

• TALK · 20 MIN

Services taggués & Compiler pass

UNE ÉTIQUETTE, POUR COMPRENDRE TOUT LE CONTAINER

Nicolas Loyer



Cas fictif : un service d'export

```
final class ExportService {  
    public function export(array $rows, string $format): string {  
        return match ($format) {  
            'csv'  => (new CsvExporter())->export($rows),  
            'json' => (new JsonExporter())->export($rows),  
            'xml'  => (new XmlExporter())->export($rows),  
            // et si un autre module veut ajouter 'pdf' ?  
            default => throw new \InvalidArgumentException($format),  
        };  
    }  
}
```

PHP

Notre cas : Des exporters sans configuration, faciles à lister, ordonner et manipuler lors du processus.



Chaque nouveau format

Il faut modifier cette classe



Des "new" en cascade

Aucune injection possible



Fermé aux autres modules

Impossible d'ajouter un format

Le vieux réflexe Drupal : un hook

```
// mon_module : on interroge les autres modules
$exporters = \Drupal::moduleHandler()
    → invokeAll('mon_module_exporter_info');
```

PHP

```
// autre_module/src/Hook/ExportHooks.php
namespace Drupal\autre_module\Hook;

use Drupal\Core\Hook\Attribute\Hook;

final class ExportHooks {

    #[Hook('mon_module_exporter_info')]
    public function exporterInfo(): array {
        return ['pdf' ⇒ new PdfExporter()];
    }
}
```

PHP



Extensible par n'importe quel module



Exécuté à chaque requête



Tous les exporters instanciés



Toujours des new à la main

Bien mais pas ouf : on veut la même extensibilité sans tout refaire à chaque requête.

Deuxième réflexe : un plugin

```
// 1. Un attribut pour déclarer les plugins
#[\Attribute(\Attribute::TARGET_CLASS)]
final class Exporter extends Plugin {}

// 2. Un manager qui les découvre (+ son service)
final class ExporterManager extends DefaultPluginManager {
    // parent::__construct('Plugin/Exporter', ..., Exporter::class)
}

// 3. Enfin, le plugin lui-même
#[Exporter(id: 'pdf')]
final class PdfExporter extends PluginBase implements
ExporterInterface {}
```

PHP



Extensible par n'importe quel module



Découverte mise en cache



Beaucoup de code avant le 1^{er} format



Pensé pour des composants configurables

Parfait pour des blocs. Lourd pour « consulter tous les exporters ». On y revient à la fin.

Et un service alors ?
(C'EST VRAI ÇA ! J'EN MET PARTOUT DE TOUTE MANIÈRE !)

A new player has entered!

[ZE CONTAINER!]

Same old drill, vous connaissez la musique

Le container de services est une **usine à objets** : elle sait comment construire chaque service et quoi lui injecter.

```
# mon_module.services.yml
services:
  mon_module.exporter.pdf:
    class: Drupal\mon_module\Export\PdfExporter
    arguments: ['@file_system']
```

YAML

```
// À éviter dans une classe
\Drupal::service('mon_module.exporter.pdf');

// À privilégier : on reçoit ses dépendances
public function __construct(private PdfExporter $exporter) {}
```

PHP

1

Service

Un objet géré par le container, identifié par un id

2

Injection

On reçoit ses dépendances, on ne va pas les chercher

3

create()

Contrôleurs et formulaires reçoivent le container

Les tags : vous en utilisez déjà

Un tag est une étiquette posée sur un service.

Il ne change rien au service : il sert juste à le retrouver.

Déjà dans vos services.yml :

• event_subscriber

• breadcrumb_builder

• access_check

• paramconverter

• cache_context

• theme_negotiator

```
# Les nôtres
mon_module.exporter.csv:
  class: Drupal\mon_module\Export\CsvExporter
  tags:
    - { name: mon_module.exporter, format: csv }

mon_module.exporter.json:
  class: Drupal\mon_module\Export\JsonExporter
  tags:
    - { name: mon_module.exporter, format: json }
```

YAML

Ok cool, mais ... comment on collecte nos services d'exporter ensuite ?

Collecter : la méthode historique (D8)

```
mon_module.export:
  class:
    Drupal\mon_module\Export\ExportService
  tags:
    - name: service_collector
      tag: mon_module.exporter
      call: addExporter
```

YAML



Pour chaque service taggué mon_module.exporter, appelle addExporter() avec ce service



Un nouveau format

C'est juste une classe + 3 lignes de YAML

```
final class ExportService {

    private array $exporters = [];

    public function addExporter(
        ExporterInterface $exporter,
        int $priority = 0,
    ): void {
        $this->exporters[] = $exporter;
    }

    public function export(array $rows, string $format):
    string {
        foreach ($this->exporters as $exporter) {
            if ($exporter->supports($format)) {
                return $exporter->export($rows);
            }
        }
        throw new \InvalidArgumentException($format);
    }
}
```

PHP

Collecter : la méthode moderne

```
# Depuis Drupal 10.3
mon_module.export:
  class: Drupal\mon_module\Export\ExportService
  arguments: [!tagged_iterator mon_module.exporter]
```

YAML

```
public function __construct(
    private readonly iterable $exporters,
) {}
// export() inchangé
```

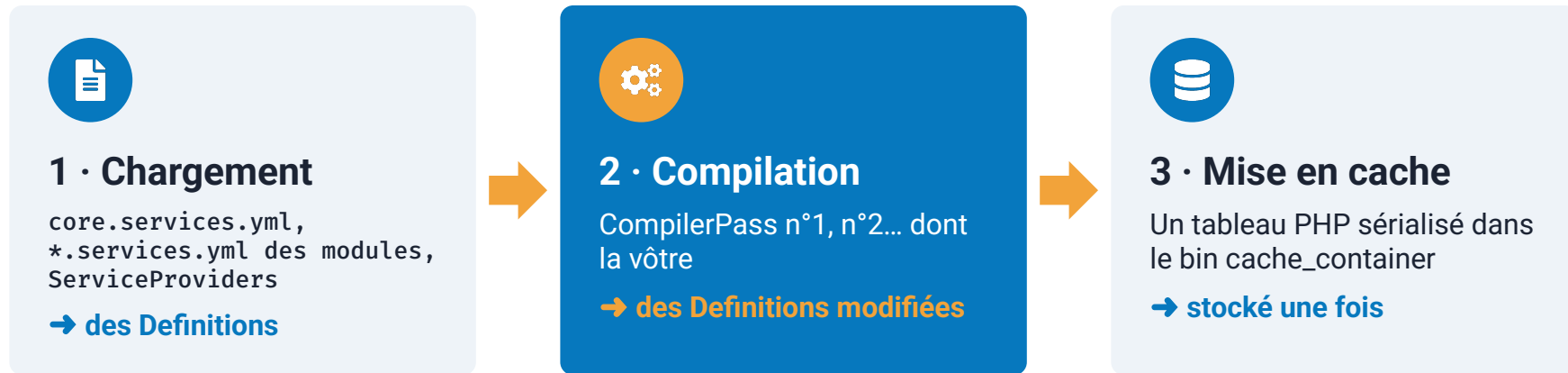
PHP



Lazy : un exporter n'est construit qu'au moment où on boucle dessus.

	service_collector	service_id_collector	!tagged_iterator
On reçoit	les objets, via une méthode	un tableau d'ids	un itérable
Instanciation	immédiate	à la demande (par vous)	à l'itération
Disponible	depuis Drupal 8	depuis Drupal 8	depuis Drupal 10.3

Sous le capot : la vie du container



Déclenché par : drush cr, installation ou désinstallation d'un module...

4 · Runtime chaque requête recharge ce tableau, sans rien recompiler

Deux classes, deux moments



Au drush cr : la compilation

`Drupal\Core\DependencyInjection\`
ContainerBuilder

étend
le ContainerBuilder de Symfony

- manipule des Definitions
- reçu par register() et process()



cache



À chaque requête : le runtime

`Drupal\Core\DependencyInjection\`
Container

étend
Drupal\Component\DependencyInjection\Container

- relit le tableau en cache
- derrière \Drupal::service() et create()

Symfony compile, Drupal exécute : en production, ce n'est pas le container de Symfony qui tourne.

Des recettes, pas des objets

Pendant la compilation, **aucun service n'existe encore**. Le container ne manipule que des Definitions.

```
$definition = new Definition(CsvExporter::class);
$definition->setArguments([new Reference('file_system')]);
$definition->addMethodCall('setDelimiter', [';']);
$definition->addTag('mon_module.exporter', ['format' => 'csv']);
```

PHP

En PHP	En YAML
new Definition()	une entrée de services.yml
new Reference('id')	'@id'
addMethodCall()	calls:
addTag()	tags:

Si votre service est **un gâteau** 🍰, sa Definition est sa **recette**.



Dans une CompilerPass

jamais de `$container->get()`, ni de `\Drupal::service()`, ni d'accès à la base ou à la config

Trucs vaudous

(ARRIVÉS ASSEZ RÉCEMMENT)

(QU'ON UTILISE TOUSTES SANS TROP SAVOIR COMMENT ÇA MARCHE)

(MAIS COMME ÇA MARCHE ALORS ON POSE PAS TROP DE QUESTIONS)

(ET PEUT ÊTRE QUE JE METS UN PEU TROP DE PARENTHÈSES...)

Ça vous parle ?

```
services:
  _defaults:
    autowire: true
    autoconfigure: true

mon_module.exporter.csv:
  class: Drupal\mon_module\Export\CsvExporter

mon_module.exporter.json:
  class: Drupal\mon_module\Export\JsonExporter
```

YAML

Autowire et autoconfigure



autowire

ce que le service reçoit

```
// Plus besoin d'écrire arguments:
// ['@file_system']
public function __construct(
    private FileSystemInterface $fileSystem,
) {}
```

PHP

- Plusieurs implémentations (caches, loggers) :
#[Autowire(service: ...)]
- Rien à deviner (iterable, chaîne) : arguments explicites



autoconfigure

ce qu'on dit du service

```
// MonModuleServiceProvider::register()
$container
→registerForAutoconfiguration(ExporterInterfa
ce::class)
→addTag('mon_module.exporter');
```

PHP

- Pose les tags, ne collecte rien
- Chaque module doit activer autoconfigure: true

Les deux agissent au drush cr : ce sont des CompilerPass qui complètent les recettes 🍰.

Écrire sa CompilerPass : le registre



Le besoin n°1

Un registre indexé par format :
get('pdf')



Le besoin n°2

Une erreur dès le drush cr si deux
modules déclarent le même format

```
# mon_module.services.yml
mon_module.exporter_registry:
  class:
    Drupal\mon_module\Export\ExporterRegistry
```

YAML

```
namespace Drupal\mon_module\Export;

final class ExporterRegistry {

    private array $exporters = [];

    public function add(
        string $format,
        ExporterInterface $e
    ): void {
        $this->exporters[$format] = $e;
    }

    public function get(string $format): ExporterInterface {
        return $this->exporters[$format]
            ?? throw new \InvalidArgumentException($format);
    }

}
```

PHP

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
    }  
}
```

PHP

1

La recette

pas l'objet registre

2

Les tags

id → liste d'attributs

3

add() ajouté

à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
  
        foreach ($container->findTaggedServiceIds('mon_module.exporter') as $id  
⇒ $tags) {  
            // La magie va apparaître ici...  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
  
        foreach ($container->findTaggedServiceIds('mon_module.exporter') as $id  
⇒ $tags) {  
  
            foreach ($tags as $attributes) {  
                // Une autre magie va apparaître ici...  
            }  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
  
        foreach ($container->findTaggedServiceIds('mon_module.exporter') as $id  
⇒ $tags) {  
  
            foreach ($tags as $attributes) {  
                $format = $attributes['format']  
                ?? throw new \LogicException("$id : attribut format manquant");  
            }  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
  
        foreach ($container->findTaggedServiceIds('mon_module.exporter') as $id  
⇒ $tags) {  
            foreach ($tags as $attributes) {  
                $format = $attributes['format']  
                ?? throw new \LogicException("$id : attribut format manquant");  
  
                if (isset($seen[$format])) {  
                    throw new \LogicException("Format $format déclaré deux fois");  
                }  
            }  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

Écrire sa CompilerPass : la pass

```
final class ExporterPass implements CompilerPassInterface {  
  
    public function process(ContainerBuilder $container): void {  
        if (!$container->hasDefinition('mon_module.exporter_registry')) {  
            return;  
        }  
        $registry = $container->getDefinition('mon_module.exporter_registry');  
        $seen = [];  
        foreach ($container->findTaggedServiceIds('mon_module.exporter') as $id  
⇒ $tags) {  
            foreach ($tags as $attributes) {  
                $format = $attributes['format']  
                ?? throw new \LogicException("$id : attribut format manquant");  
                if (isset($seen[$format])) {  
                    throw new \LogicException("Format $format déclaré deux fois");  
                }  
                $seen[$format] = $id;  
                $registry->addMethodCall('add', [$format, new Reference($id)]);  
            }  
        }  
    }  
}
```

PHP

1

La recette
pas l'objet registre

2

Les tags
id → liste d'attributs

3

add() ajouté
à la recette du
registre

L'enregistrer avec un ServiceProvider

```
// src/MonModuleServiceProvider.php PHP
namespace Drupal\mon_module;

use Drupal\Core\DependencyInjection\ContainerBuilder;
use Drupal\Core\DependencyInjection\ServiceProviderInterface;

final class MonModuleServiceProvider implements
ServiceProviderInterface {

    public function register(ContainerBuilder $container): void {
        $container->addCompilerPass(new ExporterPass());
    }
}
```

Dans le cache (schématisé)

```
'mon_module.exporter_registry' => [
    'class' => '...\ExporterRegistry',
    'calls' => [['add', ['csv', @exporter.csv]], ...],
],
```



Par convention

mon_module →
MonModuleServiceProvider, à
la racine de src/. Rien à
déclarer.



Mal nommé ?

Ignoré en silence : aucune
erreur.

Drupal le fait déjà partout

Ce que vous écrivez	Ce qui travaille au drush cr
<code>tags: [{ name: event_subscriber }]</code>	une CompilerPass enregistre vos subscribers
<code>tags: [{ name: service_collector, ... }]</code>	TaggedHandlersPass
<code>#[Hook('form_alter')] (Drupal 11.1+)</code>	HookCollectorPass

Quand écrire le vôtre ?



Valider

doublons, attribut manquant



Construire

une map calculée une fois, des données typées à ordonner, etc.



Modifier

des services selon le contexte

Sinon, !tagged_iterator ou service_collector suffisent.

Un autre exemple pratique

(PARCE QU'IL N'Y A PAS QUE LES EXPORTERS DANS LA VIE)

(Y A LE FROMAGE AUSSI... AUCUN RAPPORT)

● EN PRATIQUE

1

Deux panneaux de recherche en un

Tout doit être accessible HTMX

2

Des zones de contenus communs

boutons, listes de contenus

3

Pas de configuration

On a juste besoin d'un truc facile à maintenir

Rechercher un contenu ou choisir un territoire ⓘ

🔍 Elec| ✕

📍 Territoires 📊 Graphiques ⓘ Décryptages 🔑 Services

☐ Commune
Belecausse-et-Malin

☒ Commune
Belecausse Choisir ce territoire

☐ Commune
Saint-Elec

Thématique
Électrification

📁 Secteur d'activité
Entreprises d'électricité

📊 Consommation d'électricité
5 graphiques

📊 Bornes de recharge pour véhicules électriques
5 graphiques ⚡ Indisponible ⚠

📍 France
Que faut-il savoir sur l'électrification [en territoire]

📍 France
En 2050, à quoi pourraient ressembler la consommation et la production d'électricité ?

📍 Service 📍 Nouvelle Aquitaine
Bilan de mon territoire
PCAET-Loi APER

Tous les résultats

Choisir un territoire ⓘ

🔍 Rechercher une commune, un quartier, une région... ✕

Disponibles : Commune · EPCI · Département · Région · France

📍 Territoires ✕ 📍 CA le Grand Périgueux ⓘ📍 Comparer

France > Nouvelle Aquitaine > Dordogne >

🕒 Récents ✕

📍 EPCI
CA Le Grand Périgueux Accueil du territoire

📍 EPCI
CC Causses et Vallée de la Dordogne

Territoires similaires [Voir sur la carte →](#)

📊 Parc résidentiel similaire 5 > ⓘ

📍 EPCI
CC Nom d'EPCI

📍 EPCI
CA Saint-Nicolas-Fleury-Mérozy

📊 Parc résidentiel similaire 5 > ⓘ

📍 EPCI
CC Nom d'EPCI

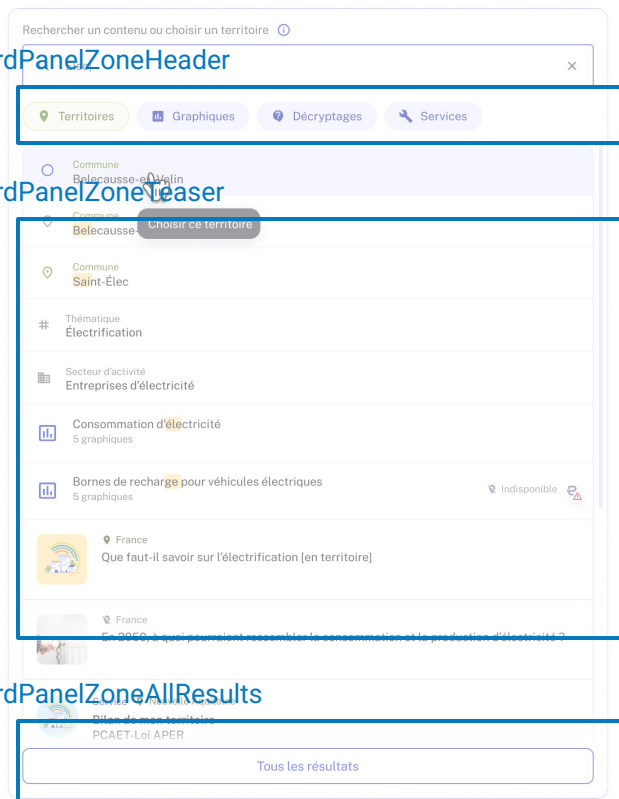
📍 EPCI
CA Saint-Nicolas-Fleury-Mérozy

Tous les résultats

SearchStandardPanelZoneHeader

SearchStandardPanelZoneLeaser

SearchStandardPanelZoneAllResults

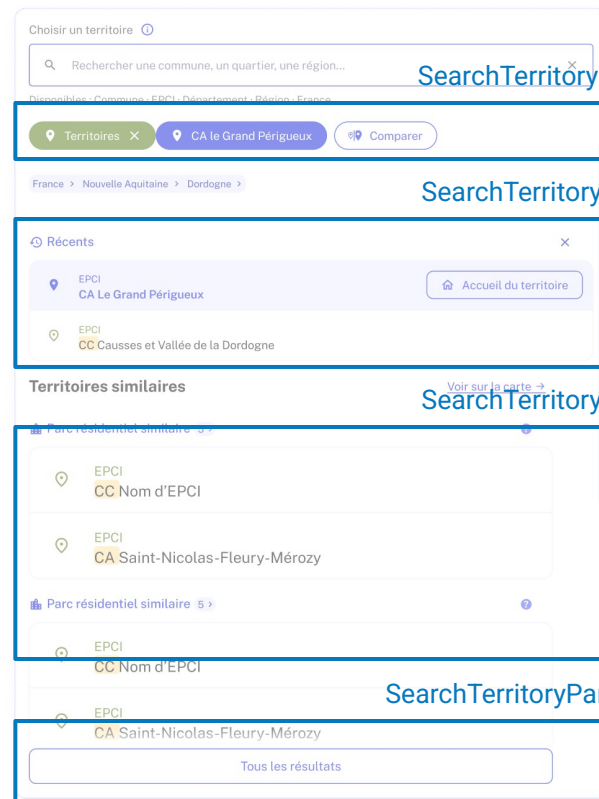


SearchTerritoryPanelZoneHeader

SearchTerritoryPanelZoneRecent

SearchTerritoryPanelZoneSimilar

SearchTerritoryPanelZoneAllResults



• EN PRATIQUE

YAML

```
my_search.standard_panel_zone.header:
  class:
    Drupal\my_search\Service\SearchStandardPanelZone\SearchStandardPanelZoneHeader
  tags:
    - name: my_search.standard_panel_zone
      weight: 1

my_search.standard_panel_zone.teaser:
  class:
    Drupal\my_search\Service\SearchStandardPanelZone\SearchStandardPanelZoneTeaser
  tags:
    - name: my_search.standard_panel_zone
      weight: 5

my_search.standard_panel_zone.allresults:
  class:
    Drupal\my_search\Service\SearchStandardPanelZone\SearchStandardPanelZoneAllResults
  tags:
    - name: my_search.standard_panel_zone
      weight: 10

# etc.
```

Services taggués & Compiler pass

PHP

```
class SearchStandardPanel implements
SearchPanelInterface {

    public function
addZone(SearchPanelZoneInterface $zone): void {
    $this->zones[] = $zone;
    }

    public function getPanel(): array {
        $sorted = $this->zones;
        usort($sorted, static fn($a, $b) =>
$a->getWeight() <=> $b->getWeight());
        // Autres opérations ici.
        $build["zone_{$index}"] = $zone->build();
    }
}
```

PHP

```
class MySearchServiceProvider extends
ServiceProviderBase {

    public function register(ContainerBuilder
$container): void {
        $container->addCompilerPass(new
SearchTerritoryPanelZonePass());

        $container->addCompilerPass(new
SearchStandardPanelZonePass());
    }
}
```

```

class MySearchStandardPanelZonePass implements PHP
CompilerPassInterface {
    public function process(ContainerBuilder
$container): void {
        if
(! $container->hasDefinition('my_search.standar
d_panel')) {
            return;
        }

        $definition =
$container->getDefinition('my_search.standard_
panel');

        foreach
(array_keys($container->findTaggedServiceIds('
my_search.standard_panel_zone')) as $id) {

            $definition->addMethodCall('addZone',
[new Reference($id)]);
        }
    }
}

```

```

// Dans un controller... PHP
$build =
$this->searchTerritoirePanel->getPanel();

$html = (string)
$this->renderer->renderInIsolation($build);

```

Avantages multiples :

- des zones de contenus “typées”
- faciles à ordonner, enrichir et maintenir
- elles peuvent être récupérées via des controllers HTMX pour rafraîchir uniquement certaines zones

Service taggué ou plugin ?

	Service taggué	Plugin
Découverte	tag dans services.yml	attribut PHP ([Block]...)
Instances	une seule, partagée	une par configuration
Configurable dans l'UI	non	oui
Exemples du core	breadcrumb builders, access checks, event subscriber	blocs, formateurs, widgets



Plugin

l'admin choisit et configure un composant



Service taggué

le code interroge tous les candidats

Debug et pièges classiques

```
# Après chaque modification
drush cr

# Lister les services (module Devel)
drush devel:services mon_module

# Vérifier ce que fournit le container
drush php:eval "var_dump(get_class(
  \Drupal::service('mon_module.exporter_registry')));"
```

BASH



drush cr oublié
le cache sert l'ancien
container



Faute dans le tag
aucune erreur,
service ignoré



Provider mal nommé
ignoré en silence



**Config dans une
CompilerPass**
pas disponible

À retenir

- 1 Un tag est une étiquette dans services.yml
- 2 service_collector, service_id_collector et !tagged_iterator les collectent
- 3 Le container est compilé une fois, au drush cr, puis mis en cache
- 4 Pendant la compilation : des Definitions (des recettes), pas des objets
- 5 Une CompilerPass, enregistrée par un ServiceProvider, modifie ces recettes

Avec des tags, vous pouvez catégoriser vos services pour mieux les maintenir.

Des questions ?