Djrill Documentation

Release 2.1.0

Djrill contributors (see AUTHORS.txt)

August 29, 2016

Contents

Version 2.1.0

Djrill integrates the Mandrill transactional email service into Django.

> **PROJECT STATUS: INACTIVE**
>
> As of April, 2016, Djrill is no longer actively maintained (other than security updates). It is likely to keep working unless/until Mandrill changes their APIs, but Djrill will not be updated for newer Django versions or Mandrill changes. (more info)
>
> You may be interested in django-anymail, a Djrill fork that supports Mailgun, Postmark, SendGrid, and other transactional ESPs (including limited support for Mandrill).

In general, Djrill "just works" with Django's built-in `django.core.mail` package. It includes:

- Support for HTML, attachments, extra headers, and other features of Django's built-in email
- Mandrill-specific extensions like tags, metadata, tracking, and MailChimp templates
- Optional support for Mandrill inbound email and other webhook notifications, via Django signals

Djrill is released under the BSD license. It is tested against Django 1.4–1.9 (including Python 3 with Django 1.6+, and PyPy support with Django 1.5+). Djrill uses semantic versioning.

CHAPTER 1

Documentation

1.1 Djrill 1-2-3

1. Install Djrill from PyPI:

```
$ pip install djrill
```

2. Edit your project's `settings.py`:

```
INSTALLED_APPS = (
    ...
    "djrill"
)

MANDRILL_API_KEY = "<your Mandrill key>"
EMAIL_BACKEND = "djrill.mail.backends.djrill.DjrillBackend"
DEFAULT_FROM_EMAIL = "you@example.com"  # if you don't already have this in settings
```

3. Now the regular Django email functions will send through Mandrill:

```
from django.core.mail import send_mail

send_mail("It works!", "This will get sent through Mandrill",
    "Djrill Sender <djrill@example.com>", ["to@example.com"])
```

You could send an HTML message, complete with custom Mandrill tags and metadata:

```
from django.core.mail import EmailMultiAlternatives

msg = EmailMultiAlternatives(
    subject="Djrill Message",
    body="This is the text email body",
    from_email="Djrill Sender <djrill@example.com>",
    to=["Recipient One <someone@example.com>", "another.person@example.com"],
    headers={'Reply-To': "Service <support@example.com>"} # optional extra headers
)
msg.attach_alternative("<p>This is the HTML email body</p>", "text/html")

# Optional Mandrill-specific extensions:
msg.tags = ["one tag", "two tag", "red tag", "blue tag"]
msg.metadata = {'user_id': "8675309"}

# Send it:
msg.send()
```

1.2 Installation

It's easiest to install Djrill from PyPI:

```
$ pip install djrill
```

If you decide to install Djrill some other way, you'll also need to install its one dependency (other than Django, of course): the requests library from Kenneth Reitz.

1.2.1 Configuration

In your project's `settings.py`:

1. Add `djrill` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    "djrill"
)
```

2. Add the following line, substituting your own *`MANDRILL_API_KEY`*:

```
MANDRILL_API_KEY = "brack3t-is-awesome"
```

3. Override your existing `EMAIL_BACKEND` with the following line:

```
EMAIL_BACKEND = "djrill.mail.backends.djrill.DjrillBackend"
```

Also, if you don't already have a `DEFAULT_FROM_EMAIL` in settings, this is a good time to add one. (Django's default is "webmaster@localhost", which won't work with Mandrill.)

1.2.2 Mandrill Webhooks (Optional)

Djrill includes optional support for Mandrill webhooks, including inbound email. See the Djrill *webhooks* section for configuration details.

1.2.3 Other Optional Settings

You can optionally add any of these Djrill settings to your `settings.py`.

MANDRILL_IGNORE_RECIPIENT_STATUS

Set to `True` to disable *`djrill.MandrillRecipientsRefused`* exceptions on invalid or rejected recipients. (Default `False`.)

New in version 2.0.

MANDRILL_SETTINGS

You can supply global default options to apply to all messages sent through Djrill. Set `MANDRILL_SETTINGS` to a dict of these options. Example:

```
MANDRILL_SETTINGS = {
    'subaccount': 'client-347',
    'tracking_domain': 'example.com',
    'track_opens': True,
}
```

See *Mandrill-Specific Options* for a list of available options. (Everything *except* *merge_vars*, *recipient_metadata*, and *send_at* can be used with MANDRILL_SETTINGS.)

Attributes set on individual EmailMessage objects will override the global MANDRILL_SETTINGS for that message. *global_merge_vars* on an EmailMessage will be merged with any global_merge_vars in MANDRILL_SETTINGS (with the ones on the EmailMessage taking precedence if there are conflicting var names).

New in version 2.0.

MANDRILL_API_URL

The base url for calling the Mandrill API. The default is MANDRILL_API_URL = "https://mandrillapp.com/api/1.0", which is the secure, production version of Mandrill's 1.0 API.

(It's unlikely you would need to change this.)

MANDRILL_SUBACCOUNT

Prior to Djrill 2.0, the MANDRILL_SUBACCOUNT setting could be used to globally set the Mandrill subaccount. Although this is still supported for compatibility with existing code, new code should set a global subaccount in *MANDRILL_SETTINGS* as shown above.

1.3 Upgrading from 1.x

Djrill 2.0 includes some breaking changes from 1.x. These changes should have minimal (or no) impact on most Djrill users, but if you are upgrading please review the major topics below to see if they apply to you.

Djrill 1.4 tried to warn you if you were using Djrill features expected to change in 2.0. If you are seeing any deprecation warnings with Djrill 1.4, you should fix them before upgrading to 2.0. (Warnings appear in the console when running Django in debug mode.)

Please see the *release notes* for a list of new features and improvements in Djrill 2.0.

1.3.1 Dropped support for Django 1.3, Python 2.6, and Python 3.2

Although Djrill may still work with these older configurations, we no longer test against them. Djrill now requires Django 1.4 or later and Python 2.7, 3.3, or 3.4.

If you require support for these earlier versions, you should not upgrade to Djrill 2.0. Djrill 1.4 remains available on pypi, and will continue to receive security fixes.

1.3.2 Removed DjrillAdminSite

Earlier versions of Djrill included a custom Django admin site. The equivalent functionality is available in Mandrill's dashboard, and Djrill 2.0 drops support for it.

Although most Djrill users were unaware the admin site existed, many did follow the earlier versions' instructions to enable it.

If you have added DjrillAdminSite, you will need to remove it for Djrill 2.0.

In your `urls.py`:

```
from djrill import DjrillAdminSite  # REMOVE this
admin.site = DjrillAdminSite()  # REMOVE this

admin.autodiscover()  # REMOVE this if you added it only for Djrill
```

In your `settings.py`:

```
INSTALLED_APPS = (
    ...
    # If you added SimpleAdminConfig only for Djrill:
    'django.contrib.admin.apps.SimpleAdminConfig',  # REMOVE this
    'django.contrib.admin',  # ADD this default back
    ...
)
```

(These instructions assume you had changed to SimpleAdminConfig solely for DjrillAdminSite. If you are using it for custom admin sites with any other Django apps you use, you should leave it SimpleAdminConfig in place, but still remove the references to DjrillAdminSite.)

1.3.3 Added exception for invalid or rejected recipients

Djrill 2.0 raises a new *djrill.MandrillRecipientsRefused* exception when all recipients of a message are invalid or rejected by Mandrill. (This parallels the behavior of Django's default `SMTP email backend`, which raises `SMTPRecipientsRefused` when all recipients are refused.)

Your email-sending code should handle this exception (along with other exceptions that could occur during a send). However, if you want to retain the Djrill 1.x behavior and treat invalid or rejected recipients as successful sends, you can set *MANDRILL_IGNORE_RECIPIENT_STATUS* to `True` in your settings.py.

1.3.4 Other 2.0 breaking changes

Code that will be affected by these changes is far less common than for the changes listed above, but they may impact some uses:

Removed unintended date-to-string conversion If your code was inadvertently relying on Djrill to automatically convert date or datetime values to strings in *merge_vars*, *metadata*, or other Mandrill message attributes, you must now explicitly do the string conversion yourself. See *Formatting Merge Data* for an explanation. (Djrill 1.4 reported a DeprecationWarning for this case.)

(This does not affect *send_at*, where Djrill specifically allows date or datetime values.)

Removed DjrillMessage class The `DjrillMessage` class has not been needed since Djrill 0.2. You should replace any uses of it with the standard `EmailMessage` class. (Djrill 1.4 reported a DeprecationWarning for this case.)

Removed DjrillBackendHTTPError This exception was deprecated in Djrill 0.3. Replace uses of it with *djrill.MandrillAPIError*. (Djrill 1.4 reported a DeprecationWarning for this case.)

Refactored Djrill backend and exceptions Several internal details of `djrill.mail.backends.DjrillBackend` and Djrill's exception classes have been significantly updated for 2.0. The intent is to make it easier to maintain and extend the backend (including creating your own subclasses to override Djrill's default behavior). As a result, though, any existing code that depended on undocumented Djrill internals may need to be updated.

1.4 Sending Mail

Djrill handles **all** outgoing email sent through Django's standard `django.core.mail` package, including `send_mail()`, `send_mass_mail()`, the `EmailMessage` class, and even `mail_admins()`.

If you'd like to selectively send only some messages through Mandrill, there is a way to *use multiple email backends*.

1.4.1 Django Email Support

Djrill supports most of the functionality of Django's `EmailMessage` and `EmailMultiAlternatives` classes.

Some notes and limitations:

Display Names All email addresses (from, to, cc, bcc) can be simple ("email@example.com") or can include a display name ("Real Name <email@example.com>").

CC and BCC Recipients Djrill properly identifies "cc" and "bcc" recipients to Mandrill.

Note that you may need to set the Mandrill option *`preserve_recipients`* to `True` if you want recipients to be able to see who else was included in the "to" list.

HTML/Alternative Parts To include an HTML version of a message, use `attach_alternative()`:

```
from django.core.mail import EmailMultiAlternatives

msg = EmailMultiAlternatives("Subject", "text body",
                             "from@example.com", ["to@example.com"])
msg.attach_alternative("<html>html body</html>", "text/html")
```

Djrill allows a maximum of one `attach_alternative()` on a message, and it must be `mimetype="text/html"`. Otherwise, Djrill will raise *`NotSupportedByMandrillError`* when you attempt to send the message. (Mandrill doesn't support sending multiple html alternative parts, or any non-html alternatives.)

Attachments Djrill will send a message's attachments. (Note that Mandrill may impose limits on size and type of attachments.)

Also, if an image attachment has a Content-ID header, Djrill will tell Mandrill to treat that as an embedded image rather than an ordinary attachment. (For an example, see `test_embedded_images()` in `tests/test_mandrill_send.py`.)

Headers Djrill accepts additional headers and passes them along to Mandrill:

```
msg = EmailMessage( ...
    headers={'Reply-To': "reply@example.com", 'List-Unsubscribe': "..."}
)
```

Note: Djrill also supports the `reply_to` param added to `EmailMessage` in Django 1.8. (If you provide *both* a 'Reply-To' header and the `reply_to` param, the header will take precedence.)

1.4.2 Mandrill-Specific Options

Most of the options from the Mandrill messages/send API `message` struct can be set directly on an `EmailMessage` (or subclass) object.

Note: You can set global defaults for common options with the *MANDRILL_SETTINGS* setting, to avoid having to set them on every message.

important
Boolean: whether Mandrill should send this message ahead of non-important ones.

track_opens
Boolean: whether Mandrill should enable open-tracking for this message. Default from your Mandrill account settings.

```
message.track_opens = True
```

track_clicks
Boolean: whether Mandrill should enable click-tracking for this message. Default from your Mandrill account settings.

Note: Mandrill has an option to track clicks in HTML email but not plaintext, but it's *only* available in your Mandrill account settings. If you want to use that option, set it at Mandrill, and *don't* set the `track_clicks` attribute here.

auto_text
Boolean: whether Mandrill should automatically generate a text body from the HTML. Default from your Mandrill account settings.

auto_html
Boolean: whether Mandrill should automatically generate an HTML body from the plaintext. Default from your Mandrill account settings.

inline_css
Boolean: whether Mandrill should inline CSS styles in the HTML. Default from your Mandrill account settings.

url_strip_qs
Boolean: whether Mandrill should ignore any query parameters when aggregating URL tracking data. Default from your Mandrill account settings.

preserve_recipients
Boolean: whether Mandrill should include all recipients in the "to" message header. Default from your Mandrill account settings.

view_content_link
Boolean: set False on sensitive messages to instruct Mandrill not to log the content.

tracking_domain
str: domain Mandrill should use to rewrite tracked links and host tracking pixels for this message. Useful if you send email from multiple domains. Default from your Mandrill account settings.

signing_domain
str: domain Mandrill should use for DKIM signing and SPF on this message. Useful if you send email from multiple domains. Default from your Mandrill account settings.

return_path_domain
str: domain Mandrill should use for the message's return-path.

merge_language
str: the merge tag language if using merge tags – e.g., "mailchimp" or "handlebars". Default from your Mandrill account settings.

global_merge_vars
dict: merge variables to use for all recipients (most useful with *Mandrill Templates*).

```
message.global_merge_vars = {'company': "ACME", 'offer': "10% off"}
```

Merge data must be strings or other JSON-serializable types. (See *Formatting Merge Data* for details.)

merge_vars
dict: per-recipient merge variables (most useful with *Mandrill Templates*). The keys in the dict are the recipient email addresses, and the values are dicts of merge vars for each recipient:

```
message.merge_vars = {
    'wiley@example.com': {'offer': "15% off anvils"},
    'rr@example.com':    {'offer': "instant tunnel paint"}
}
```

Merge data must be strings or other JSON-serializable types. (See *Formatting Merge Data* for details.)

tags
list of str: tags to apply to the message, for filtering reports in the Mandrill dashboard. (Note that Mandrill prohibits tags longer than 50 characters or starting with underscores.)

```
message.tags = ["Order Confirmation", "Test Variant A"]
```

subaccount
str: the ID of one of your subaccounts to use for sending this message.

google_analytics_domains
list of str: domain names for links where Mandrill should add Google Analytics tracking parameters.

```
message.google_analytics_domains = ["example.com"]
```

google_analytics_campaign
str or list of str: the utm_campaign tracking parameter to attach to links when adding Google Analytics tracking. (Mandrill defaults to the message's from_email as the campaign name.)

metadata
dict: metadata values Mandrill should store with the message for later search and retrieval.

```
message.metadata = {'customer': customer.id, 'order': order.reference_number}
```

Mandrill restricts metadata keys to alphanumeric characters and underscore, and metadata values to numbers, strings, boolean values, and None (null).

recipient_metadata
dict: per-recipient metadata values. Keys are the recipient email addresses, and values are dicts of metadata for each recipient (similar to *merge_vars*)

Mandrill restricts metadata keys to alphanumeric characters and underscore, and metadata values to numbers, strings, boolean values, and None (null).

async
Boolean: whether Mandrill should use an async mode optimized for bulk sending.

ip_pool
str: name of one of your Mandrill dedicated IP pools to use for sending this message.

send_at
datetime or date or str: instructs Mandrill to delay sending this message until the specified time. Example:

```
msg.send_at = datetime.utcnow() + timedelta(hours=1)
```

Mandrill requires a UTC string in the form `YYYY-MM-DD HH:MM:SS`. Djrill will convert python dates and datetimes to this form. (Dates will be given a time of 00:00:00.)

Note: Timezones

Mandrill assumes `send_at` is in the UTC timezone, which is likely *not* the same as your local time.

Djrill will convert timezone-*aware* datetimes to UTC for you. But if you format your own string, supply a date, or a *naive* datetime, you must make sure it is in UTC. See the python `datetime` docs for more information.

For example, `msg.send_at = datetime.now() + timedelta(hours=1)` will try to schedule the message for an hour from the current time, but *interpreted in the UTC timezone* (which isn't what you want). If you're more than an hour west of the prime meridian, that will be in the past (and the message will get sent immediately). If you're east of there, the message might get sent quite a bit later than you intended. One solution is to use `utcnow` as shown in the earlier example.

Note: Scheduled sending is a paid Mandrill feature. If you are using a free Mandrill account, `send_at` won't work.

All the Mandrill-specific attributes listed above work with *any* `EmailMessage`-derived object, so you can use them with many other apps that add Django mail functionality.

If you have questions about the python syntax for any of these properties, see `DjrillMandrillFeatureTests` in `tests/test_mandrill_send.py` for examples.

1.4.3 Response from Mandrill

`mandrill_response`

Djrill adds a `mandrill_response` attribute to each `EmailMessage` as it sends it. This allows you to retrieve message ids, initial status information and more.

For an EmailMessage that is successfully sent to one or more email addresses, `mandrill_response` will be set to a `list` of `dict`, where each entry has info for one email address. See the Mandrill docs for the messages/send API for full details.

For example, to get the Mandrill message id for a sent email you might do this:

```
msg = EmailMultiAlternatives(subject="subject", body="body",
                             from_email="sender@example.com",to=["someone@example.com"])
msg.send()
response = msg.mandrill_response[0]
mandrill_id = response['_id']
```

For this example, msg.mandrill_response might look like this:

```
msg.mandrill_response = [
    {
        "email": "someone@example.com",
        "status": "sent",
        "_id": "abc123abc123abc123abc123abc123"
    }
]
```

If an error is returned by Mandrill while sending the message then `mandrill_response` will be set to None.

1.4.4 Exceptions

exception djrill.**NotSupportedByMandrillError**
If the email tries to use features that aren't supported by Mandrill, the send call will raise a NotSupportedByMandrillError exception (a subclass of ValueError).

exception djrill.**MandrillRecipientsRefused**
If *all* recipients (to, cc, bcc) of a message are invalid or rejected by Mandrill (e.g., because they are your Mandrill blacklist), the send call will raise a MandrillRecipientsRefused exception. You can examine the message's *mandrill_response* attribute to determine the cause of the error.

If a single message is sent to multiple recipients, and *any* recipient is valid (or the message is queued by Mandrill because of rate limiting or *send_at*), then this exception will not be raised. You can still examine the mandrill_response property after the send to determine the status of each recipient.

You can disable this exception by setting *MANDRILL_IGNORE_RECIPIENT_STATUS* to True in your settings.py, which will cause Djrill to treat any non-API-error response from Mandrill as a successful send.

New in version 2.0: Djrill 1.x behaved as if MANDRILL_IGNORE_RECIPIENT_STATUS = True.

exception djrill.**MandrillAPIError**
If the Mandrill API fails or returns an error response, the send call will raise a MandrillAPIError exception (a subclass of requests.HTTPError). The exception's status_code and response attributes may help explain what went wrong. (Tip: you can also check Mandrill's API error log to view the full API request and error response.)

exception djrill.**NotSerializableForMandrillError**
The send call will raise a NotSerializableForMandrillError exception if the message has attached data which cannot be serialized to JSON for the Mandrill API.

See *Formatting Merge Data* for more information.

New in version 2.0: Djrill 1.x raised a generic TypeError in this case. NotSerializableForMandrillError is a subclass of TypeError for compatibility with existing code.

1.5 Sending Template Mail

1.5.1 Mandrill Templates

To use a *Mandrill* (MailChimp) template stored in your Mandrill account, set a template_name and (optionally) template_content on your EmailMessage object:

```
from django.core.mail import EmailMessage

msg = EmailMessage(subject="Shipped!", from_email="store@example.com",
                   to=["customer@example.com", "accounting@example.com"])
msg.template_name = "SHIPPING_NOTICE"           # A Mandrill template name
msg.template_content = {                        # Content blocks to fill in
    'TRACKING_BLOCK': "<a href='.../*|TRACKINGNO|*'>track it</a>"
}
msg.global_merge_vars = {                       # Merge tags in your template
    'ORDERNO': "12345", 'TRACKINGNO': "1Z987"
}
msg.merge_vars = {                              # Per-recipient merge tags
    'accounting@example.com': {'NAME': "Pat"},
    'customer@example.com':   {'NAME': "Kim"}
```

```
}
msg.send()
```

If `template_name` is set, Djrill will use Mandrill's messages/send-template API, and will ignore any `body` text set on the `EmailMessage`.

All of Djrill's other *Mandrill-specific options* can be used with templates.

Formatting Merge Data

If you're using dates, datetimes, Decimals, or anything other than strings and integers, you'll need to format them into strings for use as merge data:

```
product = Product.objects.get(123)  # A Django model
total_cost = Decimal('19.99')
ship_date = date(2015, 11, 18)

# Won't work -- you'll get "not JSON serializable" exceptions:
msg.global_merge_vars = {
    'PRODUCT': product,
    'TOTAL_COST': total_cost,
    'SHIP_DATE': ship_date
}

# Do something this instead:
msg.global_merge_vars = {
    'PRODUCT': product.name,  # assuming name is a CharField
    'TOTAL_COST': "%.2f" % total_cost,
    'SHIP_DATE': ship_date.strftime('%B %d, %Y')  # US-style "March 15, 2015"
}
```

These are just examples. You'll need to determine the best way to format your merge data as strings.

Although floats are allowed in merge vars, you'll generally want to format them into strings yourself to avoid surprises with floating-point precision.

Technically, Djrill will accept anything serializable by the Python json package – which means advanced template users can include dicts and lists as merge vars (for templates designed to handle objects and arrays). See the Python `json.JSONEncoder` docs for a list of allowable types.

Djrill will raise *djrill.NotSerializableForMandrillError* if you attempt to send a message with non-json-serializable data.

How To Use Default Mandrill Subject and From fields

To use default Mandrill "subject" or "from" field from your template definition (overriding your EmailMessage and Django defaults), set the following attrs: *use_template_subject* and/or *use_template_from* on your `EmailMessage` object:

```
msg.use_template_subject = True
msg.use_template_from = True
msg.send()
```

use_template_subject
: If `True`, Djrill will omit the subject, and Mandrill will use the default subject from the template.

use_template_from
: If `True`, Djrill will omit the "from" field, and Mandrill will use the default "from" from the template.

1.5.2 Django Templates

To compose email using *Django* templates, you can use Django's `render_to_string()` template shortcut to build the body and html.

Example that builds an email from the templates `message_subject.txt`, `message_body.txt` and `message_body.html`:

```
from django.core.mail import EmailMultiAlternatives
from django.template import Context
from django.template.loader import render_to_string

template_data = {
    'ORDERNO': "12345", 'TRACKINGNO': "1Z987"
}

plaintext_context = Context(autoescape=False)  # HTML escaping not appropriate in plaintext
subject = render_to_string("message_subject.txt", template_data, plaintext_context)
text_body = render_to_string("message_body.txt", template_data, plaintext_context)
html_body = render_to_string("message_body.html", template_data)

msg = EmailMultiAlternatives(subject=subject, from_email="store@example.com",
                             to=["customer@example.com"], body=text_body)
msg.attach_alternative(html_body, "text/html")
msg.send()
```

1.6 Mixing Email Backends

Since you are replacing Django's global `EMAIL_BACKEND`, by default Djrill will handle all outgoing mail, sending everything through Mandrill.

You can use Django mail's optional `connection` argument to send some mail through Mandrill and others through a different system.

This could be useful, for example, to deliver customer emails with Mandrill, but send admin emails directly through an SMTP server:

```
from django.core.mail import send_mail, get_connection

# send_mail connection defaults to the settings EMAIL_BACKEND, which
# we've set to DjrillBackend. This will be sent using Mandrill:
send_mail("Thanks", "We sent your order", "sales@example.com", ["customer@example.com"])

# Get a connection to an SMTP backend, and send using that instead:
smtp_backend = get_connection('django.core.mail.backends.smtp.EmailBackend')
send_mail("Uh-Oh", "Need your attention", "admin@example.com", ["alert@example.com"],
    connection=smtp_backend)
```

You can supply a different connection to Django's `send_mail()` and `send_mass_mail()` helpers, and in the constructor for an `EmailMessage` or `EmailMultiAlternatives`.

(See the django.utils.log.AdminEmailHandler docs for more information on Django's admin error logging.)

1.7 Mandrill Webhooks and Inbound Email

Mandrill webhooks are used for notification about outbound messages (bounces, clicks, etc.), and also for delivering

inbound email processed through Mandrill.

Djrill includes optional support for Mandrill's webhook notifications. If enabled, it will send a Django signal for each event in a webhook. Your code can connect to this signal for further processing.

Warning: Webhook Security
Webhooks are ordinary urls—they're wide open to the internet. You must take steps to secure webhooks, or anyone could submit random (or malicious) data to your app simply by invoking your webhook URL. For security:

- Your webhook should only be accessible over SSL (https). (This is beyond the scope of Djrill.)
- Your webhook must include a random, secret key, known only to your app and Mandrill. Djrill will verify calls to your webhook, and will reject calls without the correct key.
- You can, optionally include the two settings `DJRILL_WEBHOOK_SIGNATURE_KEY` and `DJRILL_WEBHOOK_URL` to enforce webhook signature checking

1.7.1 Configuration

To enable Djrill webhook processing you need to create and set a webhook secret in your project settings, include the Djrill url routing, and then add the webhook in the Mandrill control panel.

1. In your project's `settings.py`, add a `DJRILL_WEBHOOK_SECRET`:

```
DJRILL_WEBHOOK_SECRET = "<create your own random secret>"
```

   substituting a secret you've generated just for Mandrill webhooks. (Do *not* use your Mandrill API key or Django SECRET_KEY for this!)

   An easy way to generate a random secret is to run the command below in a shell:

```
$ python -c "from django.utils import crypto; print crypto.get_random_string(16)"
```

2. In your base `urls.py`, add routing for the Djrill urls:

```
urlpatterns = patterns('',
    ...
    url(r'^djrill/', include(djrill.urls)),
)
```

3. Now you need to tell Mandrill about your webhook:

   - For receiving events on sent messages (e.g., bounces or clickthroughs), you'll do this in Mandrill's webhooks control panel.
   - For setting up inbound email through Mandrill, you'll add your webhook to Mandrill's inbound settings under "Routes" for your domain.
   - And if you want both, you'll need to add the webhook in both places.

   In all cases, the "Post to URL" is *`https://yoursite.example.com`*`/djrill/webhook/?secret=`*`your-secret`* substituting your app's own domain, and changing *your-secret* to the secret you created in step 1.

   (For sent-message webhooks, don't forget to tick the "Trigger on Events" checkboxes for the events you want to receive.)

Once you've completed these steps and your Django app is live on your site, you can use the Mandrill "Test" commands to verify your webhook configuration. Then see the next section for setting up Django signal handlers to process the webhooks.

Incidentally, you have some control over the webhook url. If you'd like to change the "djrill" prefix, that comes from the url config in step 2. And if you'd like to change the *name* of the "secret" query string parameter, you can set `DJRILL_WEBHOOK_SECRET_NAME` in your `settings.py`.

For extra security, Mandrill provides a signature in the request header X-Mandrill-Signature. If you want to verify this signature, you need to provide the settings `DJRILL_WEBHOOK_SIGNATURE_KEY` with the webhook-specific signature key that can be found in the Mandrill admin panel and `DJRILL_WEBHOOK_URL` where you should enter the exact URL, including that you entered in Mandrill when creating the webhook.

1.7.2 Webhook Notifications

Once you've enabled webhooks, Djrill will send a `djrill.signals.webhook_event` custom Django signal for each Mandrill event it receives. You can connect your own receiver function to this signal for further processing.

Be sure to read Django's listening to signals docs for information on defining and connecting signal receivers.

Examples:

```
from djrill.signals import webhook_event
from django.dispatch import receiver

@receiver(webhook_event)
def handle_bounce(sender, event_type, data, **kwargs):
    if event_type == 'hard_bounce' or event_type == 'soft_bounce':
        print "Message to %s bounced: %s" % (
            data['msg']['email'],
            data['msg']['bounce_description']
        )

@receiver(webhook_event)
def handle_inbound(sender, event_type, data, **kwargs):
    if event_type == 'inbound':
        print "Inbound message from %s: %s" % (
            data['msg']['from_email'],
            data['msg']['subject']
        )

@receiver(webhook_event)
def handle_whitelist_sync(sender, event_type, data, **kwargs):
    if event_type == 'whitelist_add' or event_type == 'whitelist_remove':
        print "Rejection whitelist update: %s email %s (%s)" % (
            data['action'],
            data['reject']['email'],
            data['reject']['reason']
        )
```

Note that your webhook_event signal handlers will be called for all Mandrill webhook callbacks, so you should always check the `event_type` param as shown in the examples above to ensure you're processing the expected events.

Mandrill batches up multiple events into a single webhook call. Djrill will invoke your signal handler once for each event in the batch.

The available fields in the `data` param are described in Mandrill's documentation: sent-message webhooks, inbound webhooks, and whitelist/blacklist sync webooks.

1.8 Troubleshooting

Djrill throwing errors? Not sending what you want? Here are some tips...

1.8.1 Figuring Out What's Wrong

- **Check the error message:** Look for a Mandrill error message in your web browser or console (running Django in dev mode) or in your server error logs. As of v1.4, Djrill reports the detailed Mandrill error when something goes wrong. And when the error is something like "invalid API key" or "invalid email address", that's probably 90% of what you'll need to know to solve the problem.
- **Check the Mandrill API logs:** The Mandrill dashboard includes an *incredibly-helpful* list of your recent API calls – and you can click into each one to see the full request and response. Check to see if the data you thought you were sending actually made it into the request, and if Mandrill has any complaints in the response.
- **Double-check common issues:**
  - Did you set your `MANDRILL_API_KEY` in settings.py?
  - Did you add `'djrill'` to the list of `INSTALLED_APPS` in settings.py?
  - Are you using a valid from address? Django's default is "webmaster@localhost", which won't cut it. Either specify the `from_email` explicitly on every message you send through Djrill, or add `DEFAULT_FROM_EMAIL` to your settings.py.
- **Try it without Djrill:** Try switching your `EMAIL_BACKEND` setting to Django's File backend and then running your email-sending code again. If that causes errors, you'll know the issue is somewhere other than Djrill. And you can look through the `EMAIL_FILE_PATH` file contents afterward to see if you're generating the email you want.

1.8.2 Getting Help

If you've gone through the suggestions above and still aren't sure what's wrong, the Djrill community is happy to help. Djrill is supported and maintained by the people who use it – like you! (We're not Mandrill employees.)

You can ask in either of these places (but please pick only one per question!):

Ask on StackOverflow Tag your question with **both** `Django` and `Mandrill` to get our attention. Bonus: a lot of questions about Djrill are actually questions about Django itself, so by asking on StackOverflow you'll also get the benefit of the thousands of Django experts there.

Open a GitHub issue We do our best to answer questions in GitHub issues. And if you've found a Djrill bug, that's definitely the place to report it. (Or even fix it – see *Contributing*.)

Wherever you ask, it's always helpful to include the relevant portions of your code, the text of any error messages, and any exception stack traces in your question.

1.9 Contributing

Djrill is maintained by its users. Your contributions are encouraged!

The Djrill source code is on github. See AUTHORS.txt for a list of some of the people who have helped improve Djrill.

1.9.1 Bugs

You can report problems or request features in Djrill's github issue tracker.

We also have some *Troubleshooting* information that may be helpful.

1.9.2 Pull Requests

Pull requests are always welcome to fix bugs and improve support for Mandrill and Django features.

- Please include test cases.
- We try to follow the Django coding style (basically, PEP 8 with longer lines OK).
- By submitting a pull request, you're agreeing to release your changes under under the same BSD license as the rest of this project.

1.9.3 Testing

Djrill is tested on Travis against several combinations of Django and Python versions. (Full list in .travis.yml.)

Most of the included tests verify that Djrill constructs the expected Mandrill API calls, without actually calling Mandrill or sending any email. So these tests don't require a Mandrill API key, but they *do* require mock and six (`pip install mock six`).

To run the tests, either:

```
python -Wall setup.py test
```

or:

```
python -Wall runtests.py
```

If you set the environment variable `MANDRILL_TEST_API_KEY` to a valid Mandrill test API key, there are also a handful of integration tests which will run against the live Mandrill API. (Otherwise these live API tests are skipped.)

1.10 Release Notes

Djrill practices semantic versioning. Among other things, this means that minor updates (1.x to 1.y) should always be backwards-compatible, and breaking changes will always increment the major version number (1.x to 2.0).

1.10.1 Djrill 2.x

Version 2.1:

- Handle Mandrill rejection whitelist/blacklist sync event webhooks
- This is likely the final release of Djrill (other than any critical security updates). See GitHub for more on the future of Djrill.

Version 2.0:

- **Breaking Changes:** please see the *upgrade guide*.
- Add Django 1.9 support; drop Django 1.3, Python 2.6, and Python 3.2 support
- Add global *`MANDRILL_SETTINGS`* dict that can provide defaults for most Djrill message options
- Add *`djrill.NotSerializableForMandrillError`*
- Use a single HTTP connection to the Mandrill API to improve performance when sending multiple messages at once using `send_mass_mail()`. (You can also directly manage your own long-lived Djrill connection across multiple sends, by calling open and close on Django's email backend.)
- Add Djrill version to user-agent header when calling Mandrill API

- Improve diagnostics in exceptions from Djrill
- Remove DjrillAdminSite
- Remove unintended date-to-string conversion in JSON encoding
- Remove obsolete DjrillMessage class and DjrillBackendHTTPError
- Refactor Djrill backend and exceptions

1.10.2 Djrill 1.x and Earlier

Version 1.4:

- Django 1.8 support
- Support new Django 1.8 EmailMessage reply_to param. (Specifying a *Reply-To header* still works, with any version of Django, and will override the reply_to param if you use both.)
- Include Mandrill error response in str(MandrillAPIError), to make errors easier to understand.
- More-helpful exception when using a non-JSON-serializable type in merge_vars and other Djrill message attributes
- Deprecation warnings for upcoming 2.0 changes (see above)

Version 1.3:

- Use Mandrill secure https API endpoint (rather than http).
- Support *merge_language* option (for choosing between Handlebars and Mailchimp templates).

Version 1.2:

- Support Django 1.7; add testing on Python 3.3, 3.4, and PyPy
- Bug fixes

Version 1.1:

- Allow use of Mandrill template default "from" and "subject" fields, via *use_template_from* and *use_template_subject*.
- Fix `UnicodeEncodeError` with unicode attachments

Version 1.0:

- Global *MANDRILL_SUBACCOUNT* setting

Version 0.9:

- Better handling for "cc" and "bcc" recipients.
- Allow all extra message headers in send. (Mandrill has relaxed previous API restrictions on headers.)

Version 0.8:

- Expose *Response from Mandrill* on sent messages

Version 0.7:

- Support for Mandrill send options *async*, *important*, *ip_pool*, *return_path_domain*, *send_at*, *subaccount*, and *view_content_link*

Version 0.6:

- Support for signed webhooks

Version 0.5:

- Support for incoming mail and other Mandrill webhooks
- Support for Mandrill send options *auto_html*, *tracking_domain* and *signing_domain*.

Version 0.4:

- Attachments with a Content-ID are now treated as *embedded images*
- New Mandrill *inline_css* option is supported
- Remove limitations on attachment types, to track Mandrill change
- Documentation is now available on djrill.readthedocs.org

Version 0.3:

- *Attachments* are now supported
- *Mandrill templates* are now supported
- A bcc address is now passed to Mandrill as bcc, rather than being lumped in with the "to" recipients. Multiple bcc recipients will now raise an exception, as Mandrill only allows one.
- Python 3 support (with Django 1.5)
- Exceptions should be more useful: *djrill.NotSupportedByMandrillError* replaces generic ValueError; *djrill.MandrillAPIError* replaces DjrillBackendHTTPError, and is now derived from requests.HTTPError. (New exceptions are backwards compatible with old ones for existing code.)

Version 0.2:

- `MANDRILL_API_URL` is no longer required in settings.py
- Earlier versions of Djrill required use of a `DjrillMessage` class to specify Mandrill-specific options. This is no longer needed – Mandrill options can now be set directly on a Django `EmailMessage` object or any subclass. (Existing code can continue to use `DjrillMessage`.)

CHAPTER 2

Thanks

Thanks to the MailChimp team for asking us to build this nifty little app, and to all of Djrill's contributors. Oh, and, of course, Kenneth Reitz for the awesome requests library.