How to Set Up Xdebug with Laravel Herd and PhpStorm on macOS

A complete, step-by-step guide to configure Xdebug for debugging Laravel applications using Herd and PhpStorm.

Table of Contents

Prerequisites

- **macOS** (Apple Silicon or Intel)
- **Laravel Herd** installed and running
- **PhpStorm** IDE
- A Laravel project served by Herd
- Basic knowledge of terminal commands

Understanding the Setup

Before diving in, it's important to understand how the components work together:

- **Xdebug**: PHP extension that enables debugging
- **Herd**: Manages PHP, Nginx, and serves your Laravel applications
- **PhpStorm**: Listens for debugging connections from Xdebug
- **Browser Extension**: Triggers Xdebug to activate for web requests

Key Concepts

1. **PHP CLI vs PHP-FPM**: Herd uses different PHP processes for command-line (CLI) and web requests (FPM)
2. **Debug Socket**: Herd has separate sockets for normal and debug modes

3. **Port Configuration**: Xdebug connects to PhpStorm on port 9003 by default

---

Step 1: Configure Xdebug in Herd

1.1 Identify Your PHP Version and Architecture

First, determine your PHP version and Mac architecture:

```
# Check PHP version
php -v

# Check architecture (arm64 = Apple Silicon, x86 = Intel)
uname -m
```

1.2 Configure the Debug Configuration File

Herd includes Xdebug extensions but requires configuration. Edit the debug configuration file:

```
# Open the debug.ini file for your PHP version
# Replace '83' with your PHP version (82, 81, etc.)
nano ~/Library/Application\ Support/Herd/config/php/83/debug/debug.ini
```

Add the following configuration:

```
# For Apple Silicon (M1/M2/M3)
zend_extension=/Applications/Herd.app/Contents/Resources/xdebug/xdebug-83-arm64.so

# For Intel Mac, use this instead:
# zend_extension=/Applications/Herd.app/Contents/Resources/xdebug/xdebug-83-x86.so

xdebug.mode=debug,develop
xdebug.start_with_request=yes
xdebug.start_upon_error=yes
xdebug.client_host=127.0.0.1
xdebug.client_port=9003
xdebug.idekey=PHPSTORM
```

Important Notes:

- Replace `83` with your PHP version (e.g., `82` for PHP 8.2)
- Use `arm64` for Apple Silicon or `x86` for Intel
- Use `127.0.0.1` instead of `localhost` to avoid DNS resolution issues

1.3 Optional: Configure the Main php.ini (for CLI debugging)

If you also want to debug CLI commands (artisan, tests):

```
# Open php.ini
nano ~/Library/Application\ Support/Herd/config/php/83/php.ini
```

Add the same Xdebug configuration as above.

1.4 Restart Herd

```
herd restart
```

After restarting, you should NOT see connection errors in the output. If you see errors, don't worry - they appear because PhpStorm isn't listening yet.

1.5 Verify Xdebug Installation

```
# Check if Xdebug is loaded
php -v
# You should see "with Xdebug v3.x.x"

# View Xdebug configuration
php --ri xdebug | grep -E "client_host|client_port|mode"
```

You should see:

```
xdebug.mode => debug,develop
xdebug.client_host => 127.0.0.1
xdebug.client_port => 9003
```

Step 2: Configure PhpStorm

2.1 Configure PHP Interpreter

1. Open **PhpStorm**
2. Go to **Settings/Preferences** (`Cmd + ,`)
3. Navigate to **PHP**
4. Next to **CLI Interpreter**, click the **three dots** (`...`)

5. Click the `+` button
6. Select **Other Local...**
7. Browse to Herd's PHP binary: (Or let PhpStorm auto-detect it)

```
/Users/[your-username]/Library/Application Support/Herd/bin/php83
```

8. Verify that PhpStorm shows:
    - PHP version: 8.3.x
    - Xdebug: 3.3.x
9. Click **OK**

2.2 Configure Server

This is crucial and often the source of problems.

1. In **Settings**, go to **PHP Servers**
2. Click `+` to add a new server
3. Configure:
    - **Name**: `your-domain.test` (e.g., `myapp.test`)
    - **Host**: `your-domain.test` (same as name)
    - **Port**: `443` (if using HTTPS) or `80` (if using HTTP)
    - **Debugger**: `Xdebug`
    - **Use path mappings**: Leave **UNCHECKED**

Critical: Port Configuration

- If your Herd site uses HTTPS (most do by default), use port **443**
- If using HTTP, use port **80**
- To check: look at your browser URL - `https://` = port 443, `http://` = port 80

1. Click **Apply**

Pro Tip: If you're unsure about the port, delete the server configuration and PhpStorm will auto-detect it on the first connection and suggest the correct port (usually 443).

2.3 Configure Debug Settings

1. Go to **PHP Debug**
2. In the **Xdebug** section, verify:
    - **Debug port**: `9003`
    - **Can accept external connections**: Can be checked
    - **Force break at first line when no path mapping specified**: **UNCHECKED**

- **Force break at first line when a script is outside the project**: **UNCHECKED**

These last two options prevent PhpStorm from stopping at Herd's internal files.

1. Click **OK**

2.4 Start Listening for Debug Connections

In the PhpStorm toolbar (top right), find the **phone icon** ( ) and click it to activate "Start Listening for PHP Debug Connections".

The icon should turn **green** when active.

Alternative: Go to **Run** **Start Listening for PHP Debug Connections**

Step 3: Testing the Setup

3.1 CLI Debugging (Quick Test)

Create a simple test file:

```
<?php
// test-xdebug.php

echo "Starting Xdebug test...\n";

$name = "Developer";  //    Place breakpoint here
$message = "Hello " . $name;

echo $message . "\n";
echo "Test finished.\n";
```

1. **Place a breakpoint** on the line `$name = "Developer";` (click in the left margin)
2. **Save the file** (`Cmd + S`)
3. **Ensure PhpStorm is listening** (green phone icon)
4. **Run from terminal**:

```
php test-xdebug.php
```

Expected Result:

- Terminal pauses
- PhpStorm comes to the foreground
- The breakpoint line is highlighted in blue
- Variables panel shows values

If this works, Xdebug is configured correctly!

3.2 Web Debugging

For debugging web requests, you need to trigger Xdebug from the browser.

Option A: Browser Extension (Recommended)

1. **Install Xdebug Helper**:
   - Chrome: Xdebug Helper
   - Firefox: Xdebug Helper
2. **Configure the extension**:
   - Click the extension icon
   - Go to **Options/Settings**
   - Set **IDE Key** to `PHPSTORM`
   - Save
3. **Activate debugging**:
   - Click the extension icon
   - Select **Debug** (icon turns green)

Option B: URL Parameter

Add this to your URL:

```
https://your-domain.test/your-route?XDEBUG_SESSION_START=PHPSTORM
```

3.3 Debug a Laravel Route

1. Open `routes/web.php`
2. Place a breakpoint inside a route:

```
Route::get('/test-debug', function () {
    $data = "Testing Xdebug";  //    Breakpoint here
    return view('welcome', compact('data'));
});
```

1. **Save the file** (`Cmd + S`)
2. **PhpStorm is listening** (green icon)
3. **Browser extension is active** (green)
4. **Visit the route**: `https://your-domain.test/test-debug`

Expected Result:

- Browser page starts loading but pauses

- PhpStorm comes to foreground
- Breakpoint line is highlighted
- You can inspect all variables: `$data`, `$request`, etc.
- Use controls: **Resume** (F9), **Step Over** (F8), **Step Into** (F7)

Common Issues and Solutions

Issue 1: "Could not connect to debugging client" Errors

Symptoms: Errors appear when running `herd restart`

Cause: Xdebug tries to connect but PhpStorm isn't listening yet

Solution: These errors during Herd startup are **normal and harmless**. They only matter if they appear when actually trying to debug.

Issue 2: CLI Debugging Works, Web Debugging Doesn't

Symptoms: Breakpoints work with `php artisan` or terminal commands, but not from browser

Root Causes:

1. **Wrong port in PhpStorm server configuration** (most common)
2. **PHP-FPM not loading Xdebug**
3. **Browser extension not configured properly**

Solutions:

A. Fix Server Port Configuration

- If using HTTPS (default for Herd), set port to **443**
- If using HTTP, set port to **80**
- To auto-detect: Delete the server in PhpStorm, then debug from browser - PhpStorm will suggest the correct configuration

B. Verify PHP-FPM Loads Xdebug

Create `public/info.php`:

```
<?php phpinfo();
```

Visit `https://your-domain.test/info.php` and search for "xdebug".

If you DON'T see an Xdebug section, PHP-FPM isn't loading it. Verify the `debug.ini` file path and restart Herd.

C. Check Browser Extension

- Verify the extension is set to `PHPSTORM` (not `YOUR-NAME` or other values)
- Check cookies: Open DevTools Application Cookies
- You should see `XDEBUG_SESSION=PHPSTORM`

Issue 3: PhpStorm Doesn't Stop at Breakpoints

Symptoms: Everything seems configured, but PhpStorm doesn't react

Solutions:

A. Verify PhpStorm is Actually Listening

```
lsof -i :9003
```

You should see PhpStorm listening on port 9003. If not, restart PhpStorm.

B. Check Debug Settings

- Settings PHP Debug
- Verify "Ignore external connections through unregistered server configurations" is **UNCHECKED**

C. Accept First Connection

- The first time Xdebug connects, PhpStorm shows a popup: "Incoming connection from Xdebug"
- Click **Accept**
- This popup might be hidden behind other windows - check!

D. Enable Xdebug Logging

Add to `debug.ini`:

```
xdebug.log=/tmp/xdebug.log
```

Restart Herd, then check the log:

```
tail -f /tmp/xdebug.log
```

Issue 4: Breakpoint Stops at Herd's Internal Files

Symptoms: Debugger stops at files like `dump-loader.php` instead of your code

Solution:

- Settings PHP Debug
- **Uncheck**: "Force break at first line when no path mapping specified"
- **Uncheck**: "Force break at first line when a script is outside the project"

Issue 5: Using `localhost` Instead of IP

Symptoms: Connection errors even though everything is configured

Solution: Always use `127.0.0.1` instead of `localhost` in the Xdebug configuration. macOS can have DNS resolution issues with `localhost`.

Tips and Best Practices

1. Debugging Artisan Commands

Use Herd's debug command:

```
herd debug artisan your:command
```

Or set environment variable:

```
XDEBUG_CONFIG="idekey=PHPSTORM" php artisan your:command
```

2. Debugging PHPUnit Tests

```
herd debug vendor/bin/phpunit
```

Or in PhpStorm, create a PHPUnit run configuration with the Herd PHP interpreter.

3. Conditional Breakpoints

Right-click a breakpoint Add condition:

```
$user->id == 123
```

PhpStorm will only stop when the condition is true.

4. Evaluate Expressions

While debugging, select any expression in your code, right-click → **Evaluate Expression** (or press `Alt + F8`).

5. Quick Debugging Toggle

Create a keyboard shortcut for "Toggle Line Breakpoint":

- Settings → Keymap → Search for "Toggle Line Breakpoint"
- Set to `Cmd + F8` for quick breakpoint toggling

6. Xdebug Performance Impact

Disable Xdebug when not needed to maintain performance:

```
# Comment out the zend_extension line in debug.ini
# Or use Herd Pro's auto-detection feature
```

With Herd Pro, Xdebug only activates when breakpoints are detected, keeping your app fast.

7. Multiple Projects

If working on multiple projects:

- Each project can have different server configurations in PhpStorm
- The server name should match the domain: `project1.test`, `project2.test`, etc.

8. Remote Debugging (Optional)

If debugging from a different machine:

In `debug.ini`:

```
xdebug.client_host=192.168.x.x  # IP of machine running PhpStorm
```

Also configure your firewall to allow connections on port 9003.

Troubleshooting Checklist

When debugging doesn't work, verify ALL of these:

Herd Configuration:

- [ ] `debug.ini` exists and contains correct configuration

- [ ] Correct architecture (arm64/x86) specified
- [ ] `xdebug.client_host=127.0.0.1` (not localhost)
- [ ] `xdebug.client_port=9003`
- [ ] Herd restarted after config changes

PhpStorm Configuration:

- [ ] PHP interpreter shows Xdebug loaded
- [ ] Server name matches domain
- [ ] Server port is correct (443 for HTTPS, 80 for HTTP)
- [ ] Path mappings are disabled (unchecked)
- [ ] Debug port is 9003
- [ ] Force break options are unchecked
- [ ] PhpStorm is listening (green phone icon)

Browser/Testing:

- [ ] Browser extension installed and configured
- [ ] Extension IDE key is "PHPSTORM"
- [ ] Extension is activated (green icon)
- [ ] Breakpoint is placed and file is saved
- [ ] Cookie `XDEBUG_SESSION=PHPSTORM` exists

Verification:

- [ ] `php -v` shows Xdebug loaded
- [ ] `lsof -i :9003` shows PhpStorm listening
- [ ] `phpinfo()` from browser shows Xdebug section
- [ ] CLI debugging works (`php test-file.php`)

Conclusion

Setting up Xdebug with Laravel Herd and PhpStorm involves configuring three main components:

1. **Xdebug** (via Herd's `debug.ini`)
2. **PhpStorm** (interpreter, server, debug settings)
3. **Trigger mechanism** (browser extension or URL parameter)

The most common pitfall is the **server port configuration** - always verify you're using the correct port (443 for HTTPS, 80 for HTTP).

Once configured correctly, you'll have a powerful debugging environment that works seamlessly for both CLI and web debugging.

Happy debugging!

Additional Resources

- Official Xdebug Documentation
- Laravel Herd Documentation
- PhpStorm Debugging Guide
- Xdebug Helper Chrome Extension

Last Updated: December 2024

Tested With: Laravel Herd 1.x, PhpStorm 2024.x, PHP 8.3, macOS Sonoma