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 WSL2 (Windows Subsystem for Linux). This setup enables you to run ODRAS, use Docker, and work with Cursor IDE entirely within WSL2.

Table of Contents

Prerequisites

What You Need

- **Windows 10 (version 2004 or later)** or **Windows 11**
- **Internet connection** for downloading components
- **Administrator privileges** - Required for enabling virtualization and Windows features
- **Approximately 10GB free disk space** for WSL, Docker, and ODRAS

What You DON'T Need

- × Windows Store access (though it helps)
- × Special IT permissions (beyond admin access)

⚠ **IMPORTANT**: Administrator privileges are **REQUIRED** to enable virtualization features in Windows. If you don't have admin access, you must contact your IT department to:
1. Enable virtualization in BIOS/UEFI (if disabled)
2. Enable Windows Virtualization Platform and WSL features
3. Install WSL2 kernel update if needed

Enable Virtualization

⚠ **IMPORTANT**: Before installing WSL2, you must ensure virtualization is enabled. This is required for WSL2 to function properly.

⚠ **ADMINISTRATOR/IT ASSISTANCE REQUIRED**:
- **BIOS/UEFI changes**: On government-compliant or enterprise-managed machines, BIOS access is restricted and requires IT administrator assistance
- **Windows features**: Enabling Windows virtualization features requires administrator privileges

- **If you don't have admin access**: Contact your IT department before proceeding - they must enable virtualization in BIOS and Windows features for you

Step 1: Check if Virtualization is Enabled

1. **Open Task Manager** (Press `Ctrl + Shift + Esc`)
2. **Go to the "Performance" tab**
3. **Select "CPU"** from the left sidebar
4. **Look for "Virtualization"** at the bottom
5. If it shows **"Enabled"** → You're good to go! Skip to WSL Installation
6. If it shows **"Disabled"** → Continue to Step 2

Step 2: Enable Virtualization in BIOS/UEFI

If virtualization is disabled, you need to enable it in your system's BIOS/UEFI settings:

⚠ **ADMINISTRATOR/IT ASSISTANCE REQUIRED**: On government-compliant or enterprise-managed machines, BIOS/UEFI access is typically restricted and requires IT administrator assistance. You **cannot** enable virtualization in BIOS without admin privileges on these systems.

If you're on a government-compliant or enterprise-managed machine:
- **Contact your IT department immediately** - they must enable virtualization in BIOS for you
- Provide them with this document and specify you need "Virtualization Technology" enabled
- IT will need to:
1. Access BIOS/UEFI settings (may require physical access or remote management tools)
2. Enable "Virtualization Technology" (Intel VT-x) or "AMD-V" or "SVM Mode"
3. Save changes and restart the system

If you have full admin access to your machine:

1. **Restart your computer**
2. **Enter BIOS/UEFI Setup**:

3. **Dell/HP/Lenovo**: Press `F2` or `F12` during boot
4. **ASUS**: Press `F2` or `Delete` during boot
5. **Other brands**: Check your manufacturer's documentation
6. **Windows 11**: Settings → System → Recovery → Advanced startup → Restart now → Troubleshoot → Advanced options → UEFI Firmware Settings
7. **Find Virtualization Settings**:
8. Look for **"Virtualization Technology"**, **"Intel VT-x"**, **"AMD-V"**, or **"SVM Mode"**
9. Common locations:
    - **Advanced → CPU Configuration → Virtualization Technology**
    - **Advanced → Processor Configuration → Intel Virtualization Technology**
    - **Security → Virtualization**
10. **Enable Virtualization**:
11. Set the option to **"Enabled"**
12. Save and exit (usually `F10`)
13. **Restart your computer**

Step 3: Enable Windows Virtualization Features

After enabling virtualization in BIOS, enable the required Windows features:

⚠ **ADMINISTRATOR PRIVILEGES REQUIRED**: This step requires administrator privileges. You must run PowerShell as Administrator.

1. **Open PowerShell as Administrator** (REQUIRED):
2. Right-click Start menu → **Windows PowerShell (Admin)** or **Terminal (Admin)**
3. **You must select "Run as Administrator"** - this is mandatory
4. If you don't have admin access, **contact your IT department** - they must run these commands for you

5. **Enable required features** (requires admin PowerShell):

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

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

1. **Restart your computer** (if prompted)

2. **Verify features are enabled**:

```
# Check if features are enabled
   Get-WindowsOptionalFeature -Online | Where-Object {$_.FeatureName -like
   "*VirtualMachine*" -or $_.FeatureName -like "*Subsystem*"}
```

Step 4: Verify Virtualization is Enabled

After restarting, verify virtualization is enabled:

1. **Open Task Manager** → **Performance** → **CPU**
2. **Confirm "Virtualization" shows "Enabled"**

WSL Installation

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:

3. Ubuntu 22.04 LTS (recommended)

4. Ubuntu 20.04 LTS
5. Ubuntu 24.04 LTS
6. **Click "Install"** - This will automatically install WSL2 if needed
7. **Launch Ubuntu** from Start menu after installation
8. **Set up your Linux username and password** when prompted
9. This is your Linux user account (can be different from Windows username)
10. 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**:
2. Go to: https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi
3. Download and run the installer (should work without admin if your IT allows)
4. **Install Linux Distribution**:
   ```
   # Open PowerShell (not as admin)
      wsl --install -d Ubuntu-22.04
   ```
5. **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
```

⚠ **Important**: If VERSION shows "1", you need to convert to WSL2. See the Troubleshooting section for instructions.

Configure WSL (.wslconfig)

⚠ **CRITICAL**: This must be done **immediately after WSL installation** to ensure proper internet access and resource allocation. This configuration enables mirrored networking mode which is essential for internet connectivity.

Step 1: Create .wslconfig File

1. **Open Windows File Explorer**
2. **Navigate to your user profile directory**:
3. Press `Win + R`
4. Type: `%USERPROFILE%`
5. Press Enter
6. This opens: `C:\Users\YourUsername\`
7. **Create the .wslconfig file**:
8. Right-click in the folder → **New** → **Text Document**
9. Name it exactly: `.wslconfig` (including the leading dot)
10. If Windows warns about the file extension, click "Yes"
11. If you can't create a file starting with a dot, use PowerShell:

```
# Open PowerShell (not as admin)
   cd $env:USERPROFILE
   New-Item -Path .wslconfig -ItemType File
```

Step 2: Configure .wslconfig

1. **Open .wslconfig** with Notepad or any text editor
2. **Add the following configuration**:

```
[wsl2]
    networkingMode=mirrored
    memory=16384MB
    processors=16
```

1. **Save the file** (Ctrl + S)
2. **Close the file**

Step 3: Apply Configuration

1. **Shutdown WSL** to apply the new configuration:
   ```
   wsl --shutdown
   ```
2. **Wait 10-15 seconds** for WSL to fully shutdown
3. **Restart WSL** by opening Ubuntu from the Start menu or running:
   ```
   wsl
   ```

Step 4: Verify Configuration

1. **Test internet connectivity** in WSL:
   ```
   # In WSL (Ubuntu)
       ping -c 3 google.com
   ```

You should see successful ping responses. If not, see the Troubleshooting section.

1. **Verify resource allocation**:
   ```
   
   # Check memory (should show ~16GB available)
   free -h

# Check CPU cores (should show 16 processors)
nproc
```

Configuration Explanation

- **`networkingMode=mirrored`**: Enables mirrored networking mode, which provides:
- Full internet access from WSL
- Better network performance
- Proper DNS resolution
- Required for Docker and ODRAS services
- **`memory=16384MB`**: Allocates 16GB of RAM to WSL2 (adjust based on your system's available RAM)
- **`processors=16`**: Allocates 16 CPU cores to WSL2 (adjust based on your system's CPU cores)

Note: Adjust `memory` and `processors` values based on your system's resources. Ensure you don't allocate more than your system has available.

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** - Docker Engine requires WSL2's full Linux kernel

- **No Docker Desktop needed** - We'll install Docker Engine directly in WSL2
- **Better performance** - Native Docker Engine in WSL2 performs better than Docker Desktop
- This setup works entirely within WSL2

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/keyrings/docker.gpg

# Set up Docker repository
echo \
  "deb [arch=$(dpkg --print-architecture)
  signed-by=/etc/apt/keyrings/docker.gpg]
  https://download.docker.com/linux/ubuntu \
  $(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-compose-plugin
```

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
\`\`\`

1. **Make it executable**:
   `chmod +x cursor.AppImage`

2. **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
```

1. **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**:
4. File → Open Folder
5. Click the folder icon in the path bar
6. Select "\wsl$\Ubuntu-22.04\home\yourusername\working"
7. **Install WSL Extension** (if prompted):

8. 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**:

4. Username: `admin`
5. Password: `admin`
6. **Or use test account**:
7. Username: `das_service`
8. 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:

```
# Ensure WSL2 is set as default
wsl --set-default-version 2

# Convert existing distribution to WSL2
wsl --set-version Ubuntu-22.04 2

# Verify the conversion
wsl --list --verbose
```

Note: If conversion fails, ensure virtualization is enabled (see Enable Virtualization section).

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: If All Else Fails
- Ask IT to enable "Virtual Machine Platform" and "Windows Subsystem for Linux" features
- Ensure Windows is fully updated
- Verify virtualization is enabled in BIOS (see Enable Virtualization section)
- Check Windows version meets requirements (Windows 10 version 2004+ or Windows 11)

Docker Issues

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

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 **requires WSL2**. If you can't get WSL2 working, ensure virtualization is enabled (see Enable Virtualization section) and 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:
- With `networkingMode=mirrored` in `.wslconfig`, this should not be an issue
- Use `localhost` from Windows - it should work automatically
- If you still have issues, verify `.wslconfig` has `networkingMode=mirrored` set
- Check Windows firewall settings if problems persist

Problem: No internet access in WSL

Solution:

```
# Verify .wslconfig is set correctly
# From Windows PowerShell:
cat $env:USERPROFILE\.wslconfig

# Should show networkingMode=mirrored
# If not, update .wslconfig (see Configure WSL section)

# Restart WSL to apply changes
wsl --shutdown
# Then reopen WSL

# Test connectivity
ping -c 3 google.com
```

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

# Verify .wslconfig has adequate resources allocated
# Edit: C:\Users\YourName\.wslconfig
# Ensure it has:
# [wsl2]
# networkingMode=mirrored
# memory=16384MB  (adjust based on your system)
# processors=16   (adjust based on your system)

# Restart WSL to apply changes
wsl --shutdown
```

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!