Docker Installation Guide

Table of Contents

Docker Installation on Windows

System Requirements

Minimum Requirements: - Windows 10 64-bit: Pro, Enterprise, or Education (Build 19041 or higher) - OR Windows 11 64-bit - Hardware virtualization (VT-x/AMD-V) enabled in BIOS - 4GB RAM minimum (8GB+ recommended) - 64-bit processor with Second Level Address Translation (SLAT) - WSL 2 (Windows Subsystem for Linux 2)

Installation Methods

Method 1: Docker Desktop (Recommended) Step 1: Download Docker Desktop

```
1. Visit https://www.docker.com/products/docker-desktop
2. Click "Download for Windows"
3. Download Docker Desktop Installer.exe
```

Step 2: Install Docker Desktop

```
# Run the installer
# Double-click Docker Desktop Installer.exe

# During installation:
  Enable WSL 2 Windows Features (recommended)
  Add shortcut to desktop

# Click "Install"
# Wait for installation to complete (5-10 minutes)
```

Step 3: Enable WSL 2

```
# Open PowerShell as Administrator

# Enable WSL
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norest
```

```
# Enable Virtual Machine Platform
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

# Restart your computer
Restart-Computer

# After restart, set WSL 2 as default
wsl --set-default-version 2

# Install a Linux distribution (Ubuntu recommended)
wsl --install -d Ubuntu
```

Step 4: Start Docker Desktop

```
1. Search for "Docker Desktop" in Start Menu
2. Click to launch
3. Accept the service agreement
4. Sign in (optional but recommended)
5. Wait for Docker Engine to start (green icon in taskbar)
```

Step 5: Verify Installation

```
# Open PowerShell or Command Prompt
docker --version
# Output: Docker version 24.0.6, build ed223bc

docker run hello-world
# Output: Hello from Docker!
# This message shows that your installation appears to be working correctly.
```

Docker Desktop Settings (Windows)

```
Open Docker Desktop → Settings

General:
    Use WSL 2 based engine
    Start Docker Desktop when you log in

Resources:
  CPUs: 4 (adjust based on your system)
  Memory: 4 GB (adjust based on your system)
  Swap: 1 GB
  Disk image size: 60 GB

Docker Engine:
  {
    "debug": true,
```

```
  "experimental": false
}
```

Windows-Specific Commands

```
# Check Docker status
docker info

# Run a Windows container (requires Windows Server base image)
docker run -it mcr.microsoft.com/windows/servercore:ltsc2022 cmd

# Run a Linux container (default with WSL 2)
docker run -it ubuntu bash

# Switch between Windows and Linux containers
# Right-click Docker Desktop icon → Switch to Windows containers
# Right-click Docker Desktop icon → Switch to Linux containers
```

Common Windows Issues and Fixes

Issue 1: WSL 2 Installation Failed

```
# Solution: Manually install WSL 2 kernel update
# Download from: https://aka.ms/wsl2kernel
# Run: wsl_update_x64.msi

# Then set WSL 2 as default
wsl --set-default-version 2
```

Issue 2: Virtualization Not Enabled

```
Solution:
1. Restart computer
2. Enter BIOS/UEFI (usually F2, F10, F12, or Del during boot)
3. Find "Virtualization Technology" or "Intel VT-x" or "AMD-V"
4. Enable it
5. Save and exit BIOS
6. Restart Docker Desktop
```

Issue 3: Docker Daemon Not Starting

```
# Check if Hyper-V is enabled
Get-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V

# If not enabled, run:
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

```
# Restart computer
Restart-Computer
```

Using Docker in VS Code (Windows)

```
# Install Docker extension in VS Code
code --install-extension ms-azuretools.vscode-docker

# Features:
# - View containers and images
# - Build, run, and debug containers
# - Connect to container terminals
# - Dockerfile syntax highlighting
```

Docker Installation on macOS

System Requirements

Minimum Requirements: - macOS 11 or newer - Mac hardware from 2010 or newer - 4GB RAM minimum (8GB+ recommended) - VirtualBox prior to version 4.3.30 must NOT be installed

Installation Steps

Step 1: Download Docker Desktop for Mac

```
# Visit: https://www.docker.com/products/docker-desktop

# Choose your chip:
# - Apple Silicon (M1, M2, M3) → Download for Apple Silicon
# - Intel Chip → Download for Intel Chip

# Check your chip:
uname -m
# Output: arm64 (Apple Silicon) or x86_64 (Intel)
```

Step 2: Install Docker Desktop

```
# 1. Open Docker.dmg file
# 2. Drag Docker icon to Applications folder
# 3. Open Applications → Docker
# 4. Docker Desktop starts

# Grant privileged access when prompted
# Docker needs privileged access to install networking components
```

Step 3: Configure Docker Desktop (Mac)

```
Open Docker Desktop → Preferences

General:
    Start Docker Desktop when you log in
    Include VM in Time Machine backups (optional)

Resources:
  CPUs: 4 (adjust based on your Mac)
  Memory: 4 GB (adjust based on your Mac)
  Swap: 1 GB
  Disk image size: 60 GB

Advanced:
    Enable default Docker socket
```

Step 4: Verify Installation

```
# Open Terminal

# Check Docker version
docker --version
# Output: Docker version 24.0.6, build ed223bc

# Check Docker Compose version
docker-compose --version
# Output: Docker Compose version v2.21.0

# Run hello-world
docker run hello-world
# Output: Hello from Docker!

# Check Docker info
docker info
# Shows system-wide information
```

macOS-Specific Features

Apple Silicon (M1/M2/M3) Considerations

```
# Docker Desktop uses Rosetta 2 for x86 images
# Most images work seamlessly, but native ARM images are faster

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

```
# Pull x86 image (runs via emulation)
docker pull --platform linux/amd64 nginx

# Check image platform
docker image inspect nginx --format='{{.Architecture}}'
# Output: arm64 or amd64
```

File Sharing Configuration

```
# Configure which folders Docker can access
# Docker Desktop → Preferences → Resources → File Sharing

# Default shared directories:
/Users
/Volumes
/private
/tmp

# Add custom directories if needed
```

Installing via Homebrew (Alternative)

```
# Install Homebrew if not installed
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.s

# Install Docker using Homebrew Cask
brew install --cask docker

# Start Docker Desktop
open /Applications/Docker.app

# Verify installation
docker --version
```

Common macOS Issues and Fixes

Issue 1: "Docker Desktop is not running"

```
# Solution 1: Start Docker Desktop
open /Applications/Docker.app

# Solution 2: Restart Docker Desktop
killall Docker && open /Applications/Docker.app

# Solution 3: Check if daemon is running
docker info
```

Issue 2: Slow Performance

```
# Solution: Increase resources
# Docker Desktop → Preferences → Resources
# Increase CPUs to 4-6
# Increase Memory to 6-8 GB

# Use named volumes instead of bind mounts for better performance
docker volume create mydata
docker run -v mydata:/app myimage
```

Issue 3: Port Already in Use

```
# Check what's using the port
lsof -i :8080

# Kill the process
kill -9 <PID>

# Or use a different port
docker run -p 8081:80 nginx
```

Docker Installation on Linux

Ubuntu/Debian Installation

Method 1: Using the Repository (Recommended)

```
# Step 1: Update package index
sudo apt-get update

# Step 2: Install prerequisites
sudo apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Step 3: Add Docker's official GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/key

# Step 4: Set up the repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://do
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
# Step 5: Install Docker Engine
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-co

# Step 6: Verify installation
sudo docker run hello-world
```

Method 2: Using Convenience Script

```
# Download and run the convenience script
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

# Start Docker service
sudo systemctl start docker
sudo systemctl enable docker

# Verify installation
sudo docker run hello-world
```

CentOS/RHEL Installation

```
# Step 1: Remove old versions
sudo yum remove docker \
    docker-client \
    docker-client-latest \
    docker-common \
    docker-latest \
    docker-latest-logrotate \
    docker-logrotate \
    docker-engine

# Step 2: Install yum-utils
sudo yum install -y yum-utils

# Step 3: Set up repository
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

# Step 4: Install Docker Engine
sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-p

# Step 5: Start Docker
sudo systemctl start docker
```

```
sudo systemctl enable docker

# Step 6: Verify installation
sudo docker run hello-world
```

Fedora Installation

```
# Step 1: Set up repository
sudo dnf -y install dnf-plugins-core
sudo dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo

# Step 2: Install Docker Engine
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-p

# Step 3: Start Docker
sudo systemctl start docker
sudo systemctl enable docker

# Step 4: Verify installation
sudo docker run hello-world
```

Post-Installation Steps for Linux

Run Docker as Non-Root User

```
# Create docker group
sudo groupadd docker

# Add your user to docker group
sudo usermod -aG docker $USER

# Apply group changes (or logout/login)
newgrp docker

# Verify you can run docker without sudo
docker run hello-world

# If permission issues persist:
sudo chmod 666 /var/run/docker.sock
```

Configure Docker to Start on Boot

```
# Enable Docker service
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

```
# Check status
sudo systemctl status docker

# Output should show:
#   docker.service - Docker Application Container Engine
#      Loaded: loaded (/lib/systemd/system/docker.service; enabled)
#      Active: active (running)
```

Configure Docker Daemon

```
# Create or edit daemon.json
sudo nano /etc/docker/daemon.json

# Add configuration:
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "default-address-pools": [
    {
      "base": "172.17.0.0/16",
      "size": 24
    }
  ]
}

# Restart Docker
sudo systemctl restart docker

# Verify configuration
docker info
```

Linux-Specific Commands

```
# Check Docker service status
sudo systemctl status docker

# Start Docker service
sudo systemctl start docker

# Stop Docker service
sudo systemctl stop docker
```

```
# Restart Docker service
sudo systemctl restart docker

# View Docker logs
sudo journalctl -u docker.service

# View recent Docker logs
sudo journalctl -u docker.service -n 50 --no-pager

# Enable debug logging
sudo nano /etc/docker/daemon.json
# Add: "debug": true
sudo systemctl restart docker
```

Common Linux Issues and Fixes

Issue 1: Permission Denied

```
# Error: permission denied while trying to connect to Docker daemon socket

# Solution:
sudo usermod -aG docker $USER
newgrp docker

# Verify:
docker ps
```

Issue 2: Docker Service Won't Start

```
# Check service status
sudo systemctl status docker

# Check logs
sudo journalctl -u docker.service --no-pager | tail -n 50

# Common fix: Remove old Docker installations
sudo apt-get purge docker-ce docker-ce-cli
sudo rm -rf /var/lib/docker
sudo apt-get install docker-ce docker-ce-cli
```

Issue 3: Storage Driver Issues

```
# Check current storage driver
docker info | grep "Storage Driver"

# Change to overlay2 (recommended)
sudo nano /etc/docker/daemon.json
```

```
{
  "storage-driver": "overlay2"
}
```

```
sudo systemctl restart docker
```

Docker Desktop

What is Docker Desktop?

Docker Desktop is an all-in-one application for Mac, Windows, and Linux that includes: - Docker Engine - Docker CLI - Docker Compose - Kubernetes (optional) - Docker Content Trust - Credential helper - Graphical user interface

Docker Desktop Features

1. Dashboard

```
Visual interface showing:
- Running containers
- Container logs
- Resource usage
- Quick actions (start, stop, delete)
```

2. Dev Environments

```
Quick setup for development:
- Clone repo
- Automatic container setup
- Share with team
```

3. Extensions

```
Extend Docker Desktop functionality:
- Database management (MySQL, PostgreSQL)
- Log viewers
- Security scanners
- Disk usage analyzers
```

4. Volumes

```
Visual management of:
- Volume list
- Volume size
- Volume data
- Quick backup/restore
```

5. Images

```
Manage images:
- Pull from registries
- View image layers
- Scan for vulnerabilities
- Push to registries
```

Docker Desktop Settings

```
General:
    Start Docker Desktop when you log in
    Include VM in Time Machine backups (Mac only)
    Use WSL 2 based engine (Windows only)

Resources:
  CPUs: 4
  Memory: 4.00 GB
  Swap: 1 GB
  Disk image size: 60 GB

Docker Engine:
  {
    "builder": {
      "gc": {
        "defaultKeepStorage": "20GB",
        "enabled": true
      }
    },
    "debug": false,
    "experimental": false,
    "features": {
      "buildkit": true
    }
  }

Kubernetes:
    Enable Kubernetes
  Show system containers (advanced)
```

Docker Desktop CLI

```
# Docker Desktop CLI commands (Mac/Windows)

# Open Docker Desktop
open -a Docker  # Mac
start docker    # Windows
```

```
# Check if Docker Desktop is running
docker info

# Restart Docker Desktop
# Mac: Quit and reopen
# Windows: Right-click tray icon → Restart
```

Post-Installation Setup

1. Configure Docker to Use Mirror (Optional)

```
# For faster image pulls in certain regions

# Edit daemon.json
sudo nano /etc/docker/daemon.json

# Add mirror:
{
  "registry-mirrors": ["https://mirror.gcr.io"]
}

# Restart Docker
sudo systemctl restart docker  # Linux
# Or restart Docker Desktop (Mac/Windows)
```

2. Set Resource Limits

```
// /etc/docker/daemon.json (Linux)
// Docker Desktop → Settings → Resources (Mac/Windows)
{
  "default-ulimits": {
    "nofile": {
      "Name": "nofile",
      "Hard": 64000,
      "Soft": 64000
    }
  },
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

3. Enable BuildKit

```
# BuildKit provides improved build performance

# Method 1: Environment variable
export DOCKER_BUILDKIT=1

# Make permanent (add to ~/.bashrc or ~/.zshrc)
echo 'export DOCKER_BUILDKIT=1' >> ~/.bashrc

# Method 2: Daemon configuration
{
  "features": {
    "buildkit": true
  }
}
```

4. Configure Network Settings

```
{
  "bip": "172.17.0.1/16",
  "default-address-pools": [
    {
      "base": "172.80.0.0/16",
      "size": 24
    }
  ],
  "dns": ["8.8.8.8", "8.8.4.4"]
}
```

Verification and Testing

Basic Verification

```
# Check Docker version
docker --version
# Output: Docker version 24.0.6, build ed223bc

# Check Docker Compose version
docker compose version
# Output: Docker Compose version v2.21.0

# Check detailed Docker information
docker info
# Shows:
```

```
# - Server Version
# - Storage Driver
# - Logging Driver
# - Cgroup Driver
# - Plugins
# - Registry
# - etc.

# Check system-wide information
docker system info
```

Run Test Containers

```
# Test 1: Hello World
docker run hello-world
# Should output: Hello from Docker!

# Test 2: Ubuntu Interactive
docker run -it ubuntu bash
# Opens bash shell in Ubuntu container
# Type: cat /etc/os-release
# Type: exit

# Test 3: Nginx Web Server
docker run -d -p 8080:80 --name testnginx nginx
# Open browser: http://localhost:8080
# Should see "Welcome to nginx!"

# Clean up test container
docker stop testnginx
docker rm testnginx

# Test 4: Multi-container with Docker Compose
cat > docker-compose-test.yml << EOF
version: '3.8'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  db:
    image: postgres:14
    environment:
      POSTGRES_PASSWORD: secret
EOF
```

```
docker compose -f docker-compose-test.yml up -d
# Check status
docker compose -f docker-compose-test.yml ps

# Clean up
docker compose -f docker-compose-test.yml down
rm docker-compose-test.yml
```

Performance Test

```
# Test Docker performance
time docker run --rm alpine echo "Hello"
# Should complete in < 2 seconds

# Test image pull speed
time docker pull nginx
# Note the download time

# Test build speed
cat > Dockerfile.test << EOF
FROM alpine
RUN echo "Test build"
EOF

time docker build -t test-build -f Dockerfile.test .
rm Dockerfile.test
docker rmi test-build
```

Troubleshooting Commands

```
# Check Docker service status
docker info

# Check running containers
docker ps

# Check all containers (including stopped)
docker ps -a

# Check Docker logs
docker logs <container_name>

# Check Docker events
docker events

# Check disk usage
```

```
docker system df

# Clean up unused resources
docker system prune -a
```

Key Takeaways

1. **Windows**: Use Docker Desktop with WSL 2 for best performance
2. **macOS**: Docker Desktop is the recommended solution
3. **Linux**: Install Docker Engine directly for production environments
4. **Verification**: Always test installation with `docker run hello-world`
5. **Post-Setup**: Configure daemon.json for optimal performance
6. **Resources**: Allocate appropriate CPU and memory based on your workload

---

What's Next?

Now that Docker is installed, you're ready to: - Learn Docker CLI commands - Build your first Dockerfile - Run and manage containers - Work with Docker images - Use Docker Compose for multi-container applications

Happy Dockering!