WSL Windows Setup Guide for ODRAS Development

Overview

This guide provides step-by-step instructions for setting up a complete ODRAS development environment on Windows using WSL (Windows Subsystem for Linux) **without requiring administrator privileges**. This setup enables you to run ODRAS, use Docker, and work with Cursor IDE entirely within WSL.

⚠ **CMMC Compliance Note**: If you're in a CMMC-compliant environment that restricts WSL2, see the CMMC Compliance and WSL1 Constraint section for alternative approaches using Docker Desktop for Windows.

Table of Contents

Prerequisites

What You Need

- **Windows 10 (version 2004 or later)** or **Windows 11**

- **Internet connection** for downloading components
- **User account** with standard (non-admin) privileges
- **Approximately 10GB free disk space** for WSL, Docker, and ODRAS

What You DON'T Need

- ? Administrator privileges
- ? Windows Store access (though it helps)
- ? Special IT permissions

WSL Installation (No Admin Required)

Method 1: Using Windows Store (Recommended - Easiest)

If you have access to the Microsoft Store:

1. **Open Microsoft Store** (search "Microsoft Store" in Start menu)
2. **Search for "Ubuntu"** and install one of these:
    - Ubuntu 22.04 LTS (recommended)
    - Ubuntu 20.04 LTS
    - Ubuntu 24.04 LTS
3. **Click "Install"** - This will automatically install WSL2 if needed
4. **Launch Ubuntu** from Start menu after installation
5. **Set up your Linux username and password** when prompted
    - This is your Linux user account (can be different from Windows username)
    - Remember this password - you'll need it for `sudo` commands

Method 2: Manual Installation (If Store Not Available)

If you don't have Store access, you can install WSL manually:

1. **Download WSL2 Update Package**:
    - Go to: https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi
    - Download and run the installer (should work without admin if your IT allows)
2. **Install Linux Distribution**:

```
# Open PowerShell (not as admin)
wsl --install -d Ubuntu-22.04
```

3. **If the above doesn't work**, try:

```
wsl --install
wsl --list --online
wsl --install -d Ubuntu-22.04
```

Verify WSL Installation

Open PowerShell (not as admin) and run:

```
wsl --list --verbose
```

You should see:

```
  NAME            STATE           VERSION
* Ubuntu-22.04    Running         2
```

If VERSION shows "1", you need to convert to WSL2 (see troubleshooting).

WSL Configuration

Step 1: Launch WSL and Update System

1. **Open Ubuntu** from Start menu (or type `wsl` in PowerShell)

2. **Update package lists**:

```
sudo apt update
```

3. **Upgrade packages** (optional but recommended):

```
sudo apt upgrade -y
```

Step 2: Install Essential Tools

```
# Install basic development tools
sudo apt install -y \
    git \
    curl \
    wget \
    build-essential \
    ca-certificates \
    gnupg \
    lsb-release \
    software-properties-common
```

Step 3: Configure Git (Important for ODRAS)

```
# Set your Git identity
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# Verify
git config --list
```

Step 4: Set Up Your Working Directory

```
# Create a working directory (you can use any location)
mkdir -p ~/working
cd ~/working

# Verify you're in the right place
pwd
# Should show: /home/yourusername/working
```

Docker Installation in WSL

Important Notes

- **WSL2 is REQUIRED for Docker Engine** - Docker Engine cannot run directly in WSL1
- Docker Desktop for Windows is NOT required
- We'll install Docker Engine directly in WSL
- This works without admin privileges on Windows

WSL1 vs WSL2 for Docker

WSL2 (Required for Docker Engine):

- ? Full Linux kernel - required for Docker
- ? Native Docker Engine support
- ? Better performance
- ? Full container networking support

WSL1 (NOT supported for Docker Engine):

- ? No full Linux kernel - Docker Engine won't work
- ⚠ Docker Desktop for Windows can use WSL1 backend, but:
    - Limited functionality
    - Performance issues

    - May not work with ODRAS's docker-compose setup
    - Not recommended

If you're stuck with WSL1, you have these options:

1. **Get WSL2 working** (preferred) - see troubleshooting section for `0xc03a0014` error
2. **Use Docker Desktop for Windows** with WSL1 backend (may have limitations)
3. **Ask IT to enable Virtual Machine Platform** to get WSL2 working

CMMC Compliance and WSL1 Constraint

⚠ **Important for CMMC Environments**:

- Some CMMC-compliant environments restrict WSL2 (requires Virtual Machine Platform)
- WSL1 may be the only allowed option for compliance reasons
- **Docker Engine cannot run in WSL1** - requires alternative approach

Solutions for CMMC/WSL1 Environments:

Option 1: Docker Desktop for Windows (Recommended for WSL1)

1. **Install Docker Desktop for Windows**:
    - Download from: https://www.docker.com/products/docker-desktop
    - Install on Windows (may require IT approval in CMMC environments)
2. **Configure Docker Desktop for WSL1**:
    - Open Docker Desktop Settings
    - Go to "Resources" → "WSL Integration"
    - Enable integration with your WSL1 distribution
    - Apply & Restart
3. **Verify Docker in WSL1**:

```
# In WSL1, Docker should now be available
docker --version
docker ps
```

4. **Use ODRAS with Docker Desktop**:

```
cd ~/working/ODRAS
# Docker Desktop will handle docker-compose
docker compose up -d
```

Note: Docker Desktop with WSL1 backend should work with ODRAS, but you may encounter:

- Slower performance than WSL2
- Some networking limitations
- File system performance differences

Option 2: Remote Docker Host

- Run Docker on a remote server/VM that meets compliance requirements
- Configure Docker client in WSL1 to connect to remote Docker host
- Set `DOCKER_HOST` environment variable

Option 3: Native Windows Services (If Available)

- Run ODRAS services natively on Windows if supported
- May require significant configuration changes
- Not currently supported by ODRAS

Option 4: Request WSL2 Exception

- Work with IT/security to get WSL2 approved
- WSL2 can be configured for compliance with proper controls
- Document security controls and get approval

Step 1: Install Docker Engine

```
# 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/k

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

# Update package index
sudo apt update

# Install Docker Engine, CLI, and Containerd
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-comp
```

Step 2: Start Docker Service

```
# Start Docker service
sudo service docker start

# Verify Docker is running
sudo docker ps
# Should show empty list (no containers running yet)
```

Step 3: Add Your User to Docker Group (Avoid `sudo` for Docker)

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

# Apply the group change (you'll need to log out and back in)
# For now, you can use 'newgrp docker' to apply immediately
newgrp docker

# Verify you can run Docker without sudo
docker ps
```

Note: After closing and reopening WSL, the group change will be permanent. Until then, you may need to use `sudo docker` or run `newgrp docker`.

Step 4: Configure Docker to Start Automatically

Create a startup script:

```
# Create a script to start Docker on WSL startup
cat > ~/.bashrc.d/docker-start.sh << 'EOF'
#!/bin/bash
# Start Docker if not running
if ! pgrep -x "dockerd" > /dev/null; then
    sudo service docker start > /dev/null 2>&1
fi
EOF

chmod +x ~/.bashrc.d/docker-start.sh

# Add to ~/.bashrc
echo 'source ~/.bashrc.d/docker-start.sh' >> ~/.bashrc
```

Step 5: Verify Docker Installation

```
# Test Docker installation
docker run hello-world

# Check Docker version
docker --version
docker compose version
```

You should see:

```
Hello from Docker!
...
Docker version 24.x.x
Docker Compose version v2.x.x
```

Cursor Installation in WSL

Method 1: Download and Install Cursor (Recommended)

1. **Download Cursor for Linux**:

   ```
   cd ~/Downloads || mkdir -p ~/Downloads && cd ~/Downloads

   # Download Cursor (adjust version if needed)
   wget https://downloader.cursor.sh/linux/appImage/x64 -O cursor.AppImage

   # Or download .deb package if available
   # wget https://downloader.cursor.sh/linux/deb/x64 -O cursor.deb
   ```

2. **Make it executable**:

   ```
   chmod +x cursor.AppImage
   ```

3. **Create a launcher script**:

```
# Create a bin directory if it doesn't exist
mkdir -p ~/bin

# Create launcher script
cat > ~/bin/cursor << 'EOF'
#!/bin/bash
cd ~/Downloads
./cursor.AppImage "$@"
EOF

chmod +x ~/bin/cursor

# Add to PATH (if not already there)
echo 'export PATH="$HOME/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

4. **Launch Cursor**:

```
cursor
```

Method 2: Install via Snap (If Available)

```
# Install snapd if not installed
sudo apt install -y snapd

# Install Cursor
sudo snap install cursor --classic
```

Method 3: Use Windows Cursor with WSL Integration

If you prefer to use Cursor installed on Windows:

1. **Install Cursor on Windows** (download from cursor.sh)
2. **Open Cursor on Windows**
3. **Open WSL folder**:

- File → Open Folder
- Click the folder icon in the path bar
- Select "\wsl$\Ubuntu-22.04\home\yourusername\working"

4. **Install WSL Extension** (if prompted):
    - Cursor will automatically detect WSL and offer to install the Remote-WSL extension

Verify Cursor Installation

```
# Check if cursor command works
cursor --version

# Or if using AppImage
~/Downloads/cursor.AppImage --version
```

ODRAS Setup

Step 1: Clone ODRAS Repository

```
# Navigate to your working directory
cd ~/working

# Clone ODRAS
git clone https://github.com/laserpointlabs/ODRAS.git
cd ODRAS

# Verify you're in the right place
pwd
ls -la
```

Step 2: Run ODRAS Installation Script

```
# Make install script executable
chmod +x install.sh

# Run installation (this will set up Python, dependencies, etc.)
./install.sh
```

Note: The installation script will:

- Install Python dependencies
- Set up virtual environment
- Configure system requirements
- This may take 5-10 minutes

Step 3: Initialize ODRAS Database

```
# Clean any existing data (optional, for fresh start)
./odras.sh clean -y

# Initialize database
./odras.sh init-db
```

Step 4: Start ODRAS Services

```
# Start all services (Docker containers + ODRAS API)
./odras.sh start

# Wait 30-60 seconds for services to start
# Check status
./odras.sh status
```

Step 5: Verify ODRAS is Running

```
# Check logs
./odras.sh logs

# Check if API is responding
curl http://localhost:8000/api/health

# Or open in browser (from Windows)
# http://localhost:8000
```

Step 6: Open ODRAS in Browser

1. **Open your Windows browser** (Chrome, Edge, Firefox)
2. **Navigate to**: `http://localhost:8000`
3. **Login with**:
    - Username: `admin`
    - Password: `admin`
4. **Or use test account**:
    - Username: `das_service`
    - Password: `das_service_2024!`

Verification and Testing

Complete System Check

Run these commands to verify everything is working:

```
# 1. Check WSL version
wsl --list --verbose

# 2. Check Docker
docker --version
docker ps

# 3. Check Docker Compose
docker compose version

# 4. Check Cursor (if installed in WSL)
cursor --version

# 5. Check ODRAS
cd ~/working/ODRAS
./odras.sh status

# 6. Check ODRAS API
curl http://localhost:8000/api/health

# 7. Check all Docker containers
docker ps -a
```

Expected Output

You should see:

- ? WSL2 running Ubuntu
- ? Docker version 24.x or later
- ? Docker Compose version 2.x
- ? Cursor version (if installed)
- ? ODRAS services running
- ? API responding with health status
- ? Multiple Docker containers running (postgres, neo4j, qdrant, etc.)

Troubleshooting

WSL Issues

Problem: WSL won't install

Solution:

```
# Check if WSL is enabled
wsl --status

# If not, try enabling it
wsl --install

# If that fails, you may need to enable Windows features (might require admin)
# Ask your IT department to enable:
# - Windows Subsystem for Linux
# - Virtual Machine Platform
```

Problem: WSL is version 1, need version 2

Solution:

```
# Convert to WSL2
wsl --set-version Ubuntu-22.04 2

# Set WSL2 as default
wsl --set-default-version 2
```

Problem: "WSL 2 requires an update to its kernel component"

Solution:

1. Download WSL2 kernel update: https://aka.ms/wsl2kernel
2. Run the installer (should work without admin)
3. Restart WSL: `wsl --shutdown` then reopen

Problem: "WslRegisterDistribution failed with error: 0xc03a0014 - A virtual disk support provider for the specified file was not found"

Solution:

This error means WSL2 virtual disk support is missing. Try these steps in order:

Step 1: Enable Virtual Machine Platform (May require admin)

```
# From PowerShell (may need admin)
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

If you don't have admin, ask IT to enable "Virtual Machine Platform" Windows feature.

Step 2: Install WSL2 Kernel Update

1. Download: https://aka.ms/wsl2kernel
2. Run the installer (should work without admin)
3. Restart computer if prompted

Step 3: Set WSL2 as Default

```
wsl --set-default-version 2
```

Step 4: Try Installing Again

```
wsl --install -d Ubuntu-22.04
```

Step 5: If Still Failing - Check Windows Version

```
# Check Windows version
winver
```

WSL2 requires:

- Windows 10 version 1903 or higher (with KB4566116)

- Windows 10 version 2004 or higher (recommended)
- Windows 11 (recommended)

Step 6: Alternative - Use WSL1 First, Then Upgrade

```
# Install with WSL1
wsl --install -d Ubuntu-22.04 --version 1

# After installation, convert to WSL2
wsl --set-version Ubuntu-22.04 2
```

⚠ **Important**: If you install WSL1, you **cannot run Docker Engine directly**. You must upgrade to WSL2 before installing Docker. ODRAS requires WSL2 for Docker to work.

Step 7: If All Else Fails

- Ask IT to enable "Virtual Machine Platform" and "Windows Subsystem for Linux" features
- Ensure Windows is fully updated
- Check if virtualization is enabled in BIOS (if you have access)

Step 8: CMMC Compliance Restriction
If you're in a CMMC-compliant environment that restricts WSL2:

- WSL1 may be the only option allowed
- See "CMMC Compliance and WSL1 Constraint" section above for alternatives
- Docker Desktop for Windows with WSL1 backend is the recommended workaround
- Contact IT/security team for approved Docker solution

Docker Issues

Problem: "Docker requires WSL2" or Docker won't start in WSL1

Solution:

```
# Check WSL version
wsl --list --verbose

# If showing VERSION 1, convert to WSL2
wsl --set-version Ubuntu-22.04 2

# Set WSL2 as default
wsl --set-default-version 2

# Restart WSL
wsl --shutdown
```

Note: Docker Engine **cannot run in WSL1**. You must use WSL2. If you can't get WSL2 working, ask IT to enable "Virtual Machine Platform" Windows feature.

Problem: "Cannot connect to Docker daemon"

Solution:

```
# Start Docker service
sudo service docker start

# Check if Docker is running
sudo service docker status

# If still not working, check Docker group
groups | grep docker
# If docker is not in the list:
sudo usermod -aG docker $USER
newgrp docker
```

Problem: "Permission denied" when running docker commands

Solution:

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

# Apply immediately
newgrp docker

# Or use sudo temporarily
sudo docker ps
```

Problem: Docker containers won't start

Solution:

```
# Check Docker daemon logs
sudo journalctl -u docker

# Restart Docker
sudo service docker restart

# Check available disk space
df -h

# Check Docker system info
docker system info
```

Cursor Issues

Problem: Cursor won't launch in WSL

Solution:

```
# If using AppImage, make sure it's executable
chmod +x ~/Downloads/cursor.AppImage

# Try running directly
~/Downloads/cursor.AppImage

# Check for missing libraries
ldd ~/Downloads/cursor.AppImage

# Install missing dependencies if needed
sudo apt install -y libfuse2
```

Problem: Cursor can't access WSL files from Windows

Solution:

- Use Windows Cursor with WSL integration (Method 3 above)
- Or access WSL files via `\\wsl$\Ubuntu-22.04\` in Windows Explorer

ODRAS Issues

Problem: "Port already in use"

Solution:

```
# Check what's using port 8000
sudo lsof -i :8000

# Stop ODRAS
./odras.sh stop

# Clean and restart
./odras.sh clean -y
./odras.sh init-db
./odras.sh start
```

Problem: "Cannot connect to database"

Solution:

```
# Check if Docker containers are running
docker ps

# Check PostgreSQL container
docker logs odras-postgres-1

# Restart services
./odras.sh stop
./odras.sh start

# Wait 60 seconds for services to initialize
sleep 60
./odras.sh status
```

Problem: ODRAS won't start

Solution:

```
# Check logs
./odras.sh logs

# Check Docker containers
docker ps -a

# Clean and reinitialize
./odras.sh clean -y
./odras.sh init-db
./odras.sh start

# Check system resources
free -h
df -h
```

Problem: "Permission denied" on scripts

Solution:

```
# Make scripts executable
chmod +x *.sh
chmod +x scripts/*.sh

# Verify
ls -la *.sh
```

Network Issues

Problem: Can't access localhost:8000 from Windows browser

Solution:

```
# Check if ODRAS is listening
netstat -tuln | grep 8000

# Check WSL IP address
hostname -I

# Try accessing via WSL IP from Windows
# e.g., http://172.x.x.x:8000
```

Problem: WSL IP changes on restart

Solution:

- This is normal - WSL gets a new IP each time
- Use `localhost` from Windows - it should work automatically
- If not, check Windows firewall settings

Performance Issues

Problem: WSL is slow

Solution:

```
# Check WSL version (should be 2)
wsl --list --verbose

# If version 1, convert to version 2
wsl --set-version Ubuntu-22.04 2

# Check available resources
free -h
df -h
```

Problem: Docker is slow

Solution:

```
# Check Docker resources
docker system df

# Clean up unused resources
docker system prune -a

# Check if WSL2 has enough memory allocated
# Edit: C:\Users\YourName\.wslconfig
# Add:
# [wsl2]
# memory=4GB
# processors=2
```

Quick Reference Commands

WSL Commands

```
# Exit WSL
exit

# Shutdown WSL
wsl --shutdown

# List distributions
wsl --list --verbose

# Open WSL from Windows
wsl
```

Docker Commands

```
# Start Docker
sudo service docker start

# Stop Docker
sudo service docker stop

# Check Docker status
sudo service docker status

# View running containers
docker ps

# View all containers
docker ps -a

# View logs
docker logs <container-name>

# Stop all containers
docker stop $(docker ps -q)
```

ODRAS Commands

```
# Start ODRAS
./odras.sh start

# Stop ODRAS
./odras.sh stop

# Check status
./odras.sh status

# View logs
./odras.sh logs

# Clean and reset
./odras.sh clean -y
./odras.sh init-db

# Restart services
./odras.sh restart
```

Cursor Commands

```
# Launch Cursor (if installed in WSL)
cursor

# Or if using AppImage
~/Downloads/cursor.AppImage

# Open specific folder
cursor ~/working/ODRAS
```

Next Steps

Once everything is set up:

1. **Explore ODRAS**: Open `http://localhost:8000` in your browser
2. **Read Documentation**: Check `docs/` folder for detailed guides
3. **Start Simple Tasks**: Begin with small bug fixes or documentation updates
4. **Join the Team**: Ask for access to the repository and start contributing!

Recommended Reading

- README.md - Overview of ODRAS
- Development Guide - How to develop for ODRAS
- Testing Guide - How to test changes
- Git Workflow - How to contribute code

Getting Help

If you encounter issues not covered here:

1. **Check Logs**: `./odras.sh logs` or `docker logs <container>`
2. **Search Documentation**: Look in `docs/` folder
3. **Ask the Team**: Reach out to your team lead or colleagues
4. **GitHub Issues**: Check existing issues or create a new one

---

Summary

You now have:

- ? WSL2 with Ubuntu running
- ? Docker Engine installed and configured
- ? Cursor IDE ready to use
- ? ODRAS cloned and running
- ? Development environment fully functional

Welcome to the ODRAS development team! ?