

**add more content and code for uses and more infromation with the setup and all use the images also**

Absolutely! Let’s expand on the Drupal debugging blog by adding more detailed setup steps, code snippets for practical use, and suggestions for images or visual aids (note: as a text-based assistant, I can recommend what images you might use but I can’t generate or include direct images). Here’s a more comprehensive guide:

**Advanced Drupal Debugging in 2025: Step-by-Step Setup, Code, and Visual Guide**

Debugging Drupal sites is easier and more powerful than ever, thanks to robust toolsets and clear best practices. Here’s a deep dive on how to set up your Drupal debugging environment, code examples for common tasks, and guidance on what images/screenshots to include for maximum clarity in your blog.

**1. Setting Up Your Debugging Environment**

**A. Enable Error Reporting**

* Edit your settings.php (located in sites/default/):

// Enable all error reporting for development.  
error\_reporting(E\_ALL);  
ini\_set('display\_errors', TRUE);  
ini\_set('display\_startup\_errors', TRUE);  
// Recommended: Only for development, never production!

* **Image Suggestion:** Screenshot of the Drupal site showing error messages displayed at the top of the page.

**B. Install Devel Module**

1. **Install using Composer:**

composer require drupal/devel

1. **Enable via Drush:**

drush en devel --yes

1. Access Devel’s configuration and settings at /admin/config/development/devel.

* **Image Suggestion:** admin panel showing the Devel settings page.

**Practical Devel Code Examples**

* Print debugging info for a variable:

// Example: Inspect a node object in a custom module.  
dpm($node, 'Current node object');

This displays a readable dump directly in the Drupal UI, labeled for easy searching.

* Inspect database queries:

// After running complex entity queries, inspect with:  
dpq($query);

**C. Enabling Xdebug (With PhpStorm/VSCode)**

1. Install Xdebug (in your Docker/DDEV or local PHP setup):

pecl install xdebug

1. Update your php.ini with:

zend\_extension=xdebug.so  
xdebug.mode=debug  
xdebug.start\_with\_request=yes  
xdebug.client\_port=9003  
xdebug.client\_host=host.docker.internal

1. Set breakpoints in your IDE, then reload your Drupal page to trigger debugging.

* **Image Suggestion:** IDE screenshot showing a breakpoint and a stack trace during a request.

**2. Twig Theming Debugging & Template Discovery**

**Enable Twig Debug Mode**

Edit sites/default/services.yml:

parameters:  
 twig.config:  
 debug: true  
 auto\_reload: true  
 cache: false

Clear cache:

drush cr

Now, in your site’s HTML source, you’ll see comments like:

<!-- THEME DEBUG -->  
<!-- FILE NAME SUGGESTIONS:  
 \* node--article.html.twig  
 \* node.html.twig  
-->  
<!-- BEGIN OUTPUT from 'themes/custom/mytheme/templates/node--article.html.twig' -->

* **Image Suggestion:** Browser DevTools open to the HTML showing theme debug comments.

**Twig Variable Dump Example**

Inside any Twig template:

{{ dump() }}  
{{ dump(content) }}

This outputs the template variables right within your page (visible if Twig debug is enabled).

**3. Command-Line Debugging Workflows**

**Using Drush for Diagnostics**

* Get overall status:

drush status

* Check enabled modules:

drush pm:list --status=enabled

* Clear caches quickly:

drush cr

* **Image Suggestion:** Terminal output of drush status and drush cr commands.

**Automated Testing with PHPUnit**

* Write a simple test:

// In modules/custom/my\_module/tests/src/Functional/MyTest.php  
public function testHomepageLoads() {  
 $this->drupalGet('/');  
 $this->assertSession()->statusCodeEquals(200);  
}

* Run tests:

phpunit --testsuite functional

**4. Cache Configuration for Debugging**

* To disable render and page caching during development (add to settings.local.php):

$settings['cache']['bins']['render'] = 'cache.backend.null';  
$settings['cache']['bins']['dynamic\_page\_cache'] = 'cache.backend.null';

* Remember: **Never commit these lines for production.**

**5. Visual Guide Suggestions for the Blog**

* **Drupal error visible screenshot** (proves error reporting is enabled)
* **Devel module admin UI** (visual guide to its features)
* **IDE with Xdebug Breakpoint** (demonstrates backend step-debugging)
* **HTML View with Twig Comments** (shows template suggestions)
* **CLI output for Drush/Console** (easy for readers to replicate)
* **Sample Log Messages** in Reports > Recent log messages (Drupal admin screenshot)

**6. Useful Debugging Recipes**

**Backtrace Debugging Example**

Insert in your custom module to trace execution:

debug\_print\_backtrace(DEBUG\_BACKTRACE\_IGNORE\_ARGS);  
die('Stopping here for backtrace.');

* Use sparingly and never in production.

**Log a Custom Message**

\Drupal::logger('my\_module')->notice('Custom debug value: @value', ['@value' => $my\_value]);

These messages appear in **Reports > Recent log messages**.

**Conclusion**

With this expanded toolkit, you can set up a powerful debugging environment for Drupal, understand every layer of your site, and quickly resolve issues in code, configuration, or theming. Combining **clear error reporting, Devel, Xdebug+IDE, Drush/Console, Twig debug, and detailed logging**, you’ll troubleshoot with confidence and efficiency.

**Tip for the blog:** Consider including step-by-step screenshots for each major step, as visual context can dramatically accelerate learning for new and intermediate Drupal developers. Mark up key parts of screenshots (settings, buttons, code highlighted) with arrows or circles for clarity.

If you need even more specifics, want example screenshots, or want to focus on a particular area (like frontend vs backend debugging), let me know!