Food Ordering System - Docker Deployment Report

Overview

Food Ordering System is fully containerized using Docker and Docker Compose. The system consists of 4 separate microservices, each with its own `docker-compose.yml` file and can run independently or together.

Docker Architecture

```
Food-Ordering-System/
├── backend/
│   ├── user-service/          # Django service
│   │   ├── Dockerfile
│   │   └── docker-compose.yml
│   ├── menu-service/          # Node.js service
│   │   ├── Dockerfile
│   │   └── docker-compose.yml
│   └── order-service/         # Node.js service
│       ├── Dockerfile
│       └── docker-compose.yml
├── frontend/
│   └── restaurant/            # React service
│       ├── dockerfile
│       └── docker-compose.yml
├── run-all-services.bat       # Windows script
└── run-all-services.sh        # Linux/Mac script
```

System Requirements

Required Software

- **Docker** (v20.10+)
- **Docker Compose** (v2.0+)
- **Git**

Docker Installation

Windows

```
# Download Docker Desktop from https://www.docker.com/products/docker-desktop
# Install and start Docker Desktop
```

Linux (Ubuntu/Debian)

```
# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Install Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Start Docker
sudo systemctl start docker
sudo systemctl enable docker
```

Docker Configuration

1. Clone repository

```
git clone <repository-url>
cd Food-Ordering-System
```

2. Check Docker configuration

```
# Check Docker version
docker --version
docker-compose --version

# Check Docker daemon
docker info
```

Individual Service Deployment

1. User Service (Django + PostgreSQL + Redis)

docker-compose.yml configuration

```
version: '3.8'
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: food_ordering_users
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5

  user-service:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DEBUG=True
      - SECRET_KEY=django-insecure-development-key-change-in-production
      - DB_NAME=food_ordering_users
      - DB_USER=postgres
      - DB_PASSWORD=password
      - DB_HOST=db
      - DB_PORT=5432
      - REDIS_URL=redis://redis:6379/0
      - ALLOWED_HOSTS=localhost,127.0.0.1,user-service
      - CORS_ALLOWED_ORIGINS=http://localhost:3000,http://127.0.0.1:3000
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
    volumes:
      - .:/app
    command: >
      sh -c "python manage.py migrate &&
             python manage.py collectstatic --noinput &&
             gunicorn --bind 0.0.0.0:8000 --workers 3 --reload user_service.wsgi:application"

volumes:
  postgres_data:
```

Dockerfile

```
FROM python:3.11-slim
```

```
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        postgresql-client \
        build-essential \
        libpq-dev \
        curl \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt /app/
RUN pip install --no-cache-dir -r requirements.txt

COPY . /app/

RUN mkdir -p /app/staticfiles

RUN adduser --disabled-password --gecos '' appuser
RUN chown -R appuser:appuser /app
USER appuser

EXPOSE 8000

HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/api/health/ || exit 1

CMD ["sh", "-c", "python manage.py makemigrations && python manage.py collectstatic --noinput && gunicorn --bind 0.0.0.0:8000 --workers 3 user_service.wsgi:application"]
```

Deployment

```
cd backend/user-service

# Build and run
docker-compose build --no-cache
docker-compose up -d

# Check logs
docker-compose logs -f

# Check health
curl http://localhost:8000/api/health/
```

2. Menu Service (Node.js + MongoDB)

docker-compose.yml configuration

```
version: '3.8'
services:
  menu-service:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
    environment:
      - NODE_ENV=production
      - PORT=3001
      - MONGODB_URI=mongodb://mongodb:27017/food_ordering_menus
      - USER_SERVICE_URL=http://localhost:8000
      - ORDER_SERVICE_URL=http://localhost:3002
    depends_on:
      - mongodb
    networks:
      - menu-network
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider", "http://localhost:3001/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  mongodb:
    image: mongo:latest
    ports:
      - "27018:27017"
    volumes:
      - mongodb_data:/data/db
    networks:
      - menu-network
    restart: unless-stopped

networks:
  menu-network:
    name: menu-network
    driver: bridge

volumes:
  mongodb_data:
```

Dockerfile

```
FROM node:18-alpine

WORKDIR /app

RUN apk add --no-cache curl

COPY package*.json ./

RUN npm ci

COPY . .

RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

RUN chown -R nodejs:nodejs /app

USER nodejs

EXPOSE 3001

HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:3001/health || exit 1

CMD ["npm", "run", "dev"]
```

Deployment

```
cd backend/menu-service

# Build and run
docker-compose build --no-cache
docker-compose up -d

# Check logs
docker-compose logs -f

# Check health
curl http://localhost:3001/health
```

3. Order Service (Node.js + MongoDB)

docker-compose.yml configuration

```
version: '3.8'
services:
  order-service:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3002:3002"
    environment:
      - NODE_ENV=production
      - PORT=3002
      - MONGODB_URI=mongodb://mongodb:27017/food_ordering_orders
      - USER_SERVICE_URL=http://localhost:8000
      - MENU_SERVICE_URL=http://localhost:3001
    depends_on:
      - mongodb
    networks:
      - order-network
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider", "http://localhost:3002/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
    networks:
      - order-network
    restart: unless-stopped

networks:
  order-network:
    name: order-network
    driver: bridge

volumes:
  mongodb_data:
```

Dockerfile

```
FROM node:20-alpine

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3001

CMD ["npm", "start"]
```

Deployment

```
cd backend/order-service

# Build and run
docker-compose build --no-cache
docker-compose up -d

# Check logs
docker-compose logs -f

# Check health
curl http://localhost:3002/health
```

4. Frontend (React + Vite)

docker-compose.yml configuration

```
version: '3.8'
services:
  frontend:
    build:
      context: .
      dockerfile: Dockerfile
      args:
        - VITE_API_URL=http://localhost:8000/api
        - VITE_MENU_SERVICE_URL=http://localhost:3001
        - VITE_ORDER_SERVICE_URL=http://localhost:3002
    ports:
      - "3000:3000"
    restart: unless-stopped
    networks:
      - frontend-network

networks:
  frontend-network:
    name: frontend-network
    driver: bridge
```

Dockerfile

```
# Build stage
FROM node:20-alpine as build

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

ARG VITE_API_URL
ARG VITE_MENU_SERVICE_URL
ARG VITE_ORDER_SERVICE_URL

RUN echo "VITE_API_URL: ${VITE_API_URL}"
RUN echo "VITE_MENU_SERVICE_URL: ${VITE_MENU_SERVICE_URL}"
RUN echo "VITE_ORDER_SERVICE_URL: ${VITE_ORDER_SERVICE_URL}"

ENV VITE_API_URL=${VITE_API_URL}
ENV VITE_MENU_SERVICE_URL=${VITE_MENU_SERVICE_URL}
ENV VITE_ORDER_SERVICE_URL=${VITE_ORDER_SERVICE_URL}

RUN npm run build

# Production stage
FROM node:20-alpine

WORKDIR /app

RUN npm install -g serve

COPY --from=build /app/dist /app/dist

EXPOSE 3000

CMD ["serve", "-s", "dist", "-l", "3000"]
```

Deployment

```
cd frontend/restaurant

# Build and run
docker-compose build --no-cache
docker-compose up -d

# Check logs
docker-compose logs -f

# Check health
curl http://localhost:3000
```

Automation Scripts

Windows Script (run-all-services.bat)

This script provides commands to manage the entire system:

```
# Syntax
run-all-services.bat [command]

# Available commands
run-all-services.bat start           # Start sequentially (recommended)
run-all-services.bat start-concurrent # Start concurrently
```

```
run-all-services.bat stop             # Stop all services
run-all-services.bat restart          # Restart all services
run-all-services.bat status           # Show status
run-all-services.bat logs             # Show logs
run-all-services.bat health           # Check health
run-all-services.bat clean            # Clean containers, volumes, images
run-all-services.bat help             # Show help
```

Key Features:

- **Prerequisites check**: Docker, Docker Compose
- **File validation**: Ensure all docker-compose.yml files exist
- **Sequential startup**: User Service → Menu Service → Order Service → Frontend
- **Health checks**: Check health of each service
- **Logging**: Display logs from all services
- **Cleanup**: Complete system cleanup

Linux/Mac Script (run-all-services.sh)

Similar script for Linux/Mac with additional features:

```
# Make executable
chmod +x run-all-services.sh

# Syntax
./run-all-services.sh [command]

# Available commands
./run-all-services.sh start           # Start sequentially
./run-all-services.sh start-concurrent # Start concurrently
./run-all-services.sh stop            # Stop all services
./run-all-services.sh restart         # Restart
./run-all-services.sh status          # Status
./run-all-services.sh logs            # Logs
./run-all-services.sh health          # Health check
./run-all-services.sh clean           # Cleanup
./run-all-services.sh help            # Help
```

Additional Features:

- **Colored output**: Color-coded display for better readability
- **Concurrent execution**: Run services concurrently
- **Better error handling**: Improved error handling
- **PID tracking**: Track process IDs

Complete System Deployment

Method 1: Using scripts (Recommended)

Windows

```
# Start entire system
run-all-services.bat start

# Check status
run-all-services.bat status

# View logs
run-all-services.bat logs

# Check health
run-all-services.bat health
```

Linux/Mac

```
# Make executable
chmod +x run-all-services.sh

# Start entire system
./run-all-services.sh start
```

```
# Check status
./run-all-services.sh status

# View logs
./run-all-services.sh logs

# Check health
./run-all-services.sh health
```

Method 2: Manual service deployment

```
# 1. Start User Service
cd backend/user-service
docker-compose up -d

# 2. Start Menu Service
cd backend/menu-service
docker-compose up -d

# 3. Start Order Service
cd backend/order-service
docker-compose up -d

# 4. Start Frontend
cd frontend/restaurant
docker-compose up -d
```

Monitoring and Health Checks

Health Checks

```
# Check health of all services
curl http://localhost:8000/api/health/  # User Service
curl http://localhost:3001/health       # Menu Service
curl http://localhost:3002/health       # Order Service
curl http://localhost:3000              # Frontend
```

Useful Docker Commands

```
# View all containers
docker ps -a

# View container logs
docker logs <container_name>

# View resource usage
docker stats

# View networks
docker network ls

# View volumes
docker volume ls

# View images
docker images
```

Access URLs

- **Frontend**: http://localhost:3000
- **User Service API**: http://localhost:8000/api
- **Menu Service API**: http://localhost:3001/api
- **Order Service API**: http://localhost:3002/api
- **Django Admin**: http://localhost:8000/admin

Troubleshooting

Common Issues

1. Port conflicts

```
# Check which ports are in use
netstat -tulpn | grep :3000
netstat -tulpn | grep :3001
netstat -tulpn | grep :3002
netstat -tulpn | grep :8000

# Stop service using port
docker-compose down
```

2. Build failures

```
# Clean build
docker-compose build --no-cache

# Remove old images
docker system prune -a
```

3. Database connection issues

```
# Check database containers
docker ps | grep -E "(postgres|mongo|redis)"

# Check database logs
docker logs <db_container_name>

# Restart database
docker-compose restart db
```

4. Network issues

```
# Check networks
docker network ls

# Create new network if needed
docker network create food-ordering-network
```

Debugging

View detailed logs

```
# Logs from all services
docker-compose logs -f

# Logs from specific service
docker-compose logs -f user-service

# Logs with timestamps
docker-compose logs -f -t
```

Enter container for debugging

```
# Enter container
docker exec -it <container_name> /bin/bash

# Check processes
docker exec -it <container_name> ps aux

# Check network
docker exec -it <container_name> netstat -tulpn
```

Security

Security best practices

1. **Non-root users**: All containers run with non-root users
2. **Health checks**: Each service has health check
3. **Resource limits**: Can add resource limits
4. **Secrets management**: Use Docker secrets for production
5. **Network isolation**: Each service has separate network

Production considerations

```
# Add to docker-compose.yml
services:
  app:
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 512M
        reservations:
          cpus: '0.25'
          memory: 256M
    security_opt:
      - no-new-privileges:true
    read_only: true
    tmpfs:
      - /tmp
      - /var/cache
```

Backup and Recovery

Backup volumes

```
# Backup PostgreSQL data
docker run --rm -v food-ordering-system_postgres_data:/data -v $(pwd):/backup alpine tar czf /backup/postgres_backup.tar.gz -C /data .

# Backup MongoDB data
docker run --rm -v food-ordering-system_mongodb_data:/data -v $(pwd):/backup alpine tar czf /backup/mongodb_backup.tar.gz -C /data .
```

Restore volumes

```
# Restore PostgreSQL
docker run --rm -v food-ordering-system_postgres_data:/data -v $(pwd):/backup alpine tar xzf /backup/postgres_backup.tar.gz -C /data

# Restore MongoDB
docker run --rm -v food-ordering-system_mongodb_data:/data -v $(pwd):/backup alpine tar xzf /backup/mongodb_backup.tar.gz -C /data
```

Performance and Scaling

Monitoring

```
# View resource usage
docker stats

# View container metrics
docker system df

# Cleanup unused resources
docker system prune
```

Scaling

```
# Scale service
docker-compose up -d --scale user-service=3

# Load balancing
# Use nginx or traefik for load balancing
```

Conclusion

The Food Ordering System has been fully containerized with Docker and Docker Compose. Deployment is very simple, just run the script `run-all-services.bat` (Windows) or `run-all-services.sh` (Linux/Mac).

Advantages of Docker deployment:

1. **Isolation**: Each service runs independently
2. **Consistency**: Same environment on every machine

3. **Scalability**: Easy to scale and deploy
4. **Portability**: Runs on any platform
5. **Versioning**: Easy version management
6. **Automation**: Fully automated scripts

Next steps:

1. Configure CI/CD pipeline
2. Setup monitoring and alerting
3. Implement load balancing
4. Configure SSL/TLS
5. Setup backup automation