Debugging

TABLE OF CONTENTS

views 147

Overview

Using a combination of PhpStorm, DDEV and Xdebug makes debugging a pleasure. PhpStorm is not essential. Xdebug works fine with other IDE's also. In my experience, many Drupal developers have not experienced using a true debugger, but once they do, they wonder how they ever delivered any code without it.

Enable error reporting

If you experience a WSOD (White Screen Of Death), enabling verbose error messages can often give you some useful clue:

```
/**
 * Show all error messages, with backtrace information.
 *
 * In case the error level could not be fetched from the database, as for
 * example the database connection failed, we rely only on this value.
 */
$config['system.logging']['error_level'] = 'verbose';
```

For more, check out https://drupal.stackexchange.com/questions/127182/how-do-i-enable-developer-debug-mode#:~:text=%24config%5B'system.,can%20always%20comment%20them%20out. Also in https://github.com/drupal/drupal/blob/10.1.x/core/includes/bootstrap.inc Error reporting levels are defined:

```
/**
 * Error reporting level: display no errors.
 */
const ERROR_REPORTING_HIDE = 'hide';

/**
 * Error reporting level: display errors and warnings.
 */
const ERROR_REPORTING_DISPLAY_SOME = 'some';

/**
 * Error reporting level: display all messages.
 */
const ERROR_REPORTING_DISPLAY_ALL = 'all';

/**
 * Error reporting level: display all messages, plus backtrace information.
 */
const ERROR_REPORTING_DISPLAY_VERBOSE = 'verbose';
```

Disable caches and enable Twig debugging

This will cause twig debugging information to be displayed in the HTML code like the following:

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'toolbar' -->
<!-- BEGIN OUTPUT from 'core/themes/stable/templates/navigation/toolbar.html.twig' -->
```

and

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'page' -->
<!-- FILE NAME SUGGESTIONS:
   * page--teks--admin--srp--program--expectation--correlation--vote-all.html.twig
   * page--teks--admin--srp--program--expectation--correlation--852136.html.twig
   * page--teks--admin--srp--program--expectation--correlation--%.html.twig
   * page--teks--admin--srp--program--expectation--correlation.html.twig
   * page--teks--admin--srp--program--expectation--852131.html.twig
   * page--teks--admin--srp--program--expectation--%.html.twig
   * page--teks--admin--srp--program--expectation.html.twig
   * page--teks--admin--srp--program--852061.html.twig
   * page--teks--admin--srp--program--%.html.twig
   * page--teks--admin--srp--program.html.twig
   * page--teks--admin--srp.html.twig
   x page--teks--admin.html.twig
   * page--teks.html.twig
   * page.html.twig
-->
```

In `sites/default/development.services.yml` in the `parameters`, `twig.config`, set `debug:true`. See `core.services.yml` for lots of other items to change for development.

```
# Local development services.
#
parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    debug: true
    auto_reload: true
    cache: false

# To disable caching, you need this and a few other items
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in `settings.local.php`:

```
/**
 * Enable local development services.
 */
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

Disable caches in `settings.local.php`:

```
$config['system.performance']['css']['preprocess'] = FALSE;
$config['system.performance']['js']['preprocess'] = FALSE;
$settings['cache']['bins']['render'] = 'cache.backend.null';
$settings['cache']['bins']['page'] = 'cache.backend.null';
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
```

Enable/Disable Xdebug

To enable or disable Xdebug when using DDEV use:

```
$ ddev xdebug on

$ ddev xdebug off
```

Note. Enabling Xdebug will slow down your app because xdebug has a significant performance impact so be sure to disable it when you are finished debugging.

Add this to your `.zshrc` or `.bash` file for `xon` and `xoff` shortcut

```
alias xon='ddev xdebug on'
alias xoff='ddev xdebug off'
```

Xdebug Port

DDEV now sets the default port to 9003. If you want to change it to another port, use the steps below. From https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/#using-xdebug-on-a-port-other-than-the-default-9003:

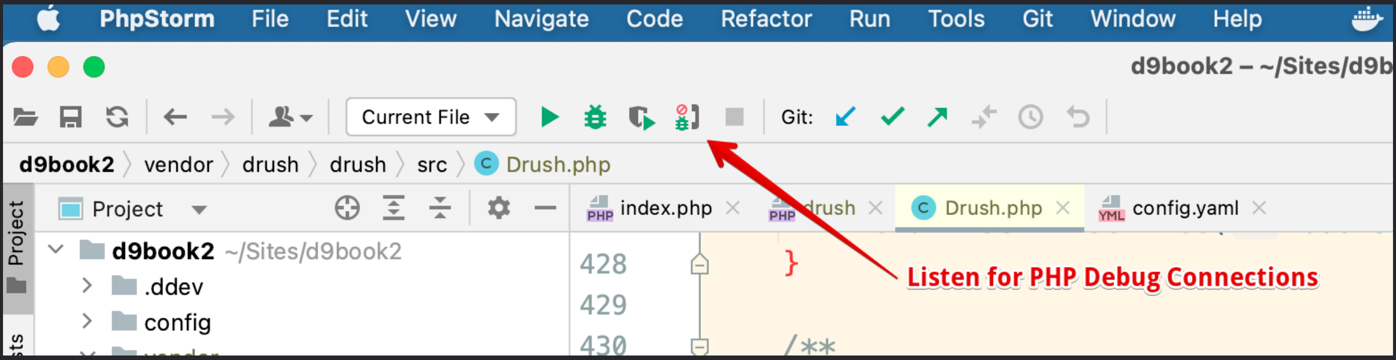
To override the port, add an override file in the project's `.ddev/php` directory. For example, add the file `.ddev/php/xdebug_client_port.ini` like this to specify the legacy port 9000:

```
[PHP]

xdebug.client_port=9000
```

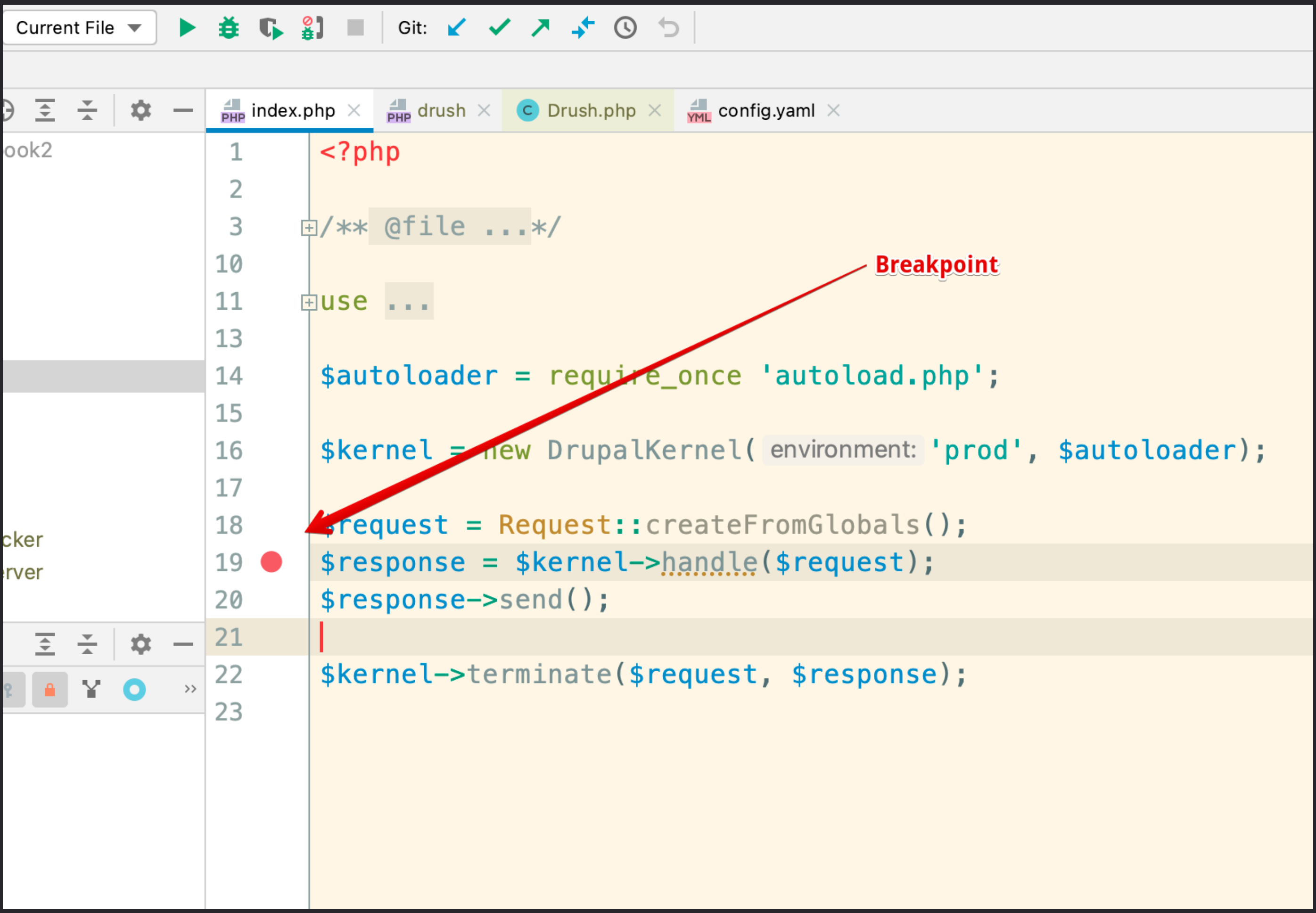
Drupal code debugging

Phpstorm and DDEV make this process as painless as possible. Once you enable Xdebug in DDEV, simply click the "start listening for PHP Debug Connections" button.

To start debugging, open the index.php file and set a breakpoint by clicking on a line number.



Select a breakpoint:

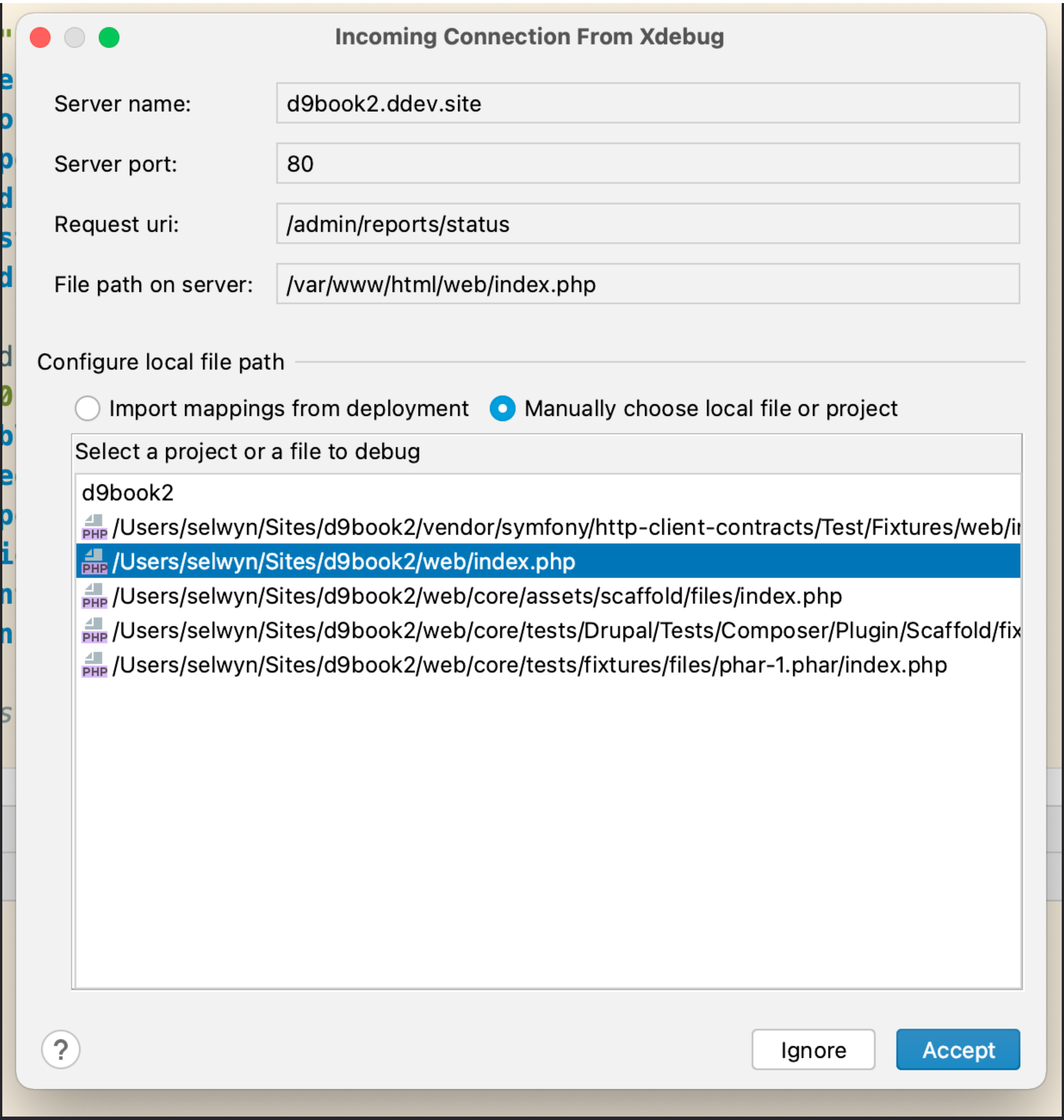


```php
<?php

/** @file ...*/

use ...

$autoloader = require_once 'autoload.php';

$kernel = new DrupalKernel( environment: 'prod', $autoloader);

$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();

$kernel->terminate($request, $response);
```

Next refresh the Drupal home page in a browser

You should immediately see a dialog pop up in PhpStorm asking you to to configure your local path. Be sure to click the `site/web/index php` and click `Accept`:

Note. If you select one of the other lines you will see a different php file pop up and you won't be debugging Drupal, but probably some Symfony file.


Incoming Connection From Xdebug

Server name: d9book2.ddev.site

Server port: 80

Request uri: /admin/reports/status

File path on server: /var/www/html/web/index.php

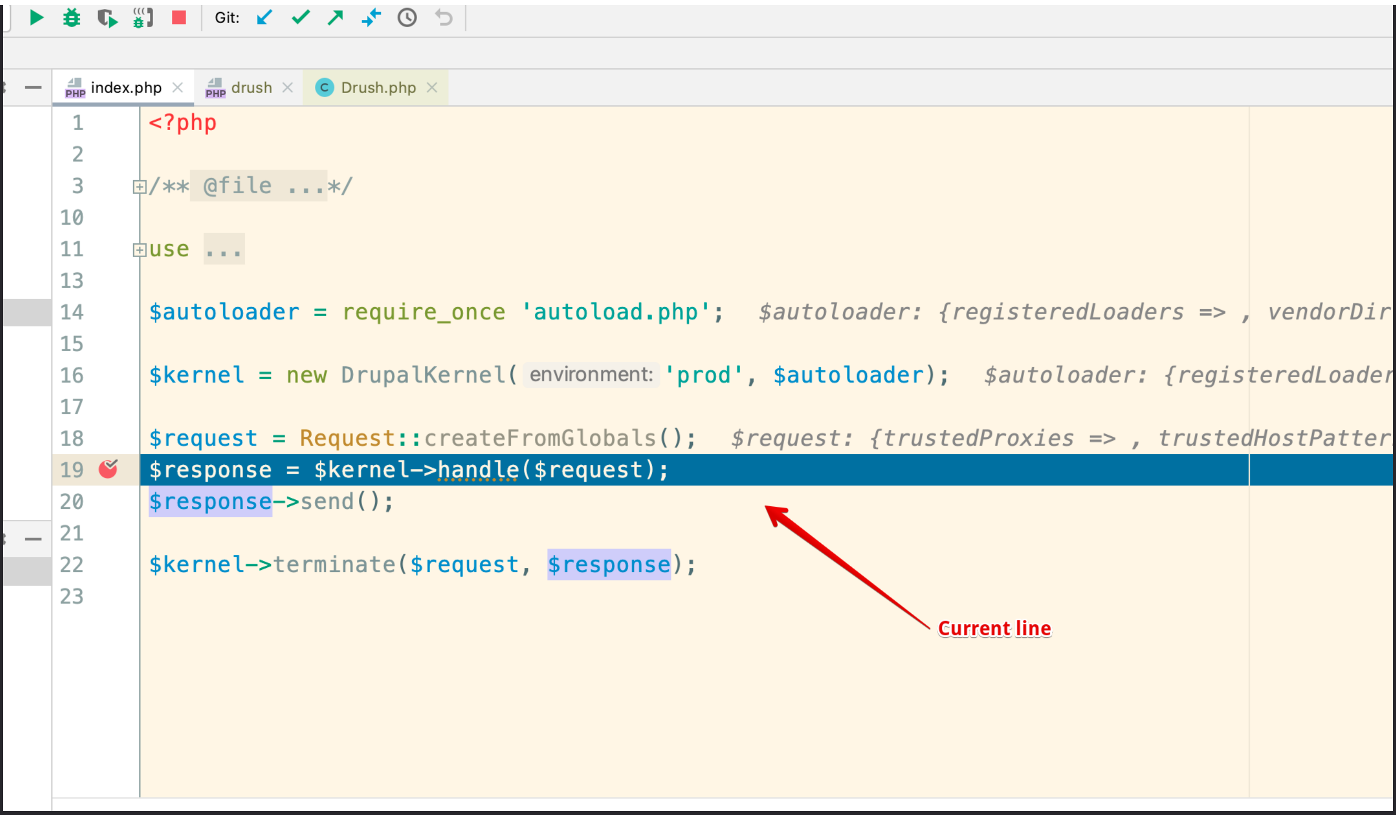
Configure local file path

Import mappings from deployment [✓ Manually choose local file or project]

Select a project or a file to debug

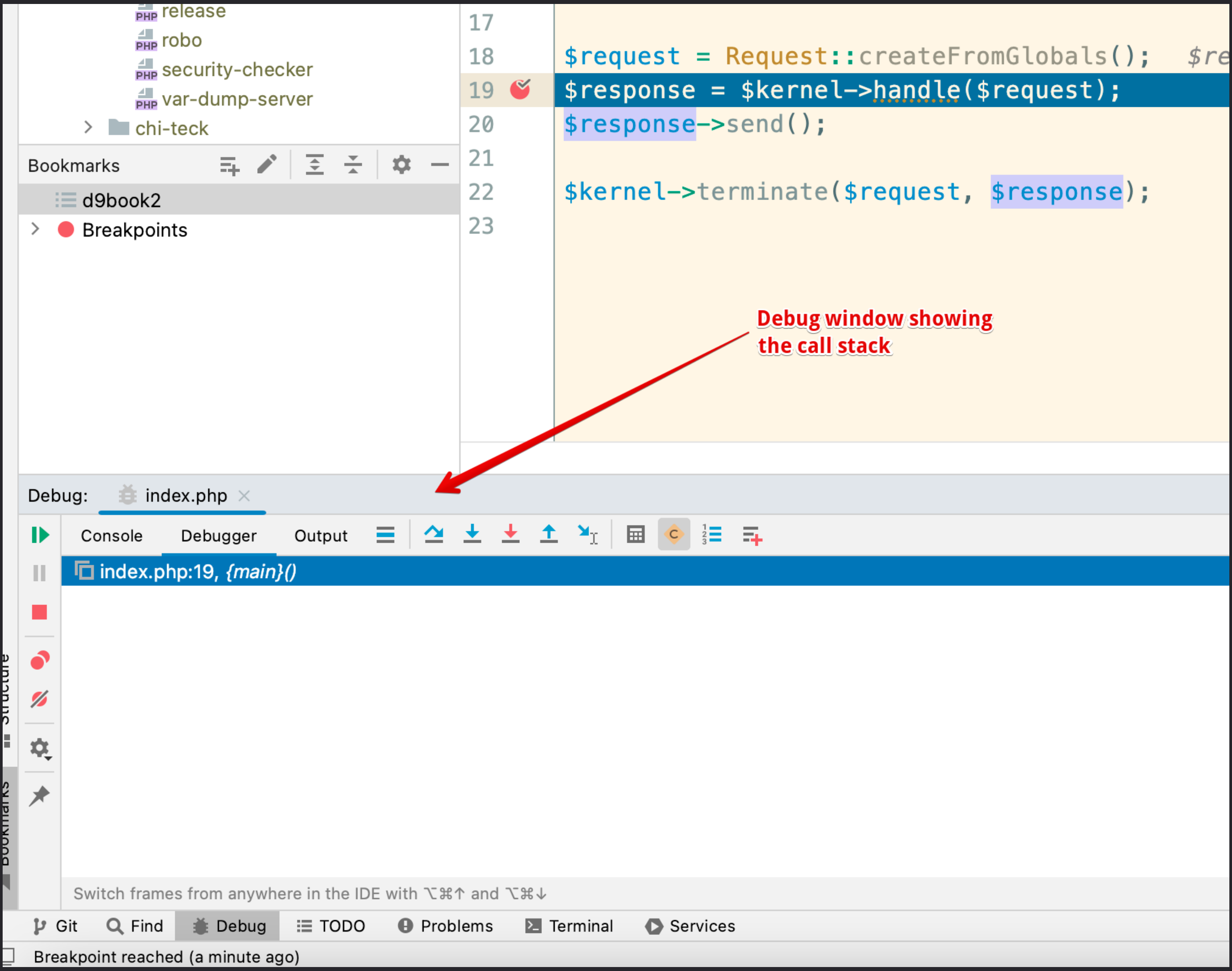
d9book2
/Users/selwyn/Sites/d9book2/vendor/symfony/http-client-contracts/Test/Fixtures/web/i
/Users/selwyn/Sites/d9book2/web/index.php
/Users/selwyn/Sites/d9book2/web/core/assets/scaffold/files/index.php
/Users/selwyn/Sites/d9book2/web/core/tests/Drupal/Tests/Composer/Plugin/Scaffold/fix
/Users/selwyn/Sites/d9book2/web/core/tests/fixtures/files/phar-1.phar/index.php

If you accidentally selected the wrong local path, in PhpStorm, go to Settings, PHP, Servers and delete all servers that are displayed. The correct one will be recreated after you retry the operation above.



```
<?php

/** @file ...*/

use ...

$autoloader = require_once 'autoload.php';  $autoloader: {registeredLoaders => , vendorDir

$kernel = new DrupalKernel( environment: 'prod', $autoloader);  $autoloader: {registeredLoader

$request = Request::createFromGlobals();  $request: {trustedProxies => , trustedHostPatter
$response = $kernel->handle($request);
$response->send();

$kernel->terminate($request, $response);
```

Once you accept the local path, you should see a highlighted line indicating the current line. The debug window will appear below showing the call stack



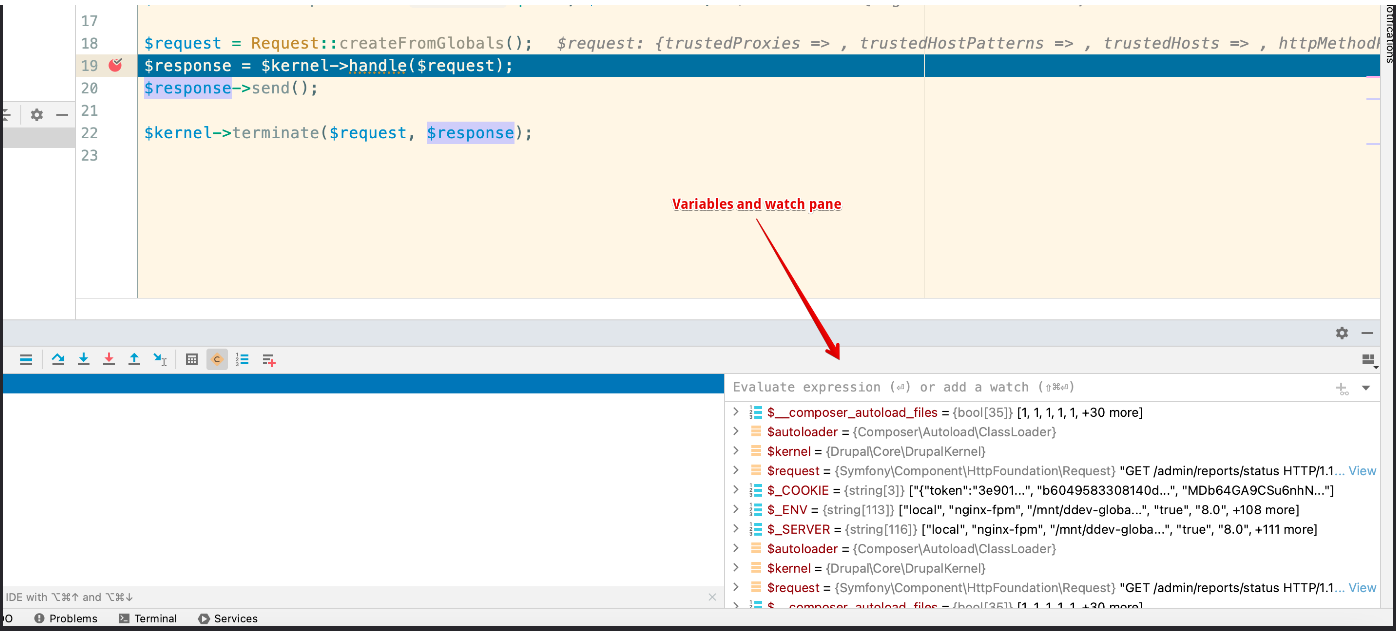
```
$request = Request::createFromGlobals();  $re
$response = $kernel->handle($request);
$response->send();

$kernel->terminate($request, $response);
```

index.php:19, {main}()

Switch frames from anywhere in the IDE with ⌥⌘↑ and ⌥⌘↓

Breakpoint reached (a minute ago)

You will also see the variables and watch pane:



```
$request = Request::createFromGlobals();   $request: {trustedProxies => , trustedHostPatterns => , trustedHosts => , httpMethod
$response = $kernel->handle($request);
$response->send();

$kernel->terminate($request, $response);
```

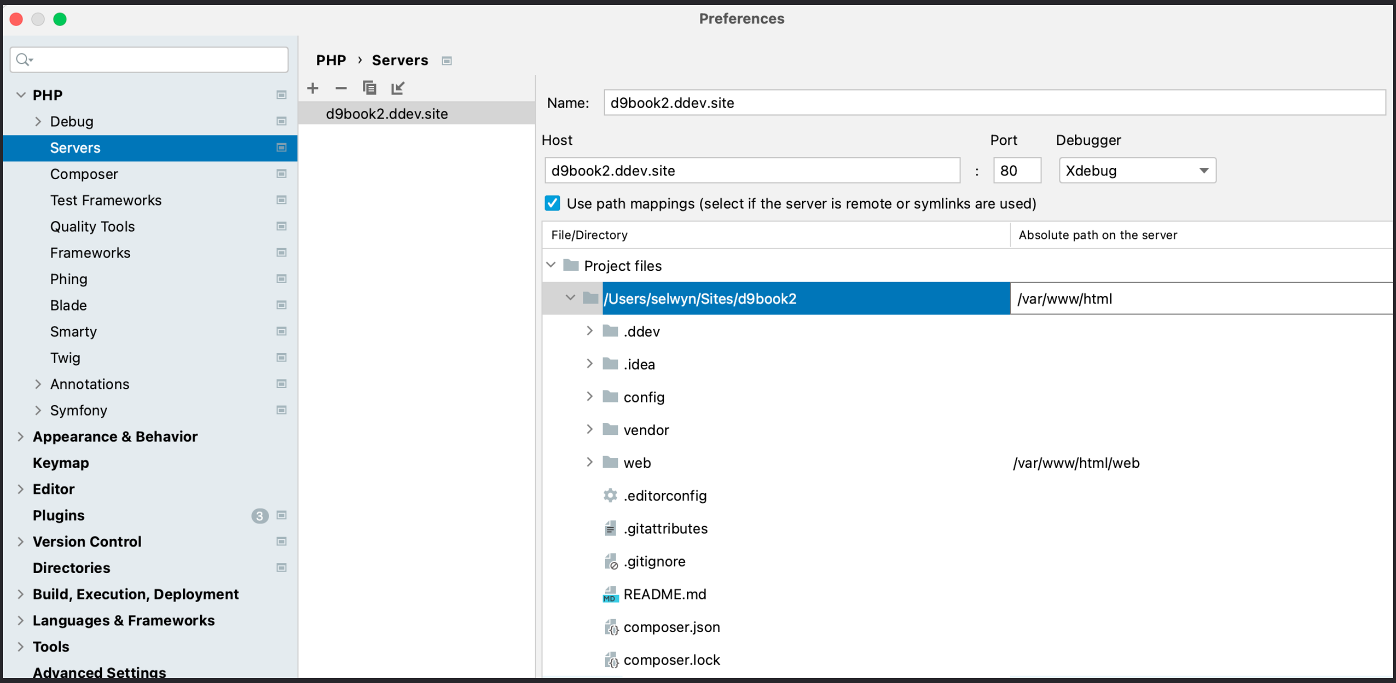
```
$__composer_autoload_files = {bool[35]} [1, 1, 1, 1, 1, +30 more]
$autoloader = {Composer\Autoload\ClassLoader}
$kernel = {Drupal\Core\DrupalKernel}
$request = {Symfony\Component\HttpFoundation\Request} "GET /admin/reports/status HTTP/1.1... View
$_COOKIE = {string[3]} ["{"token":"3e901...", "b6049583308140d...", "MDb64GA9CSu6nhN..."]
$_ENV = {string[113]} ["local", "nginx-fpm", "/mnt/ddev-globa...", "true", "8.0", +108 more]
$_SERVER = {string[116]} ["local", "nginx-fpm", "/mnt/ddev-globa...", "true", "8.0", +111 more]
$autoloader = {Composer\Autoload\ClassLoader}
$kernel = {Drupal\Core\DrupalKernel}
$request = {Symfony\Component\HttpFoundation\Request} "GET /admin/reports/status HTTP/1.1... View
```

Command line or drush debugging

For command line or drush debugging you must run your code from within the DDEV container. This means you must 'ssh' into the DDEV (Docker) container and execute the command you want to debug this way:

```
$ ddev ssh

$ vendor/bin/drush status
```

To setup command line debugging, follow the steps above to setup for Drupal Code debugging to confirm that you have debugging working. Then look in PhpStorm's: settings, PHP, Servers, and select the server you set up from the previous steps. Specify the top level path as shown below. Usually it will be /var/www/html



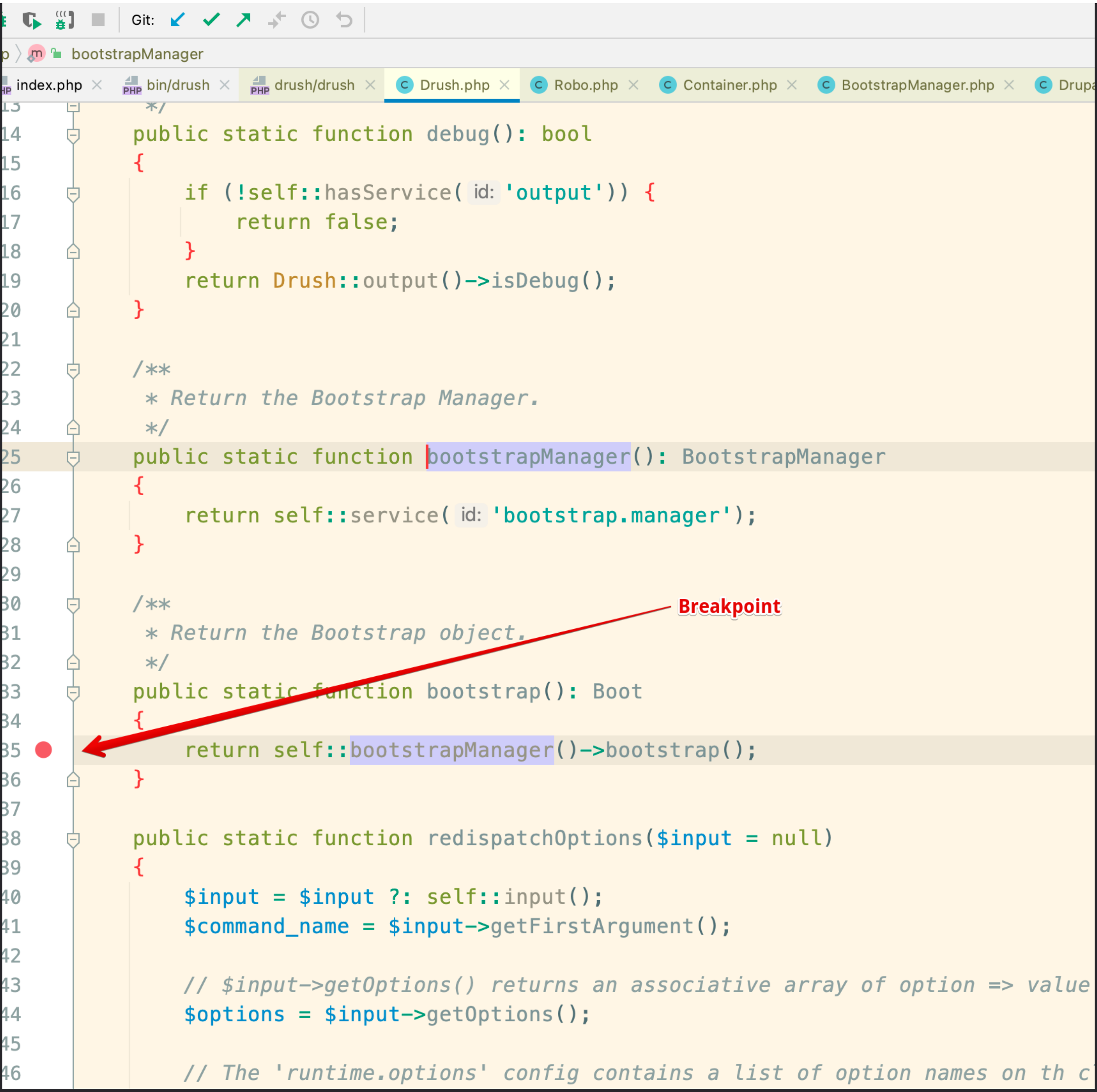
Name: d9book2.ddev.site

Host	Port	Debugger
d9book2.ddev.site	80	Xdebug

[✓] Use path mappings (select if the server is remote or symlinks are used)

File/Directory	Absolute path on the server
Project files	
/Users/selwyn/Sites/d9book2	/var/www/html
.ddev	
.idea	
config	
vendor	
web	/var/www/html/web
.editorconfig	
.gitattributes	
.gitignore	
README.md	
composer.json	
composer.lock	

Next open vendor/drush/drush/src/Drush.php and specify a breakpoint like this:

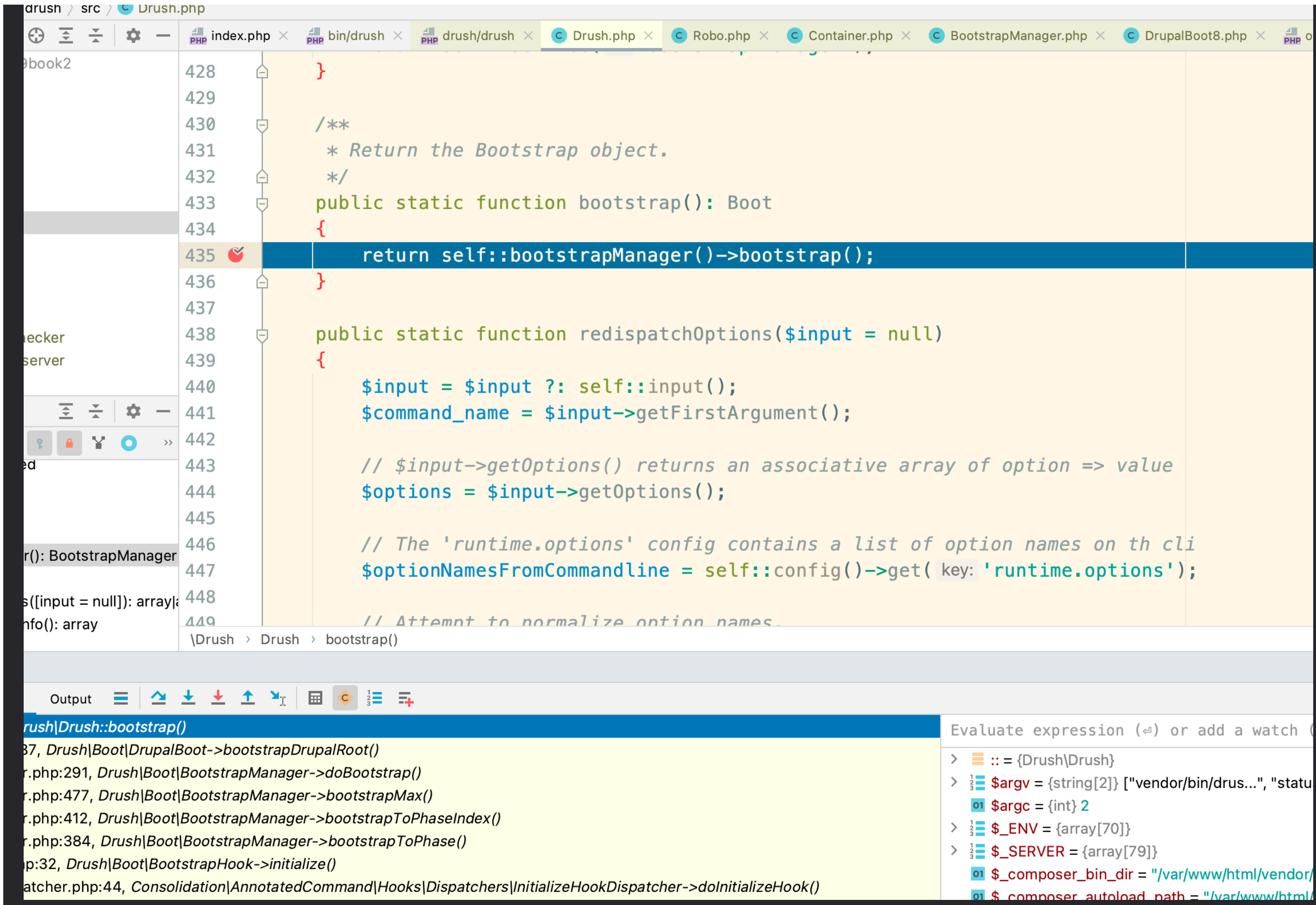


```
    */
    public static function debug(): bool
    {
        if (!self::hasService( id: 'output')) {
            return false;
        }
        return Drush::output()->isDebug();
    }

    /**
     * Return the Bootstrap Manager.
     */
    public static function bootstrapManager(): BootstrapManager
    {
        return self::service( id: 'bootstrap.manager');
    }

    /**
     * Return the Bootstrap object.
     */
    public static function bootstrap(): Boot
    {
        return self::bootstrapManager()->bootstrap();
    }

    public static function redispatchOptions($input = null)
    {
        $input = $input ?: self::input();
        $command_name = $input->getFirstArgument();

        // $input->getOptions() returns an associative array of option => value
        $options = $input->getOptions();

        // The 'runtime.options' config contains a list of option names on th c
```

Now in the terminal ssh into the DDEV container and execute drush:

```
$ ddev ssh

$ vendor/bin/drush status
```

PhpStorm will pop up and display the current line and you can debug to your heart's content:



```
    }

    /**
     * Return the Bootstrap object.
     */
    public static function bootstrap(): Boot
    {
        return self::bootstrapManager()->bootstrap();
    }

    public static function redispatchOptions($input = null)
    {
        $input = $input ?: self::input();
        $command_name = $input->getFirstArgument();

        // $input->getOptions() returns an associative array of option => value
        $options = $input->getOptions();

        // The 'runtime.options' config contains a list of option names on th cli
        $optionNamesFromCommandline = self::config()->get( key: 'runtime.options');

        // Attempt to normalize option names.
```

```
rush|Drush::bootstrap()
37, Drush|Boot|DrupalBoot->bootstrapDrupalRoot()
r.php:291, Drush|Boot|BootstrapManager->doBootstrap()
r.php:477, Drush|Boot|BootstrapManager->bootstrapMax()
r.php:412, Drush|Boot|BootstrapManager->bootstrapToPhaseIndex()
r.php:384, Drush|Boot|BootstrapManager->bootstrapToPhase()
p:32, Drush|Boot|BootstrapHook->initialize()
atcher.php:44, Consolidation|AnnotatedCommand|Hooks|Dispatchers|InitializeHookDispatcher->doInitializeHook()
```

```
:: = {Drush\Drush}
$argv = {string[2]} ["vendor/bin/drus...", "statu
$argc = {int} 2
$_ENV = {array[70]}
$_SERVER = {array[79]}
$_composer_bin_dir = "/var/www/html/vendor/
$_composer_autoload_path = "/var/www/html/
```

More at https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/

Add a breakpoint in code

To add a breakpoint in code, you can use: xdebug_break()

more at https://xdebug.org/docs/all_functions

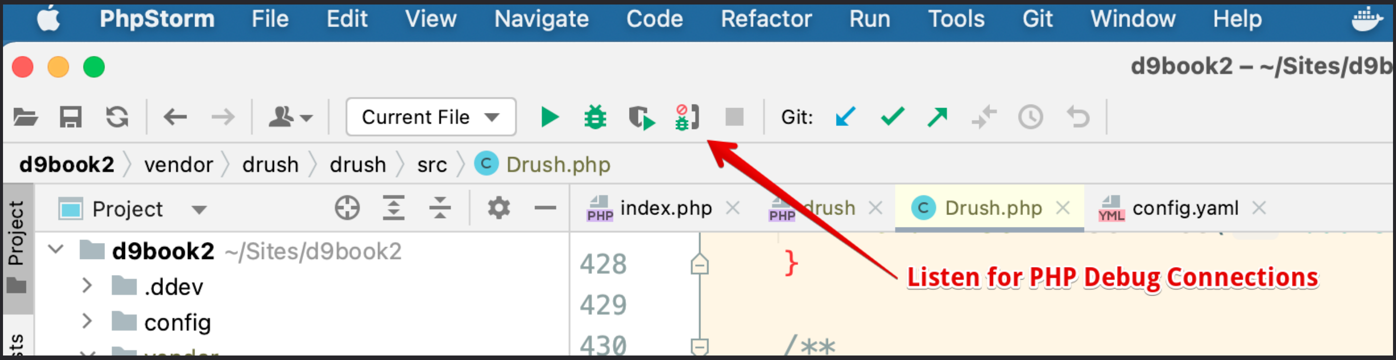
Troubleshooting Xdebug with DDEV

Could not connect to debugging client

When debugging command line e.g. drush commands etc. if you see:

```
$ vendor/bin/drush status


Xdebug: \[Step Debug\] Could not connect to debugging client. Tried: host.docker.internal:9003 (through xdebug.client_host/xdebug.client_port).
```

This means you have not clicked the PhpStorm button: "Start listening for PHP Debug Connections". Just click it and try again



PhpStorm refuses to debug

Here are some steps to try:

CURL

Use curl or a browser to create a web request. For example:

```
$ curl https://d9.ddev.site
```

Replace d9.ddev.site with the correct URL for your site.

LOGS

If the IDE doesn't respond, take a look at ddev logs. Use ddev logs to display current logs for the project's web server. See https://ddev.readthedocs.io/en/stable/users/basics/commands/#logs for more.

If you see a message like

```
"PHP message: Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9000 (through xdebug.client_host/xdebug client_port)"
```

then php/xdebug (inside the container) is not able to make a connection to port 9000.

This means you have not clicked the “Start listening for PHP Debug Connections” button in PhpStorm. Just click it and try again.

Note. Port 9003 is more current.

TELNET

With PhpStorm NOT listening for a PHP Debug connection, try to telnet to see what is listening to port 9003 with this:

```
$ telnet d9book2.ddev.site 9003
```

You should get connection refused. If you get a connection (see below) something is listening on port 9003 and you should either disable it or use a different port. Note. You must set that in both PhpStorm as well as DDEV.

This shows a connection succeeding. You can try it with PhpStorm listening for a debug connection:

```
$ telnet d9book2.ddev.site 9003

Trying 127.0.0.1\...

Connected to d9book2.ddev.site.

Escape character is \'\^\]\'.
```

Use Control] to exit and then type quit to return to exit telnet.

On a mac, use `sudo lsof -i :9003 -sTCP:LISTEN` to find out what is listening on that port and stop it, or change the xdebug port and configure both DDEV and PhpStorm to use the new one .

More about changing ports at https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/#using-xdebug-on-a-port-other-than-the-default-9003

Note. In the past, php-fpm was likely to be one of the apps using port 9000.

IS XDEBUG ENABLED?

• To check to make sure that Xdebug is enabled, you can use `php -i | grep xdebug` inside the container. You can also use other techniques to view the output of phpinfo(), including Drupal's `admin/reports/status/php`. Below you can see the expected output when Xdebug is enabled assuming you have Xdebug v3.2.0 or later.

```
`$ php -i | grep "xdebug.remote_enable"`

xdebug.remote_enable => (setting renamed in Xdebug 3) => (setting renamed in Xdebug 3)
```

See https://ddev.readthedocs.io/en/stable/users/step-debugging

What is listening on the debug port?

To check if something is listening on port 9003, it's best to use `lsof` as it will actually list the name of the process listening.

i.e. Here we see phpstorm listening:

```
lsof -i TCP:9003
COMMAND    PID   USER   FD   TYPE            DEVICE SIZE/OFF NODE NAME
phpstorm 78897 selwyn  209u  IPv4 0x4d118c80f54422d7      0t0  TCP *:9003 (LISTEN)
```

You can also use `nc` and `netstat` but they is not quite as informative:

```
$ nc -z localhost 9003
Connection to localhost port 9003 [tcp/*] succeeded!
```

The message `Connection to localhost port 9003 [tcp/*] succeeded!` means that there is a program listening on port 9003. If you get nothing when you type the command, then nothing is listening.

Here `netstat` reports that something is listening on port 9003. Again, if you get nothing when you type the command, then nothing is listening.

```
$ netstat -an | grep 9003
tcp4       0      0  *.9003                 *.*                    LISTEN
```

Here is an example running `lsof` and finding php-fpm listening on port 9000

```
$ lsof -i TCP:9000

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

php-fpm 732 selwyn 7u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)

php-fpm 764 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)

php-fpm 765 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)
```

Enable twig debugging output in source

In `sites/default/development.services.yml` in the `parameters`, `twig.config`, set `debug:true`. See `core.services.yml` for lots of other items to change for development.

```
# Local development services.
#
parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    debug: true
    auto_reload: true
    cache: false

# To disable caching, you need this and a few other items
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in settings.local.php:

```
/**
 * Enable local development services.
 */
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in `settings.local.php` with:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';
```

Devel and Devel Kint Extras

From https://www.webwash.net/how-to-print-variables-using-devel-and-kint-in-drupal/

Setup

We need both the Devel and Devel Kint Extras module. [Devel Kint Extras]() ships with the kint-php library so installing this via Composer will take care of it automatically.

Install using Composer:

```
composer require drupal/devel drupal/devel_kint_extras
```

Enable both with the following Drush command:

```
$ ddev drush en devel_kint_extras -y
```

Finally, enable `Kint Extended` as the `Variables Dumper`. To do this go to:

```
admin/config/development/devel
```

and select `Kint Extender` and Save the configuration

Add kint to a custom module

```
function custom_kint_preprocess_page(&$variables) {
  kint($variables['page']);
 }
```

Dump variables in a TWIG template

{{ kint(attributes) }}

Kint::dump

From [Migrate Devel contrib module](), in `docroot/modules/contrib/migrate_devel/src/EventSubscriber/MigrationEventSubscriber.php`

This is used in migrate to dump the source and destination values.

```
// We use kint directly here since we want to support variable naming.
kint_require();
\Kint::dump($Source, $Destination, $DestinationIDValues);
```

Set max levels to avoid running out of memory

Kint can run really slowly so you may have to set maxLevels this way:

Add this to settings.local.php

```
// Change kint maxLevels setting:
include_once(DRUPAL_ROOT . '/modules/contrib/devel/kint/kint/Kint.class.php');
if(class_exists('Kint')){
  // Set the maxlevels to prevent out-of-memory. Currently there doesn't seem to be a cleaner way to set this:
  Kint::$maxLevels = 4;
}
```

Resources

- DDEV Documentation
- Debugging with Xdebug in DDEV docs
- Debug Drush commands with PhpStorm at
- Configuring PhpStorm Drupal.org docs updated September 2022
- Debugging Drush commands
- DDEV docs on using a different port for debugging
- How to setup Devel and Kint on Drupal 9 by Alex Aug 2021

Back to top