Docker Compose Cheat Sheet

Docker Compose Cheat Sheet

A practical quick-reference guide for Docker Compose multi-container applications. Commands are organized by workflow with realistic examples.

Getting Started

Initialize a Project

```
# Generate a basic docker-compose.yml from an existing container
docker compose convert > docker-compose.yml

# Generate from a Dockerfile
docker compose build

# Verify the configuration is valid
docker compose config
```

Start Services

```
# Start all services in the background
docker compose up -d

# Start a specific service
docker compose up -d web

# Start with a custom compose file
docker compose -f docker-compose.yml -f docker-compose.override.yml up -d

# Start and attach to logs
docker compose up

# Start, attach to logs, and detach with Ctrl+C
docker compose up --detach-attach
```

Service Management

Control Services

```
# Stop all services (containers remain)
docker compose stop

# Stop and remove containers
docker compose down

# Stop, remove containers, and networks
docker compose down --remove-orphans

# Stop, remove everything including volumes
docker compose down -v

# Restart a specific service
docker compose restart web

# Pause/resume services
docker compose pause
docker compose unpause

# Start/stop a single service
docker compose start web
docker compose stop web
```

View Logs

```
# View all service logs
docker compose logs

# View logs for a specific service
docker compose logs web

# Follow logs in real-time
docker compose logs -f web

# Show last 50 lines
docker compose logs --tail=50 web

# Show logs from the last hour
docker compose logs --since=1h web
```

Execute Commands Inside Containers

```
# Run a command in a running service
docker compose exec web ls -la

# Open an interactive shell in a service
docker compose exec web bash

# Run as a different user
docker compose exec -u www-data web whoami

# Execute in a different service
docker compose exec db psql -U postgres
```

```
# Run in the background, capture output
docker compose exec web cat /etc/os-release
```

Images and Builds

Build Services

```
# Build all services
docker compose build

# Build a specific service
docker compose build web

# Force rebuild without cache
docker compose build --no-cache web

# Build with build arguments
docker compose build --build-arg NODE_ENV=production web

# Build and start simultaneously
docker compose up --build

# Pull newer base images
docker compose pull

# Pull images and build services
docker compose up --build --pull always
```

Image Management

```
# List images created by compose
docker compose ls

# View image details for a service
docker compose images web

# Remove unused images
docker compose up --remove-orphans --prune

# Tag images for registry
docker compose run --rm web bash -c "echo $IMAGE"
```

Networking

Network Commands

```
# List networks created by compose
docker compose networks

# Inspect network configuration
docker compose run --rm web ip addr show
```

```
# Ping another service from within a container
docker compose exec web ping db

# Connect a running container to a compose network
docker network connect myapp_default container_id

# Test connectivity between services
docker compose run --rm web curl http://db:5432
```

Service Discovery

Docker Compose provides automatic DNS-based service discovery. Services can reference each other by their service name:

```
# Inside your application, connect to the database using the service name:
DATABASE_URL=postgresql://postgres:password@db:5432/myapp

# Similarly for Redis:
REDIS_URL=redis://redis:6379
```

Volumes

Volume Commands

```
# List all volumes
docker compose volumes

# Create a volume
docker compose volume create mydata

# Remove unused volumes
docker compose down -v

# Remove specific volume
docker compose volume rm myapp_mydata

# Inspect a volume
docker volume inspect myapp_mydata
```

Backup and Restore

```
# Backup a volume
docker compose run --rm postgres pg_dump -U postgres myapp > backup.sql

# Restore from backup
cat backup.sql | docker compose exec -T postgres psql -U postgres myapp

# Backup a volume to a tar file
docker run --rm \
  -v myapp_mydata:/data:ro \
  -v $(pwd):/backup \
  alpine tar czf /backup/backup.tar.gz -C /data .
```

```
# Restore from tar file
docker run --rm \
  -v myapp_mydata:/data \
  -v $(pwd):/backup \
  alpine tar xzf /backup/backup.tar.gz -C /data
```

Environment Variables

Loading Variables

```
# Use .env file (automatically loaded)
# .env:
#   DATABASE_URL=postgresql://...
#   API_KEY=secret123

# Override specific variables
docker compose up -d -e DATABASE_URL=postgresql://other:5432/db

# Load from a specific file
docker compose --env-file .env.prod up -d

# View resolved environment variables
docker compose run --rm web env

# Set variables inline
docker compose run -e FOO=bar web echo $FOO
```

Variable Interpolation

```
# docker-compose.yml
services:
  web:
    image: myapp
    environment:
      DB_HOST: ${DB_HOST:-db}
      DB_PORT: ${DB_PORT:-5432}
      NODE_ENV: ${NODE_ENV:-development}
```

Docker Compose File Reference

Common Top-Level Keys

```
version: "3.9"

services:
  web:
    build: .
    image: myapp:latest
    ports:
      - "3000:3000"
      - "9229:9229"
```

```
    environment:
      - NODE_ENV=production
      - DATABASE_URL=${DATABASE_URL}
    volumes:
      - ./src:/app/src
      - /app/node_modules
    depends_on:
      - db
      - redis
    networks:
      - frontend
      - backend
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  db:
    image: postgres:15-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    networks:
      - backend
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data
    networks:
      - backend
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:

networks:
  frontend:
  backend:
    internal: true
```

Key Directives

Directive	Purpose
`build`	Build image from Dockerfile
`image`	Pull specific image
`ports`	Map host port to container port

Directive	Purpose
`expose`	Document container port (internal only)
`environment`	Set environment variables
`env_file`	Load variables from file
`volumes`	Mount host paths or named volumes
`depends_on`	Define service startup order
`networks`	Assign to custom networks
`restart`	Restart policy (no, always, unless-stopped, on-failure)
`healthcheck`	Define container health check
`command`	Override default command
`entrypoint`	Override default entrypoint
`deploy`	Production deployment settings
`profiles`	Conditional service activation

Profiles

```
# List available profiles
docker compose config --format json | jq '.profile'

# Start with specific profile
docker compose --profile worker up -d

# Start with multiple profiles
docker compose --profile worker --profile scheduler up -d

# Start only services without profiles
docker compose --profile development up -d
```

```
# docker-compose.yml
services:
  web:
    # Always starts
    image: myapp:web

  worker:
    # Only starts with --profile worker
    image: myapp:worker
    profiles: ["worker"]

  scheduler:
    # Only starts with --profile scheduler
    image: myapp:scheduler
    profiles: ["scheduler"]
```

Troubleshooting

Debugging Services

```
# View service configuration
docker compose config

# View running containers with ports
docker compose ps

# Check health status
docker compose ps --filter "health=healthy"

# View resource usage
docker compose top

# View cgroup limits
docker compose run --rm web cat /sys/fs/cgroup/memory/memory.limit_in_bytes
```

Common Issues

```
# Port already in use
# 1. Find what's using the port
# lsof -i :3000        (macOS/Linux)
# netstat -ano | findstr :3000  (Windows)

# 2. Kill the process or change the port mapping
# docker compose up -d  # after fixing

# Container exits immediately
docker compose logs web
docker compose run --rm web bash  # interact with it

# DNS resolution fails inside compose
# Services can't find each other by name
docker compose exec web nslookup db
# Fix: ensure services are on the same network

# Volume permission issues
docker compose exec web chown -R www-data:www-data /app/data

# Build fails with cache error
docker compose build --no-cache
docker system prune -a  # clear all unused data
```

Health Check Patterns

```
# HTTP health check
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
  interval: 30s
  timeout: 10s
  retries: 3
  start_period: 40s

# Command-based health check
healthcheck:
```

```
  test: ["CMD", "pg_isready", "-U", "postgres"]
  interval: 10s
  timeout: 5s
  retries: 5

# Shell-based health check
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} || exit 1"]
  interval: 10s
  timeout: 5s
  retries: 5
```

Production Deployment

Docker Compose vs Docker Swarm

```
# Deploy with docker swarm (legacy swarm mode)
docker stack deploy -c docker-compose.yml myapp

# Docker Compose v3 supports swarm deploy syntax
# docker-compose.yaml can be used with both docker compose and swarm

# For Kubernetes deployment, use Kompose
kompose convert -f docker-compose.yml -o k8s/
kubectl apply -f k8s/

# For systemd-based deployment
# Install compose as a systemd service
# https://github.com/manaime/compose2systemd
```

Security Best Practices

```
# Never do this in production:
services:
  web:
    privileged: true  # DANGEROUS
    network_mode: "host"  # DANGEROUS

# Do this instead:
services:
  web:
    read_only: true
    tmpfs:
      - /tmp
    security_opt:
      - no-new-privileges:true
    user: "1000:1000"
    networks:
      - backend
```