

Operation Policy

Operation Policy

Anonymous · 10/10/2026

Operation Policy

Controlling Which Operations Can Be Dispatched:

OperationPolicyInterface

By default, any public method of a worker is dispatchable — the historical behavior, kept as `AllowAllOperationPolicy`, `DirectDispatcher`'s 4th constructor argument's default value. **This default is convenient, not recommended:** “any public method” includes infrastructure methods a worker gets from `JobsAwareTrait`/`HandlersAwareTrait`/`OptionsAwareTrait` (`getJobs()`, `setOptions()`, ...) and from `ServiceInterface` itself (`getId()`, `getName()`, `getDescription()`) — none of it business logic, all of it just as dispatchable as a real operation while this policy is active. `TaggedOperationPolicy` is the recommended choice for anything reached from outside PHP: it only allows what is explicitly tagged `#[Operation]`, which is the one real signal for “this is meant to be exposed” — reflection alone cannot tell “a public method that happens to exist” apart from “an operation” (see `AllowAllOperationPolicy`'s own docblock). Two other policies ship with the package, and swapping the one wired into `DirectDispatcher` is the only change needed — nothing else in the three-tier chain knows a policy exists:

```
use Derafu\BackboneDispatcher\Service\Policy\TaggedOperationPolicy;
use Derafu\BackboneDispatcher\Service\Policy\AllowListOperationPolicy;

// Only methods tagged with derafu/backbone's #[Operation] attribute.
$policy = new TaggedOperationPolicy($registry, $inspector);

// Only operations matching one of these ids. fnmatch() wildcards allowed,
// e.g. "billing.invoice.builder::*" for every operation of that worker,
// or "billing.*" for an entire package.
$policy = new AllowListOperationPolicy([
    'billing.invoice.builder::build',
    'billing.invoice.builder::cancel',
]);

$directDispatcher = new DirectDispatcher(
    $registry,
    $inspector,
    $resolver,
    $policy,
```

```
);
```

Two guards run in `DirectDispatcher::dispatch()`, in this order, before an operation is ever resolved or invoked — so every tier built on top of it enforces both without knowing they exist:

- Does the operation exist at all, as a public method declared on the worker? Independent of which policy is configured — throws `OperationNotFoundException` otherwise.
- Does the active `OperationPolicyInterface` allow it? Throws `OperationNotAllowedException` otherwise.

Explorer accepts the same `OperationPolicyInterface` as an optional constructor argument, so documentation never advertises an operation the dispatcher would then reject — without one, it lists everything, matching `AllowAllOperationPolicy`. With one, the pruning goes all the way up the tree: `getOperations()` drops operations it rejects, and `getWorkers()/getComponents()/getPackages()` drop any branch left with zero visible operations underneath — a worker nobody may call does not show up at all, not even as an empty entry.

Implementing `OperationPolicyInterface` yourself (a single `isAllowed()` method) covers any other rule — e.g. combining policies, or checking against the current user's permissions.

Last updated on 10/10/2026