

Docker Containers: Running and Management

Container Basics

A container is a running instance of a Docker image. Multiple containers can run from the same image, each isolated with its own filesystem, network interface, and process space.

Container Lifecycle

```
Created → Running → Paused ↔ Running → Stopped → Removed
          ↓           ↓
          Running       Restart
```

Running Containers

Basic Run Command

```
# Run a simple container
docker run nginx:latest

# Output (container logs):
# /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
# /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
# /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
# Running nginx server...
```

Run in Detached Mode

```
# Run container in background
docker run -d --name web-app nginx:latest

# Output: Container ID
# a1b2c3d4e5f6a0b1c2d3e4f5a6b7c8d9

# Check it's running
docker ps
```

```
# Output:  
# CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS  
# a1b2c3d4e5f6        nginx:latest        "/docker-entrypoint.s"   10 seconds ago      Up 9s
```

Interactive Mode

```
# Run container with interactive terminal  
docker run -it ubuntu:22.04 /bin/bash  
  
# Output: You're now inside the container  
# root@a1b2c3d4e5f6:/#  
  
# Execute commands inside  
# root@a1b2c3d4e5f6:/# apt-get update  
# Reading package lists... Done  
# root@a1b2c3d4e5f6:/# python3 --version  
# Python 3.10.6  
# root@a1b2c3d4e5f6:/# exit
```

Port Mapping

Expose Ports

```
# Map port 8080 on host to 80 in container  
docker run -d -p 8080:80 --name web nginx:latest  
  
# Output:  
# a1b2c3d4e5f6  
  
# Map multiple ports  
docker run -d \  
  -p 8080:80 \  
  -p 3000:3000 \  
  -p 5432:5432 \  
  --name myapp myapp:latest  
  
# Map to specific host interface  
docker run -d -p 127.0.0.1:8080:80 nginx:latest  
  
# Random port mapping  
docker run -d -P nginx:latest  
  
# View port mappings  
docker port web
```

```
# Output:  
# 80/tcp -> 0.0.0.0:8080
```

Environment Variables

Set Environment Variables

```
# Single environment variable  
docker run -d \  
  -e PYTHONUNBUFFERED=1 \  
  --name myapp \  
  myapp:latest  
  
# Multiple variables  
docker run -d \  
  -e DATABASE_URL=postgres://localhost:5432/mydb \  
  -e DEBUG=true \  
  -e LOG_LEVEL=debug \  
  -e API_KEY=secret123 \  
  --name app \  
  myapp:latest  
  
# From file  
docker run -d \  
  --env-file .env \  
  --name app \  
  myapp:latest
```

View Environment Variables

```
# Check environment in running container  
docker exec myapp env  
  
# Output:  
# PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
# HOSTNAME=a1b2c3d4e5f6  
# DATABASE_URL=postgres://localhost:5432/mydb  
# DEBUG=true  
# LOG_LEVEL=debug  
# PYTHONUNBUFFERED=1  
# HOME=/root
```

Resource Limits

CPU and Memory Constraints

```
# Limit memory to 512MB
docker run -d \
  -m 512m \
  --name app \
  myapp:latest

# Limit memory and swap
docker run -d \
  -m 512m \
  --memory-swap 1g \
  --name app \
  myapp:latest

# Limit CPU shares (out of 1024)
docker run -d \
  --cpu-shares 512 \
  --name app \
  myapp:latest

# Limit to specific CPUs
docker run -d \
  --cpus="2.0" \
  --name app \
  myapp:latest

# Verify resource settings
docker inspect -f \
  '{{json .HostConfig | json .Memory}}' \
  my_container

# Output:
# 536870912 (bytes, which is 512MB)
```

Container Communication

Linking Containers (Legacy)

```
# Start database container
docker run -d \
  --name db \
  postgres:15-alpine

# Start app linked to database
```

```
docker run -d \
  --link db:database \
  --name app \
  myapp:latest

# Inside app container, access as 'database'
docker exec app bash -c 'echo $DATABASE_PORT_5432_TCP_ADDR'

# Output:
# 172.17.0.2
```

User-Defined Networks

```
# Create custom network
docker network create mynet

# Run containers on network
docker run -d \
  --network mynet \
  --name db \
  postgres:15

docker run -d \
  --network mynet \
  --name app \
  myapp:latest

# Containers can communicate by name
docker exec app ping db

# Output:
# PING db (172.18.0.2): 56 data bytes
# 64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.123 ms
```

Container Inspection and Monitoring

View Logs

```
# View container logs
docker logs d4e5f6a7b8c9

# Output:
# INFO Starting application...
# INFO Listening on port 8000
# INFO Health check passed
```

```
# Follow logs in real-time
docker logs -f d4e5f6a7b8c9

# Output (streaming):
# INFO Starting application...
# INFO Listening on port 8000
# [continuously streams new logs]

# View last 100 lines
docker logs --tail 100 d4e5f6a7b8c9

# View logs with timestamps
docker logs -t d4e5f6a7b8c9

# Output:
# 2024-02-17T14:30:00.123456789Z INFO Starting...
# 2024-02-17T14:30:01.234567890Z INFO Ready...
```

Inspect Container Details

```
# Get detailed container information
docker inspect d4e5f6a7b8c9

# Output (abbreviated):
# [
#     {
#         "Id": "d4e5f6a7b8c9...",
#         "Created": "2024-02-17T14:30:00.123456789Z",
#         "State": {
#             "Status": "running",
#             "Running": true,
#             "Paused": false,
#             "Pid": 1234
#         },
#         "Image": "sha256:abc123def456...",
#         "Name": "/myapp",
#         "NetworkSettings": {
#             "IPAddress": "172.17.0.2",
#             "Ports": {"80/tcp": [{"HostIp": "0.0.0.0", "HostPort": "8080"}]}
#         }
#     }
# ]

# Get specific information
docker inspect -f '{{.NetworkSettings.IPAddress}}' d4e5f6a7b8c9
```

```
# Output:  
# 172.17.0.2
```

View Container Process Status

```
# See processes running in container  
docker top d4e5f6a7b8c9  
  
# Output:  
#  UID      PID      PPID      C      STIME      TTY  STAT      TIME  COMMAND  
#  root      1        0        0      14:30      ?      Ss      0:00  /sbin/docker-init -- python  
#  root      9        1        1      14:30      ?      S1      0:05  python app.py  
  
# Get resource usage stats  
docker stats d4e5f6a7b8c9  
  
# Output (streaming):  
# CONTAINER ID      NAME      CPU %      MEM USAGE / LIMIT  
# d4e5f6a7b8c9      myapp      2.15%      124.5MiB / 512MiB
```

Executing Commands in Containers

Execute Commands

```
# Run one-off command  
docker exec d4e5f6a7b8c9 ls -la  
  
# Output:  
# total 12  
# drwxr-xr-x  1 root  root  4096 Feb 17 14:30 .  
# drwxr-xr-x  1 root  root  4096 Feb 17 14:30 ..  
# -rw-r--r--  1 root  root   220 Feb 17 14:30 app.py  
  
# Run interactive command  
docker exec -it d4e5f6a7b8c9 /bin/bash  
  
# Output: Interactive bash prompt  
# root@d4e5f6a7b8c9:/#  
  
# Run with specific user  
docker exec -u appuser d4e5f6a7b8c9 whoami  
  
# Output:  
# appuser
```

Container Lifecycle Management

Start and Stop Containers

```
# Create container without starting
docker create --name app-backup nginx:latest

# Output:
# a1b2c3d4e5f6

# Start container
docker start app-backup

# Restart container
docker restart app-backup

# Pause container (freeze processes)
docker pause app-backup

# Unpause container
docker unpause app-backup

# Stop container gracefully (30 second timeout)
docker stop -t 30 app-backup

# Kill container immediately
docker kill app-backup

# Remove container
docker rm app-backup

# Remove running container
docker rm -f app-backup

# Remove container and volumes
docker rm -v app-backup
```

Copying Files

Copy Between Host and Container

```
# Copy from host to container
docker cp ./config.json d4e5f6a7b8c9:/app/
```

```
# Copy from container to host
docker cp d4e5f6a7b8c9:/var/log/app.log ./logs/

# Copy entire directory
docker cp d4e5f6a7b8c9:/data ./container-data
```

Container Cleanup

Remove Unused Containers

```
# Remove stopped containers
docker container prune

# Output:
# Deleted Containers:
# a1b2c3d4e5f6
# b2c3d4e5f6a7
# Total reclaimed space: 2.1MB

# Remove all stopped containers
docker rm $(docker ps -aq)

# Remove containers matching pattern
docker rm $(docker ps -a | grep "Exited" | awk '{print $1}')
```

Common Container Operations

Rename Container

```
# Rename a container
docker rename old-name new-name

# Verify
docker ps --filter "name=new-name"
```

Create Checkpoint

```
# Checkpoint container (experimental)
docker checkpoint create --checkpoint-id backup app

# Restore from checkpoint
docker start --checkpoint=backup app
```

Container Commit (Create Image)

```
# Create image from running container
docker commit d4e5f6a7b8c9 myapp:modified

# Output:
# sha256:abc123def456ghi789jkl012mno345pqr678stu

# With author info
docker commit -a "John Doe" -m "Added dependencies" d4e5f6a7b8c9 myapp:v2
```

Best Practices

1. **One process per container** - Each container should run one main process
2. **Use health checks** - Monitor container health
3. **Implement graceful shutdown** - Handle SIGTERM signal
4. **Use resource limits** - Prevent runaway processes
5. **Keep logs manageable** - Configure log rotation
6. **Non-root user** - Run applications as non-root

Next Steps

- Learn about [Networking and Storage](#)
- Explore [Docker Compose](#) for multi-container apps