# **Docker Compose**

## **Multi-container Applications with Docker Compose**

Multi-container applications with Docker Compose are a common use case in modern application development, especially when dealing with microservices architectures. Here's a real-time scenario to illustrate its application:

### Scenario: E-commerce Web Application

#### Components:

1. **Frontend (React.js)**: Serves the user interface.
2. **Backend (Node.js/Express)**: Handles API requests and business logic.
3. **Database (MongoDB)**: Stores user data, product information, orders, etc.
4. **Cache (Redis)**: Caches frequently accessed data to improve performance.
5. **Nginx**: Acts as a reverse proxy to manage traffic between the frontend and backend services.

#### Docker Compose File (docker-compose.yml):

yaml

Copy code

version: '3.8'

services:

frontend:

image: react-app:latest

build: ./frontend

ports:

- "3000:3000"

depends\_on:

- backend

networks:

- app-network

backend:

image: node-app:latest

build: ./backend

ports:

- "5000:5000"

environment:

- MONGO\_URL=mongodb://mongo:27017/ecommerce

- REDIS\_URL=redis://redis:6379

depends\_on:

- mongo

- redis

networks:

- app-network

mongo:

image: mongo:latest

ports:

- "27017:27017"

volumes:

- mongo-data:/data/db

networks:

- app-network

redis:

image: redis:latest

ports:

- "6379:6379"

networks:

- app-network

nginx:

image: nginx:latest

ports:

- "80:80"

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf

depends\_on:

- frontend

- backend

networks:

- app-network

volumes:

mongo-data:

networks:

app-network:

driver: bridge

#### Explanation:

1. **Frontend Service**:
   * Uses a React.js Docker image and is built from the ./frontend directory.
   * Exposes port 3000 for development.
   * Depends on the backend service to be up and running.
2. **Backend Service**:
   * Uses a Node.js Docker image and is built from the ./backend directory.
   * Exposes port 5000.
   * Connects to MongoDB and Redis using environment variables.
   * Depends on mongo and redis services.
3. **MongoDB Service**:
   * Uses the official MongoDB image.
   * Exposes port 27017.
   * Persists data using a Docker volume mongo-data.
4. **Redis Service**:
   * Uses the official Redis image.
   * Exposes port 6379.
5. **Nginx Service**:
   * Uses the official Nginx image.
   * Exposes port 80.
   * Custom configuration is provided through a volume ./nginx.conf.
   * Depends on both frontend and backend services to be up and running.
6. **Volumes and Networks**:
   * mongo-data volume is used to persist MongoDB data.
   * app-network is a custom bridge network to allow communication between all services.

### Benefits of Using Docker Compose:

* **Isolation**: Each service runs in its own container, providing isolation and reducing the risk of conflicts.
* **Reproducibility**: The entire environment can be spun up or down with a single command, ensuring consistency across different environments (development, staging, production).
* **Scalability**: Individual services can be scaled independently by adjusting the docker-compose.yml file.
* **Ease of Management**: Docker Compose simplifies the orchestration of multi-container applications, making it easier to manage complex setups.

### Commands to Manage the Setup:

* **Start Services**: docker-compose up -d
* **Stop Services**: docker-compose down
* **View Logs**: docker-compose logs -f
* **Scale a Service**: docker-compose up -d --scale backend=3

This scenario demonstrates how Docker Compose can be used to manage a multi-container application, providing a scalable and maintainable solution for running complex applications.