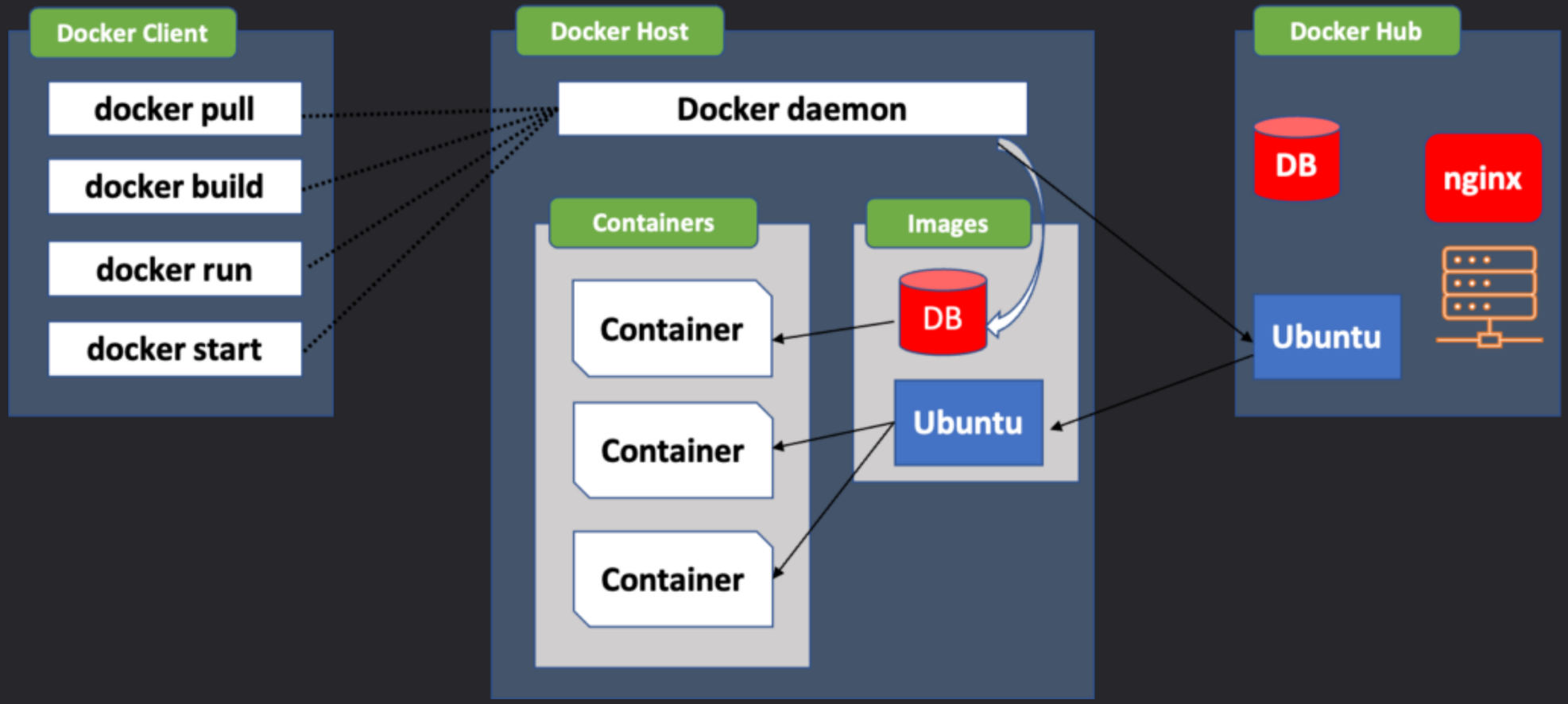
Docker Client

CLI for user interaction.

Docker Hub

Registry for Docker images.

Docker Architecture



Docker Images

Read-only Templates

Used to create containers.

Image Layers

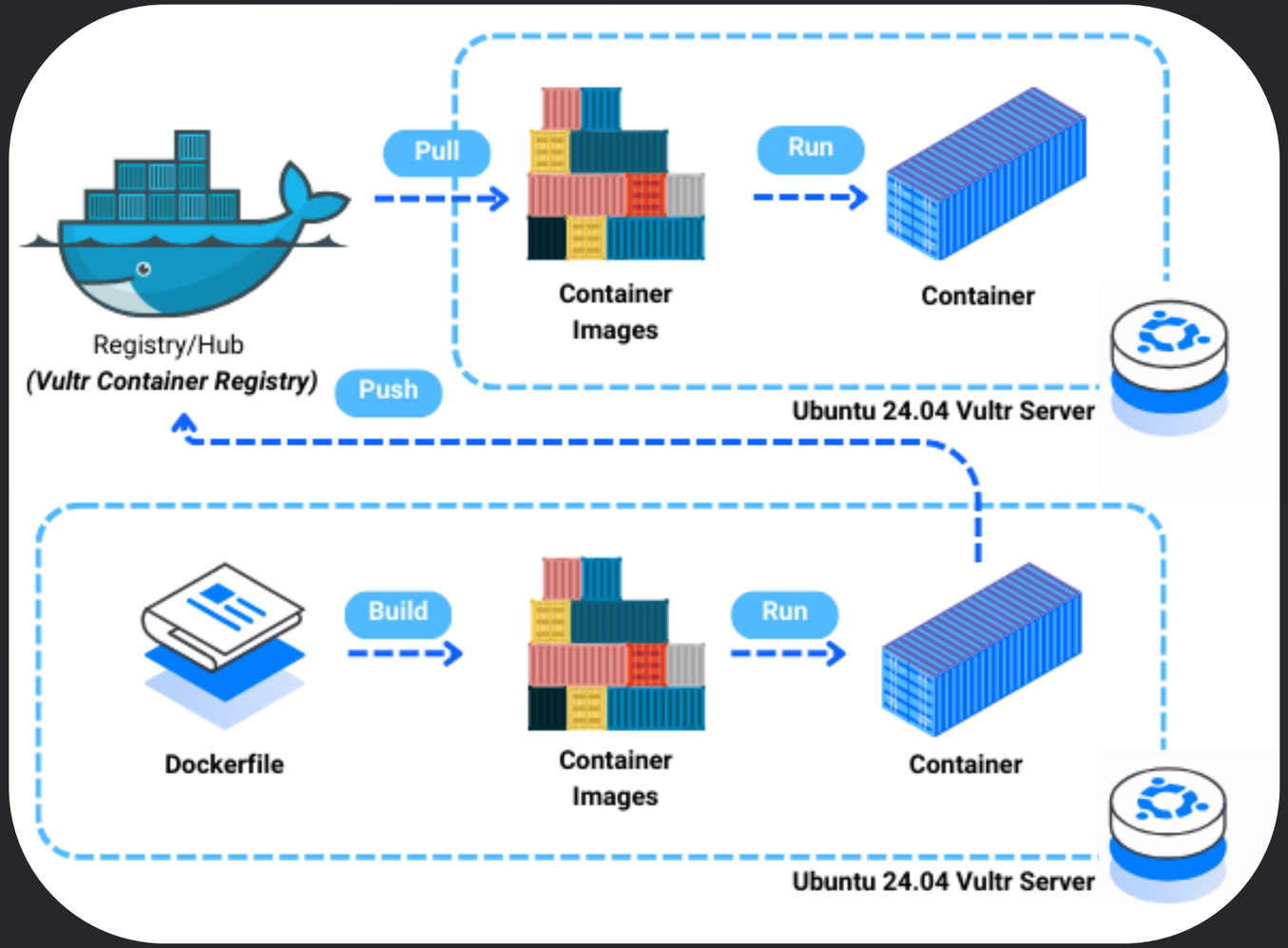
Efficient storage and reuse.

Docker file

Instructions for building images.

Docker Hub

Repositories for sharing images.



Images are built in layers for efficiency. Dockerfiles contain instructions like FROM, RUN, and COPY.

Basic Docker Commands

docker pull

Downloads an image.

docker run

Creates and starts a container.

docker ps — docker ps -a

Lists running containers.

docker stop — docker resume

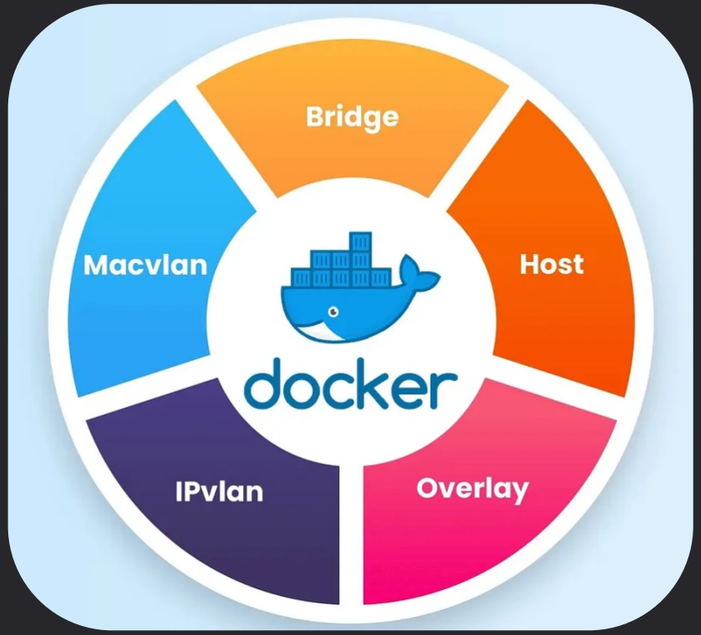
Stops a container.

Commands include pull, run, ps, stop, start, rm, images, rmi, and build. These commands allow users to manage images and containers.

Docker Networking

- **Bridge Network:**

Description: The default network type you create a container without specifying a network. It provides a private internal network on the host where containers can



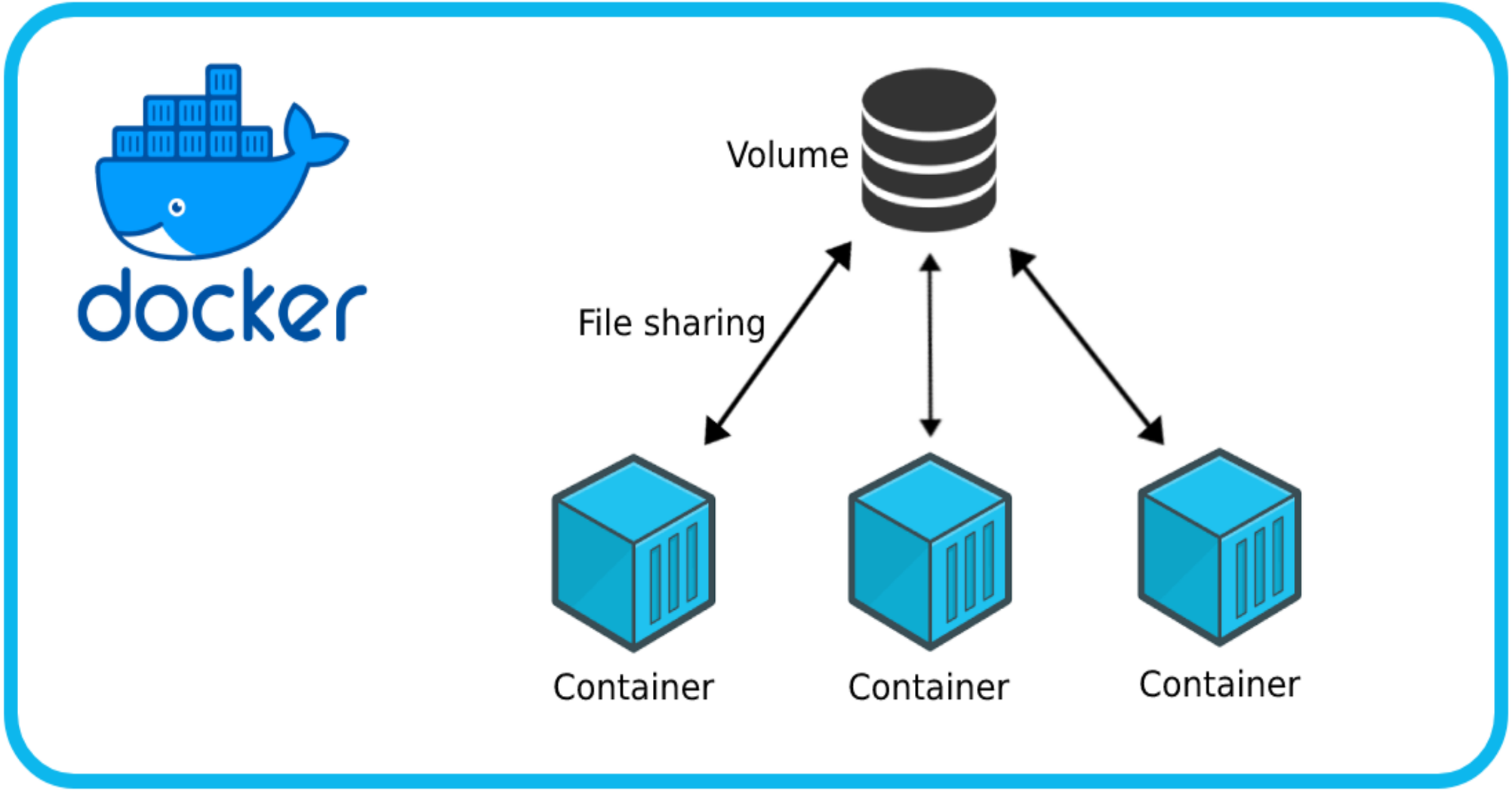
- **Host Network**

Description: Removes network isolation between the container and the Docker host, allowing the container to use the host's network stack directly.

- **Overlay Network**

Description: Enables communication between containers across multiple Docker hosts, typically used with Docker Swarm or Kubernetes for multi-host deployments.

```
docker network create --driver bridge --subnet 172.28.0.0/16 my-network
```


docker
Volume
File sharing
Container
Container
Container

Docker Volumes

Named Volumes:

- **Description**: These are managed by Docker and stored in a designated directory on the host (typically /var/lib/docker/volumes/). They are explicitly named for easy reference.

- **Use Case**: Ideal for persisting data across container restarts or when sharing data between containers in a simple, managed way.

```
docker run -v my-volume:/app/data my-image
```

Anonymous Volumes:

- **Description**: These are temporary volumes created automatically by Docker when a volume is specified without a name. They are tied to the lifecycle of a specific container.

- **Use Case**: Useful for temporary storage needs within a single container run, where persistence beyond the container’s lifecycle isn’t required.

```
docker run -v docker run -v /app/data my-image
```

Docker Volumes

Bind Mounts:

- **Description**: These map a specific directory or file from the host filesystem into a container. Unlike named or anonymous volumes, bind mounts are not managed by Docker.

- **Use Case**: Useful for development environments (e.g., mounting source code) or when you need direct access to host files.

```
docker run -v /host/path:/container/path my-image
```

tmpfs Mounts:

- **Description**: These are in-memory volumes that store data in the host's RAM rather than on disk. They are not persisted to the filesystem.

- **Use Case**: Ideal for temporary, sensitive data (e.g., caches, secrets) that shouldn't persist after the container stops.

```
docker run --tmpfs /tmp:size=100M my-image
```

Build image (01)

```
# Use an official base image
FROM ubuntu:20.04

# Set working directory
WORKDIR /app

# Copy files to container
COPY . /app

# Install dependencies (example)
RUN apt-get update && apt-get install -y \
    python3 \
    && rm -rf /var/lib/apt/lists/*

# Set command to run when container starts
CMD ["python3", "app.py"]
```

Build image

1- Instruction: FROM ubuntu:20.04 :

- What it does: Specifies the starting point for your image. Every Dockerfile begins with a FROM instruction, which pulls an existing image from Docker Hub (or another registry) to use as a foundation.

Details:

- ubuntu is the base image name (a popular Linux distribution).
- 20.04 is the tag, indicating a specific version of Ubuntu.
- You could use other base images like python:3.9, node:16, or alpine depending on your needs.

2- Instruction: WORKDIR /app

- What it does: Defines the default directory inside the container where subsequent commands (like COPY, RUN, or CMD) will execute.

Details:

- /app is a common choice, but you could use any path (e.g., /usr/src/myapp).
- If the directory doesn’t exist, Docker creates it automatically.
- This helps keep your container organized and avoids cluttering the root directory.

Build image (02)

3- Instruction: COPY . /app

- What it does: Copies files from your local directory (where the Dockerfile is) into the container’s filesystem.

Details:

- “ . ” means "everything in the current directory" on your host machine.
- /app is the destination inside the container (matches the WORKDIR).

4- Instruction: RUN “something”

- What it does: Executes commands inside the container during the build process to set up the environment.

Details:

- apt-get update: Updates the package list for Ubuntu’s package manager.
- apt-get install -y python3: Installs Python 3 without prompting for confirmation (-y means "yes").
- rm -rf /var/lib/apt/lists/*: Cleans up cached files to reduce image size.
- The && chains commands to run them in sequence in a single layer (more efficient).

Build image (03)

5- Instruction: CMD ["python3", "app.py"]

- What it does: Defines the default command that runs when a container starts from this image.

Details:

- Uses exec form ["executable", "arg1", "arg2"] (recommended over shell form CMD python3 app.py).
- python3 is the command, and app.py is the argument (runs your script).
- This only executes when you docker run the image, not during the build.

```
docker build -t my-python-app:latest .
```

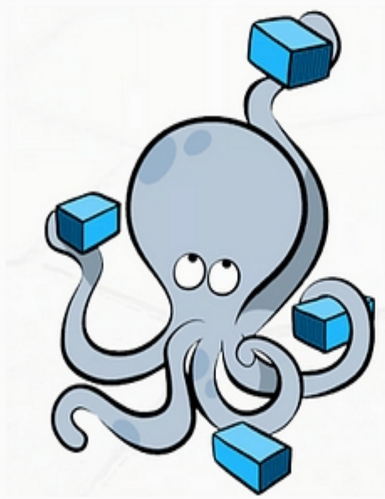
```
docker run my-python-app:latest
```

Docker Compose

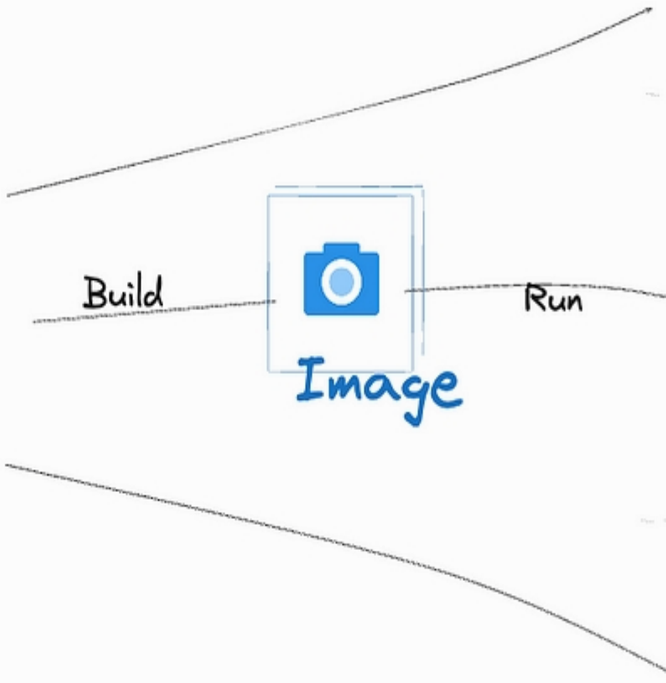
docker-compose.yaml

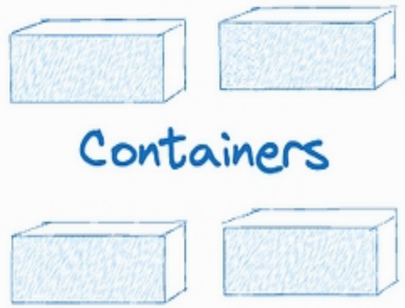
```yaml
services:
  app:
    image: node:18-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 127.0.0.1:3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:8.0
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Config YAML



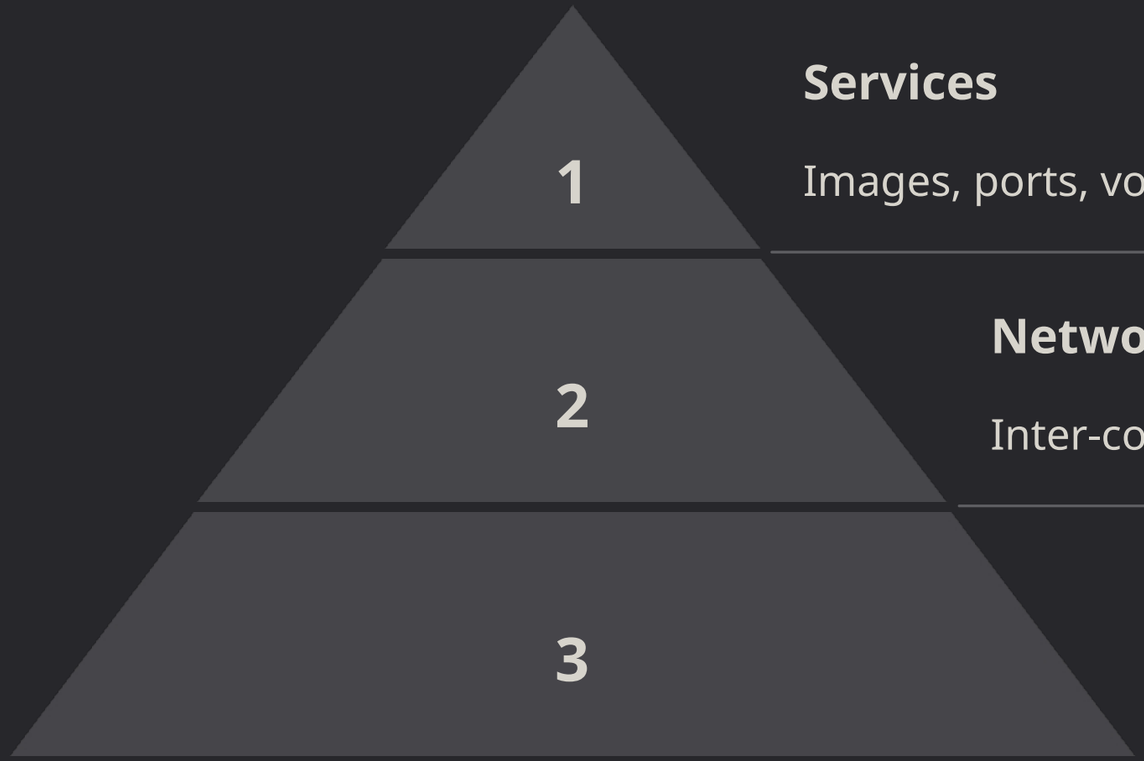
Docker Compose





Docker Compose

Docker_compose_link



Services

Images, ports, volumes.

Networks

Inter-container communication.

Volumes

Data persistence.

Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to configure your application's services, networks, and volumes in a single YAML file, making it easier to manage and deploy complex applications.

Docker Use Cases

Microservices

Deploy and manage services.

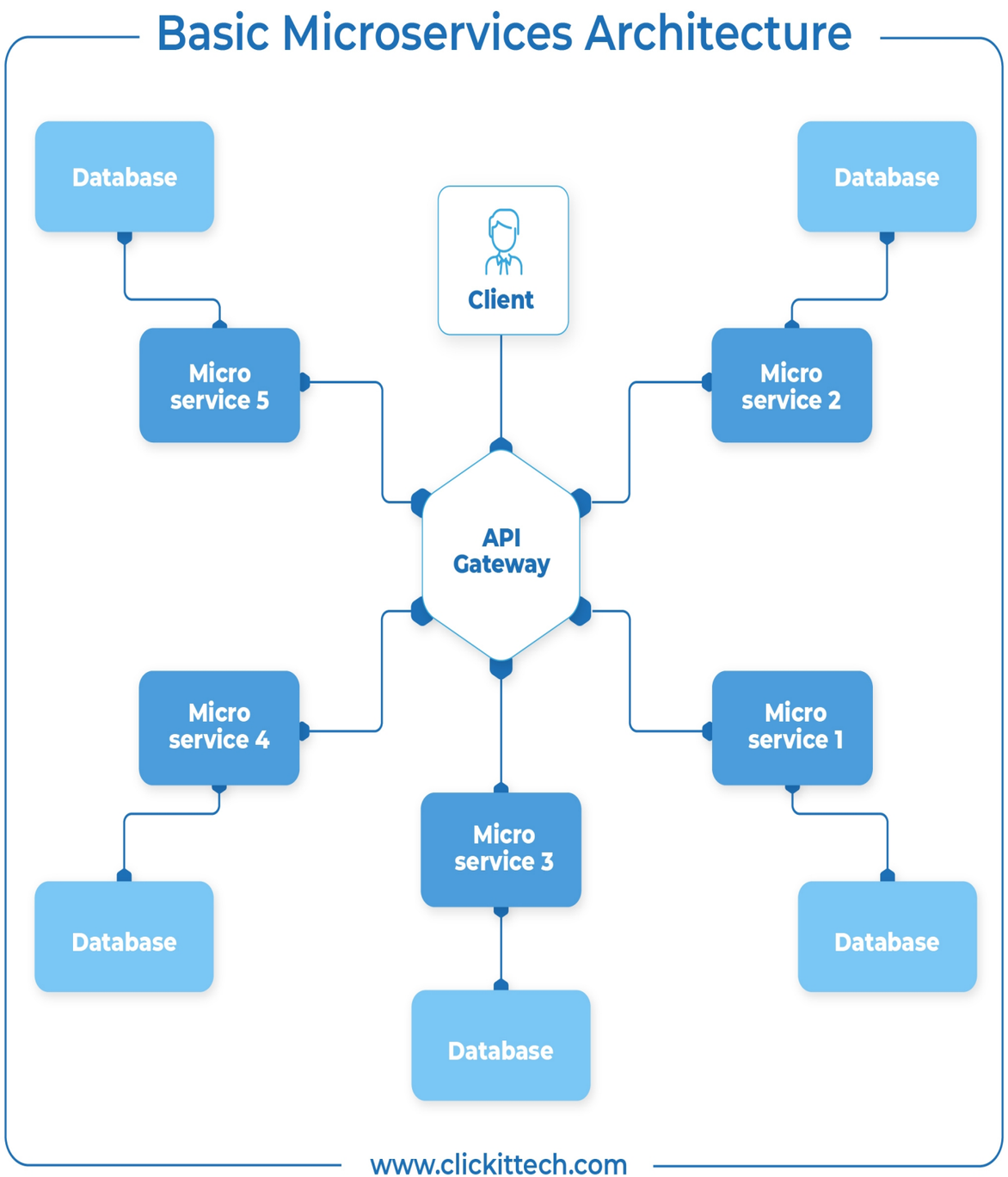
CI/CD

Automate testing and deployment.

Development

Reproducible environments.

Docker is used in microservices, CI/CD, development environments, web applications, and databases. It enables consistent and reproducible environments.



Conclusion

1

Portability

Run anywhere.

2

Scalability

Easily scale services.

3

Efficiency

Optimize resource use.

Docker offers portability, scalability, and efficiency. Explore Docker documentation, tutorials, and online courses for more information.