Feature Flags with Symfony

SymfonyCon Madrid 2014

Benjamin Eberlei <benjamin@qafoo.com>

27.11.2014

Qafoo
passion for software quality

Me

- Working at Qafoo

passion for software quality

We promote high quality code with trainings and consulting

`http://qafoo.com`

- Doctrine and Symfony Contributor
- Blogging at `www.whitewashing.de`
- Twitter `@beberlei` and `@qafoo`

Outline

Introduction

```php
<?php

if (is_feature_enabled('billing')) {
    handle_billing_form();
} else {
    handle_beta_form();
}
```

One Year later

```
<?php

if (is_fea[illegible]enabled('[illegible]ng')) {
    /* if (i[illegible]ture_en[illegible]d('billing_provider_foo')) {
        hand[illegible]oo_bi[illegible]g();
    }*/
    if (is_featu[illegible]bled('billing_new_vat_law')) {
        if (is_fe[illegible]_enabled('billing_new_vat_law2')) {
            ha[illegible]ng_new2();
        } els[illegible]
            [illegible]le_bi[illegible]new();
        }
    } els[illegible]
        [illegible]dle_billing_for[illegible]
    }
} else {
    handle_beta_form();
}
```

Qafoo
passion for software quality

History

- ”Flipping Out” by Flickr (2009)
- ”FeatureToggle” by Martin Fowler (2010)
- Names
  - Flags
  - Toggles
  - Flippers
  - Switches

Feature Flags are branching on the code-level

passion for software quality

Branches for Features Flags

```php
<?php
// branch "master"
handle_beta_form();
```

```php
<?php
// branch "billing"
handle_billing_form();
```

```php
<?php
// branch "billing_new_vat_law"
handle_billing_new_vat_law();
```

Qafoo
passion for software quality

Feature Flags vs VCS Branches

Feature Flags allow arbitrary combination of branches

VCS don’t have this flexibility!

- Allow trunk-based development
- Increase complexity

Outline

API for Feature Flags

```php
<?php

interface FeatureFlags
{
    function isEnabled($flag);
}
```

passion for software quality

Static Feature Flags

```php
<?php
class StaticFlags implements FeatureFlags
{
    public function isEnabled($flag)
    {
        if ($flag === 'billing') {
            return true;
        }

        return false;
    }
}
```

passion for software quality

Feature Flags Service

```
<service
    id="feature_flags"
    class="Acme\DemoBundle\Util\StaticFlags">
</service>
```

Feature Flags Usage

```php
<?php
class BillingController
{
    public function signupAction()
    {
        $features = $this->get('feature_flags');

        if (!$features->isEnabled('billing')) {
            throw new NotFoundHttpException();
        }
        // ..
    }
}
```

Implementation

- Symfony Configuration
- SQL-Database
- Redis
- Any kind of implementation is usually simple.

Outline

Design Considerations

- Avoid if/elseif/else hell
- Maintainable Solution
  - Cleanup old code
  - Cleanup deprecated flags
- Integrate nicely into Symfony
- Reusable, generic solutions preferred
- Seperate development from activation/testing

passion for software quality

Solution

Move all toggle decisions outside of your code

Integration Points

- Twig Templates
- Routing
- Controllers
- Services
- Event Listeners

Twig Templates

Decide what a user can see

- Show Links
- Load Sub-Controllers

Twig Templates

```
{% if is_feature_enabled('billing') %}
    <a href="{{ path('billing') }}">Pay</a>
{% endif %}
```

Twig Templates

```
{% if is_feature_enabled('billing') %}
    {{ render(controller(
        "AcmeDemoBundle:Billing:show"))
    }}
{% endif %}
```

Routing

Decide what a user can access

- Conditional routes
- Show 404 if it the feature is disabled

Routing

```
billing:
    pattern: /billing/signup
    defaults:
        _feature_flag: billing
```

Routing: EventListener

```php
<?php
public function onKernelRequest($event)
{
    $request = $event->getRequest();
    $flag = $request->attributes
        ->get('_feature_flag');

    if (!$this->features->isEnabled($flag)) {
        throw new NotFoundHttpException();
    }
}
```

Selecting Controllers

Decide what controller is called

- Execute different actions based on flags
- Manipulate Controller Resolver

Deciding about Controllers

```
billing:
  pattern: /billing/signup
  defaults:
    _controller: "AccountBundle:Billing:signup"
    _alternative: "AccountBundle:Billing:signup2"
    _when_feature: billing
```

Deciding about Controllers

```php
<?php

public function onKernelRequest($event)
{
        // ...
        if ($this->features->isEnabled($whenFlag)) {
            $request->attributes->set(
                '_controller',
                $alternative
            );
        }
}
```

passion for software quality

Deciding about Services

Decide what business logic is called

- Construct different services based on feature flags
- Requires a common interface the services implement
- Interface Segregation (SOLID principles)

Symfony Dependency Injection

- Delegate construction of a service to a factory
- Use `factory-service` and `factory-method`
- Implement a generic Factory for the task only once

Feature Flag Service Factory

```php
<?php
class FeatureFlagFactory
{
    private $container;

    public function create($when, $then, $else)
    {
        return $this->flags->isEnabled($when)
            ? $this->container->get($then)
            : $this->container->get($else);
    }
}
```

passion for software quality

Feature Flag Service Definition

```
<service id="feature_flag_factory"
      class="Acme\DemoBundle\FeatureFlagFactory">

      <argument type="service"
            id="service_container" />
</service>
```

Feature Flag Service

```
<service id="payment" class=".."
      factory-service="feature_flag_factory"
      factory-method="create">

      <argument>billing_stripe</argument>
      <argument>payment.stripe</argument>
      <argument>payment.paypal</argument>
</service>
```

passion for software quality

Using the Feature Flag Service

```php
<?php

public function paymentAction()
{
    $provider = $this->get('payment');
    // ...
}
```

Deciding about EventListeners

Decide what event listeners are called

- Add a custom event attribute tag for feature flags.
- Make sure listeners are only called when flag is enabled.
- It is too complicated to do this generically.

Simple Solution

```
class AwesomeListener
{
    public function onKernelRequest($event)
    {
        if (! $this->features->isEnabled('awesome')) {
            return;
        }

        // ...
    }
}
```

Outline

passion for software quality

What about Context?

- A dynamic feature flag system needs context.
  - User Information
  - Request Information
- Gather very early in `kernel.request` event.
- Obviously before any dynamic feature flag is used.

API with Context

```php
<?php

interface FeatureFlags
{
    function setContext($variable, $value);
    function isEnabled($flag);
}
```

Gather Context

```php
<?php

public function onKernelRequest($event)
{
    // ...
    $this->featureFlags->setContext(
        'user_id',
        $user->getId()
    );
    $this->featureFlags->setContext(
        'ip_address',
        $request->getClientIp()
    );
}
```

Links

- http://code.flickr.net/2009/12/02/flipping-out/
- http://martinfowler.com/bliki/FeatureToggle.html
- http://labs.qandidate.com/blog/2014/09/04/feature-toggles-in-symfony2/

Qafoo

passion for software quality

THANK YOU

Rent a quality expert

qafoo.com

Outline

A/B Testing

- Consider small experiments activated with feature toggles
- Let 50% of users see the new feature
- Measure success of the new variant compared to the old
- Decide to keep the old or switch to the new variant
- Requires user context (groups of users)

Circuit Breaker

- Use dynamic feature toggles to deactivate defunct backends
- Example: Deactivate Search when Elasticsearch is down
- Requires feature toggle to be always present in code
- Requires datastorage to measure number of failures of backend services.