{ Drupal 8 }

{ Drupal 8 }

Or, How I Learned to Stop Worrying and Love the Service

What Has Changed?

- PSR-4
- Symfony
- Everything is a Service
  - Except when it’s a Plugin
  - Or a Utility
  - Or a Hook
  - Or something else
- Instead of Menus we have Routes and Controllers (and Links)
- There is caching (it’s complicated)
- Everything is registered with YAML
  - Except when it uses Annotations
  - Or info Hooks
- Composer and Console and Config (oh my!)

PSR-4

- The current standard for auto-loading classes in PHP
  - Forget about PSR-0

I want to load my utility class called SpecialString. It lives in:

```
/modules/custom/hello_module/src/Utility/SpecialString.php
```

And it has the namespace declaration:

```
namespace \Drupal\hello_module\Utility;
```

So I can include it in another file with:

```
use \Drupal\hello_module\Utility\SpecialString;
```

- The important bits are the namespace and file path
- The autoloader fills in the intermediate parts, and will look in other places
  - e.g `core/modules/module_name/src/..`

PSR-4

- If you want to see how this works, check out `autoload_real.php`
- Executed via Composer in the `/vendor` directory
- Some Drupal secret sauce makes the paths work seamlessly
- There are some conventions that make it easier to navigate Drupal code
  - Plugins generally have their own subdirectory e.g. `src/Plugin/Block`
- You can probably guess what goes in these directories
  - `src/Annotation`
  - `src/Controller`
  - `src/Entity`
  - `src/Form`
  - `src/Plugin/FieldFormatter`

Symfony

- Symfony works under the hood
- Drupal 8 uses some core Symfony concepts:
    - Services
    - Controllers
    - Routes
    - Config
    - Events
- Some related concepts and libraries:
    - Dependency Injection
    - Annotations
    - Twig
- And some things that are layered on top:
    - Plugins

Services

- Largely replace hooks for specific functionality
  - Sending emails
  - Caching
  - Storage
  - Logging
  - Serialization
- You can see core Services (and Plugins) in `core.services.yml`
- Should always be accessed via the service container
  - `\Drupal::service('date')`
  - `\Drupal::translation()`
- These are fairly standard Symfony components, so the docs are useful

Plugins

- Largely replace hooks for specific functionality **with a user-configurable GUI**
  - Sending emails (I lied)
  - Blocks
  - Field Formatters
  - Views components
- Registered and discovered with Annotations
  - These use a standard syntax (Doctrine) with custom formats
  - You can look these up for each Plugin type if you need a reference
- A Plugin implementation requires
  - Annotation definition
  - Plugin Manager (and interface), to deal with management and DI
  - Plugin Base (and interface)
- The core Block code in `core/lib/Drupal/Core/Block` is a good example

Components and Utilities

- Some really basic functionality is provided as core classes
  - Xss
  - Image
  - Number
  - Bytes
  - Unicode
  - DateTime
  - Transliteration
  - Diff
  - etc.
- You can see these in `core/lib/Drupal/Component`
- I'm unclear as to how overridable these are

Events

- Another core Symfony concept
- A small set of core Events are available
  - https://api.drupal.org/api/drupal/core!core.api.php/group/events
- Allow Services to dispatch events when things happen
- You can create your own Service to react to them
- This is used in core to handle things like
  - Rendering the page to the user
    - `/core/lib/Drupal/Core/MainContentViewSubscriber`
  - Logging Exceptions
    - `/core/lib/Drupal/Core/ExceptionLoggingSubscriber`
  - Altering Routes
    - `/core/lib/Drupal/Core/EventSubscriber/EntityRouteAlterSubscriber`

Hooks

- They still exist in core
  - https://api.drupal.org/api/drupal/core!core.api.php/group/hooks
- Altering primarily still uses Hooks (rather than, say, Events)
- This is the source of a lot of ‘messy’ code
  - Mixing of old and new paradigms
  - Don’t have convenient ways to access relevant Services
  - Don’t have convenient methods a Service or Plugin would provide
- These will still generally live in a `.module` file or an include

Routes

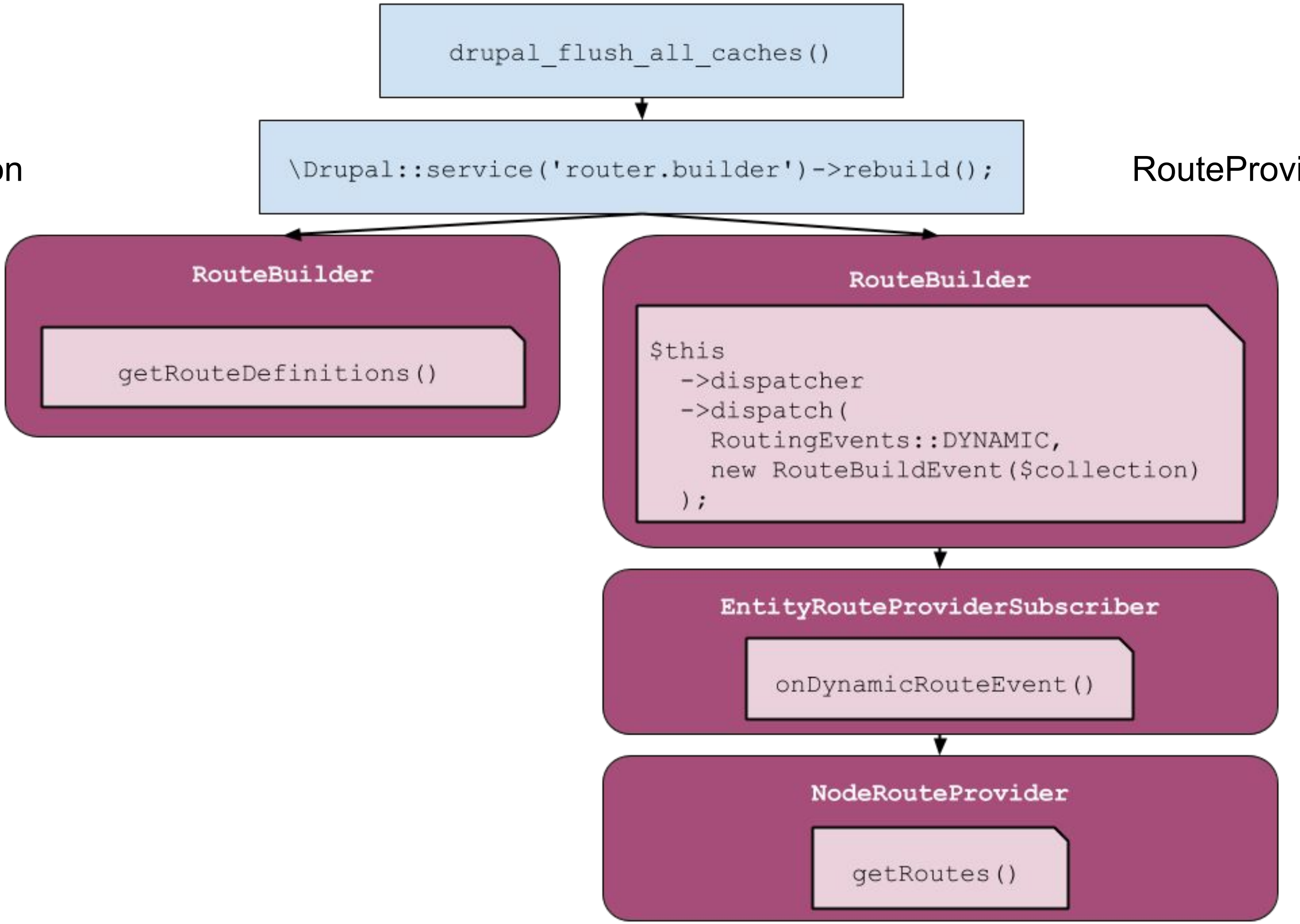
- The actual URL paths and the behaviour associated with them are much more separate in Drupal 8
- Routes manage the URL paths
- They're generally defined in `module_name.routing.yml`
- They still do some of the magic that you got in Drupal 7
- There are other ways of defining routes e.g. entity.{type}.canonical
    - See `\Drupal\Core\Entity\Routing\DefaultHtmlRouteProvider.php`
    - For nodes we have `NodeRouteProvider.php`

Routes

- Manual route discovery is a bit confusing, because routes are either defined:
  - In a `module_name.routing.yml` file
  - In a `RouteProvider` class
- However, they're all rebuilt via the core `RouteBuilder`
  - This is the `route.builder` service, and can be invoked with `\Drupal::service('router.builder')->rebuild()`
- Routes defined in `*.routing.yml` are handled directly in `RouteBuilder`
  - `RouteBuilder->rebuild()` then rebuilds all routes via `getRouteDefinitions()`
  - It also dispatches the events that `RouteSubscribers` can respond to
- A `RouteProvider` is invoked by a `RouteSubscriber`
  - For entities this is the `EntityRouteProviderSubscriber`
  - `onDynamicRouteEvent()` invokes the defined route provider
  - This `RouteProvider` is provided as part of the Entity Plugin annotation
  - See `\Drupal\core\modules\node\src\Entity\Node.php`

YAML Definition

RouteProvider Definition



```
drupal_flush_all_caches()
```

```
\Drupal::service('router.builder')->rebuild();
```

```
getRouteDefinitions()
```

```
$this
  ->dispatcher
  ->dispatch(
    RoutingEvents::DYNAMIC,
    new RouteBuildEvent($collection)
  );
```

```
onDynamicRouteEvent()
```

```
getRoutes()
```

Routes

- Still have a lot of the same magic properties as in Drupal 7
  - Wildcards
  - Named placeholders
- Some new functionality
  - If a placeholder has the same name as an entity type, will try and upcast it and pass it on
  - See https://www.drupal.org/node/2122223
- There’s a lot of complexity you can add
- Not something I’ve done much with
  - https://www.drupal.org/docs/8/api/routing-system

Controllers

- Deliver the content from a Route
  - Replace menu callbacks from Drupal 7
- Pretty simple—most of the complexity exists in the Route definition

```yaml
example.content:
  path: '/example'
  defaults:
    _controller: '\Drupal\example\Controller\ExampleController::content'
    _title: 'Hello World'
  requirements:
    _permission: 'access content'
```

```php
namespace Drupal\example\Controller;

use Drupal\Core\Controller\ControllerBase;

/**
 * An example controller.
 */
class ExampleController extends ControllerBase {

  /**
   * {@inheritdoc}
   */
  public function content() {
    $build = [
      '#markup' => t('Hello World!'),
    ];
    return $build;
  }

}
```

```php
namespace Drupal\dino_roar\Controller;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Logger\LoggerChannelFactoryInterface;
use Drupal\dino_roar\Jurassic\RoarGenerator;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Symfony\Component\HttpFoundation\Response;

/**
 * Class RoarController.
 */
class RoarController extends ControllerBase {

  private $roarGenerator;
  protected $loggerFactory;

  public function __construct(RoarGenerator $roarGenerator, LoggerChannelFactoryInterface $loggerFactory) {
    $this->roarGenerator = $roarGenerator;
    $this->loggerFactory = $loggerFactory;
  }

  public static function create(ContainerInterface $container) {
    /* @var $roarGenerator \Drupal\dino_roar\Jurassic\RoarGenerator */
    $roarGenerator = $container->get('dino_roar.roar_generator');

    /* @var $loggerFactory \Drupal\Core\Logger\LoggerChannelFactoryInterface */
    $loggerFactory = $container->get('logger.factory');

    return new static($roarGenerator, $loggerFactory);
  }

  public function roar($count) {
    $roar = $this->roarGenerator->getRoar($count);
    $this->loggerFactory->get('default')->debug($roar);
    return new Response($roar);
  }
}
```

Menu Links

- A mechanism to associate URLs with routes
- This is much more flexible than the Drupal 7 system
  - Easy to point multiple URLs at the same route
  - Routes do not have to follow the same hierarchy as links
- Various kinds of links that can be defined
  - Similar to `type` from `hook_menu` definition
- Not something I’ve done much with
  - https://www.drupal.org/docs/8/api/menu-api/comparison-of-menu-api-in-drupal-7-and-8

Caching

- Instead of cache bins we now have three ways to manage render caching:
  - **Tags** (these items relate to a particular node)
  - **Contexts** (this item relates to a particular theme)
  - **Max-age** (this should only be cached for 10 minutes)
- **Much** easier to clear related cache items by tag
  - This is why caching is now enabled by default
  - Caching information is also exposed to external caches
- To turn off caching, you can replace the cache backend service
  - Drupal provides configuration you can enable to do this
  - You can toggle this using Console with `drupal site:mode`
  - See https://www.drupal.org/node/2598914
- This is one of the parts I'm less familiar with
  - https://www.drupal.org/docs/8/api/cache-api/cache-api
  - https://dri.es/making-drupal-8-fly

YAML

- YAML is now the markup language of choice
  - Replaces the .ini format used in Drupal 7
- Used in lots of places for registering code or functionality
  - `*.services.yml`
  - `*.libraries.yml`
  - `*.routing.yml`
  - `*.links.*.yml`
- Not used for registering Plugins
  - There are lots of discussions about Annotations vs. other formats
  - The main justification for not using a separate YAML manifest is to keep the metadata in the same file as the Plugin
  - Personally I'd prefer this to be implemented in the same way as Services for consistency
- It is theoretically possible to use YAML-based discovery if you want to

Composer

- Modules are now managed via composer
  - This also takes care of PHP package dependencies
  - This does not manage non-PHP libraries (e.g. a WYSIWYG editor)
- You can install modules from drupal.org with:
  - `composer require drupal/module_name`
- You can install specific versions or ranges with semantic versions:
  - `composer require drupal/module_name:1.2.3`
  - `composer require drupal/module_name:~1.0`
- The documentation is pretty good
  - https://www.drupal.org/docs/develop/using-composer
  - https://www.drupal.org/docs/develop/using-composer/using-composer-to-manage-drupal-site-dependencies
- Don’t use drupal-composer-init unless you have a good reason

Console

- This does not replace Drush (but does overlap with it)
  - A lot of the core commands have a synonym, but the contrib ones don't
- Similar to the Symfony console
- Does useful things like:
  - Code scaffolding with `drupal generate:*`
  - Debugging of code definitions with `drupal debug:*`
  - Dummy content generation with `drupal create:*`
- It also does some things Drush does:
  - `drupal site:status`
  - `drupal site:maintenance`
- Requires both a global and per-site installation
- See the documentation for many more commands
  - https://hechoendrupal.gitbooks.io/drupal-console/content/en/commands/available-commands.html

A Note on Drush

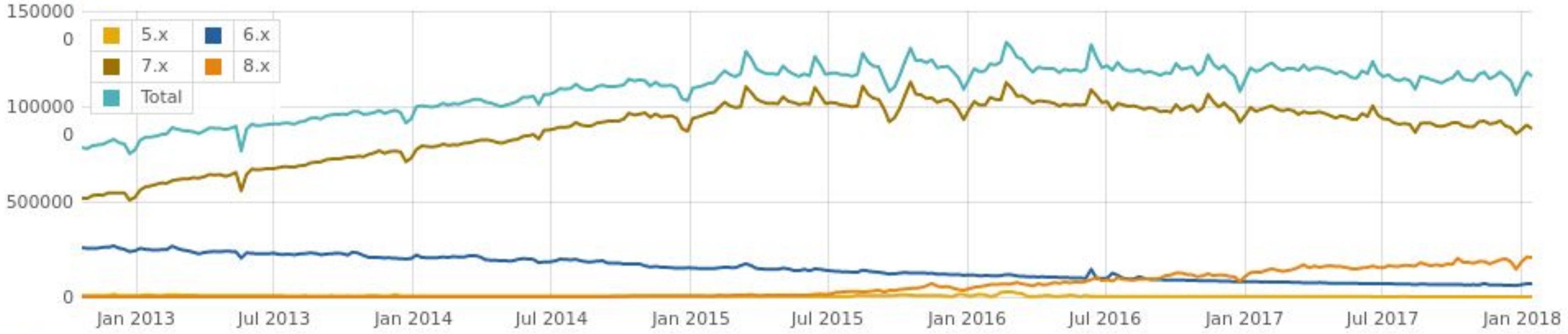
- There is now a stable release of Drush 9
- This does not work with Drupal 7
- You’ll need to either have both versions available, or stick with Drush 8 for now
  - Multi-versioning is probably preferable if you’re working with Drupal 8
  - https://www.lullabot.com/articles/switching-drush-versions

Config

- A core mechanism to define complex, exportable configuration
- This exists in conjunction with State
    - State effectively replaces Variables for ephemeral, per-env settings
    - Config is for permanent, exportable, cross-env settings
- Config can be exported from core as `.yml` files
    - It’s quite a monolithic process
- Would strongly encourage the use of Config Filter and Config Split
- Requires a level of process to prevent tricky merging
- Effectively deprecates Features

Drupal 7

- Will continue to get security fixes until Drupal 9 enters LTS
    - **Drupal 6.0** February 13th 2008
    - **Drupal 7.0** January 5th 2011
    - **Drupal 8.0.0** November 19th 2015
    - **Drupal 9.0.0** 2019/2020?
- It’s not dead yet



My Take

- Drupal 8 struggled to get a stable release
  - It’s still having issues two years later
  - Adoption is pretty poor, considering how quick the D7 uptake was
  - It feels like they severely underestimated the amount of onboarding needed for the development community
- There’s a lot of good stuff
  - But some of it is bloody complicated or lacks decent documentation
- Issues are somewhat understandable given the level of architectural changes
- The main problem is that you’re forced to think in two paradigms
  - It’s the offspring of Symfony and Drupal 7
- I think we’ll see the shift in D8>D9 as we did in D6>D7
  - No major architectural changes, but a solidification of the core concepts
  - Unfortunately this is a long way off

The End