DevOps: Docker & Docker Compose - Complete Tutorial

Table of Contents

The Dockerfile

Basic Django Dockerfile

dockerfile

```dockerfile
# Basic Dockerfile for Django Application

# 1. Base Image - The foundation
FROM python:3.11

# 2. Set working directory inside container
WORKDIR /app

# 3. Copy requirements first (for better caching)
COPY requirements.txt .

# 4. Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# 5. Copy application code
COPY . .

# 6. Expose the port Django runs on
EXPOSE 8000

# 7. Command to run when container starts
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Multi-Stage Django Dockerfile

dockerfile

```
# Multi-Stage Dockerfile for Django Application
# Stage 1: Build Stage (includes build tools)
FROM python:3.11 AS builder

# Install system dependencies needed for building Python packages
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Create virtual environment
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Copy and install requirements
COPY requirements.txt .
RUN pip install --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

# Stage 2: Runtime Stage (minimal dependencies)
FROM python:3.11-slim AS runtime

# Install only runtime dependencies
RUN apt-get update && apt-get install -y \
    libpq5 \
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# Copy virtual environment from builder stage
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Create non-root user for security
RUN groupadd -r django && useradd -r -g django django

# Set working directory
WORKDIR /app

# Copy application code
COPY . .

# Change ownership to django user
RUN chown -R django:django /app
USER django
```

```
# Expose port
EXPOSE 8000

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

# Run application
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Size-Optimized Dockerfile

dockerfile

```
# Size-Optimized Dockerfile with Best Practices

# Use slim variant (smaller base image)
FROM python:3.11-slim

# Combine RUN commands to reduce layers
RUN apt-get update && apt-get install -y \
    # Only essential packages
    libpq5 \
    curl \
    # Clean up in same layer to reduce image size
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# Create non-root user early
RUN groupadd -r django && useradd -r -g django django

# Set working directory
WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .

# Install Python packages with optimizations
RUN pip install --upgrade pip && \
    pip install --no-cache-dir \
    --no-compile \
    --no-deps \
    -r requirements.txt && \
    # Remove pip cache and temporary files
    pip cache purge

# Create necessary directories
RUN mkdir -p /app/static /app/media /app/logs && \
    chown -R django:django /app

# Copy application code (do this last to maximize cache usage)
COPY --chown=django:django . .

# Switch to non-root user
USER django

EXPOSE 8000
```

```
# Use exec form for better signal handling
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

.dockerignore

dockerignore

```
.git/
.gitignore
README.md
Dockerfile
.dockerignore
node_modules/
__pycache__/
*.pyc
.pytest_cache/
.coverage
.env
.env.local
venv/
env/
```

The docker-compose.yml

Basic docker-compose.yml

yaml

```yaml
# Basic docker-compose.yml for Django + PostgreSQL + React

version: '3.8'

services:
  # PostgreSQL Database Service
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password123
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  # Django Web Application Service
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DEBUG=1
      - DATABASE_URL=postgresql://postgres:password123@db:5432/myapp

  # React Frontend Service
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - /app/node_modules
    depends_on:
      - web

# Named volumes for data persistence
volumes:
```

```
  postgres_data:
```

Production-Ready docker-compose.yml

yaml

```yaml
# Production-ready docker-compose.yml with best practices

version: '3.8'

services:
  # PostgreSQL Database
  db:
    image: postgres:13
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${DB_NAME:-myapp}
      POSTGRES_USER: ${DB_USER:-postgres}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backup:/backup
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER:-postgres}"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s

  # Redis Cache
  redis:
    image: redis:6-alpine
    restart: unless-stopped
    command: redis-server --appendonly yes --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis_data:/data
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 5

  # Django Web Application
  web:
    build:
      context: .
      target: production
    restart: unless-stopped
    command: >
      sh -c "python manage.py migrate &&
```

```
        python manage.py collectstatic --noinput &&
        gunicorn myapp.wsgi:application --bind 0.0.0.0:8000"
  volumes:
    - static_volume:/app/static
    - media_volume:/app/media
  ports:
    - "8000:8000"
  depends_on:
    db:
      condition: service_healthy
    redis:
      condition: service_healthy
  environment:
    - DEBUG=0
    - DATABASE_URL=postgresql://${DB_USER:-postgres}:${DB_PASSWORD}@db:5432/${DB_NAME:-myapp}
    - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379/0
  env_file:
    - .env
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/health/"]
    interval: 30s
    timeout: 10s
    retries: 3

# React Frontend
frontend:
  build:
    context: ./frontend
    dockerfile: Dockerfile.prod
  restart: unless-stopped
  ports:
    - "3000:80"
  depends_on:
    - web
  environment:
    - REACT_APP_API_URL=http://localhost:8000/api

# Nginx Reverse Proxy
nginx:
  image: nginx:alpine
  restart: unless-stopped
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
    - static_volume:/static
```

```
      - media_volume:/media
      - ssl_certs:/etc/nginx/ssl
    depends_on:
      - web
      - frontend

  # Celery Worker (for background tasks)
  worker:
    build:
      context: .
      target: production
    restart: unless-stopped
    command: celery -A myapp worker --loglevel=info
    volumes:
      - media_volume:/app/media
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
    environment:
      - DATABASE_URL=postgresql://${DB_USER:-postgres}:${DB_PASSWORD}@db:5432/${DB_NAME:-myapp}
      - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379/0
    env_file:
      - .env

# Named volumes for data persistence
volumes:
  postgres_data:
  redis_data:
  static_volume:
  media_volume:
  ssl_certs:

# Custom network (optional)
networks:
  default:
    driver: bridge
```

docker-compose.override.yml (Development)

yaml

```yaml
# docker-compose.override.yml - Development overrides (auto-loaded)
version: '3.8'

services:
  web:
    volumes:
      - .:/app  # Mount source code for live reloading
    ports:
      - "8000:8000"
    environment:
      - DEBUG=1
      - RELOAD=1

  db:
    ports:
      - "5432:5432"  # Expose database port for development tools
```

docker-compose.prod.yml (Production)

yaml

```yaml
# docker-compose.prod.yml - Production configuration
version: '3.8'

services:
  web:
    restart: unless-stopped
    environment:
      - DEBUG=0
      - SECURE_SSL_REDIRECT=1
    # No volume mounts in production

  db:
    restart: unless-stopped
    # No exposed ports for security

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.prod.conf:/etc/nginx/nginx.conf
```

Service Linking

Django Settings for Service Linking

python

```python
# Django settings.py - Using service names for connections

import os

# Database Configuration
# Instead of localhost, use the service name 'db'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'myapp'),
        'USER': os.environ.get('DB_USER', 'postgres'),
        'PASSWORD': os.environ.get('DB_PASSWORD', 'password123'),
        'HOST': 'db',  # Service name, not localhost!
        'PORT': '5432',
    }
}

# Redis Configuration (for caching/sessions)
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://redis:6379/1',  # Service name: redis
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}

# Celery Configuration (if using background tasks)
CELERY_BROKER_URL = 'redis://redis:6379/0'  # Service name: redis
CELERY_RESULT_BACKEND = 'redis://redis:6379/0'

# Email Backend (if using MailHog for testing)
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'mailhog'  # Service name
EMAIL_PORT = 1025

# Common Mistake - DON'T DO THIS:
# 'HOST': 'localhost',     # Won't work in containers!
# 'HOST': '127.0.0.1',     # Won't work in containers!
# 'HOST': '172.18.0.2',    # Don't hardcode IPs!
```

Frontend API Configuration

javascript

```javascript
// React Frontend - API calls to Django backend

// Method 1: Using environment variable
const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000';

// Method 2: Different URLs for different environments
const getApiUrl = () => {
  if (process.env.NODE_ENV === 'production') {
    return 'https://api.myapp.com';  // Production API
  } else if (process.env.NODE_ENV === 'development') {
    // In development with Docker Compose
    return 'http://web:8000';  // Service name
  } else {
    return 'http://localhost:8000';  // Local development
  }
};

// API service
class ApiService {
  constructor() {
    this.baseURL = getApiUrl();
  }

  async fetchUsers() {
    try {
      // This will call http://web:8000/api/users/
      const response = await fetch(`${this.baseURL}/api/users/`);
      return await response.json();
    } catch (error) {
      console.error('API call failed:', error);
    }
  }

  async createUser(userData) {
    const response = await fetch(`${this.baseURL}/api/users/`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(userData),
    });
    return await response.json();
  }
}
```

Network Isolation Example

yaml

```yaml
# Network Isolation in Docker Compose

version: '3.8'

services:
  # Frontend services (public-facing)
  web:
    build: .
    ports:
      - "8000:8000"
    networks:
      - frontend_network
      - backend_network
    depends_on:
      - db

  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    networks:
      - frontend_network

  # Backend services (internal only)
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: myapp
    networks:
      - backend_network
    # No ports exposed - only internal access

  redis:
    image: redis:6
    networks:
      - backend_network
    # No ports exposed - only internal access

  # Worker processes
  worker:
    build: .
    command: celery -A myapp worker
    networks:
      - backend_network
```

```
    depends_on:
      - db
      - redis

# Custom networks
networks:
  frontend_network:
    driver: bridge
  backend_network:
    driver: bridge
```

Environment Variables

.env File Template

bash

```
# .env file - Main environment variables (DO NOT COMMIT TO GIT)

# Database Configuration
DB_NAME=myapp
DB_USER=postgres
DB_PASSWORD=super_secure_password_123
DB_HOST=db
DB_PORT=5432

# Django Settings
SECRET_KEY=your-very-secret-django-key-here
DEBUG=1
ALLOWED_HOSTS=localhost,127.0.0.1,web

# Redis Configuration
REDIS_PASSWORD=redis_password_456
REDIS_URL=redis://:redis_password_456@redis:6379/0

# Email Configuration
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_HOST_USER=your-email@gmail.com
EMAIL_HOST_PASSWORD=your-app-password

# API Keys (External Services)
AWS_ACCESS_KEY_ID=your_aws_access_key
AWS_SECRET_ACCESS_KEY=your_aws_secret_key
STRIPE_SECRET_KEY=sk_test_your_stripe_key

# Environment Type
ENVIRONMENT=development
```

.env.example File

bash

```
# .env.example (COMMIT THIS TO GIT)
# This shows required variables without exposing secrets

# Database Configuration
DB_NAME=myapp
DB_USER=postgres
DB_PASSWORD=your_db_password_here
DB_HOST=db
DB_PORT=5432

# Django Settings
SECRET_KEY=your_django_secret_key_here
DEBUG=1
ALLOWED_HOSTS=localhost,127.0.0.1

# Redis Configuration
REDIS_PASSWORD=your_redis_password_here

# Instructions for new developers:
# 1. Copy this file to .env
# 2. Fill in the actual values
# 3. Never commit .env to git
```

Django Settings with Environment Variables

python

```python
# Django settings.py - Using environment variables properly

import os
from pathlib import Path

# Build paths inside the project
BASE_DIR = Path(__file__).resolve().parent.parent

# Security Settings
SECRET_KEY = os.environ.get('SECRET_KEY', 'fallback-key-for-development')

# NEVER use fallback for production!
if os.environ.get('ENVIRONMENT') == 'production' and SECRET_KEY == 'fallback-key-for-development':
    raise ValueError("SECRET_KEY must be set in production!")

DEBUG = os.environ.get('DEBUG', '0').lower() in ['true', '1', 'yes', 'on']

ALLOWED_HOSTS = os.environ.get('ALLOWED_HOSTS', 'localhost').split(',')

# Database Configuration
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'myapp'),
        'USER': os.environ.get('DB_USER', 'postgres'),
        'PASSWORD': os.environ.get('DB_PASSWORD', ''),
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}

# Alternative: Using DATABASE_URL (more common)
import dj_database_url
DATABASES = {
    'default': dj_database_url.parse(
        os.environ.get('DATABASE_URL', 'postgresql://postgres:@localhost:5432/myapp')
    )
}

# Redis Configuration
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': os.environ.get('REDIS_URL', 'redis://localhost:6379/1'),
```

```
    }
}

# Email Configuration
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = os.environ.get('EMAIL_HOST', 'localhost')
EMAIL_PORT = int(os.environ.get('EMAIL_PORT', '587'))
EMAIL_USE_TLS = os.environ.get('EMAIL_USE_TLS', '1').lower() in ['true', '1']
EMAIL_HOST_USER = os.environ.get('EMAIL_HOST_USER', '')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_HOST_PASSWORD', '')

# AWS Configuration (for file storage)
AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')
AWS_STORAGE_BUCKET_NAME = os.environ.get('AWS_STORAGE_BUCKET_NAME')

# Only use S3 if credentials are provided
if AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY:
    DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
    AWS_S3_REGION_NAME = os.environ.get('AWS_S3_REGION_NAME', 'us-east-1')

# Environment-specific settings
ENVIRONMENT = os.environ.get('ENVIRONMENT', 'development')

if ENVIRONMENT == 'production':
    # Production-specific settings
    SECURE_SSL_REDIRECT = True
    SECURE_HSTS_SECONDS = 31536000
    SECURE_HSTS_INCLUDE_SUBDOMAINS = True
    SECURE_HSTS_PRELOAD = True

elif ENVIRONMENT == 'development':
    # Development-specific settings
    CORS_ALLOW_ALL_ORIGINS = True

# Helper functions for environment variables
def get_bool_env(var_name, default=False):
    """Convert environment variable to boolean."""
    value = os.environ.get(var_name, '').lower()
    if value in ['true', '1', 'yes', 'on']:
        return True
    elif value in ['false', '0', 'no', 'off']:
        return False
    return default

def get_int_env(var_name, default=None):
    """Convert environment variable to integer with validation."""
```

```python
    value = os.environ.get(var_name)
    if value is None:
        return default
    try:
        return int(value)
    except ValueError:
        raise ValueError(f"Invalid integer value for {var_name}: {value}")

# Usage:
DEBUG = get_bool_env('DEBUG', False)
MAX_CONNECTIONS = get_int_env('MAX_CONNECTIONS', 10)
```

Docker Compose with Environment Variables

yaml

```yaml
# docker-compose.yml with environment variables

version: '3.8'

services:
  # Method 1: Direct environment variables
  web1:
    build: .
    environment:
      - DEBUG=1
      - DATABASE_URL=postgresql://user:pass@db:5432/myapp

  # Method 2: Using .env file
  web2:
    build: .
    env_file:
      - .env          # Load variables from .env file
      - .env.local    # Override with local settings

  # Method 3: Mixed approach (recommended)
  web3:
    build: .
    environment:
      # Non-sensitive variables directly
      - DEBUG=1
      - ALLOWED_HOSTS=localhost,127.0.0.1
      # Sensitive variables from .env
      - DATABASE_URL=${DATABASE_URL}
      - SECRET_KEY=${SECRET_KEY}
      - REDIS_PASSWORD=${REDIS_PASSWORD}
    env_file:
      - .env

  database:
    image: postgres:13
    environment:
      # Using variable substitution
      POSTGRES_DB: ${DB_NAME:-myapp}
      POSTGRES_USER: ${DB_USER:-postgres}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
```

Complete Setup Files

.gitignore

gitignore

```
# Environment variables
.env
.env.local
.env.production
.env.staging

# Docker
.dockerignore

# Python
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
env/
venv/
*.egg-info/
.pytest_cache/
.coverage

# Django
db.sqlite3
media/
staticfiles/

# Node.js
node_modules/
npm-debug.log*

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db
```

Environment Validation Script

python

```python
#!/usr/bin/env python3
# validate_env.py

import os
import sys

# Required environment variables
REQUIRED_VARS = [
    'SECRET_KEY',
    'DB_PASSWORD',
    'DB_NAME',
    'DB_USER',
]

# Optional but recommended
RECOMMENDED_VARS = [
    'REDIS_PASSWORD',
    'EMAIL_HOST_PASSWORD',
]

def validate_environment():
    missing = []
    warnings = []

    # Check required variables
    for var in REQUIRED_VARS:
        if not os.environ.get(var):
            missing.append(var)

    # Check recommended variables
    for var in RECOMMENDED_VARS:
        if not os.environ.get(var):
            warnings.append(var)

    # Validate SECRET_KEY length
    secret_key = os.environ.get('SECRET_KEY', '')
    if len(secret_key) < 32:
        warnings.append('SECRET_KEY should be at least 32 characters')

    # Check DEBUG setting in production
    debug = os.environ.get('DEBUG', '0').lower()
    environment = os.environ.get('ENVIRONMENT', 'development')
    if environment == 'production' and debug in ['true', '1', 'yes']:
        warnings.append('DEBUG should be False in production')
```

```
    # Report results
    if missing:
        print("❌ Missing required environment variables:")
        for var in missing:
            print(f"  - {var}")
        return False

    if warnings:
        print("⚠️  Warnings:")
        for warning in warnings:
            print(f"  - {warning}")

    print("✅ Environment validation passed!")
    return True

if __name__ == '__main__':
    # Load .env file if it exists
    try:
        from dotenv import load_dotenv
        load_dotenv()
    except ImportError:
        pass

    if not validate_environment():
        sys.exit(1)
```

Health Check Script

python

```python
#!/usr/bin/env python3
# healthcheck.py

import os
import sys
import psycopg2
import redis

def check_database():
    try:
        conn = psycopg2.connect(
            host=os.environ.get('DB_HOST', 'db'),
            port=os.environ.get('DB_PORT', '5432'),
            user=os.environ.get('DB_USER', 'postgres'),
            password=os.environ.get('DB_PASSWORD'),
            database=os.environ.get('DB_NAME', 'myapp'),
            connect_timeout=5
        )
        conn.close()
        return True
    except Exception as e:
        print(f"Database check failed: {e}")
        return False

def check_redis():
    try:
        r = redis.Redis(
            host=os.environ.get('REDIS_HOST', 'redis'),
            port=int(os.environ.get('REDIS_PORT', '6379')),
            password=os.environ.get('REDIS_PASSWORD'),
            socket_connect_timeout=5
        )
        r.ping()
        return True
    except Exception as e:
        print(f"Redis check failed: {e}")
        return False

if __name__ == '__main__':
    checks = [
        ('Database', check_database),
        ('Redis', check_redis),
    ]
```

```
failed = []
for name, check_func in checks:
    if not check_func():
        failed.append(name)
    else:
        print(f"✅ {name} connection OK")

if failed:
    print(f"❌ Failed checks: {', '.join(failed)}")
    sys.exit(1)

print("🎉 All health checks passed!")
```

Development Setup Script

bash

```bash
#!/bin/bash
# setup_dev.sh

set -e

echo "🚀 Setting up development environment..."

# Check if .env exists
if [ ! -f .env ]; then
    echo "📝 Creating .env from template..."
    cp .env.example .env
    echo "⚠️  Please edit .env with your actual values"
fi

# Generate secure passwords if needed
if ! grep -q "super_secure_password" .env; then
    echo "🔐 Generating secure passwords..."
    DB_PASSWORD=$(openssl rand -base64 32)
    SECRET_KEY=$(python3 -c 'import secrets; print(secrets.token_urlsafe(50))')
    REDIS_PASSWORD=$(openssl rand -base64 24)

    echo "Generated passwords (update your .env file):"
    echo "DB_PASSWORD=$DB_PASSWORD"
    echo "SECRET_KEY=$SECRET_KEY"
    echo "REDIS_PASSWORD=$REDIS_PASSWORD"
fi

# Validate environment
echo "🔍 Validating environment..."
python3 validate_env.py

# Build and start services
echo "🏗️  Building Docker containers..."
docker-compose build

echo "🚀 Starting services..."
docker-compose up -d

# Wait for services to be ready
echo "⏳ Waiting for services to be ready..."
sleep 10

# Run health checks
echo "🏥 Running health checks..."
```

```
docker-compose exec web python healthcheck.py

echo "✅ Development environment is ready!"
echo "🌐 Access your application at:"
echo "   - Web: http://localhost:8000"
echo "   - Frontend: http://localhost:3000"
echo "   - Database: localhost:5432"
```

Common Docker Commands

bash

```bash
# Common Docker Compose Commands

# Start services
docker-compose up
docker-compose up -d                # Detached mode
docker-compose up --build           # Rebuild images
docker-compose up --scale web=3     # Scale specific service

# Stop services
docker-compose down                 # Stop and remove containers
docker-compose down -v              # Also remove volumes
docker-compose stop                 # Stop without removing

# View logs
docker-compose logs                 # All services
docker-compose logs web             # Specific service
docker-compose logs -f web          # Follow logs

# Execute commands
docker-compose exec web bash        # Interactive shell
docker-compose exec web python manage.py migrate
docker-compose run --rm web python manage.py createsuperuser

# Build and push
docker-compose build                # Build all services
docker-compose build web            # Build specific service
docker-compose push                 # Push to registry

# Environment-specific commands
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d
docker-compose --profile production up -d

# Debugging
docker-compose config               # View resolved configuration
docker-compose ps                   # List running containers
docker-compose top                  # Show running processes
```

Security Checklist

markdown

```
# Docker Security Checklist

## Container Security
- ✅ Use non-root user in containers
- ✅ Use multi-stage builds to reduce image size
- ✅ Scan images for vulnerabilities
- ✅ Use specific image tags, not 'latest'
- ✅ Remove unnecessary packages and files

## Environment Variables
- ✅ Never commit .env files to version control
- ✅ Use strong, randomly generated passwords
- ✅ Validate required environment variables at startup
- ✅ Use Docker secrets for production
- ✅ Rotate secrets regularly

## Network Security
- ✅ Don't expose database ports in production
- ✅ Use custom networks for service isolation
- ✅ Implement proper firewall rules
- ✅ Use HTTPS in production
- ✅ Validate SSL certificates

## Access Control
- ✅ Limit container capabilities
- ✅ Use read-only filesystems where possible
- ✅ Implement proper authentication
- ✅ Use least privilege principle
- ✅ Regular security updates

## Monitoring & Logging
- ✅ Implement health checks
- ✅ Monitor container resources
- ✅ Centralized logging
- ✅ Security event monitoring
- ✅ Regular backup procedures
```

Quick Start Commands

```bash
# 1. Clone or create your project
mkdir myproject && cd myproject

# 2. Create necessary files
touch Dockerfile docker-compose.yml .env.example .env .gitignore

# 3. Copy the configurations from above into respective files

# 4. Generate secure environment variables
python3 -c "
import secrets
print(f'SECRET_KEY={secrets.token_urlsafe(50)}')
print(f'DB_PASSWORD={secrets.token_urlsafe(32)}')
print(f'REDIS_PASSWORD={secrets.token_urlsafe(24)}')
"

# 5. Build and start your application
docker-compose up --build

# 6. Run Django migrations (in another terminal)
docker-compose exec web python manage.py migrate
docker-compose exec web python manage.py createsuperuser

# 7. Access your application
# Web: http://localhost:8000
# Frontend: http://localhost:3000
```

Troubleshooting Guide

Common Issues and Solutions

bash

```bash
# Issue 1: Port already in use
# Error: bind: address already in use
# Solution: Change ports or stop conflicting services
docker-compose down
sudo lsof -i :8000  # Find what's using the port
kill -9 <PID>       # Kill the process

# Issue 2: Permission denied
# Error: Permission denied when accessing files
# Solution: Fix file permissions
sudo chown -R $USER:$USER .
chmod -R 755 .

# Issue 3: Database connection refused
# Error: could not connect to server
# Solution: Wait for database to be ready, use health checks
docker-compose exec web python -c "
import time
import psycopg2
import os

for i in range(30):
    try:
        conn = psycopg2.connect(
            host='db',
            user=os.environ['DB_USER'],
            password=os.environ['DB_PASSWORD'],
            database=os.environ['DB_NAME']
        )
        print('Database connected!')
        break
    except:
        print(f'Attempt {i+1}: Database not ready, waiting...')
        time.sleep(2)
"

# Issue 4: Environment variables not loading
# Solution: Check .env file and docker-compose configuration
docker-compose config  # View resolved configuration
docker-compose exec web env | grep DB_  # Check loaded variables

# Issue 5: Image build failures
# Solution: Clear Docker cache and rebuild
docker system prune -a  # Remove all unused images
```

```
docker-compose build --no-cache
docker-compose up --force-recreate

# Issue 6: Volume permission issues
# Solution: Use proper ownership in Dockerfile
# Add to Dockerfile:
# RUN chown -R django:django /app
# USER django

# Issue 7: Service linking not working
# Solution: Use service names, not localhost
# Wrong: DATABASE_HOST=localhost
# Right: DATABASE_HOST=db

# Issue 8: Container keeps restarting
# Solution: Check logs and fix application errors
docker-compose logs web  # Check application logs
docker-compose exec web ps aux  # Check running processes
```

Debugging Commands

bash

```bash
# Debug Docker Compose
docker-compose config                # View final configuration
docker-compose ps                    # List containers status
docker-compose logs -f web           # Follow logs for specific service
docker-compose exec web bash         # Interactive shell in container
docker-compose top                   # Show running processes

# Debug Networks
docker network ls                    # List networks
docker network inspect myproject_default # Inspect network details
docker-compose exec web nslookup db  # Test DNS resolution
docker-compose exec web ping db      # Test connectivity

# Debug Volumes
docker volume ls                     # List volumes
docker volume inspect myproject_postgres_data  # Inspect volume
docker-compose exec web ls -la /app  # Check mounted files

# Debug Environment Variables
docker-compose exec web env          # List all environment variables
docker-compose exec web printenv | grep DB  # Filter specific variables

# Debug Application
docker-compose exec web python manage.py shell  # Django shell
docker-compose exec web python manage.py check  # Django system check
docker-compose exec web python -c "import django; print(django.VERSION)"

# Performance Debugging
docker stats                         # Show container resource usage
docker-compose exec web ps aux       # Show processes inside container
docker system df                     # Show Docker disk usage
```

Performance Optimization

yaml

```yaml
# Performance optimizations in docker-compose.yml

version: '3.8'

services:
  web:
    build: .
    # Resource limits
    deploy:
      resources:
        limits:
          memory: 512M
          cpus: '0.5'
        reservations:
          memory: 256M
          cpus: '0.25'

    # Restart policy
    restart: unless-stopped

    # Health check
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health/"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 60s

    # Environment optimizations
    environment:
      - PYTHONUNBUFFERED=1      # Don't buffer Python output
      - PYTHONDONTWRITEBYTECODE=1  # Don't create .pyc files
      - DJANGO_SETTINGS_MODULE=myapp.settings.production

    # Volume optimizations
    volumes:
      # Use cached or delegated consistency for better performance on macOS
      - .:/app:cached
      - /app/node_modules       # Prevent overwriting node_modules

    # Logging configuration
    logging:
      driver: "json-file"
      options:
```

```
      max-size: "10m"
      max-file: "3"

  db:
    image: postgres:13
    # PostgreSQL performance tuning
    command: >
      postgres
      -c shared_buffers=256MB
      -c effective_cache_size=1GB
      -c maintenance_work_mem=64MB
      -c checkpoint_completion_target=0.9
      -c wal_buffers=16MB
      -c default_statistics_target=100
      -c random_page_cost=1.1
      -c effective_io_concurrency=200

    # Resource limits for database
    deploy:
      resources:
        limits:
          memory: 1G
          cpus: '1.0'
        reservations:
          memory: 512M
          cpus: '0.5'
```

Production Deployment Checklist

markdown

```
# Production Deployment Checklist

## Before Deployment
- ✅ All environment variables set and validated
- ✅ Database migrations tested
- ✅ Static files configuration verified
- ✅ SSL certificates configured
- ✅ Backup strategy implemented
- ✅ Monitoring and logging set up
- ✅ Health checks implemented
- ✅ Resource limits configured
- ✅ Security scan completed
- ✅ Performance testing done

## Deployment Process
- ✅ Use production docker-compose file
- ✅ Pull latest images
- ✅ Run database migrations
- ✅ Collect static files
- ✅ Verify all services are healthy
- ✅ Test critical functionality
- ✅ Monitor logs for errors
- ✅ Set up automated backups
- ✅ Configure log rotation
- ✅ Set up alerts and monitoring

## Post-Deployment
- ✅ Verify all services are running
- ✅ Check application functionality
- ✅ Monitor performance metrics
- ✅ Test backup and restore procedures
- ✅ Verify SSL certificate validity
- ✅ Check security configurations
- ✅ Document deployment process
- ✅ Plan rollback procedure
```

Advanced Docker Compose Features

yaml

```yaml
# Advanced docker-compose.yml features

version: '3.8'

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
      args:
        - BUILD_ENV=production
        - USER_ID=1001
      target: production  # Multi-stage build target
      cache_from:
        - myapp:latest

    # Multiple environment files (loaded in order)
    env_file:
      - .env.common
      - .env.production
      - .env.local

    # Complex environment variable setup
    environment:
      - DEBUG=${DEBUG:-0}
      - WORKERS=${WORKERS:-4}
      - TIMEOUT=${TIMEOUT:-30}

    # Health check with custom script
    healthcheck:
      test: ["CMD-SHELL", "/app/healthcheck.sh"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 60s

    # Resource constraints
    deploy:
      mode: replicated
      replicas: 3
      resources:
        limits:
          memory: 1G
          cpus: '1.0'
```

```yaml
    reservations:
      memory: 512M
      cpus: '0.5'
  restart_policy:
    condition: on-failure
    delay: 5s
    max_attempts: 3
    window: 120s

# Complex volume configuration
volumes:
  - type: bind
    source: ./app
    target: /app
    consistency: cached
  - type: volume
    source: static_data
    target: /app/static
    read_only: false
  - type: tmpfs
    target: /tmp
    tmpfs:
      size: 100M

# Network configuration
networks:
  frontend:
    aliases:
      - web-app
  backend:
    ipv4_address: 172.20.0.10

# Dependencies with conditions
depends_on:
  db:
    condition: service_healthy
  redis:
    condition: service_started

# External links (deprecated, use networks instead)
# external_links:
#   - redis_1
#   - project_db_1:mysql

# Init process (helps with signal handling)
init: true
```

```yaml
    # Override container command
    command: >
      sh -c "python manage.py migrate &&
             python manage.py collectstatic --noinput &&
             gunicorn myapp.wsgi:application
             --bind 0.0.0.0:8000
             --workers ${WORKERS:-4}
             --timeout ${TIMEOUT:-30}"

    # Container labels (for organization)
    labels:
      - "com.myapp.description=Main web application"
      - "com.myapp.department=engineering"
      - "com.myapp.environment=production"

# External networks (created outside compose)
networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.20.0.0/16
  external_network:
    external: true
    name: shared_network

# External volumes (created outside compose)
volumes:
  static_data:
    driver: local
    driver_opts:
      type: nfs
      o: addr=10.40.0.199,nolock,soft,rw
      device: ":/docker/example"
  postgres_data:
    external: true
    name: production_postgres_data

# Configuration for external secrets
secrets:
  db_password:
    external: true
  ssl_certificate:
```

```
    file: ./ssl/cert.pem
  ssl_private_key:
    file: ./ssl/private.key

# Configuration templates
configs:
  nginx_config:
    file: ./nginx.conf
  app_config:
    external: true
    name: production_app_config
```

Summary

This complete tutorial covers everything you need to know about Docker and Docker Compose for DevOps. You now have:

1. **Dockerfile expertise** - From basic to multi-stage builds with optimization
2. **Docker Compose mastery** - Multi-service orchestration with best practices
3. **Service linking knowledge** - How containers communicate seamlessly
4. **Environment management** - Secure, flexible configuration across environments
5. **Production-ready templates** - Copy-paste configurations for real projects
6. **Debugging skills** - Troubleshooting common issues
7. **Security practices** - Keeping your applications safe
8. **Performance optimization** - Making your containers efficient

Key Takeaways:

- Always use service names (not localhost) for inter-container communication
- Keep secrets in `.env` files, never commit them to git
- Use multi-stage builds for smaller, more secure images
- Implement health checks for reliable deployments
- Use environment-specific compose files for different stages
- Follow the principle of least privilege for security
- Monitor and log everything in production

You're now ready to containerize and deploy production applications with confidence! 🚀