

Introduction

Specialized PHP data containers, from simple bags to stores validated with JSON Schema, with a common interface and typed getters

Anonymous · 20/10/2026

Flexible Data Containers for PHP

[github](#) [derafu/container](#) [last commit](#) [today](#) [CI](#) [passing](#) [code size](#) [122 KiB](#) [open issues](#) [0](#)

[downloads](#) [8.23 k](#) [downloads](#) [1.28 k this month](#)

A collection of specialized PHP data containers that provide different levels of structure and validation, from simple bags to schema-validated stores.

Features

- ☐ Purpose-built containers for different needs.
- ☐ Validation with a simple schema (`Vault`) or with JSON Schema (`Store`).
- ☐ Validation on every change: a container never holds data that breaks its schema.
- ☐ Typed getters (`getString()`, `requireInt()`, `getList(of: 'string')...`) that PHPStan understands.
- ☐ Simple to complex data structures, with dot notation.
- ☐ Sequential data handling.
- ☐ Common interface across containers.
- ☐ Translatable exceptions.
- ☐ Minimal dependencies.
- ☐ Comprehensive test coverage.

Why Derafu\Container?

Unlike generic data structures, Derafu\Container provides specialized containers each designed for specific use cases:

- **Bag**: Simple, flexible data storage without restrictions.
- **Vault**: Structured data validated with a simple schema (types, required keys, choices, defaults, normalizers).
- **Store**: Structured data validated with JSON Schema.
- **Journal**: Sequential data storage with LIFO access.

All of them share the same interface, so code that reads data does not care which container it

received.

Installation

Install via Composer:

```
composer require derafu/container
```

A Quick Tour

```
use Derafu\Container\Vault;

$options = new Vault(
    ['host' => 'imap.example.com'],
    [
        'host' => ['types' => 'string', 'required' => true],
        'port' => ['types' => 'int', 'default' => 993],
    ]
);

$host = $options->requireString('host'); // string: "imap.example.com"
$port = $options->requireInt('port');    // int: 993 (the default of the schema).

$options->set('port', 'abc');              // Throws: the port must be an int.
$options->getInt('port');                  // Still 993: the failed change did nothing.
```

Container Comparison

Feature	Bag	Vault	Store	Journal
Schema	No	Simple	JSON	No
Validation on every change	No	Yes	Yes	No
Defaults and normalizers	No	Yes	Defaults	No
Nested data	Yes	Yes	Yes	No
Dot notation	Yes	Yes	Yes	No
Sequential	No	No	No	Yes
Typed getters	Yes	Yes	Yes	Yes

Where to Go Next

- [Containers](#): Bag, Vault, Store and Journal one by one.
- [Typed Getters](#): read values with the type you expect.
- [Validation](#): schemas, when they are checked and what happens when they fail.
- [Translations](#): the exceptions of the package and how to translate them.

Last updated on 20/10/2026