Docker CLI vs Dockerfile vs Docker Compose: Basic to Advanced Cheat Sheet

🚀 Level 1: BASIC FUNDAMENTALS

What Are They?

Tool	Purpose	Format	Scope
Docker CLI	Interactive container management	Command line	Single container operations
Dockerfile	Build custom images	Text file	Image creation blueprint
Docker Compose	Multi-container orchestration	YAML file	Service stack management

Basic Container Operations

Operation	Docker CLI	Dockerfile	Docker Compose
Run Container	`docker run nginx`	N/A	`docker-compose up`
Stop Container	`docker stop myapp`	N/A	`docker-compose stop`
Remove Container	`docker rm myapp`	N/A	`docker-compose down`

Basic Image Operations

Operation	Docker CLI	Dockerfile	Docker Compose
Use Image	`docker run ubuntu`	`FROM ubuntu`	`image: ubuntu`
Build Image	`docker build -t myapp .`	*Instructions in file*	`build: .`
List Images	`docker images`	N/A	`docker-compose images`

🎯 Level 2: ESSENTIAL CONFIGURATION

Container Naming & Basic Settings

Feature	Docker CLI	Dockerfile	Docker Compose
Container Name	`--name webapp`	N/A	`container_name: webapp`
Port Mapping	`-p 8080:80`	`EXPOSE 80`	`ports: - "8080:80"`
Environment Variables	`-e NODE_ENV=prod`	`ENV NODE_ENV=prod`	`environment: NODE_ENV=prod`
Working Directory	`--workdir /app`	`WORKDIR /app`	`working_dir: /app`

Basic Volume Management

Type	Docker CLI	Dockerfile	Docker Compose
Bind Mount	`-v /host/path:/container/path`	N/A	`- /host/path:/container/path`
Named Volume	`-v myvolume:/data`	`VOLUME /data`	`- myvolume:/data`

Basic Examples

Simple Web Server

Docker CLI:

```bash
docker run -d --name nginx-web -p 8080:80 nginx
```

Dockerfile:

```dockerfile
FROM nginx
EXPOSE 80
```

Docker Compose:

```yaml
version: '3.8'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
```

⚙️ Level 3: INTERMEDIATE FEATURES

Advanced Container Configuration

Feature	Docker CLI	Dockerfile	Docker Compose
Memory Limit	`--memory 512m`	N/A	`mem_limit: 512m`
CPU Limit	`--cpus 1.5`	N/A	`cpus: 1.5`
Restart Policy	`--restart unless-stopped`	N/A	`restart: unless-stopped`
Hostname	`--hostname myhost`	N/A	`hostname: myhost`
User	`--user 1000:1000`	`USER 1000:1000`	`user: "1000:1000"`

Networking

Feature	Docker CLI	Dockerfile	Docker Compose
Custom Network	`--network mynet`	N/A	`networks: - mynet`
Network Alias	`--network-alias web`	N/A	`aliases: - web`
DNS	`--dns 8.8.8.8`	N/A	`dns: - 8.8.8.8`
Static IP	`--ip 172.18.0.100`	N/A	`ipv4_address: 172.18.0.100`

Multi-Stage Builds (Dockerfile)

dockerfile

```dockerfile
# Build stage
FROM node:16 AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 80
```

Environment Management

Method	Docker CLI	Dockerfile	Docker Compose
Single Variable	`-e KEY=value`	`ENV KEY=value`	`environment: KEY=value`
Environment File	`--env-file .env`	N/A	`env_file: .env`
Multiple Variables	`-e KEY1=val1 -e KEY2=val2`	`ENV KEY1=val1 KEY2=val2`	`environment:` list

Intermediate Examples

Node.js Application with Database

Docker CLI:

```bash
# Database
docker run -d --name postgres-db \
  -e POSTGRES_PASSWORD=secret \
  -v db_data:/var/lib/postgresql/data \
  postgres:13

# Application
docker run -d --name node-app \
  --link postgres-db:db \
  -p 3000:3000 \
  -e DATABASE_URL=postgres://postgres:secret@db:5432/myapp \
  node:16
```

Docker Compose:

yaml

```yaml
version: '3.8'
services:
  app:
    image: node:16
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgres://postgres:secret@db:5432/myapp
    depends_on:
      - db

  db:
    image: postgres:13
    environment:
      POSTGRES_PASSWORD: secret
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

🔧 Level 4: ADVANCED CONFIGURATION

Security & Privileges

Feature	Docker CLI	Dockerfile	Docker Compose
Read-only Filesystem	`--read-only`	N/A	`read_only: true`
Drop Capabilities	`--cap-drop ALL`	N/A	`cap_drop: - ALL`
Add Capabilities	`--cap-add NET_ADMIN`	N/A	`cap_add: - NET_ADMIN`
Security Options	`--security-opt no-new-privileges`	N/A	`security_opt: - no-new-privileges`
Privileged Mode	`--privileged`	N/A	`privileged: true`
PID Limit	`--pids-limit 100`	N/A	`pids_limit: 100`

Advanced Storage

Feature	Docker CLI	Dockerfile	Docker Compose
Tmpfs Mount	`--tmpfs /tmp:size=100m,mode=1777`	N/A	`tmpfs: - /tmp:size=100m,mode=1777`
Volume Driver	`--volume-driver local`	N/A	`driver: local`
Volume Options	`--mount type=volume,src=vol,dst=/data,volume-driver=local`	N/A	Complex volume configuration

Health Checks

Aspect	Docker CLI	Dockerfile	Docker Compose
Command	`--health-cmd "curl -f http://localhost"`	`HEALTHCHECK CMD curl -f http://localhost`	`test: ["CMD", "curl", "-f", "http://localhost"]`
Interval	`--health-interval 30s`	`HEALTHCHECK --interval=30s`	`interval: 30s`
Timeout	`--health-timeout 10s`	`HEALTHCHECK --timeout=10s`	`timeout: 10s`
Retries	`--health-retries 3`	`HEALTHCHECK --retries=3`	`retries: 3`
Start Period	`--health-start-period 60s`	`HEALTHCHECK --start-period=60s`	`start_period: 60s`

Logging Configuration

Feature	Docker CLI	Dockerfile	Docker Compose
Log Driver	`--log-driver json-file`	N/A	`driver: json-file`
Log Options	`--log-opt max-size=10m`	N/A	`options: max-size: "10m"`
Log Rotation	`--log-opt max-file=3`	N/A	`max-file: "3"`
Syslog	`--log-driver syslog`	N/A	`driver: syslog`

Advanced Examples

Production-Ready Nginx with Security

Docker CLI:

bash

```bash
docker run -d \
  --name secure-nginx \
  --hostname nginx-prod \
  --memory 256m \
  --cpus 0.5 \
  --pids-limit 100 \
  --read-only \
  --cap-drop ALL \
  --cap-add NET_BIND_SERVICE \
  --security-opt no-new-privileges:true \
  --tmpfs /var/cache/nginx:size=50m \
  --tmpfs /var/log/nginx:size=10m \
  --tmpfs /tmp:size=10m \
  --health-cmd "wget -q --spider http://localhost || exit 1" \
  --health-interval 30s \
  --health-timeout 10s \
  --health-retries 3 \
  -p 443:443 \
  -v /etc/ssl:/etc/ssl:ro \
  nginx:alpine
```

Docker Compose:

yaml

```yaml
version: '3.8'
services:
  nginx:
    image: nginx:alpine
    container_name: secure-nginx
    hostname: nginx-prod
    mem_limit: 256m
    cpus: 0.5
    pids_limit: 100
    read_only: true
    cap_drop:
      - ALL
    cap_add:
      - NET_BIND_SERVICE
    security_opt:
      - no-new-privileges:true
    tmpfs:
      - /var/cache/nginx:size=50m
      - /var/log/nginx:size=10m
      - /tmp:size=10m
    healthcheck:
      test: ["CMD-SHELL", "wget -q --spider http://localhost || exit 1"]
      interval: 30s
      timeout: 10s
      retries: 3
    ports:
      - "443:443"
    volumes:
      - /etc/ssl:/etc/ssl:ro
```

💎 Level 5: EXPERT LEVEL

Advanced Dockerfile Techniques

Multi-Architecture Builds

dockerfile

```dockerfile
# syntax=docker/dockerfile:1
FROM --platform=$BUILDPLATFORM golang:1.19 AS build
ARG TARGETPLATFORM
ARG BUILDPLATFORM
WORKDIR /src
COPY . .
RUN CGO_ENABLED=0 GOOS=${TARGETOS} GOARCH=${TARGETARCH} go build -o app

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=build /src/app /app
ENTRYPOINT ["/app"]
```

Advanced Build Arguments

dockerfile

```dockerfile
ARG BUILD_DATE
ARG VCS_REF
ARG VERSION
LABEL org.label-schema.build-date=$BUILD_DATE \
      org.label-schema.vcs-ref=$VCS_REF \
      org.label-schema.version=$VERSION
```

Docker Compose Advanced Features

Override Files & Profiles

yaml

```yaml
# docker-compose.yml
version: '3.8'
services:
  app:
    image: myapp
    profiles: ["dev", "prod"]

  debug:
    image: myapp:debug
    profiles: ["dev"]

# docker-compose.override.yml
version: '3.8'
services:
  app:
    volumes:
      - .:/app
    environment:
      DEBUG: "true"
```

Advanced Networking

```yaml
version: '3.8'
networks:
  frontend:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.0.0/24
  backend:
    driver: overlay
    attachable: true

services:
  web:
    networks:
      frontend:
        ipv4_address: 172.16.0.100
      backend:
        aliases:
          - web-service
```

Secrets Management

```yaml
version: '3.8'
services:
  app:
    image: myapp
    secrets:
      - db_password
      - api_key

secrets:
  db_password:
    file: ./secrets/db_password.txt
  api_key:
    external: true
```

Advanced CLI Operations

Container Inspection & Debugging

bash

```bash
# Detailed inspection
docker inspect --format='{{.NetworkSettings.IPAddress}}' container

# Resource monitoring
docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"

# Process monitoring
docker top container_name aux

# File system changes
docker diff container_name

# Export/Import
docker export container_name | gzip > backup.tar.gz
gunzip -c backup.tar.gz | docker import - restored_image
```

Advanced Volume Operations

bash

```bash
# Volume with specific driver and options
docker volume create --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.1.100,rw \
  --opt device=:/path/to/share \
  nfs_volume

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

Expert Examples

Microservices Stack with Service Discovery

yaml

```yaml
version: '3.8'

x-common-variables: &common-variables
  CONSUL_HOST: consul
  LOG_LEVEL: info

services:
  consul:
    image: consul:latest
    command: consul agent -server -ui -node=server-1 -bootstrap-expect=1 -client=0.0.0.0
    ports:
      - "8500:8500"
    networks:
      - service-mesh

  api-gateway:
    image: nginx:alpine
    depends_on:
      - consul
      - user-service
      - order-service
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    networks:
      - service-mesh
      - frontend

  user-service:
    build:
      context: ./user-service
      args:
        BUILD_DATE: ${BUILD_DATE}
        VCS_REF: ${VCS_REF}
    environment:
      <<: *common-variables
      SERVICE_NAME: user-service
    deploy:
      replicas: 3
      resources:
        limits:
          memory: 256M
```

```
        cpus: '0.5'
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 40s
  networks:
    - service-mesh
    - backend

order-service:
  build:
    context: ./order-service
    target: production
  environment:
    <<: *common-variables
    SERVICE_NAME: order-service
    DATABASE_URL: postgres://postgres:${POSTGRES_PASSWORD}@postgres:5432/orders
  depends_on:
    postgres:
      condition: service_healthy
  networks:
    - service-mesh
    - backend

postgres:
  image: postgres:13
  environment:
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    POSTGRES_DB: orders
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres"]
    interval: 10s
    timeout: 5s
    retries: 5
  networks:
```

```
      - backend

  redis:
    image: redis:alpine
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    networks:
      - backend

networks:
  frontend:
    driver: bridge
  service-mesh:
    driver: overlay
    attachable: true
  backend:
    driver: bridge
    internal: true

volumes:
  postgres_data:
    driver: local
  redis_data:
    driver: local

secrets:
  postgres_password:
    file: ./secrets/postgres_password.txt
```

🛠️ TROUBLESHOOTING & DEBUGGING

Debug Commands by Level

Basic Debugging

```bash
# View logs
docker logs container_name
docker-compose logs service_name

# Check container status
docker ps -a
docker-compose ps

# Inspect configuration
docker inspect container_name
docker-compose config
```

Intermediate Debugging

```bash
# Resource usage
docker stats
docker system df

# Network inspection
docker network ls
docker network inspect network_name

# Volume inspection
docker volume ls
docker volume inspect volume_name
```

Advanced Debugging

```bash
# System events
docker events --filter container=myapp
docker-compose events

# Process inspection
docker top container_name
docker exec container_name ps aux

# File system analysis
docker diff container_name
docker exec container_name find / -name "*.log" -mtime -1
```

📚 BEST PRACTICES BY LEVEL

Basic Best Practices

- Always tag your images
- Use specific versions, not `latest`
- Clean up unused containers and images
- Use meaningful names for containers

Intermediate Best Practices

- Use multi-stage builds to reduce image size
- Implement health checks
- Use `.dockerignore` files
- Set resource limits

Advanced Best Practices

- Use secrets management for sensitive data
- Implement proper logging strategies
- Use read-only filesystems when possible
- Apply security hardening

Expert Best Practices

- Implement container scanning in CI/CD

- Use distroless or minimal base images
- Implement proper monitoring and observability
- Use orchestration platforms for production

🚀 QUICK REFERENCE COMMANDS

Docker CLI Essentials

bash

```bash
# Container lifecycle
docker run -d --name app nginx          # Create and start
docker start/stop/restart app           # Control container
docker rm app                           # Remove container
docker exec -it app bash                # Access container

# Images
docker build -t myapp .                 # Build image
docker images                           # List images
docker rmi myapp                        # Remove image
docker pull nginx:alpine                # Pull image

# System maintenance
docker system prune -a                  # Clean everything
docker container prune                  # Remove stopped containers
docker image prune                      # Remove unused images
docker volume prune                     # Remove unused volumes
```

Docker Compose Essentials

bash

```bash
# Service management
docker-compose up -d                    # Start services
docker-compose down                     # Stop and remove
docker-compose restart                  # Restart services
docker-compose logs -f                  # Follow logs

# Scaling and building
docker-compose scale web=3              # Scale service
docker-compose build                    # Build services
docker-compose pull                     # Pull images
```

Dockerfile Essentials

```dockerfile
# Base and setup
FROM node:16-alpine                    # Base image
WORKDIR /app                           # Working directory
COPY package*.json ./                  # Copy dependency files
RUN npm ci --only=production           # Install dependencies

# Application setup
COPY . .                               # Copy source code
EXPOSE 3000                            # Expose port
USER node                              # Run as non-root
CMD ["npm", "start"]                   # Default command
```

This comprehensive cheat sheet progresses from basic concepts to expert-level configurations, providing practical examples and real-world scenarios at each level.