




Tighten up your Drupal code using PHPStan

Finding bugs before your end users do!

Matt Glaman

Maintainer of `phpstan-drupal` & `drupal-check`

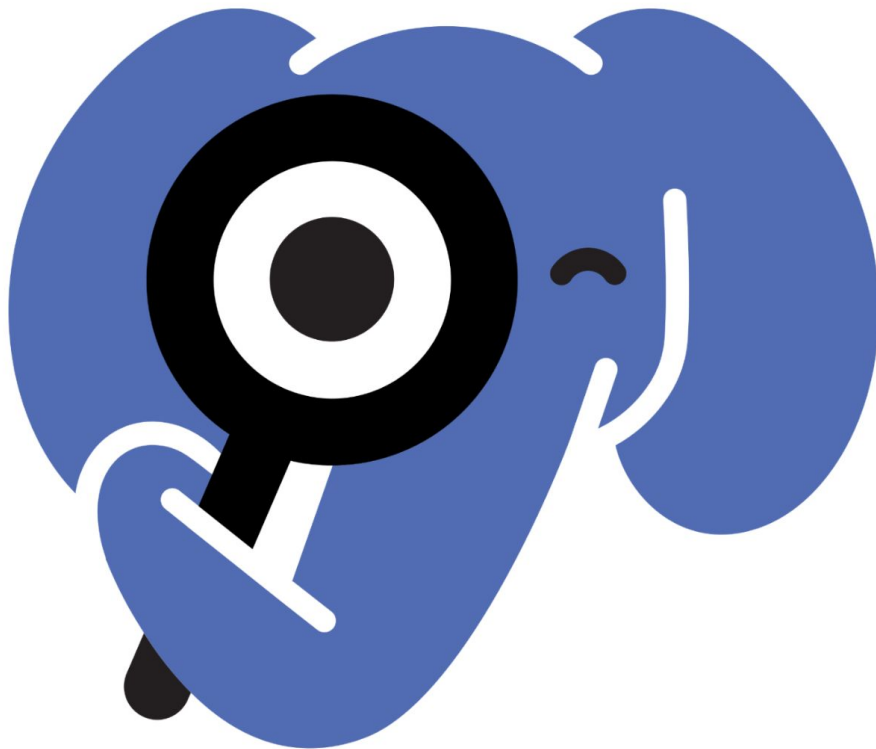
 [/u/mglaman](#)  [@nmdmatt](#)  [mglaman.dev](#)

PHPStan

PHP static analysis tool

PHPStan finds bugs in your code without writing tests.

<https://phpstan.org/>



phpstan-drupal

Extension for PHPStan to integrate with Drupal.

[mglaman/phpstan-drupal](https://mglaman.com/phpstan-drupal)



drupal-check

Wrapper around PHPStan and phpstan-drupal for generic config.

[mglaman/drupal-check](https://github.com/mglaman/drupal-check)



But first, what about X?

Can your existing tools catch the typo in the method name?

Linting

- Using `php -l` you can lint your code for syntax errors
- Great first step in your continuous integration pipelines
- Doesn't catch typos or calls to invalid methods

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(EntityInterface $node): void {
```

```
    if ($node->isPublihed()) {
```

```
    }
```

```
}
```



PHP CodeSniffer

- Uses `token_get_all` to tokenize a given source code
- Analyzes files individually and line by line
- Can detect calls to undesired functions, but not classes
- Great for coding standards and basic “code smell” checks
- Keeps code tidy, doesn’t find bugs.

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(EntityInterface $node): void {
```

```
    if ($node->isPublihed()) {
```

```
    }
```

```
}
```



Phan / Psalm

- Phan is another static analysis tool, which requires the php-ast extension (From Etsy)
- Psalm is another static analysis tool, with security analysis tools (From Vimeo)
- Drupal's autoloading is dynamic, unlike most PHP applications. **This makes it difficult to work with other tools**

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;

/**
 * Implements hook_ENTITY_TYPE_insert().
 */
function mymodule_node_insert(EntityInterface $node): void {
    if ($node->isPublished()) {

    }
}
```



PHPStan

- Uses nikic/php-parser to create an abstract syntax tree of your code base.
- Verifies calls to classes and their methods (class exists, method visibility)
- Verifies types passed to functions and methods
- Has a system for defining dynamic returns types (and Drupal **is very dynamic!**)

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(EntityInterface $node): void {
```

```
    if ($node->isPublished()) {
```

```
    }
```

```
}
```



What PHPStan can do for *you*!

PHPStan Rule Levels

- **0**: unknown classes/functions/methods (**\$this**), argument count, undefined variables
- **1**: possibly undefined variables, unknown magic methods or properties
- **2**: checks for unknown methods to all objects, validating PHPDocs
- **3**: return types, types assigned to properties
- **4**: dead code checking, redundant code
- **5**: type checks of arguments passed to functions/methods
- **6**: report missing typehints
- **7**: report wrong method calls on union types (**EntityInterface|NodeInterface**),
- **8**: report calling methods and accessing properties on nullable types
- **9**: strict on **mixed** type usage

10.0.x

Drupal 10 *is now running* PHPStan at level 0

Let's analyze the example code with PHPStan

(This is running PHPStan at level 2)

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(EntityInterface $node): void {
```

```
    if ($node->isPublished()) {
```

```
    }
```

```
}
```

Call to an undefined method Drupal\Core\Entity\EntityInterface::isPublished().

```
<?php
```

```
use Drupal\Core\Entity\EntityInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(EntityInterface $node): void {
```

```
    if ($node->isPublished()) {
```

```
        ?#!?!
```

```
    }
```

```
}
```

Call to an undefined method Drupal\Core\Entity\EntityInterface::isPublished().

```
<?php
```

```
use Drupal\node\NodeInterface;
```

```
/**
```

```
* Implements hook_ENTITY_TYPE_insert().
```

```
*/
```

```
function mymodule_node_insert(NodeInterface $node): void {
```

```
    if ($node->isPublished()) {
```

```
    }
```

```
}
```

isPublished comes from **EntityPublishedInterface**, which **NodeInterface** extends!

PHPStan & Extensions

PHPStan & Extensions overview

PHPStan

- Checks that a class exists (can be autoloaded)
- Detects incorrect namespacing
- Functions exist, methods on classes exist and are visible
- Can resolve variable values and verify their types (!!!)

phpstan-drupal

- Container services return the correct types
- Entity storage and query return types
- Class resolver service return types
- Checking if using @internal classes
- Support for checking deprecated global constants

PHPStan & Extensions overview

phpstan/phpstan-deprecation-rules

- PHPStan rules for detecting usage of deprecated classes, methods, properties, constants and traits.
- The special sauce used by the Drupal community in **drupal-check** and the **Upgrade Status** module.

phpstan/phpstan-phpunit

- PHPUnit extensions and rules for PHPStan
- Uses assertions to understand types, support for mocks, and more.

jangregor/phpstan-prophecy

- Provides a phpstan/phpstan extension for phpspec/prophecy
- Makes PHPStan understand prophesied mocks

**Because you are all
developers and
want to play...**

Adding PHPStan to your Drupal codebase

(What I run when setting up PHPStan for a Drupal code base)

phpstan/extension-installer

- Automatically configures PHPStan to use installed extensions
- Simplifies setting up PHPStan by not needing to include extension configurations
- Pssst: this is a problem **drupal-check** tried to solve for Drupal users, before the extension installer existed.

```
composer require --dev phpstan/phpstan \  
    phpstan/extension-installer \  
    mglaman/phpstan-drupal \  
    phpstan/phpstan-deprecation-rules
```

Use Composer to add PHPStan to require-dev

```
php vendor/bin/phpstan analyze \  
  --level 2 \  
  web/modules/custom
```

Run PHPStan against custom modules

Adding PHPStan to your Drupal codebase (how I do it)

```
composer require --dev phpstan/phpstan \  
    phpstan/extension-installer \  
    mglaman/phpstan-drupal \  
    phpstan/phpstan-deprecation-rules \  
    phpstan/phpstan-phpunit \  
    jangregor/phpstan-prophecy
```

Use Composer to add PHPStan to require-dev

parameters:

level: 5

paths:

- web/modules/custom
- web/themes/custom

A basic phpstan.neon

```
php vendor/bin/phpstan
```

Run PHPStan against custom modules (no arguments requires with `paths` defined)

phpstan-drupal

Bringing PHPStan magic to Drupal ✨

Autoloading

Autoloading extensions and functions

- PHPStan supports path based autoloading, but the goal is to mimic the Drupal bootstrap process
- Drupal has various includes for “legacy” functions not registered in its autoloader
- **All** extension namespaces are registered at runtime with the autoloader and their extension file loaded
- Loads files for hook includes (**views.inc**, **tokens.inc**, **pathauto.inc**)
- Loads Drush includes for functions as well

Service container

Services return types and deprecations

- Scans for *all* extensions and loads their extension file, along with registering their **services.yml** definitions.
- A service map is maintained to allow rules and return type extensions to interact with services that would exist in Drupal's container
- Reports when retrieving a deprecated service (**`$container->get / \Drupal::service`**)
- Allows detecting if invalid or deprecated method is called from the service

Entity integration

Entity mapping

- Contains a repository of entity information
- Correct storage class returned from entity type manager
- Correct entity class returned from entity storage methods
- Contrib can define their own mappings to be included ([link](#))

drupal:

entityMapping:

block:

class: Drupal\block\Entity\Block

block_content:

class: Drupal\block_content\Entity\BlockContent

node:

class: Drupal\node\Entity\Node

storage: Drupal\node\NodeStorage

taxonomy_term:

class: Drupal\taxonomy\Entity\Term

storage: Drupal\taxonomy\TermStorage

```
$etm = \Drupal::entityTypeManager();  
  
assertType('Drupal\node\NodeStorage', $etm->getStorage('node'));  
assertType('Drupal\user\UserStorage', $etm->getStorage('user'));  
assertType('Drupal\taxonomy\TermStorage', $etm->getStorage('taxonomy_term'));
```

Example of entity storage type assertions

```
$nodeStorage = \Drupal::entityTypeManager()->getStorage('node');  
  
assertType('Drupal\node\Entity\Node', $nodeStorage->create(['type' => 'page']));  
assertType('Drupal\node\Entity\Node|null', $nodeStorage->load(42));  
assertType('Drupal\node\Entity\Node|null', $nodeStorage->loadUnchanged(42));  
assertType('array<int, Drupal\node\Entity\Node>', $nodeStorage->loadMultiple());
```

Example of entity storage method assertions

Entity queries

- Determines the array return type for queries

`array<int, string>` vs.
`array<string, string>`

- Returns correct type if turned into a **count** query.
- **TODO!** Verify that **accessCheck** has been invoked (now required in 10.0.x)

```
assertType(  
    'array<int, string>',  
    $nodeStorage->getQuery()  
        ->accessCheck(TRUE)  
        ->execute()  
);  
  
assertType(  
    'int',  
    $nodeStorage->getQuery()  
        ->accessCheck(TRUE)  
        ->count()  
        ->execute()  
);
```

kudos!

beram (Benjamin Rambaud) for the major contributions to the entity storage
dynamic return type extensions

Render arrays

Trusted callbacks

- Verifies callbacks are closures or implement **TrustedCallbackInterface** or **RenderCallbackInterface**
- Checks **#pre_render**, **#post_render**, **#access_callback**, and **#lazy_builder**
- Supports normal and service name callable format
- Warns if using a closure within a form class (serialization = 🌟)

Loaded includes

Loaded includes

- Handles **ModuleHandlerInterface::loadIncludes** or the deprecated **module_load_include** function
- Verifies that the extension exists
- Verifies the file exists
- Performs **require_once** to bring the file into scope to make the functions within the file accessible

Miscellaneous awesome

Class resolver

- Correct object types from the class resolver
- **getInstanceFromDefinition** will return an instance of the correct class
- Allows proper inspections from this dynamic class instantiation

```
function workspaces_entity_type_build(array &$entity_types) {  
    return \Drupal::service('class_resolver')  
        ->getInstanceFromDefinition(EntityTypeInfo::class)  
        ->entityTypeBuild($entity_types);  
}
```

```
function workspaces_entity_type_alter(array &$entity_types) {  
    \Drupal::service('class_resolver')  
        ->getInstanceFromDefinition(EntityTypeInfo::class)  
        ->entityTypeAlter($entity_types);  
}
```

Entity access results

- Checks if calls to an entity access method should return **AccessResultInterface** or **bool**
- Handles **access**, **createAccess**, **fieldAccess**.

```
assertType(  
    'bool',  
    $accessControlHandler->access(Node::create(), 'view')  
);
```

```
assertType(  
    AccessResultInterface::class,  
    $accessControlHandler->access(  
        Node::create(),  
        'view label',  
        null,  
        true  
    )  
);
```

Extending `@internal` code

- Checks if a class extends `@internal` code outside of its namespace
- Only flags an error when using `@internal` outside of shared namespace
- Shared namespace? `\Drupal\{Core|Component|module|theme}`
- The second part of the namespace must match

How to add PHPStan to your codebase


```
composer require --dev phpstan/phpstan \  
    phpstan/extension-installer \  
    mglaman/phpstan-drupal \  
    phpstan/phpstan-deprecation-rules
```

Use Composer to add PHPStan to require-dev

```
php vendor/bin/phpstan analyze \  
  --level 2 \  
  web/modules/custom
```

Run PHPStan against custom modules

What's on the horizon?

What's on the horizon?

- Better tracking of change records from Drupal core to make sure phpstan-drupal has the rules or return types to PHPStan to detect the change.
- Better support for entity fields and field properties
- Drush command to help generate entity mapping and field information for phpstan-drupal 🤔
- And all of your suggestions 😊

Resources

#phpstan

Join the **#phpstan** channel on Drupal Slack.

GitHub bot will notify of new releases.

Links

- <https://www.drupal.org/docs/develop/development-tools/phpstan>
- <https://phpstan.org/>
- <https://github.com/mglaman/phpstan-drupal>
- <https://beram-presentation.gitlab.io/php-static-analysis-101/>
- <https://www.twitch.tv/mglaman> (live coding, Wednesdays 2PM US Central)