Tighten up your Drupal code using PHPStan

Finding bugs before your end users do!

Matt Glaman

Maintainer of phpstan-drupal

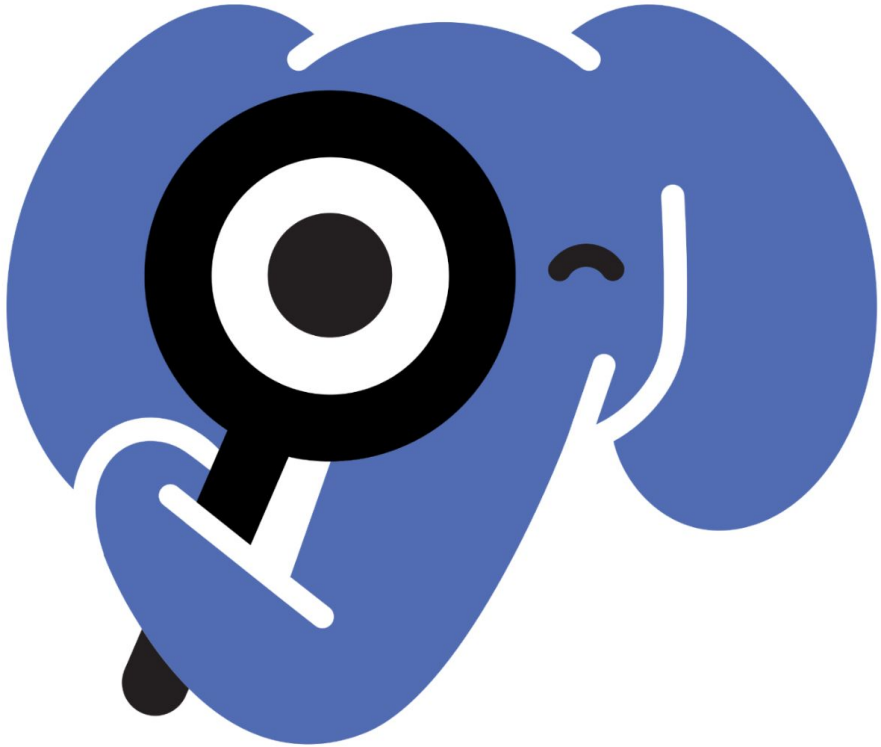
/u/mglaman @nmdmatt @mglaman@phpc.social mglaman.dev

PHPStan

PHP static **an**alysis 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]

But first, what about X?

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

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


function mymodule_node_insert(
  EntityInterface $node
): void {
    if ($node->isPublihed()) {
    }
}
```

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;


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;


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 Estsy)
- 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;

function mymodule_node_insert(
  EntityInterface $node
): void {
  if ($node->isPublihed()) {
  }
}
```

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;

function mymodule_node_insert(
  EntityInterface $node
): void {
  if ($node->isPublihed()) {
  }
}
```

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 type hints
- **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

`level: 1`

Drupal runs PHPStan at level 1!

`PHPStan-2` issue tag for bumping to level 2

level: 0

GitLab CI runs PHPStan level 0

Fall 2023

phpstan-baseline.neon

Accept existing reported errors without having to fix them all

Allows starting at a higher level for new code

Read The Baseline documentation for more

Let’s analyze the example code with PHPStan

(This is running PHPStan at level 2)

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


function mymodule_node_insert(
  EntityInterface $node
): void {
    if ($node->isPublihed()) {
    }
}
```

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

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


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;


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 exists, methods on classes exist and are visible
- Can resolve variable values and verify their types (!!!)

phpstan/extension-installer

- Automatically configures PHPStan to use installed extensions
- Simplifies setting up PHPStan by not needing to include extension configurations
- Used by Drupal core

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 the **Upgrade Status** module.
- Became a dependency of phpstan-drupal, used by Drupal core

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-phpunit

- PHPUnit extensions and rules for PHPStan
- Uses assertions to understand types, support for mocks, and more.
- Used by Drupal core (greatly reduced level 2 errors)

jangregor/phpstan-prophecy

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

PHPStan & Extensions overview

phpstan/phpstan-strict-rules

- Enforces more defensive coding practices
- Disallow `empty`
- Stronger enforcement of types

Find more on Packagist

`packagist.org?type=phpstan-extension`

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

Adding PHPStan to your Drupal codebase

```
composer require --dev drupal/core-dev
```

Use Composer to add PHPStan to require-dev

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

2022

Use Composer to add PHPStan to require-dev

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

Run PHPStan against custom modules, no configuration required

How I configure my `phpstan.neon`

```
parameters:
    level: 9
    paths:
        - web/modules/custom
        - web/themes/custom
includes:
    - vendor/phpstan/phpstan/conf/bleedingEdge.neon
    - phpstan-baseline.neon
```

My normal **phpstan.neon**

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

Run PHPStan against custom modules, no need to specify paths or level

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
```

Entity storage class detection

phpstan-drupal makes PHPStan under stand more of the dynamic calls that are used in Drupal.

```php
$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')
);
```

Entity class returned from storage methods

phpstand-drupal makes PHPStan aware of the class returned from entity type storage methods.

```
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()
);
```

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.

- Provides checks that **`accessCheck`** has been invoked

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

Render arrays

Trusted callbacks

- Verifies callbacks are closures or implement `TrustedCallbackInterface`, `RenderCallbackInterface`, or the `TrustedCallback` attribute.
- 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

Stub files

Stub files

- Improved field support by stubbing `FieldItemListInterface` and `ListInterface`.
- Uses generics to handle traversing and accessing values from entity fields
- Allows for field item lists to more easily specify the field type they contain

```
/**
 * @template T of FieldItemInterface
 * @extends ListInterface<T>
 * @property mixed $value
 */
interface FieldItemListInterface

/**
 * @template T of TypedDataInterface
 * @extends \Traversable<int, T>
 * @extends \ArrayAccess<int,T>
 */
interface ListInterface
```

Stub files

```
/**
 * @template T of EntityInterface
 * @extends FieldItemListInterface<EntityReferenceItem<T>>
 * @property int|string|null $target_id
 * @property ?T $entity
 */
interface EntityReferenceFieldItemListInterface extends FieldItemListInterface {

  /**
   * @return array<int, T>
   */
  public function referencedEntities();
}
```

Stub files

```
/**
 * @phpstan-type CacheObject object{
        data: mixed, created: int, tags: string[], valid: bool,
        expire: int, checksum: string, serialized: int }
 */
interface CacheBackendInterface {

    /**
     * @return CacheObject|false
     */
    public function get(string $cid, bool $allow_invalid = FALSE);

    /**
    * @param string[] $cids
    * @return CacheObject[]
     */
    public function getMultiple(array &$cids, bool $allow_invalid = FALSE): array;
}
```

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

```php
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`**.

```php
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 drupal/core-dev
```

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?

- Improved container support, to avoid issues with autowiring or complex service definitions
- Drush command to help generate entity mapping and field information for phpstan-drupal 🤔
- Plugin manager rule clean up
- Better Drush support for its own deprecated global constants
- And all of your suggestions 😄

`level: 9`

Drupal running PHPStan at level 9?!

Adding *all* existing errors to the baseline while improving all new code

#3426047

Resources

#phpstan

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

GitHub bot will notify of new releases.

Links

- Drupal + PHPStan documentation on Drupal.org
  https://www.drupal.org/docs/develop/development-tools/phpstan
- PHPStan website and documentation
  https://phpstan.org/
- Github repository
  https://github.com/mglaman/phpstan-drupal
- PHP Static Analysis 101
  https://beram-presentation.gitlab.io/php-static-analysis-101/
- What we learned introducing PHPStan to a large scale project
  https://www.youtube.com/live/rIriFId9j2M?si=IcX00cjOTDTK3Q-2