Docker CLI and Commands

Table of Contents

Docker CLI Overview

What is Docker CLI?

The **Docker Command Line Interface (CLI)** is the primary way users interact with Docker. It sends commands to the Docker daemon via REST API.

Basic Syntax

```
docker [OPTIONS] COMMAND [ARG...]

# Examples:
docker ps                        # No options, simple command
docker run -d nginx              # Option (-d) with command (run) and argument (nginx)
docker build -t myapp:1.0 .      # Multiple options and arguments
```

Getting Help

```
# General help
docker --help

# Help for specific command
docker run --help
docker build --help

# View Docker version
docker --version
docker version   # More detailed

# View system-wide information
docker info
```

How the CLI Works

Architecture Flow

```
  User Types
  Command




   Docker CLI
   (Parser)

        REST API Request


  Docker Daemon
    (dockerd)




  Container/Image
  Operations
```

Example Flow

```
# User command
docker run -d --name web nginx

# What happens:
1. Docker CLI receives command
2. CLI validates syntax
3. CLI sends REST API request to daemon:
   POST /containers/create
   {
     "Image": "nginx",
     "name": "web",
     "Detach": true
   }
4. Daemon processes request:
   - Checks if image exists
   - Pulls image if needed
   - Creates container
   - Starts container
5. Daemon sends response to CLI
6. CLI displays output to user
```

Docker Context

```
# View current context (where commands are sent)
docker context ls

# Output:
# NAME      DESCRIPTION         DOCKER ENDPOINT
# default * Current context     unix:///var/run/docker.sock

# Use remote Docker host
docker context create remote --docker "host=tcp://192.168.1.100:2376"
docker context use remote

# Switch back to local
docker context use default
```

Docker Image Commands

Listing Images

```
# List all images
docker images
docker image ls

# Output:
# REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
# nginx         latest    a1b2c3d4e5f6   2 weeks ago    142MB
# ubuntu        22.04     b2c3d4e5f6g7   3 weeks ago    77MB
# node          18        c3d4e5f6g7h8   1 month ago    993MB

# List images with digests
docker images --digests

# List images with specific repository
docker images nginx

# List images with filter
docker images --filter "dangling=true"  # Untagged images
docker images --filter "before=nginx"    # Before nginx was created
docker images --filter "since=ubuntu"    # After ubuntu was created
```

```
# Format output
docker images --format "table {{.Repository}}\t{{.Tag}}\t{{.Size}}"
```

Pulling Images

```
# Pull latest version
docker pull nginx

# Pull specific version
docker pull nginx:1.25
docker pull nginx:1.25-alpine

# Pull from specific registry
docker pull docker.io/library/nginx
docker pull ghcr.io/username/myapp

# Pull all tags of a repository
docker pull --all-tags nginx

# Pull image for specific platform
docker pull --platform linux/amd64 nginx
docker pull --platform linux/arm64 nginx
```

Searching Images

```
# Search Docker Hub
docker search nginx

# Output:
# NAME            DESCRIPTION                    STARS  OFFICIAL  AUTOMATED
# nginx           Official build of Nginx.       18000  [OK]
# jwilder/nginx...                               2000             [OK]

# Search with filters
docker search --filter "stars=1000" nginx
docker search --filter "is-official=true" nginx

# Limit results
docker search --limit 5 nginx
```

Downloading Docker Images

```
# How image pull works:
docker pull redis:7.0

# Output:
```

```
# 7.0: Pulling from library/redis
# a2abf6c4d29d: Pull complete     ← Layer 1
# c7a4e4382001: Pull complete     ← Layer 2
# 4044b9ba67c9: Pull complete     ← Layer 3
# c8388a79482f: Pull complete     ← Layer 4
# 413c8bb60be2: Pull complete     ← Layer 5
# 1abfd3011519: Pull complete     ← Layer 6
# Digest: sha256:db485f2e245b5b3329fdc7eff4eb00f913e09d8feb9ca720788059fdc2ed8339
# Status: Downloaded newer image for redis:7.0
# docker.io/library/redis:7.0

# What's happening:
# 1. Docker contacts registry (hub.docker.com)
# 2. Downloads image manifest
# 3. Downloads each layer (if not already cached)
# 4. Verifies integrity using SHA256
# 5. Stores in local image store
```

How a Docker Image Works

```
# Images are built from layers
docker history nginx

# Output:
# IMAGE          CREATED      CREATED BY                                      SIZE
# a1b2c3d4e5f6   2 weeks ago  /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
# <missing>      2 weeks ago  /bin/sh -c #(nop)  STOPSIGNAL SIGQUIT           0B
# <missing>      2 weeks ago  /bin/sh -c #(nop)  EXPOSE 80                    0B
# <missing>      2 weeks ago  /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
# <missing>      2 weeks ago  /bin/sh -c #(nop) COPY file:...                 4.62kB
# <missing>      2 weeks ago  /bin/sh -c apt-get update && apt-get install…   57.4MB
# <missing>      3 weeks ago  /bin/sh -c #(nop)  CMD ["bash"]                 0B
# <missing>      3 weeks ago  /bin/sh -c #(nop) ADD file:... in /             77.8MB

# Each layer is immutable and cached
# Layers are shared between images to save space

# Inspect image details
docker image inspect nginx

# Output (JSON):
# {
#   "Id": "sha256:a1b2c3...",
#   "RepoTags": ["nginx:latest"],
#   "Size": 142000000,
#   "Architecture": "amd64",
#   "Os": "linux",
#   "Layers": [
#     "sha256:layer1...",
#     "sha256:layer2...",
#     "sha256:layer3..."
#   ],
#   ...
# }
```

Building Images

```
# Build from Dockerfile
docker build -t myapp:1.0 .

# Build with build arguments
docker build --build-arg VERSION=1.0 -t myapp .

# Build with no cache
docker build --no-cache -t myapp .

# Build with specific Dockerfile
docker build -f Dockerfile.prod -t myapp:prod .

# Build for multiple platforms
docker buildx build --platform linux/amd64,linux/arm64 -t myapp .

# Build and push
docker build -t username/myapp:latest . && docker push username/myapp:latest
```

Tagging Images

```
# Tag an existing image
docker tag nginx myregistry.com/nginx:v1

# Create multiple tags
docker tag myapp:latest myapp:1.0
docker tag myapp:latest myapp:stable

# Tag for Docker Hub
docker tag myapp username/myapp:latest

# View all tags
docker images myapp
```

Pushing Images

```
# Login to Docker Hub
docker login
# Enter username and password

# Login to private registry
docker login myregistry.com

# Push image
docker push username/myapp:latest

# Push all tags
docker push --all-tags username/myapp
```

Removing Docker Images

```
# Remove specific image
docker rmi nginx
docker image rm nginx

# Remove image by ID
docker rmi a1b2c3d4e5f6

# Force remove (even if container using it)
docker rmi -f nginx

# Remove multiple images
docker rmi nginx redis postgres

# Remove all unused images
docker image prune

# Remove all images
docker image prune -a

# Remove images with filter
docker image prune --filter "until=24h"
```

Saving and Loading Images

```
# Save image to tar file
docker save nginx > nginx.tar
docker save -o nginx.tar nginx

# Save multiple images
docker save -o images.tar nginx redis postgres
```

```
# Load image from tar file
docker load < nginx.tar
docker load -i nginx.tar

# Export container filesystem
docker export mycontainer > container.tar

# Import as image
docker import container.tar myimage:latest
```

Docker Container Commands

Running Containers

```
# Basic run
docker run nginx

# Run in detached mode (background)
docker run -d nginx

# Run with name
docker run --name web nginx

# Run with port mapping
docker run -p 8080:80 nginx
docker run -p 127.0.0.1:8080:80 nginx  # Bind to specific IP

# Run with multiple ports
docker run -p 8080:80 -p 8443:443 nginx

# Run with environment variables
docker run -e "ENV=production" -e "DEBUG=false" myapp

# Run with volume
docker run -v /host/path:/container/path nginx
docker run -v myvolume:/data nginx

# Run with resource limits
docker run --cpus="1.5" --memory="512m" nginx

# Run with restart policy
docker run --restart=always nginx
docker run --restart=on-failure:3 nginx
```

```
# Run with network
docker run --network mynetwork nginx

# Run in interactive mode
docker run -it ubuntu bash

# Run with automatic removal after exit
docker run --rm ubuntu echo "Hello"

# Complete example
docker run -d \
  --name myapp \
  --restart=always \
  -p 8080:80 \
  -e "ENV=production" \
  -v mydata:/data \
  --memory="512m" \
  --cpus="1" \
  nginx:latest
```

Listing Containers

```
# List running containers
docker ps
docker container ls

# Output:
# CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS
# a1b2c3d4e5f6   nginx     "/docker-entrypoint.…"   5 minutes ago   Up 5 minutes   0.0.0.0:8

# List all containers (including stopped)
docker ps -a
docker container ls -a

# List only container IDs
docker ps -q

# List with size
docker ps -s

# List with custom format
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

# List latest created container
docker ps -l
```

```
# Filter containers
docker ps --filter "status=running"
docker ps --filter "name=web"
docker ps --filter "label=env=production"
```

Managing Running Containers

```
# Start stopped container
docker start mycontainer

# Stop running container
docker stop mycontainer

# Stop with timeout
docker stop -t 30 mycontainer  # Wait 30 seconds before force kill

# Restart container
docker restart mycontainer

# Pause container
docker pause mycontainer

# Unpause container
docker unpause mycontainer

# Kill container (immediate termination)
docker kill mycontainer

# Kill with specific signal
docker kill --signal=SIGTERM mycontainer

# Rename container
docker rename oldname newname

# Update container configuration
docker update --cpus="2" --memory="1g" mycontainer
```

Accessing a Container Locally

```
# Execute command in running container
docker exec mycontainer ls /app

# Interactive bash shell
docker exec -it mycontainer bash
docker exec -it mycontainer sh  # For Alpine-based images
```

```
# Execute as specific user
docker exec -u root -it mycontainer bash

# Execute with environment variable
docker exec -e "VAR=value" mycontainer env

# Run multiple commands
docker exec mycontainer sh -c "cd /app && ls -la"

# Example: Database operations
docker exec mysql_container mysql -u root -p'password' -e "SHOW DATABASES;"

# Example: View logs
docker exec mycontainer cat /var/log/app.log

# Example: Check process
docker exec mycontainer ps aux
```

Viewing Container Logs

```
# View logs
docker logs mycontainer

# Follow logs (like tail -f)
docker logs -f mycontainer

# Show timestamps
docker logs -t mycontainer

# Show last 100 lines
docker logs --tail 100 mycontainer

# Show logs since specific time
docker logs --since 2023-01-01 mycontainer
docker logs --since 30m mycontainer  # Last 30 minutes

# Show logs until specific time
docker logs --until 2023-01-01 mycontainer

# Combine options
docker logs -f --tail 50 --since 10m mycontainer
```

Container Inspection

```
# Inspect container details
docker inspect mycontainer
```

```
# Get specific info with format
docker inspect --format='{{.NetworkSettings.IPAddress}}' mycontainer
docker inspect --format='{{.State.Status}}' mycontainer
docker inspect --format='{{.Config.Image}}' mycontainer

# View container processes
docker top mycontainer

# View resource usage statistics
docker stats
docker stats mycontainer

# View real-time events
docker events --filter container=mycontainer

# View port mappings
docker port mycontainer

# View container changes
docker diff mycontainer
# Output:
# A /app/newfile.txt      (Added)
# C /app/config.json      (Changed)
# D /app/oldfile.txt      (Deleted)
```

Docker Commit

```
# Create image from container
docker commit mycontainer mynewimage:v1

# Commit with message and author
docker commit -m "Added new feature" -a "John Doe" mycontainer mynewimage:v1

# Commit with changes
docker commit --change="ENV DEBUG=true" mycontainer mynewimage:v1

# Example workflow:
# 1. Start container
docker run -it --name mycontainer ubuntu bash

# 2. Make changes inside container
apt-get update && apt-get install -y curl

# 3. Exit container
exit
```

```
# 4. Commit changes to new image
docker commit mycontainer ubuntu-with-curl:latest

# 5. Run new image
docker run ubuntu-with-curl:latest curl --version
```

Copying Files

```
# Copy from container to host
docker cp mycontainer:/app/file.txt ./local-file.txt
docker cp mycontainer:/app/ ./local-folder/

# Copy from host to container
docker cp ./local-file.txt mycontainer:/app/file.txt
docker cp ./local-folder/ mycontainer:/app/

# Copy with ownership preservation
docker cp --archive mycontainer:/app/ ./backup/

# Example: Backup database
docker cp mysql_container:/var/lib/mysql/backup.sql ./backup.sql

# Example: Deploy new config
docker cp ./new-config.yml app_container:/app/config.yml
docker restart app_container
```

Removing Containers

```
# Remove stopped container
docker rm mycontainer

# Force remove running container
docker rm -f mycontainer

# Remove multiple containers
docker rm container1 container2 container3

# Remove all stopped containers
docker container prune

# Remove containers with filter
docker container prune --filter "until=24h"

# Remove all containers (running and stopped)
docker rm -f $(docker ps -aq)
```

Docker Network Commands

Listing Networks

```
# List networks
docker network ls

# Output:
# NETWORK ID     NAME      DRIVER    SCOPE
# a1b2c3d4e5f6   bridge    bridge    local
# b2c3d4e5f6g7   host      host      local
# c3d4e5f6g7h8   none      null      local

# Inspect network
docker network inspect bridge
```

Creating Networks

```
# Create bridge network
docker network create mynetwork

# Create with specific driver
docker network create --driver bridge mynetwork

# Create with subnet
docker network create --subnet=172.18.0.0/16 mynetwork

# Create with gateway
docker network create --subnet=172.18.0.0/16 --gateway=172.18.0.1 mynetwork

# Create with custom options
docker network create \
  --driver=bridge \
  --subnet=172.28.0.0/16 \
  --ip-range=172.28.5.0/24 \
  --gateway=172.28.5.254 \
  mynetwork
```

Connecting Containers to Networks

```
# Connect running container to network
docker network connect mynetwork mycontainer

# Connect with specific IP
docker network connect --ip 172.18.0.10 mynetwork mycontainer
```

```
# Connect with alias
docker network connect --alias web mynetwork mycontainer

# Disconnect from network
docker network disconnect mynetwork mycontainer

# Run container on specific network
docker run -d --network mynetwork --name web nginx
```

Network Use Cases

```
# Example: Multi-tier application

# 1. Create network
docker network create app-network

# 2. Run database
docker run -d \
  --name db \
  --network app-network \
  -e POSTGRES_PASSWORD=secret \
  postgres:14

# 3. Run backend (can access db by hostname "db")
docker run -d \
  --name backend \
  --network app-network \
  -e DATABASE_HOST=db \
  mybackend:latest

# 4. Run frontend (can access backend by hostname "backend")
docker run -d \
  --name frontend \
  --network app-network \
  -p 80:80 \
  myfrontend:latest

# Containers can communicate using their names!
# backend can connect to: postgresql://db:5432
# frontend can connect to: http://backend:3000
```

Removing Networks

```
# Remove network
docker network rm mynetwork
```

```
# Remove all unused networks
docker network prune

# Force remove (disconnect containers first)
docker network rm -f mynetwork
```

Docker Volume Commands

Listing Volumes

```
# List volumes
docker volume ls

# Output:
# DRIVER    VOLUME NAME
# local     mydata
# local     postgres_data
# local     redis_cache

# Inspect volume
docker volume inspect mydata

# Output:
# [{
#     "CreatedAt": "2023-10-28T10:00:00Z",
#     "Driver": "local",
#     "Mountpoint": "/var/lib/docker/volumes/mydata/_data",
#     "Name": "mydata",
#     "Options": null,
#     "Scope": "local"
# }]
```

Creating Volumes

```
# Create volume
docker volume create mydata

# Create with driver
docker volume create --driver local mydata

# Create with labels
docker volume create --label env=production mydata

# Create with options
```

```
docker volume create \
  --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.1.1,rw \
  --opt device=:/path/to/dir \
  nfs_volume
```

Using Volumes

```
# Run container with volume
docker run -d -v mydata:/app/data nginx

# Run with multiple volumes
docker run -d \
  -v mydata:/app/data \
  -v logs:/app/logs \
  nginx

# Run with read-only volume
docker run -d -v mydata:/app/data:ro nginx

# Anonymous volume (Docker creates name)
docker run -d -v /app/data nginx

# Bind mount (mount host directory)
docker run -d -v /host/path:/container/path nginx

# Example: Database with persistent storage
docker run -d \
  --name postgres \
  -v pgdata:/var/lib/postgresql/data \
  -e POSTGRES_PASSWORD=secret \
  postgres:14

# Data persists even if container is removed!
docker rm -f postgres
docker run -d --name postgres -v pgdata:/var/lib/postgresql/data postgres:14
# Data is still there!
```

Volume Backup and Restore

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

# Restore volume
docker run --rm \
  -v mydata:/data \
  -v $(pwd):/backup \
  ubuntu \
  tar xzf /backup/mydata-backup.tar.gz -C /
```

Removing Volumes

```
# Remove volume
docker volume rm mydata

# Remove all unused volumes
docker volume prune

# Remove volumes with filter
docker volume prune --filter "label=env=development"

# Remove container and its volumes
docker rm -v mycontainer
```

Docker System Commands

System Information

```
# Show system-wide information
docker info

# Show version information
docker version

# Show disk usage
docker system df

# Output:
# TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
# Images          10        5         2.5GB     1.2GB (48%)
# Containers      15        3         500MB     400MB (80%)
# Local Volumes   8         4         1.5GB     800MB (53%)
# Build Cache     25        0         3.2GB     3.2GB (100%)

# Detailed view
docker system df -v
```

System Cleanup

```
# Remove all unused data
docker system prune

# Remove all unused data including volumes
docker system prune --volumes

# Remove everything (including unused images)
docker system prune -a

# Remove with filter
docker system prune --filter "until=24h"

# Remove without confirmation
docker system prune -f

# Complete cleanup (DANGER: Removes everything!)
docker system prune -a --volumes -f
```

Monitoring Events

```
# Show real-time events
docker events

# Filter events
docker events --filter type=container
docker events --filter event=start
docker events --filter container=mycontainer

# Show events since specific time
docker events --since '2023-10-28'
docker events --since '1h'

# Show events in JSON format
docker events --format '{{json .}}'
```

Additional Docker Commands

Docker Registry

```
# Login to registry
docker login
docker login myregistry.com
docker login -u username -p password
```

```
# Logout
docker logout

# Search Docker Hub
docker search nginx --limit 5
```

Docker Plugin

```
# List plugins
docker plugin ls

# Install plugin
docker plugin install plugin-name

# Enable/disable plugin
docker plugin enable plugin-name
docker plugin disable plugin-name

# Remove plugin
docker plugin rm plugin-name
```

Docker Config

```
# Create config (for Swarm)
docker config create myconfig config-file.conf

# List configs
docker config ls

# Inspect config
docker config inspect myconfig

# Remove config
docker config rm myconfig
```

Docker Secret

```
# Create secret (for Swarm)
echo "mypassword" | docker secret create db_password -

# List secrets
docker secret ls

# Inspect secret
docker secret inspect db_password
```

```
# Remove secret
docker secret rm db_password
```

Docker Context

```
# List contexts
docker context ls

# Create context
docker context create mycontext --docker "host=tcp://192.168.1.100:2376"

# Use context
docker context use mycontext

# Inspect context
docker context inspect mycontext

# Remove context
docker context rm mycontext
```

Docker Checkpoint (Experimental)

```
# Enable experimental features
export DOCKER_EXPERIMENTAL=1

# Create checkpoint
docker checkpoint create mycontainer checkpoint1

# List checkpoints
docker checkpoint ls mycontainer

# Restore from checkpoint
docker start --checkpoint checkpoint1 mycontainer

# Remove checkpoint
docker checkpoint rm mycontainer checkpoint1
```

Useful Command Combinations

Container Management

```
# Stop and remove all containers
docker stop $(docker ps -aq) && docker rm $(docker ps -aq)

# Remove all containers with specific name pattern
```

```
docker rm -f $(docker ps -aq --filter "name=test_")

# Restart all running containers
docker restart $(docker ps -q)

# View logs from all containers
docker ps -q | xargs -L1 docker logs --tail 10
```

Image Management

```
# Remove all untagged images
docker rmi $(docker images -f "dangling=true" -q)

# Remove all images with specific pattern
docker rmi $(docker images | grep "myapp" | awk '{print $3}')

# Pull all tags of an image
docker images --format "{{.Repository}}:{{.Tag}}" | grep myapp | xargs -L1 docker pull
```

System Information

```
# Show running containers with resource usage
docker stats $(docker ps --format={{.Names}})

# Show container IPs
docker ps -q | xargs docker inspect --format '{{.Name}} - {{range .NetworkSettings.Networks

# Show container port mappings
docker ps --format "table {{.Names}}\t{{.Ports}}"
```

Quick Reference Cheat Sheet

Most Common Commands

```
# Images
docker pull <image>          # Download image
docker images                # List images
docker rmi <image>           # Remove image
docker build -t <name> .     # Build image

# Containers
docker run <image>           # Run container
docker ps                    # List running containers
docker ps -a                 # List all containers
docker stop <container>      # Stop container
docker start <container>       # Start container
docker rm <container>          # Remove container
docker logs <container>        # View logs
docker exec -it <container> bash  # Access shell

# System
docker info                    # System information
docker system prune            # Clean up
docker volume prune            # Remove unused volumes
docker network prune           # Remove unused networks
```

Key Takeaways

1. **Docker CLI** communicates with Docker daemon via REST API
2. **Images** are templates; **containers** are running instances
3. **Volumes** provide persistent storage for containers
4. **Networks** enable container communication
5. **System commands** help monitor and clean up resources
6. **Use `--help`** with any command to see options and usage

What's Next?

Now that you know the CLI commands, you're ready to: - Create your first Dockerfile - Build custom images - Work with Docker Compose - Implement real-world projects

Master the CLI, master Docker!