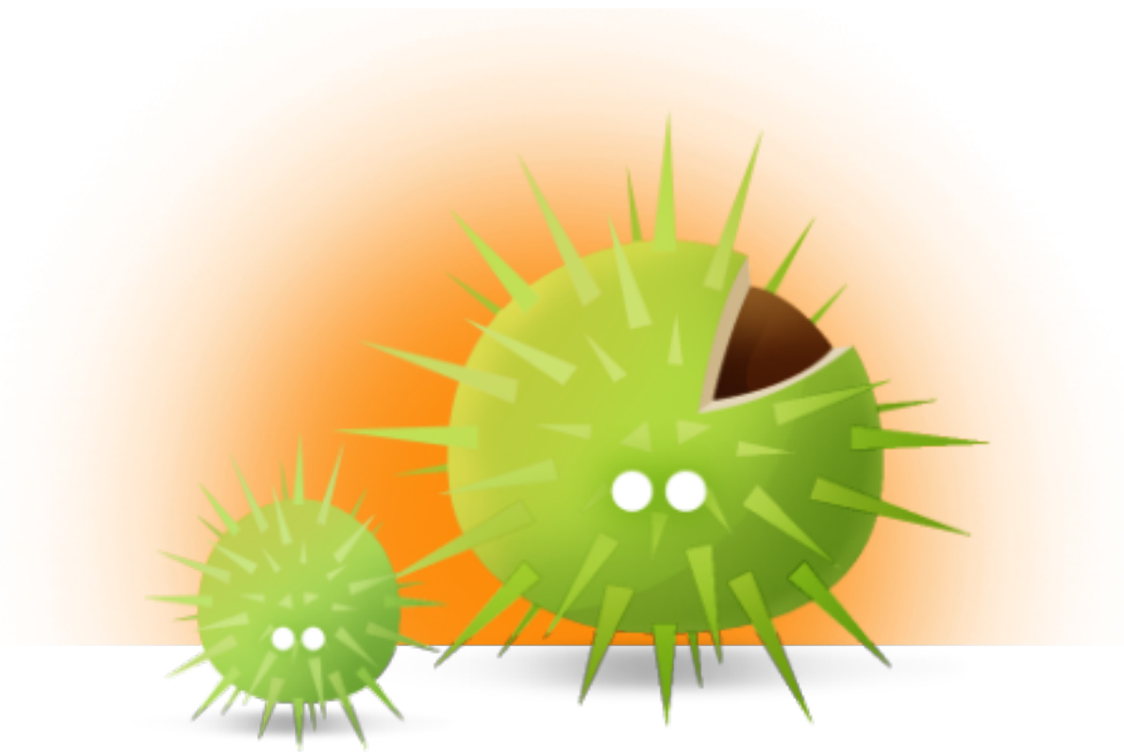
Inventing The Container

Or: WTH Is A Container And Why Should I Care?

(Not a Docker talk. :-)

Paul Mitchum
Mile23 on drupal.org
@PaulMitchum
@DrupalExamples



Caveat

- This is NOT A DOCKER TALK. :-)
- Code is pseudocode.
- We will move very quickly through the material.

What We'll Do

- Define Some Terms.
- Design a router and a database service.
- Make Mistakes.
- Learn About Dependency Injection.
- Learn How To Make A Container.
- See It In Action With Pimple.
- Explore Container Awareness.

What Is A Container?

- The thing with the services.

No Really: What Is A Container?

- Software architecture that enables *systematic* inversion of control across the application.

- Restated: A *design decision* to invert control, which has been implemented as an object in your application.

- You're deciding that all the stuff in your application will be a service. (Example: Silex)

What Is Inversion Of Control?

- A strategy of injecting dependencies so that behavior is not magical.
- `$result = someFunction(// MAGIC!);`
- `$result = someFunction(new Dependency());`

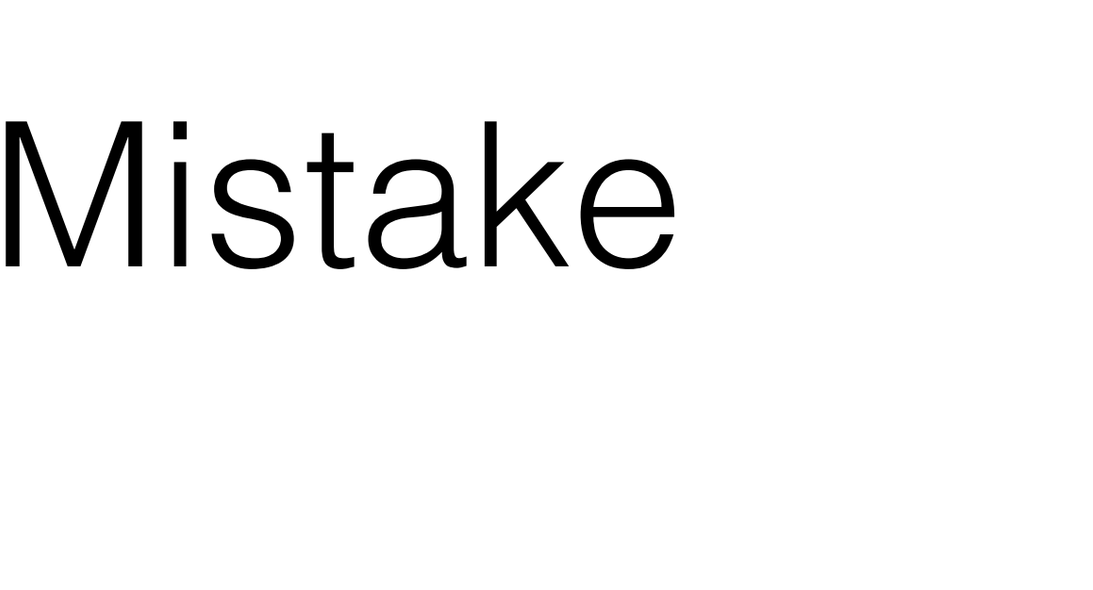
What Is Dependency Injection?

- Passing dependencies as parameters.

- (See previous slide.)

Our First Mistake

```
// index.php

return handle();

function handle() {
    $path = $__SERVER['PATH_INFO'];
    if ($path == 'about') {
        return aboutPage();
    }
    if ($path == 'blog') {
        return blogPage();
    }
}
```



Solving The Mistake…

```
// index.php

return handle($__SERVER['PATH_INFO']);

function handle($path) {
   if ($path == 'about') {
      return aboutPage();
   }
   if ($path == 'blog') {
      return blogPage();
   }
}
```

Dependency Injection

What About These?

```
// index.php

return handle($__SERVER['PATH_INFO']);

function handle($path) {
   if ($path == 'about') {
      return aboutPage();
   }
   if ($path == 'blog') {
      return blogPage();
   }
}
```

Inject Callables?

```
function handle($path,
   callable $about,
   callable $blog) {
  if ($path == ‘about’) {
     return $about();
   }
   if ($path == ‘blog’) {
     return $blog();
   }
}
```

Inversion of Control (IoC)

Better… Maybe?

(No.)

The Downside…

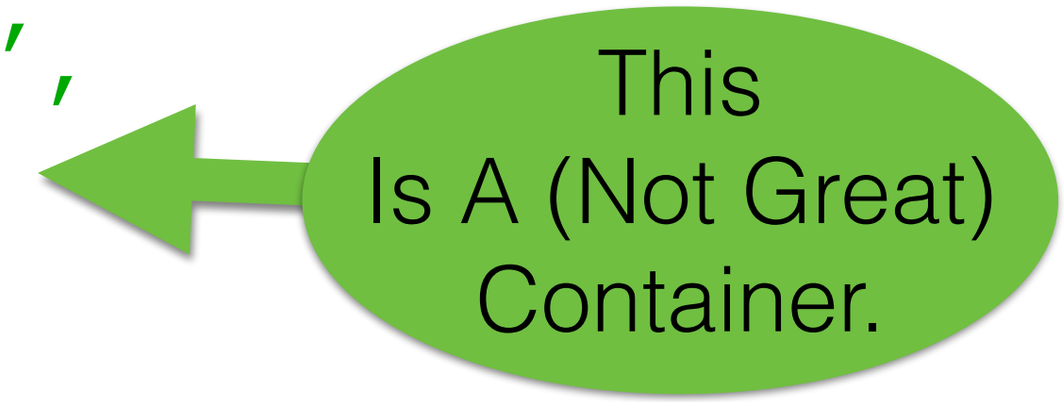
```
return handle($global_path,
  ‘aboutPage’, ‘blogPage’);

function handle($path,
   callable $about,
   callable $blog) {
  if ($path == ‘about’) {
     return $about();
   }
   if ($path == ‘blog’) {
     return $blog();
   }
}
```

Disorganized. How Many Will We Need?

Unclear. Why are we passing callables?

We Make A Router

```
$routes = [
   ‘about’ => ‘aboutPage’,
   ‘blog’ => ‘blogPage’,
];
```




```
return handle($global_path, $routes);

function handle($path, array $container) {
   if (isset($container[$path]) {
      $route_function = $container[$path];
      return $route_function();
   }
}
```

Finish The Job..?

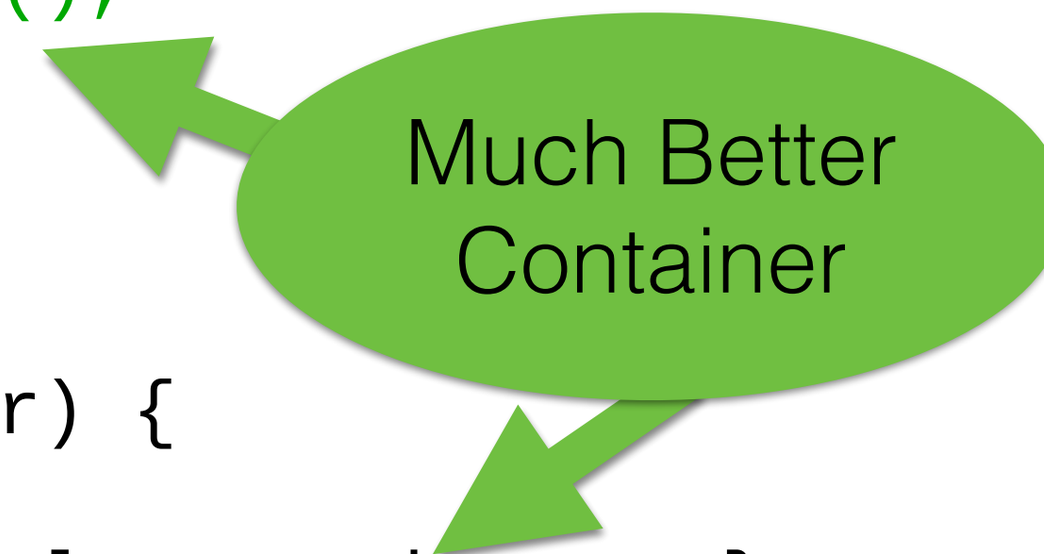
```
$container = [
   ‘path’ => $__SERVER[‘PATH_INFO’],
   ‘about’ => ‘aboutPage’,
   ‘blog’ => ‘blogPage’,
];

return handle($container);

function handle(array $container) {
   $path = $container[‘path’];
   $route_function = $container[$path];
   return $route_function();
}
```

This Is (Still) A (Bad) Container.

Because…

Router Is A Service

```
$container = [
   ‘path’ => $__SERVER[‘PATH_INFO’],
   ‘router’ => new RouterService(),
];

return handle($container);

function handle(array $container) {
   $path = $container[‘path’];
   if ($path == ‘foo’) { //Special cases here.. }
   $route_service = $container[‘router’];
   return $route_service->route($path);
}
```



Always Initialized

```
$container = [
  ‘path’ => $__SERVER[‘PATH_INFO’],
  ‘router’ => new RouterService(),
];

return handle($container);

function handle(array $container) {
  $path = $container[‘path’];
  if ($path == ‘foo’) { exit(1); }
  $route_service = $container[‘router’];
  return $route_service->route($path);
}
```

Ewps!

Let’s Try A Database

- Requirements:
  - A service to load database credentials from a file: connection.yml
  - A service to create a PDO connection using the credentials service.

Mixed Results…

```
$container['db.credentials'] =
  loadDbCredentials('connection.yml');
$container['database'] =
  new PDO(
    $container['db.credentials']
  );

handle($container);


function handle(array $container) {
  // Might not use the database.
}
```

Always Gets Initialized!

Dependency

Use Factory Functions

```
$container['db.credentials'] =
  function () {
    return loadDbCredentials('con
  };
$container['database'] =
  function ($container) {
    return new PDO(
      $container['db.credentials'
    );
  }

$pdo_factory = $container['database']
$pdo = $pdo_factory($container);
```

Defers Service Creation

Still Re-Initializes

One-Time Factory Functions

```
$container['db.credentials'] =
  function ($c) {
    $c['db.credentials'] =
      loadDbCredentials('connection.yml');
    return $c['db.credentials'];
  };
$container['database'] =
  function ($c) {
    $c['database'] =
      new PDO($c['db.credentials']);
    return $c['database'];
  };

$pdo = $container['database'];
```

Initialized Once

http://pimple.sensiolabs.org/

Pimple

There Has To Be A Framework...

```
use Pimple\Container;

$container = new Container();

// One-Time Service Factories
$container[‘db.credentials’] =
  $container->factory(function ($c) {
    return loadDbCredentials(‘connection.yml’);
  });
$container[‘database’] =
  $container->factory(function ($c) {
    return new \PDO($c[db.credentials]);
  });
```

The Whole Thing

```
use Pimple\Container;
$container = new Container();
$container[‘request’] =
  $container->factory(function ($c) {
    return Request::createFromGlobals();});
$container[‘router’] =
  $container->factory(function($c) {
    return new RouterService($c);};
$container[‘db.credentials’] =
  $container->factory(function ($c) {
    return loadDbCredentials(‘connection.yml’);};
$container[‘database’] =
  $container->factory(function ($c) {
    return new \PDO($c[db.credentials]);};

function handle(Container $c) {
  $path = $c[‘request’]->getPath();
  if ($path == ‘foo’) { exit(1); }
  return $c[‘router’]->route($path);
}
```

Magic

Psst… Magic Is BAD.

```
$container[‘router’] =
  $container->factory(function($container) {
    return new RouterService($container);
  });

function handle(Container $c) {
  $c[‘router’]->route(?);
}
```

Router could use ANY service!

Let’s Unmagic It

UnMagic v.1

```
$container[‘router’] =
  $container->factory(function($c) {
   return new RouterService($c[‘request’]);
  });

function handle(Container $c) {
  // Special business logic here.
  // ..
  return
    $c[‘router’]->route();
}
```

UnMagic v.2

```
$container[‘router’] =
  $container->factory(function($c) {
    return new RouterService();
  });

function handle(Container $c) {
  // Special business logic here.
  // ..
  return
    $c[‘router’]->route($c[‘request’]);
}
```

Automatic UnMagic

```
class RouterService {
  protected $request;

  public _constructor(Request $r) {
    $this->request = $r; }

  public static create(Container $c) {
    return new static($c[‘request’]);
  }
}

$container[‘router’] = function($c) {
  return RouterService::create($c);
};
```

Automatic UnMagic

Type Hinting Throws Errors

ContainerAware pattern

```
class R          Service {
  public _constructor(Request $r) { … }
  public static create(Container $c) {
    return new static($c[‘request’]);
  }
}
```

Requires Design Phase

Simplified Service Definition

```
$container[‘router’] = function($c) {
  return new RouterService::create($c);
};
```

Real-World Example

- App definition in drupalci_api.

Real-World Example

- Services definition file in Drupal 8.
- Requires compilation.
- Cached to avoid re-compliation.

Summary

- Defined some terms: Container, IoC, DI
- Made mistakes.
- Learned about dependency injection, IoC strategies.
- Learned How To Make A Container with one-time factory functions.
- Saw it in action with Pimple.
- Explore Container Awareness.
- Saw Silex and D8 examples.

Inventing The Container

Or: WTH Is A Container And Why Should I Care?

Paul Mitchum
Mile23 on drupal.org
@PaulMitchum
@DrupalExamples