django-push Documentation

Release 1.0

Bruno Renié

Apr 25, 2017

Contents

PuSH is the other name of PubSubHubbub, a publish/subscribe protocol based on HTTP and allowing near-instant notifications of topic updates.

- Publishers are entities that publish their updates via HTTP resources. When a resource is updated with a new entry, they ping their *hub* saying they have some new content. The hub is also declared in the resource.
- Subscribers are feed readers or followers. When they fetch a resource, they notice a hub is declared and subscribe to the resource's updates with the hub.
- Hubs fetch the published resource when it gets a ping from the publisher and takes care of notifying all the subscribers.

This library provides hooks to add PubSubHubbub support to your Django project: you can use it to be a publisher and/or subscriber.

The PubSubHubbub spec was initially designed for Atom feeds. The 0.3 version of the spec defines resources as feeds. The 0.4 version allows arbitrary content types. The 0.4 spec is supported since version **0.5** of django-push. We unfortunately missed the chance of having version numbers match properly.

CHAPTER 1

Installation

```
pip install django-push
```

CHAPTER 2

Manual

Being a publisher

Declare your hub

First, you need a hub. You can either use your own or use a public hub. See the hub's documentation for adding a new feed and add your hub's URL as a `PUSH_HUB` setting (the URL **must** be a full URL):

```
PUSH_HUB = 'https://pubsubhubbub.appspot.com'
```

Finally, use *django-push*'s base feed to declare your feeds. Instead of importing `django.contrib.syndication.views.Feed`, do it this way:

```
from django_push.publisher.feeds import Feed

class MyFeed(Feed):
    title = 'My Feed'
    link = '...'

    def items(self):
        return MyModel.objects.filter(...)
```

Django-push will take care of adding the hub declaration to the feeds. By default, the hub is set to your `PUSH_HUB` setting. If you want to change it, see *Use different hubs for each feed*.

Django-push's feed is just a slightly modified version of the `Feed` class from the `contrib.syndication` app, however its type is forced to be an Atom feed. While some hubs may be compatible with RSS and Atom feeds, the PubSubHubbub specifications encourages the use of Atom feeds. Make sure you use the Atom attributes, like `subtitle` instead of `description` for instance. If you're already publishing Atom feeds, you're fine.

Use different hubs for each feed

If you want to use different hubs for different feeds, just set the `hub` attribute to the URL you want:

```
from django_push.publisher.feeds import Feed

class MyFeed(Feed):
    title = 'My Feed'
    link = '...'
    hub = 'http://hub.example.com'

class MyOtherFeed(Feed):
    hub = 'http://some-other-hub.com'
```

By default, the `Feed` class will use the `PUSH_HUB` setting.

If you need to compute the hub URL at runtime, override the `get_hub` method on your feed subclass:

```
from django_push.publisher.feeds import Feed

class MyFeed(Feed):
    def get_hub(self, obj):
        return some_dynamic_url
```

The `get_hub` method was added in django-push 0.5.

Ping the hub on feed updates

Once your feeds are configured, you need to ping the hub each time a new item/entry is published. Since you may have your own publishing mechanics, you need to call a `ping_hub` function when a new entry is made available. For example, if a model has a `publish()` method:

```
from django.contrib.sites.models import get_current_site
from django.core.urlresolvers import reverse
from django.db import models
from django.utils import timezone

from django_push.publisher import ping_hub

class MyModel(models.Model):
    def publish(self):
        self.published = True
        self.timestamp = timezone.now()
        self.save()

        ping_hub('http://%s%s' % (get_current_site().domain,
                                  reverse('feed_for_mymodel')))
```

`ping_hub` has to be called with the full URL of the Atom feed as parameter, using either the Sites framework or your own mechanism to add the domain name. By default, `ping_hub` will ping the hub declared in the `PUSH_HUB` setting. A different hub can be set using an optional `hub_url` keyword argument:

```
from django_push.publisher import ping_hub

ping_hub('http://example.com/feed.atom',
         hub_url='http://hub.example.com')
```

Being a subscriber

- Add `django_push.subscriber` to your `INSTALLED_APPS` and run `manage.py migrate`.
- Include `django_push.subscriber.urls` in your main urlconf:

```
urlpatterns = [
    # ...
    url(r'^subscriber/', include('django_push.subscriber.urls')),
]
```

- If you have `django.contrib.sites` installed, make sure it is correctly configured: check that `Site.objects.get_current()` actually returns the domain of your publicly accessible website.
- If you don't use `django.contrib.sites`, set `PUSH_DOMAIN` to your site's domain in your settings.
- Additionally if your site is available via HTTPS, set `PUSH_SSL_CALLBACK` to `True`.

Initial subscription

Let's assume you're already parsing feeds. Your code may look like this:

```
import feedparser


parsed = feedparser.parse('http://example.com/feed/')
for entry in parsed.entries:
    # Do something with the entries: store them, email them...
    do_something()
```

You need to modify this code to check if the feed declares a hub and initiate a subscription for this feed.

```
parsed = feedparser.parse('http://example.com/feed/')

if 'links' in parsed.feed:
    for link in parsed.feed.links:
        if link.rel == 'hub':
            # Hub detected!
            hub = link.href
```

Now that you found a hub, you can create a subscription:

```
from django_push.subscriber.models import Subscription


subscription = Subscription.objects.subscribe(feed_url, hub=hub,
                                              lease_seconds=12345)
```

If a subscription for this feed already exists, no new subscription is created but the existing subscription is renewed.

`lease_seconds` is optional and **only a hint** for the hub. If the hub has a custom expiration policy it may chose another value arbitrarily. The value chose by the hub is saved in the subscription object when the subscription gets verified.

If you want to set a default `lease_seconds`, you can use the `PUSH_LEASE_SECONDS` setting.

If there's a danger of hub freezing the connection (it happens in the wild) you can use the `PUSH_TIMEOUT` setting. Its value should be the number of seconds (float) to wait for the subscription request to finish. Good number might be 60.

Renewing the leases

As we can see, the hub subscription can be valid for a certain amount of time.

Version 0.3 of the PubSubHubbub spec explains that hub must recheck with subscribers before subscriptions expire to automatically renew subscriptions. This is not the case in version 0.4 of the spec.

In any case you can renew the leases before the expire to make sure they are not forgotten by the hub. For instance, this could be run once a day:

```
import datetime

from django.utils import timezone

from django_push.subscriber.models import Subscription


tomorrow = timezone.now() + datetime.timedelta(days=1)

for subscription in Subscription.objects.filter(
    verified=True,
    lease_expiration__lte=tomorrow
):
    subscription.subscribe()
```

Unsubscribing

If you want to stop receiving notification for a feed's updates, you need to unsubscribe. This is as simple as doing:

```
from django_push.subscriber.models import Subscription

subscription = Subscription.objects.get(topic='http://example.com/feed')
subscription.unsubscribe()
```

The hub is notified to cancel the subscription and the Subscription object is deleted. You can also specify the hub if a topic uses several hubs:

```
subscription = Subscription.objects.get(topic=feed_url, hub=hub_url)
subscription.unsubscribe()
```

Authentication

Some hubs may require basic auth for subscription requests. Django-PuSH provides a way to supply authentication information via a callable that takes the hub URL as a parameter and returns None (no authentication required) or a (username, password) tuple. For instance:

```
def custom_hub_credentials(hub_url):
    if hub_url == 'http://superfeedr.com/hubbub':
        return ('my_superfeedr_username', 'password')
```

And then, set the PUSH_CREDENTIALS setting to the dotted path to your custom function:

```
PUSH_CREDENTIALS = 'path.to.custom_hub_credentials'
```

This way you have full control of the way credentials are stored (database, settings, filesystem...)

Using HTTPS Callback URLs

By default, callback URLs will be constructed using HTTP. If you would like to use HTTPS for callback URLs, set the PUSH_SSL_CALLBACK setting to True:

```
PUSH_SSL_CALLBACK = True
```

Listening to Hubs' notifications

Once subscriptions are setup, the hubs will start to send notifications to your callback URLs. Each time a notification is received, the django_push.subscriber.signals.updated signal is sent. You can define a receiver function:

```
import feedparser

from django_push.subscriber.signals import updated

def listener(notification, **kwargs):
    parsed = feedparser.parse(notification)
    for entry in parsed.entries:
        print entry.title, entry.link

updated.connect(listener)
```

The notification parameter is the raw payload from the hub. If you expect an RSS/Atom feed you should process the payload using a library such as the universal feedparser.

kwargs also contains the raw HTTP request object and the parsed Link header if it is present. You can take advantage of them to validate the notification:

```
def listener(notification, request, links, **kwargs):
    if links is not None:
        for link in links:
            if link['rel'] == 'self':
                break
        url = link['url']  # This is the topic URL
```

Listening with a view instead of the updated signal

If Django signals are not your thing, you can inherit from the base subscriber view to listen for notifications:

```
from django_push.subscriber.views import CallbackView

class MyCallback(CallbackView):
    def handle_subscription(self):
        payload = self.request.body
        parsed = feedparser.parse(payload)
        for entry in payload.entries:
```

```
            do_stuff_with(entry)
callback = MyCallback.as_view()
```

Then instead of including `django_push.subscriber.urls` in your urlconf, define a custom URL with `subscriber_callback` as a name and a `pk` named parameter:

```
from django.conf.urls import patterns, url

from .views import callback

urlpatterns = patterns(
    '',
    url(r'^subscriber/(?P<pk>\d+)/$', callback, name='subscriber_callback'),
)
```

In the `handle_subscription` method of the view, you can access `self.request`, `self.subscription` and `self.links`.

Logging

You can listen for log messages by configuring the `django_push` logger:

```
LOGGING = {
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'django_push': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
    },
}
```

CHAPTER 3

Changelog

- **1.0** (2017-04-25):
  - Confirm support for Django 1.11 (no code changes required).
- **0.9** (2016-07-13):
  - Remove support for Django 1.7.
  - Drop support for Python 3.2.
  - Confirm support for Django 1.10.
- **0.8** (2015-09-29):
  - Remove support for Django < 1.7.
  - Use a transaction hook in `Subscription.objects.subscribe()` when available (Django 1.9+).
- **0.7** (2015-07-10):
  - Remove warnings with Django versions up to 1.8.
- **0.6.1** (2014-01-14):
  - Added `PUSH_TIMEOUT` setting for passing timeouts to the subscribe/unsubscribe HTTP calls.
- **0.6** (2013-07-10):
  - Removed `get_hub()`.
  - Removed the `unsubscribe()` manager method. Unsubscribing must be done with subscription instances.
  - Added `request` and `links` keyword arguments to the `updated` signal. `request` is the raw HTTP request object, `links` is a parsed version of the `Link` header, if present.
- **0.5** (2013-06-24):
  - Python 3 support, Django >= 1.4.1 support.
  - HTTP handling via requests instead of urllib2.

  - Deprecation of `Subscription.objects.unsubscribe()` in favor of an instance method on the subscription object. The `unsubscribe()` manager method will be removed in version 0.6.
  - `Subscription.objects.subscribe()` raises a warning if the `hub` kwarg is not provided. It will become mandatory in version 0.6.
  - Removed `hub.verify_token` from subscription requests. It's optional in the 0.3 spec and absent from the 0.4 spec.
  - Secret generation code uses `django.utils.crypto` instead of the `random` module. In addition, subscriptions over HTTP don't use a secret anymore (as recommended in the spec).
  - The `updated` signal is sent with the raw payload instead of the result of a `feedparser.parse` call. This allows other content types than feeds to be processed, as suggested in version 0.4 of the PubSubHubbub spec.
  - The callback view is now a class-based view, allowing listening for content distribution via a custom view if the `updated` signal is not suitable.
  - `django.contrib.sites` is no longer a hard requirement. You can set `PUSH_DOMAIN` in your settings to your site's canonical hostname.
  - South migrations support. If you don't use South, you should. If you're upgrading from 0.4, just **fake the first migration** and apply the others:

```
./manage.py migrate subscriber 0001_initial --fake
./manage.py migrate
```

  - Tremendously improved admin support. If you were using a custom ModelAdmin for subscriptions, you might want to try the built-in one.

- **0.4** (2011-06-30):
  - Support for hub authentication via `PUSH_HUB_CREDENTIALS`.
  - Support for SSL callback URLs.
- **0.3** (2010-08-18):
  - Subscribers can unsubscribe.
- **0.2** (2010-08-12):
  - Signature handling of content distribution requests.
- **0.1** (2010-08-11):
  - Initial release.