AI-03922 Ubuntu Docker CLI Cheat Sheet

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	June 11, 2026 7:53 PM

Create a comprehensive Ubuntu Server and Docker CLI Cheat Sheet

Here is a comprehensive, production-focused **Ubuntu Server and Docker CLI Cheat Sheet** designed for quick reference, high scannability, and rapid troubleshooting.

1. Ubuntu Server Essentials

System Performance & Hardware Diagnostics

```
top                     # Real-time process monitoring (class
ic)
htop                    # Interactive interactive process vie
wer (preferred)
df -h                   # Human-readable disk space usage
du -sh * # Summary of disk usage for current directory conten
ts
free -m                 # Total, used, and free memory in Meg
abytes
lsblk                   # List block devices (disks and parti
tions)
lscpu                   # Display CPU architecture details
journalctl -xe          # View recent system logs with failur
e explanations
```

Network Administration & Diagnostics

```
ip a                    # Show all network interfaces and IP addresses
ip route                # Display current routing table
ping -c 4 8.8.8.8       # Send 4 ICMP echo requests to verify internet connectivity
ss -tulpn               # List active listening ports with PID and process names
traceroute google.com   # Trace network path to a destination
curl -I <https://xxx.xxx> # Fetch HTTP response headers to test web endpoints
```

System Updates & Package Management (apt)

```
sudo apt update         # Refresh package index
sudo apt upgrade -y     # Upgrade all installed packages safely
sudo apt install [pkg]  # Install a specific package
sudo apt autoremove -y  # Remove orphaned packages and dependencies
```

2. Docker Daemon & System Administration

Daemon State & Service Management

```
sudo systemctl status docker   # Check if Docker daemon is running
sudo systemctl start docker    # Start Docker service
sudo systemctl restart docker  # Restart Docker daemon
sudo systemctl enable docker   # Enable Docker to start on system boot
```

System Information & Diagnostics

```
docker version          # Detailed client and server engine v
ersions
docker info             # System-wide information (storage dr
iver, containers, etc.)
docker system df        # Show docker disk usage (images, con
tainers, volumes)
```

Nuclear Options (Aggressive Cleanup)

```
docker system prune -f             # Remove unused container
s, networks, and images
docker system prune -a --volumes   # Deep clean: clears ALL s
topped containers, unused images, and unused volumes
```

3. Container Management Lifecycle

Creation and Execution

```
# Run a detached container with name and port mapping (Host:C
ontainer)
docker run -d --name my-app -p 8080:80 nginx:alpine

# Run an interactive container with a pseudo-TTY (ideal for d
ebugging)
docker run -it --name debug-box ubuntu:latest /bin/bash

# Run a container that automatically destroys itself upon exi
t
docker run --rm alpine ping -c 4 google.com
```

Runtime Control & State

```
docker ps                # List only running containers
docker ps -a             # List all containers (running, exited, stopped)
docker stop [id/name]    # Gracefully stop a running container
docker start [id/name]   # Start a stopped container
docker restart [id/name]# Restart a container
docker rm [id/name]      # Delete a specific container (must be stopped)
docker rm -f $(docker ps -aq) # Nuclear: Force remove ALL local containers
```

Inspection & Live Diagnostics

```
docker logs -f [name]          # Follow live container log output
docker logs --tail 100 [name] # View only the last 100 log lines
docker inspect [id/name]       # Return low-level JSON configuration data
docker stats                   # Live stream of resource usage statistics (CPU/Memory)
docker top [id/name]           # Display the running processes of a container
```

Internal Interaction

```
# Execute an interactive bash shell inside a running container
docker exec -it [id/name] /bin/bash

# Copy files between host machine and container
docker cp /path/on/host [name]:/path/in/container
docker cp [name]:/path/in/container /path/on/host
```

4. Docker Image Management

```
docker images                    # List all locally stored imag
es
docker pull nginx:alpine         # Fetch image from Docker Hub
registry
docker rmi [image_id]            # Delete a specific local imag
e
docker image prune -a            # Remove all unused local imag
es
```

Building & Tagging

```
# Build an image from a Dockerfile in the current directory
docker build -t my-app:1.0 .

# Build an image ignoring cached layers (fresh package downlo
ads)
docker build --no-cache -t my-app:1.1 .

# Tag an existing image for a target registry
docker tag my-app:1.0 registry.example.com/my-app:1.0
```

5. Persistent Data & Network Volumes

Docker Volumes (Named storage managed by Docker)

```
docker volume create my-data   # Create a persistent volume
docker volume ls               # List all managed volumes
docker volume inspect my-data  # Find physical mountpath loca
tion on host
docker volume rm my-data       # Delete volume (fails if atta
ched to container)
```

Mounting Patterns at Runtime

```
# Pattern 1: Named Volume Mount
docker run -d -v my-data:/var/lib/mysql mysql:latest

# Pattern 2: Bind Mount (Direct mapping to a specific host di
rectory)
docker run -d -v /home/user/app/config:/etc/nginx/conf.d ngin
x:alpine
```

6. Container Networking

```
docker network ls                # List all configured network
s
docker network create --driver bridge my-net  # Create an iso
lated bridge network
docker network inspect my-net    # Inspect network subnet deta
ils and connected containers
```

Connecting Containers at Runtime

```
# Run a container inside a specific custom network
docker run -d --name backend --network my-net backend-image:l
atest

# Dynamically connect an already running container to a netwo
rk
docker network connect my-net frontend
```

7. Multi-Container Deployment (docker compose)

> **Note:** The modern v2 CLI uses docker compose (space), replacing the legacy standalone binary docker-compose (hyphen).

```
docker compose up -d      # Create and start entire stack in
the background
docker compose down       # Stop and completely remove contai
ners and networks
docker compose down -v    # Stop stack and destroy attached n
amed volumes
docker compose ps         # View state of services within the
current stack
docker compose logs -f    # Tail aggregated logs from all ser
vices in stack
docker compose restart    # Safely restart all containers in
the stack
```

What core scenarios or edge cases do you find yourself troubleshooting most often with this stack?