LabEx

labex.io

Docker Cheatsheet

Essential commands for container management and deployment

This cheatsheet provides a quick reference to fundamental Docker operations, commands, and best practices, ideal for both beginners and experienced developers for efficient container management and deployment.

Container Management
Create, run, and control containers

Image Operations
Build, manage, and distribute images

System Monitoring
Inspect and troubleshoot containers

Docker Compose
Multi-container application management

Registry Operations
Push, pull, and share images

Installation & Setup

Linux Installation

Install Docker on Ubuntu/Debian systems.

```
# Update package manager
sudo apt update
# Install prerequisites
sudo apt install apt-transport-https ca-certificates curl software-properties-common
# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# Add Docker repository
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
# Install Docker
sudo apt update && sudo apt install docker-ce
# Start Docker service
sudo systemctl start docker
sudo systemctl enable docker
```

Windows & macOS

Install Docker Desktop for GUI-based management.

```
# Windows: Download Docker Desktop from docker.com
# macOS: Use Homebrew or download from docker.com
brew install --cask docker
# Or download directly from:
# https://www.docker.com/products/docker-desktop
```

Post-Installation Setup

Configure Docker for non-root usage and verify installation.

```
# Add user to docker group (Linux)
sudo usermod -aG docker $USER
# Log out and back in for group changes
# Verify Docker installation
docker --version
docker run hello-world
```

Docker Compose Installation

Install Docker Compose for multi-container applications.

```
# Linux: Install via curl
sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
# Verify installation
docker-compose --version
# Note: Docker Desktop includes Compose
```

Basic Docker Commands

Essential commands to get started with Docker containers and images.

01

System Information: `docker version` / `docker system info`

Check Docker installation and environment details.

```
# Display Docker version information
docker version
# Show system-wide Docker information
docker system info
# Display help for Docker commands
docker help
docker <command> --help
```

02

Running Containers: `docker run`

Create and start a container from an image.

```
# Run a container interactively
docker run -it ubuntu:latest bash
# Run container in background (detached)
docker run -d --name my-container nginx
# Run with port mapping
docker run -p 8080:80 nginx
# Run with auto-removal after exit
docker run --rm hello-world
```

03

List Containers: `docker ps`

View running and stopped containers.

```
# List running containers
docker ps
# List all containers (including stopped)
docker ps -a
# List container IDs only
docker ps -q
# Show latest created container
docker ps -l
```

Container Management

Container Lifecycle: `start` / `stop` / `restart`

Control container execution state.

```
# Stop a running container
docker stop container_name
# Start a stopped container
docker start container_name
# Restart a container
docker restart container_name
# Pause/unpause container processes
docker pause container_name
docker unpause container_name
```

Execute Commands: `docker exec`

Run commands inside running containers.

```
# Execute interactive bash shell
docker exec -it container_name bash
# Execute a single command
docker exec container_name ls -la
# Execute as different user
docker exec -u root container_name whoami
# Execute in specific directory
docker exec -w /app container_name pwd
```

Container Removal: `docker rm`

Remove containers from the system.

```
# Remove a stopped container
docker rm container_name
# Force remove a running container
docker rm -f container_name
# Remove multiple containers
docker rm container1 container2
# Remove all stopped containers
docker container prune
```

Container Logs: `docker logs`

View container output and debug issues.

```
# View container logs
docker logs container_name
# Follow logs in real-time
docker logs -f container_name
# Show only recent logs
docker logs --tail 50 container_name
# Show logs with timestamps
docker logs -t container_name
```

Image Management

Building Images: `docker build`

Create Docker images from Dockerfiles.

```
# Build image from current directory
docker build .
# Build and tag an image
docker build -t myapp:latest .
# Build with build arguments
docker build --build-arg VERSION=1.0 -t myapp .
# Build without using cache
docker build --no-cache -t myapp .
```

Image Inspection: `docker images` / `docker inspect`

List and examine Docker images.

```
# List all local images
docker images
# List images with specific filters
docker images nginx
# Show image details
docker inspect image_name
# View image build history
docker history image_name
```

Registry Operations: `docker pull` / `docker push`

Download and upload images to registries.

```
# Pull image from Docker Hub
docker pull nginx:latest
# Pull specific version
docker pull ubuntu:20.04
# Push image to registry
docker push myusername/myapp:latest
# Tag image before pushing
docker tag myapp:latest myusername/myapp:v1.0
```

Image Cleanup: `docker rmi` / `docker image prune`

Remove unused images to free disk space.

```
# Remove a specific image
docker rmi image_name
# Remove unused images
docker image prune
# Remove all unused images (not just dangling)
docker image prune -a
# Force remove image
docker rmi -f image_name
```

Dockerfile Basics

Create reproducible container images using Dockerfiles.

Essential Instructions

Core Dockerfile commands for building images.

```
# Base image
FROM ubuntu:20.04
# Set maintainer information
LABEL maintainer="user@example.com"
# Install packages
RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip \
    && rm -rf /var/lib/apt/lists/*
# Copy files from host to container
COPY app.py /app/
# Set working directory
WORKDIR /app
# Expose port
EXPOSE 8000
```

Runtime Configuration

Configure how the container runs.

```
# Set environment variables
ENV PYTHON_ENV=production
ENV PORT=8000
# Create user for security
RUN useradd -m appuser
USER appuser
# Define startup command
CMD ["python3", "app.py"]
# Or use ENTRYPOINT for fixed commands
ENTRYPOINT ["python3"]
CMD ["app.py"]
# Set health check
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost:8000/ || exit 1
```

Docker Compose

Define and manage multi-container applications with docker-compose.yml.

Basic Compose Commands: `docker-compose up` / `docker-compose down`

Start and stop multi-container applications.

```
# Start services in foreground
docker-compose up
# Start services in background
docker-compose up -d
# Build and start services
docker-compose up --build
# Stop and remove services
docker-compose down
# Stop and remove with volumes
docker-compose down -v
```

Service Management

Control individual services within Compose applications.

```
# List running services
docker-compose ps
# View service logs
docker-compose logs service_name
# Follow logs for all services
docker-compose logs -f
# Restart a specific service
docker-compose restart service_name
```

Sample docker-compose.yml

Example multi-service application configuration.

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "8000:8000"
    environment:
      -
DATABASE_URL=postgresql://user:pass@db:5432/myapp
    depends_on:
      - db
    volumes:
      - .:/app

  db:
    image: postgres:13
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

Networking & Volumes

Container Networking

Connect containers and expose services.

```
# List networks
docker network ls
# Create a custom network
docker network create mynetwork
# Run container on specific network
docker run --network mynetwork nginx
# Connect running container to network
docker network connect mynetwork container_name
# Inspect network details
docker network inspect mynetwork
```

Port Mapping

Expose container ports to the host system.

```
# Map single port
docker run -p 8080:80 nginx
# Map multiple ports
docker run -p 8080:80 -p 8443:443 nginx
# Map to specific host interface
docker run -p 127.0.0.1:8080:80 nginx
# Expose all ports defined in image
docker run -P nginx
```

Data Volumes: `docker volume`

Persist and share data between containers.

```
# Create a named volume
docker volume create myvolume
# List all volumes
docker volume ls
# Inspect volume details
docker volume inspect myvolume
# Remove volume
docker volume rm myvolume
# Remove unused volumes
docker volume prune
```

Volume Mounting

Mount volumes and host directories in containers.

```
# Mount named volume
docker run -v myvolume:/data nginx
# Mount host directory (bind mount)
docker run -v /host/path:/container/path nginx
# Mount current directory
docker run -v $(pwd):/app nginx
# Read-only mount
docker run -v /host/path:/container/path:ro nginx
```

Container Inspection & Debugging

Container Details: `docker inspect`

Get detailed information about containers and images.

```
# Inspect container configuration
docker inspect container_name
# Get specific information using format
docker inspect --format='{{.State.Status}}'
container_name
# Get IP address
docker inspect --format='{{.NetworkSettings.IPAddress}}'
container_name
# Get mounted volumes
docker inspect --format='{{.Mounts}}' container_name
```

Resource Monitoring

Monitor container resource usage and performance.

```
# Show running processes in container
docker top container_name
# Display live resource usage statistics
docker stats
# Show stats for specific container
docker stats container_name
# Monitor events in real-time
docker events
```

File Operations: `docker cp`

Copy files between containers and host system.

```
# Copy file from container to host
docker cp container_name:/path/to/file ./
# Copy file from host to container
docker cp ./file container_name:/path/to/destination
# Copy directory
docker cp ./directory
container_name:/path/to/destination/
# Copy with archive mode to preserve permissions
docker cp -a ./directory container_name:/path/
```

Troubleshooting

Debug container issues and connectivity problems.

```
# Check container exit code
docker inspect --format='{{.State.ExitCode}}'
container_name
# View container processes
docker exec container_name ps aux
# Test network connectivity
docker exec container_name ping google.com
# Check disk usage
docker exec container_name df -h
```

Registry & Authentication

Docker Hub Operations: `docker login` / `docker search`

Authenticate and interact with Docker Hub.

```
# Login to Docker Hub
docker login
# Login to specific registry
docker login registry.example.com
# Search for images on Docker Hub
docker search nginx
# Search with filter
docker search --filter stars=100 nginx
```

Image Tagging & Publishing

Prepare and publish images to registries.

```
# Tag image for registry
docker tag myapp:latest username/myapp:v1.0
docker tag myapp:latest
registry.example.com/myapp:latest
# Push to Docker Hub
docker push username/myapp:v1.0
# Push to private registry
docker push registry.example.com/myapp:latest
```

Private Registry

Work with private Docker registries.

```
# Pull from private registry
docker pull registry.company.com/myapp:latest
# Run private registry locally
docker run -d -p 5000:5000 --name registry registry:2
# Push to local registry
docker tag myapp localhost:5000/myapp
docker push localhost:5000/myapp
```

Image Security

Verify image integrity and security.

```
# Enable Docker Content Trust
export DOCKER_CONTENT_TRUST=1
# Sign and push image
docker push username/myapp:signed
# Verify image signatures
docker trust inspect username/myapp:signed
```

System Cleanup & Maintenance

Keep your Docker environment clean and optimized.

System Cleanup: `docker system prune`

Remove unused Docker resources to free disk space.

```
# Remove unused containers, networks, images
docker system prune
# Include unused volumes in cleanup
docker system prune -a --volumes
# Remove everything (use with caution)
docker system prune -a -f
# Show space usage
docker system df
```

Targeted Cleanup

Remove specific types of unused resources.

```
# Remove stopped containers
docker container prune
# Remove unused images
docker image prune -a
# Remove unused volumes
docker volume prune
# Remove unused networks
docker network prune
```

Bulk Operations

Perform operations on multiple containers/images.

```
# Stop all running containers
docker stop $(docker ps -q)
# Remove all containers
docker rm $(docker ps -aq)
# Remove all images
docker rmi $(docker images -q)
# Remove dangling images only
docker rmi $(docker images -f "dangling=true" -q)
```

Resource Limits

Control container resource consumption.

```
# Limit memory usage
docker run --memory=512m nginx
# Limit CPU usage
docker run --cpus="1.5" nginx
# Limit both CPU and memory
docker run --memory=1g --cpus="2.0" nginx
# Set restart policy
docker run --restart=always nginx
```

Docker Configuration & Settings

Configure Docker daemon and client settings for optimal performance.

Daemon Configuration

Configure the Docker daemon for production use.

```
# Edit daemon configuration
sudo nano /etc/docker/daemon.json
# Example configuration:
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2"
}
# Restart Docker service
sudo systemctl restart docker
```

Environment Variables

Configure Docker client behavior with environment variables.

```
# Set Docker host
export DOCKER_HOST=tcp://remote-docker:2376
# Enable TLS verification
export DOCKER_TLS_VERIFY=1
export DOCKER_CERT_PATH=/path/to/certs
# Set default registry
export DOCKER_REGISTRY=registry.company.com
# Debug output
export DOCKER_BUILDKIT=1
```

Performance Tuning

Optimize Docker for better performance.

```
# Enable experimental features
echo '{"experimental": true}' | sudo tee /etc/docker/daemon.json
# Set storage driver options
{
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
# Configure logging
{
  "log-driver": "syslog",
  "log-opts": {"syslog-address": "udp://logs.company.com:514"}
}
```

Best Practices

Essential guidelines for efficient and secure Docker usage.

Security Best Practices

Keep your containers secure and production-ready.

```
# Run as non-root user in Dockerfile
RUN groupadd -r appuser && useradd -r -g appuser appuser
USER appuser
# Use specific image tags, not 'latest'
FROM node:16.20.0-alpine
# Scan images for vulnerabilities
docker scan myapp:latest
# Use read-only filesystems when possible
docker run --read-only nginx
```

Performance Optimization

Optimize containers for speed and resource efficiency.

```
# Use multi-stage builds to reduce image size
FROM node:16 AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

FROM node:16-alpine
WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY . .
CMD ["node", "server.js"]
```

Reference: This cheatsheet covers essential Docker commands and modern practices for efficient container management and deployment in development and production environments.