zamboni Documentation

Release 0.9

Marketplace developers

May 09, 2017

Contents

Zamboni is one of the codebases for https://marketplace.firefox.com/

The source lives at https://github.com/mozilla/zamboni

CHAPTER 1

Installation

Before you install zamboni, we strongly recommend you start with the Marketplace Documentation which illustrates how the Marketplace is comprised of multiple components, one of which is zamboni.

What are you waiting for?! *Install Zamboni!*

Want to know about how development at Mozilla works, including style guides? Mozilla Bootcamp

CHAPTER 2

Contents

Install Zamboni

Installing Zamboni

We're going to use all the hottest tools to set up a nice environment. Skip steps at your own peril. Here we go!

Need help?

Come talk to us on irc://irc.mozilla.org/marketplace if you have questions, issues, or compliments.

1. Installing dependencies

On OS X

The best solution for installing UNIX tools on OS X is Homebrew.

The following packages will get you set for zamboni:

```
brew install python libxml2 mysql openssl swig304 jpeg pngcrush redis
```

On Ubuntu

The following command will install the required development files on Ubuntu or, if you're running a recent version, you can install them automatically:

```
sudo aptitude install python-dev python-virtualenv libxml2-dev libxslt1-dev libmysqlclient-dev libssl
```

Services

Zamboni has three dependencies you must install and have running:

- MySQL should require no configuration.
- Redis should require no configuration.
- Seee *elasticsearch* for setup and configuration.

2. Grab the source

Grab zamboni from github with:

```
git clone --recursive git://github.com/mozilla/zamboni.git
cd zamboni
```

`zamboni.git` is all the source code. updating is detailed later on.

If at any point you realize you forgot to clone with the recursive flag, you can fix that by running:

```
git submodule update --init --recursive
```

3. Setup a virtualenv

virtualenv is a tool to create isolated Python environments. This will let you put all of Zamboni's dependencies in a single directory rather than your global Python directory. For ultimate convenience, we'll also use virtualenvwrapper which adds commands to your shell.

Since each shell setup is different, you can install everything you need and configure your shell using the virtualenv-burrito. Type this:

```
curl -sL https://raw.githubusercontent.com/brainsik/virtualenv-burrito/master/virtualenv-burrito.sh
```

Open a new shell to test it out. You should have the `workon` and `mkvirtualenv` commands.

4. Getting Packages

Now we're ready to go, so create an environment for zamboni:

```
mkvirtualenv zamboni
```

That creates a clean environment named zamboni using Python 2.7. You can get out of the environment by restarting your shell or calling `deactivate`.

To get back into the zamboni environment later, type:

```
workon zamboni  # requires virtualenvwrapper
```

Note: Zamboni requires at least Python 2.7.0, production is using Python 2.7.5.

Note: If you want to use a different Python binary, pass the name (if it is on your path) or the full path to mkvirtualenv with `--python`:

```
mkvirtualenv --python=/usr/local/bin/python2.7 zamboni
```

Note: If you are using an older version of virtualenv that defaults to using system packages you might need to pass `--no-site-packages`:

```
mkvirtualenv --no-site-packages zamboni
```

First make sure you have a recent **'pip'**_ for security reasons. From inside your activated virtualenv, install the required python packages:

```
make update_deps
```

Issues at this point? See Trouble-shooting the development installation.

5. Settings

Most of zamboni is already configured in `mkt.settings.py`, but there's one thing you'll need to configure locally, the database. The easiest way to do that is by setting an environment variable (see next section).

Optionally you can create a local settings file and place anything custom into `settings_local.py`.

Any file that looks like `settings_local*` is for local use only; it will be ignored by git.

Environment settings

Out of the box, zamboni should work without any need for settings changes. Some settings are configurable from the environment. See the marketplace docs for information on the environment variables and how they affect zamboni.

6. Setting up a Mysql Database

Django provides commands to create the database and tables needed, and load essential data:

```
./manage.py migrate
./manage.py loaddata init
```

Database Migrations

Each incremental change we add to the database is done with Django migrations. To keep your local DB fresh and up to date, run migrations like this:

```
./manage.py migrate
```

Loading Test Apps

Fake apps and feed collections can be created by running:

```
./manage.py generate_feed
```

Specific example applications can be loaded by running:

```
./manage.py generate_apps_from_spec data/apps/test_apps.json
```

See Fake App Data for details of the JSON format.

If you just want a certain number of public apps in various categories to be created, run:

```
./manage.py generate_apps N
```

where N is the number of apps you want created in your database.

7. Check it works

If you've gotten the system requirements, downloaded `zamboni`, set up your virtualenv with the compiled packages, and configured your settings and database, you're good to go:

```
./manage.py runserver
```

Hit:

```
http://localhost:2600/services/monitor
```

This will report any errors or issues in your installation.

8. Create an admin user

Chances are that for development, you'll want an admin account.

After logging in, run this management command:

```
./manage.py addusertogroup <your email> 1
```

9. Setting up the pages

To set up the assets for the developer hub, reviewer tools, or admin pages:

```
npm install
python manage.py compress_assets
```

For local development, it would also be good to set:

```
TEMPLATE_DEBUG = True
```

Post installation

To keep your zamboni up to date with the latest changes in source files, requrirements and database migrations run:

```
make full_update
```

Advanced Installation

In production we use things like memcached, rabbitmq + celery and Stylus. Learn more about installing these on the Optional installs page.

Note: Although we make an effort to keep advanced items as optional installs you might need to install some components in order to run tests or start up the development server.

Optional installs

MySQL

On your dev machine, MySQL probably needs some tweaks. Locate your my.cnf (or create one) then, at the very least, make UTF8 the default encoding:

```
[mysqld]
character-set-server=utf8
```

Here are some other helpful settings:

```
[mysqld]
default-storage-engine=innodb
character-set-server=utf8
skip-sync-frm=OFF
innodb_file_per_table
```

On Mac OS X with homebrew, put my.cnf in `/usr/local/Cellar/mysql/5.5.15/my.cnf` then restart like:

```
launchctl unload -w ~/Library/LaunchAgents/com.mysql.mysqld.plist
launchctl load -w ~/Library/LaunchAgents/com.mysql.mysqld.plist
```

Note: some of the options above were renamed between MySQL versions

Here are more tips for optimizing MySQL on your dev machine.

Memcached

By default zamboni uses an in memory cache. To install memcached `libmemcached-dev` on Ubuntu and `libmemcached` on OS X. Alter your local settings file to use:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': ['localhost:11211'],
        'TIMEOUT': 500,
    }
}
```

RabbitMQ and Celery

By default zamboni automatically processes jobs without needing Celery.

See the Celery page for installation instructions. The example settings set `CELERY_ALWAYS_EAGER = True`. If you're setting up RabbitMQ and want to use `celery worker` you will need to alter your local settings file to set this up.

See Celery for more instructions.

Node.js

Node.js is needed for Stylus and LESS, which in turn are needed to precompile the CSS files.

If you want to serve the CSS files from another domain than the webserver, you will need to precompile them. Otherwise you can have them compiled on the fly, using javascript in your browser, if you set `LESS_PREPROCESS = False` in your local settings.

First, we need to install node and npm:

```
brew install node
curl http://npmjs.org/install.sh | sh
```

Optionally make the local scripts available on your path if you don't already have this in your profile:

```
export PATH="./node_modules/.bin/:${PATH}"
```

Not working?

- If you're having trouble installing node, try http://shapeshed.com/journal/setting-up-nodejs-and-npm-on-mac-osx/. You need brew, which we used earlier.
- If you're having trouble with npm, check out the README on https://github.com/isaacs/npm

Stylus CSS

Learn about Stylus at http://learnboost.github.com/stylus/

```
cd zamboni
npm install
```

In your `settings_local.py` (or `settings_local_mkt.py`) ensure you are pointing to the correct executable for `stylus`:

```
STYLUS_BIN = path('node_modules/stylus/bin/stylus')
```

Celery

Celery is a task queue powered by RabbitMQ. You can use it for anything that doesn't need to complete in the current request-response cycle. Or use it wherever Les tells you to use it.

For example, each addon has a `current_version` cached property. This query on initial run causes strain on our database. We can create a denormalized database field called `current_version` on the `addons` table.

We'll need to populate regularly so it has fairly up-to-date data. We can do this in a process outside the request-response cycle. This is where Celery comes in.

Installation

RabbitMQ

Celery depends on RabbitMQ. If you use `homebrew` you can install this:

```
brew install rabbitmq
```

Setting up rabbitmq involves some configuration. You may want to define the following

```
# On a Mac, you can find this in System Preferences > Sharing
export HOSTNAME='<laptop name>.local'
```

Then run the following commands:

```
# Set your host up so it's semi-permanent
sudo scutil --set HostName $HOSTNAME

# Update your hosts by either:
# 1) Manually editing /etc/hosts
# 2) `echo 127.0.0.1 $HOSTNAME >> /etc/hosts`

# RabbitMQ insists on writing to /var
sudo rabbitmq-server -detached

# Setup rabitty things (sudo is required to read the cookie file)
sudo rabbitmqctl add_user zamboni zamboni
sudo rabbitmqctl add_vhost zamboni
sudo rabbitmqctl set_permissions -p zamboni zamboni ".*" ".*" ".*"
```

Back in safe and happy django-land you should be able to run:

```
./manage.py celery worker -Q priority,devhub,images,limited  $OPTIONS
```

Celery understands python and any tasks that you have defined in your app are now runnable asynchronously.

Celery Tasks

Any python function can be set as a celery task. For example, let's say we want to update our `current_version` but we don't care how quickly it happens, just that it happens. We can define it like so:

```
@task(rate_limit='2/m')
def _update_addons_current_version(data, **kw):
    task_log.debug("[%s@%s] Updating addons current_versions." %
                   (len(data), _update_addons_current_version.rate_limit))
    for pk in data:
        try:
            addon = Webapp.objects.get(pk=pk)
            addon.update_version()
        except Webapp.DoesNotExist:
            task_log.debug("Missing addon: %d" % pk)
```

`@task` is a decorator for Celery to find our tasks. We can specify a `rate_limit` like `2/m` which means `celery worker` will only run this command 2 times a minute at most. This keeps write-heavy tasks from killing your database.

If we run this command like so:

```
from celery.task.sets import TaskSet

all_pks = Webapp.objects.all().values_list('pk', flat=True)
ts = [_update_addons_current_version.subtask(args=[pks])
      for pks in mkt.site.utils.chunked(all_pks, 300)]
TaskSet(ts).apply_async()
```

All the Webapps with ids in `pks` will (eventually) have their `current_versions` updated.

Cron Jobs

This is all good, but let's automate this. In Zamboni we can create cron jobs like so:

```
@cronjobs.register
def update_addons_current_version():
    """Update the current_version field of the addons."""
    d = Webapp.objects.valid().values_list('id', flat=True)

    with establish_connection() as conn:
        for chunk in chunked(d, 1000):
            print chunk
            _update_addons_current_version.apply_async(args=[chunk],
                                                       connection=conn)
```

This job will hit all the addons and run the task we defined in small batches of 1000.

We'll need to add this to both the `prod` and `preview` crontabs so that they can be run in production.

Better than Cron

Of course, cron is old school. We want to do better than cron, or at least not rely on brute force tactics.

For a surgical strike, we can call `_update_addons_current_version` any time we add a new version to that addon. Celery will execute it at the prescribed rate, and your data will be updated ... eventually.

During Development

`celery worker` only knows about code as it was defined at instantiation time. If you change your `@task` function, you'll need to `HUP` the process.

However, if you've got the `@task` running perfectly you can tweak all the code, including cron jobs that call it without need of restart.

Elasticsearch

Elasticsearch is a search server. Documents (key-values) get stored, configurable queries come in, Elasticsearch scores these documents, and returns the most relevant hits.

Installation

You can download the Elasticsearch code and run elasticsearch directly from this folder. This makes it easy to upgrade or test new versions as needed. Optionally you can install Elasticsearch using your preferred system package manager.

We are currently using Elasticsearch version 1.6.2. You can install by doing the following:

```
curl -O https://download.elasticsearch.org/elasticsearch/elasticsearch/elasticsearch-1.6.2.tar.gz
tar -xvzf elasticsearch-1.6.2.tar.gz
cd elasticsearch-1.6.2
```

For running Marketplace you must install the ICU Analysis Plugin:

```
./bin/plugin -install elasticsearch/elasticsearch-analysis-icu/2.6.0
```

For more about the ICU plugin, see the ICU Github Page.

Settings

```
cluster.name: wooyeah

# Don't try to cluster with other machines during local development.
# Remove the following 3 lines to enable default clustering.
network.host: localhost
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["localhost"]

script.disable_dynamic: false

path:
  logs: /usr/local/var/log
  data: /usr/local/var/data
```

We use a custom analyzer for indexing add-on names since they're a little different from normal text.

To get the same results as our servers, configure Elasticsearch by copying the scripts/elasticsearch/elasticsearch.yml (available in the `scripts/elasticsearch/` folder of your install) to your system.

For example, copy it to the local directory so it's nearby when you launch Elasticsearch:

```
cp /path/to/zamboni/scripts/elasticsearch/elasticsearch.yml .
```

If you don't do this your results will be slightly different, but you probably won't notice.

Launching and Setting Up

Launch the Elasticsearch service:

```
./bin/elasticsearch -Des.config=elasticsearch.yml
```

Zamboni has commands that sets up mappings and indexes for you. Setting up the mappings is analagous to defining the structure of a table, indexing is analagous to storing rows.

It is worth noting that the index is maintained incrementally through post_save and post_delete hooks.

Use this to create the apps index and index apps:

```
./manage.py reindex --index=apps
```

Or you could use the makefile target (using the `settings_local.py` file):

```
make reindex
```

If you need to use another settings file and add arguments:

```
make SETTINGS=settings_other ARGS='--force' reindex
```

Querying Elasticsearch in Django

We use Elasticsearch DSL, a Python library that gives us a search API to elasticsearch.

On Marketplace, apps use `mkt/webapps/indexers:WebappIndexer` as its interface to Elasticsearch:

```
query_results = WebappIndexer.search().query(...).filter(...).execute()
```

Testing with Elasticsearch

All test cases using Elasticsearch should inherit from `mkt.site.tests.ESTestCase`. All such tests will be skipped by the test runner unless:

```
RUN_ES_TESTS = True
```

This is done as a performance optimization to keep the run time of the test suite down, unless necessary.

Troubleshooting

I got a CircularReference error on .search() - check that a whole object is not being passed into the filters, but rather just a field's value.

I indexed something into Elasticsearch, but my query returns nothing - check whether the query contains upper-case letters or hyphens. If so, try lowercasing your query filter. For hyphens, set the field's mapping to not be analyzed:

```
'my_field': {'type': 'string', 'index': 'not_analyzed'}
```

Packaging in Zamboni

There are two ways of getting packages for zamboni. The first is to install everything using pip. We have our packages separated into three files:

requirements/compiled.txt All packages that require (or go faster with) compilation. These can't be distributed cross-platform, so they need to be installed through your system's package manager or pip.

requirements/prod.txt The minimal set of packages you need to run zamboni in production. You also need to get `requirements/compiled.txt`.

requirements/dev.txt All the packages needed for running tests and development servers. This automatically includes `requirements/prod.txt`.

Installing through pip

You can get a development environment with

```
pip install --no-deps -r requirements/dev.txt
```

Adding new packages

Note: this is deprecated, all packages should be added in requirements.

The vendor repo was seeded with

```
pip install --no-install --build=vendor/packages --src=vendor/src -I -r requirements/dev.txt
```

Then I added everything in `/packages` and set up submodules in `/src` (see below). We'll be keeping this up to date through Hudson, but if you add new packages you should seed them yourself.

If we wanted to add a new dependency called `cheeseballs` to zamboni, you would add it to `requirements/prod.txt` or `requirements/dev.txt` and then do

```
pip install --no-install --build=vendor/packages --src=vendor/src -I cheeseballs
```

Then you need to update `vendor/zamboni.pth`. Python uses `.pth` files to dynamically add directories to `sys.path` (docs).

I created `zamboni.pth` with this:

```
find packages src -type d -depth 1 > zamboni.pth
```

`html5lib` and `selenium` are troublesome, so they need to be sourced with `packages/html5lib/src` and `packages/selenium/src`. Hopefully you won't hit any snags like that.

Adding submodules

Note: this is deprecated, all packages should be added in requirements.

```
for f in src/*
    pushd $f >/dev/null && REPO=$(git config remote.origin.url) && popd > /dev/null && git submodule
```

Holy readability batman!

Trouble-shooting the development installation

M2Crypto installation

If you are on a Linux box and get a compilation error while installing M2Crypto like the following:

```
SWIG/_m2crypto_wrap.c:6116:1: error: unknown type name 'STACK'

... snip a very long output of errors around STACK...

SWIG/_m2crypto_wrap.c:23497:20: error: expected expression before ')' token

   result = (STACK *)pkcs7_get0_signers(arg1,arg2,arg3);

                    ^

error: command 'gcc' failed with exit status 1
```

It may be because of a few reasons:

- comment the line starting with `M2Crypto` in `requirements/compiled.txt`
- install the patched package from the Debian repositories (replace `x86_64-linux-gnu` by `i386-linux-gnu` if you're on a 32bits platform):

```
    DEB_HOST_MULTIARCH=x86_64-linux-gnu pip install -I --exists-action=w "git+git://anonscm.debian.c
    pip install --no-deps -r requirements/dev.txt
```

- revert your changes to `requirements/compiled.txt`:

```
    git checkout requirements/compiled.txt
```

Pillow

As of Mac OS X Mavericks, you might see this error when pip builds Pillow:

```
clang: error: unknown argument: '-mno-fused-madd' [-Wunused-command-line-argument-hard-error-in-futu

clang: note: this will be a hard error (cannot be downgraded to a warning) in the future

error: command 'cc' failed with exit status 1
```

You can solve this by setting these environment variables in your shell before running `pip install ...`:

```
export CFLAGS=-Qunused-arguments
export CPPFLAGS=-Qunused-arguments
pip install ...
```

More info: http://stackoverflow.com/questions/22334776/installing-pillow-pil-on-mavericks/22365032

Image processing isn't working

If adding images to apps or extensions doesn't seem to work then there's a couple of settings you should check.

Checking your PIL installation (Ubuntu)

When you run you should see at least JPEG and ZLIB are supported

If that's the case you should see this in the output of `pip install -I PIL`:

```
--------------------------------------------------------------------
*** TKINTER support not available
--- JPEG support available
--- ZLIB (PNG/ZIP) support available
*** FREETYPE2 support not available
*** LITTLECMS support not available
--------------------------------------------------------------------
```

If you don't then this suggests PIL can't find your image libraries:

To fix this double-check you have the necessary development libraries installed first (e.g: `sudo apt-get install libjpeg-dev zlib1g-dev`)

Now run the following for 32bit:

```
sudo ln -s /usr/lib/i386-linux-gnu/libz.so /usr/lib
sudo ln -s /usr/lib/i386-linux-gnu/libjpeg.so /usr/lib
```

Or this if your running 64bit:

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libz.so /usr/lib
sudo ln -s /usr/lib/x86_64-linux-gnu/libjpeg.so /usr/lib
```

Note: If you don't know what arch you are running run `uname -m` if the output is `x86_64` then it's 64-bit, otherwise it's 32bit e.g. `i686`

Now re-install PIL:

```
pip install -I PIL
```

And you should see the necessary image libraries are now working with PIL correctly.

ES is timing out

This can be caused by `number_of_replicas` not being set to 0 for the local indexes.

To check the settings run:

```
curl -s localhost:9200/_cluster/state\?pretty | fgrep number_of_replicas -B 5
```

If you see any that aren't 0 do the following:

Set `ES_DEFAULT_NUM_REPLICAS` to `0` in your local settings.

To set them to zero immediately run:

```
curl -XPUT localhost:9200/_settings -d '{ "index" : { "number_of_replicas" : 0 } }'
```

Hacking

Testing

We're using a mix of Django's Unit Testing and `nose`.

Running Tests

To run the whole shebang use:

```
python manage.py test
```

There are a lot of options you can pass to adjust the output. Read the docs for the full set, but some common ones are:

- `-P` to prevent nose adding the *lib* directory to the path.
- `--noinput` tells Django not to ask about creating or destroying test databases.
- `--logging-clear-handlers` tells nose that you don't want to see any logging output. Without this, our debug logging will spew all over your console during test runs. This can be useful for debugging, but it's not that great most of the time. See the docs for more stuff you can do with `nose and logging`.

Our continuous integration server adds some additional flags for other features (for example, coverage statistics). To see what those commands are check out the .travis.yml file.

There are a few useful makefile targets that you can use:

Run all the tests:

```
make test
```

If you need to rebuild the database:

```
make test_force_db
```

To fail and stop running tests on the first failure:

```
make tdd
```

If you wish to add arguments, or run a specific test, overload the variables (check the Makefile for more information):

```
make SETTINGS=settings_mkt ARGS='--verbosity 2 zamboni.mkt.site.tests.test_url_prefix:MiddlewareTest
```

Those targets include some useful options, like the `--with-id` which allows you to re-run only the tests failed from the previous run:

```
make test_failed
```

Database Setup

If you want to re-use your database instead of making a new one every time you run tests, set the environment variable `REUSE_DB`.

```
REUSE_DB=1 python manage.py test
```

Writing Tests

We support two types of automated tests right now and there are some details below but remember, if you're confused look at existing tests for examples.

Unit/Functional Tests

Most tests are in this category. Our test classes extend `django.test.TestCase` and follow the standard rules for unit tests. We're using JSON fixtures for the data.

External calls

Connecting to remote services in tests is not recommended, developers should **mock_** out those calls instead.

To enforce this we run Jenkins with the nose-blockage plugin, that will raise errors if you have an HTTP calls in your tests apart from calls to the domains *127.0.0.1* and *localhost*.

Why Tests Fail

Tests usually fail for one of two reasons: The code has changed or the data has changed. An third reason is **time**. Some tests have time-dependent data usually in the fixtues. For example, some featured items have expiration dates.

We can usually save our future-selves time by setting these expirations far in the future.

Localization Tests

If you want test that your localization works then you can add in locales in the test directory. For an example see `devhub/tests/locale`. These locales are not in the normal path so should not show up unless you add them to the *LOCALE_PATH*. If you change the .po files for these test locales, you will need to recompile the .mo files manually, for example:

```
msgfmt --check-format -o django.mo django.po
```

Testing emails

By default in a non-production enviroment the setting *REAL_EMAIL* is set to False, which prevents emails from being sent to addresses during testing with live data. The contents of the emails are saved in the database instead and can be read with the Fake email admin tool at `/admin/mail`.

Sending actual email

In some circumstance you want to still recieve some emails, even when *REAL_EMAIL* is False. To allow addresses to receive emails, rather than be redirected to `/admin/mail`, use `mkt.zadmin.models.set_config` to set the `real_email_allowed_regex` key to a comma separated list of valid emails in regex format:

```
from mkt.zadmin.models import set_config
set_config('real_email_allowed_regex', '.+@mozilla\.com$,you@who\.ca$')
```

Contributing

The easiest way to let us know about your awesome work is to send a pull request on github or in IRC. Point us to a branch with your new code and we'll go from there. You can attach a patch to a bug if you're more comfortable that way.

Please read the style.

The Perfect Git Configuration

We're going to talk about two git repositories:

- *origin* will be the main zamboni repo at http://github.com/mozilla/zamboni.
- *mine* will be your fork at http://github.com/:user/zamboni.

There should be something like this in your `.git/config` already:

```
[remote "origin"]
    url = git://github.com/mozilla/zamboni.git
    fetch = +refs/heads/*:refs/remotes/origin/*
```

Now we'll set up your master to pull directly from the upstream zamboni:

```
[branch "master"]
    remote = origin
    merge = master
    rebase = true
```

This can also be done through the `git config` command (e.g. `git config branch.master.remote origin`) but editing `.git/config` is often easier.

After you've forked the repository on github, tell git about your new repo:

```
git remote add -f mine git@github.com:user/zamboni.git
```

Make sure to replace *user* with your name.

Working on a Branch

Let's work on a bug in a branch called *my-bug*:

```
git checkout -b my-bug master
```

Now we're switched to a new branch that was copied from master. We like to work on feature branches, but the master is still moving along. How do we keep up?

```
git fetch origin && git rebase origin/master
```

If you want to keep the master branch up to date, do it this way:

```
git checkout master && git pull && git checkout @{-1} && git rebase master
```

That updated master and then switched back to update our branch.

Publishing your Branch

The syntax is `git push <repository> <branch>`. Here's how to push the `my-bug` branch to your clone:

```
git push mine my-bug
```

Push From Master

We deploy from the master branch once a week. If you commit something to master that needs additional QA time, be sure to use a waffle feature flag.

Local Branches

Most new code is developed in local one-off branches, usually encompassing one or two patches to fix a bug. Upstream doesn't care how you do local development, but we don't want to see a merge commit every time you merge a single patch from a branch. Merge commits make for a noisy history, which is not fun to look at and can make it difficult to cherry-pick hotfixes to a release branch. We do like to see merge commits when you're committing a set of related patches from a feature branch. The rule of thumb is to rebase and use fast-forward merge for single patches or a branch of unrelated bug fixes, but to use a merge commit if you have multiple commits that form a cohesive unit.

Here are some tips on Using topic branches and interactive rebasing effectively.

Access Control Lists

ACL versus Django Permissions

Currently we use the `is_superuser` flag in the `User` model to indicate that a user can access the admin site.

Outside of that we use the `GroupUser` to define what access groups a user is a part of. We store this in `request.groups`.

How permissions work

Permissions that you can use as filters can be either explicit or general.

For example `Admin:EditAddons` means only someone with that permission will validate.

If you simply require that a user has *some* permission in the *Admin* group you can use `Admin:%`. The `%` means "any."

Similarly a user might be in a group that has explicit or general permissions. They may have `Admin:EditAddons` which means they can see things with that same permission, or things that require `Admin:%`.

If a user has a wildcard, they will have more permissions. For example, `Admin:*` means they have permission to see anything that begins with `Admin:`.

The notion of a superuser has a permission of `*:*` and therefore they can see everything.

Fake App Data

The *generate_apps_from_spec* command-line tool loads a JSON file containing an array of fake app objects. The fields that can be specified in these objects are:

type Required. One of "hosted", "web", or "privileged", to specify a hosted app, unprivileged packaged app, or privileged packaged app.

status Required. A string representing the app status, as listed in `mkt.constants.base.STATUS_CHOICES_API`.

name The display name for the app.

num_ratings Number of user ratings to create for this app.

num_previews Number of screenshots to create for this app.

preview_files List of paths (relative to the JSON file) of preview images.

video_files List of paths (relative to the JSON file) of preview videos in webm format.

num_locales Number of locales to localize this app's name in (max 5).

versions An array of objects with optional version-specific fields: "status", "type", and "permissions". Additional versions with these fields are created, oldest first.

permissions An array of app permissions, to be placed in the manifest.

manifest_file Path (relative to the JSON file) of a manifest to create this app from.

description Description text for the app.

categories A list of category names to create the app in.

developer_name Display name for the app author.

developer_email An email address for the app author.

device_types A list of devices the app is available on: one or more of "desktop", "mobile", "tablet", and "firefoxos".

premium_type Payment status for the app. One of "free", "premium", "premium-inapp", "free-inapp", or "other".

price The price (in dollars) for the app.

privacy_policy Privacy policy text.

rereview Boolean indicating whether to place app in rereview queue.

special_regions An object with region names as keys, and status strings as values. Adds app to special region with given status.

Logging

Logging is fun. We all want to be lumberjacks. My muscle-memory wants to put `print` statements everywhere, but it's better to use `log.debug` instead. Plus, `django-debug-toolbar` can hijack the logger and show all the log statements generated during the last request. When `DEBUG = True`, all logs will be printed to the development console where you started the server. In production, we're piping everything into `syslog`.

Configuration

The root logger is set up from `log_settings.py` in the base of zamboni's tree. It sets up sensible defaults, but you can twiddle with these settings:

LOG_LEVEL This setting is required, and defaults to `loggging.DEBUG`, which will let just about anything pass through. To reconfigure, import logging in your settings file and pick a different level:

```
import logging
LOG_LEVEL = logging.WARN
```

HAS_SYSLOG Set this to `False` if you don't want logging sent to syslog when `DEBUG` is `False`.

LOGGING See PEP 391 and log_settings.py for formatting help. Each section of LOGGING will get merged into the corresponding section of log_settings.py. Handlers and log levels are set up automatically based on LOG_LEVEL and DEBUG unless you set them here. Messages will not propagate through a logger unless propagate: True is set.

```
LOGGING = {
    'loggers': {
        'foobar': {'handlers': ['null']},
    },
}
```

If you want to add more to this in `settings_local.py`, do something like this:

```
LOGGING['loggers'].update({
    'z.paypal': {
        'level': logging.DEBUG,
    },
    'z.elasticsearch': {
        'handlers': ['null'],
    },
})
```

Using Loggers

The `logging` package uses global objects to make the same logging configuration available to all code loaded in the interpreter. Loggers are created in a pseudo-namespace structure, so app-level loggers can inherit settings from a root logger. zamboni's root namespace is just `"z"`, in the interest of brevity. In the foobar package, we create a logger that inherits the configuration by naming it `"z.foobar"`:

```
import commonware.log

log = commonware.log.getLogger('z.foobar')

log.debug("I'm in the foobar package.")
```

Logs can be nested as much as you want. Maintaining log namespaces is useful because we can turn up the logging output for a particular section of zamboni without becoming overwhelmed with logging from all other parts.

commonware.log vs. logging

`commonware.log.getLogger` should be used inside the request cycle. It returns a `LoggingAdapter` that inserts the current user's IP address into the log message.

Complete logging docs: http://docs.python.org/library/logging.html

Services

Services contain a couple of scripts that are run as seperate wsgi instances on the services. Usually they are hosted on seperate domains. They are stand alone wsgi scripts. The goal is to avoid a whole pile of Django imports, middleware, sessions and so on that we really don't need.

To run the scripts you'll want a wsgi server. You can do this using gunicorn, for example:

```
pip install gunicorn
```

Then you can do:

```
cd services
gunicorn --log-level=DEBUG -c wsgi/receiptverify.py -b 127.0.0.1:9000 --debug verify:application
```

To test:

```
curl -d "this is a bogus receipt" http://127.0.0.1:9000/verify/123
```

Translations

gettext in JavaScript

We have gettext in JavaScript! Just mark your strings with `gettext()` or `ngettext()`. There isn't an _ alias right now, since underscore.js has that. If we end up with a lot of JS translations, we can fix that. Check it out:

```
cd locale
./extract-po.py -d javascript
pybabel init -l en_US -d . -i javascript.pot -D javascript
perl -pi -e 's/fuzzy//' en_US/LC_MESSAGES/javascript.po
pybabel compile -d . -D javascript
open http://0:8000/en-US/jsi18n/
```

Model fields

The `translations` app defines a `Translation` model, but for the most part, you shouldn't have to use that directly. When you want to create a foreign key to the `translations` table, use `translations.fields.TranslatedField`. This subclasses Django's `django.db.models.ForeignKey` to make it work with our special handling of translation rows.

A minimal model with translations in zamboni would look like this:

```
from django.db import models

import mkt.site.models
import mkt.translations

class MyModel(mkt.site.models.ModelBase):
    description = translations.fieldsTranslatedField()

models.signals.pre_save.connect(translations.fields.save_signal,
                                sender=MyModel,
                                dispatch_uid='mymodel_translations')
```

How it works behind the scenes

As mentioned above, a `TranslatedField` is actually a `ForeignKey` to the `translations` table. However, to support multiple languages, we use a special feature of MySQL allowing you to have a `ForeignKey` pointing to multiple rows.

When querying

Our base manager has a `_with_translations()` method that is automatically called when you instanciate a queryset. It does 2 things:

- Stick an extra lang=lang in the query to prevent query caching from returning objects in the wrong language
- Call `translations.transformers.get_trans()` which does the black magic.

`get_trans()` is called, and calls in turn `translations.transformer.build_query()` and builds a custom SQL query. This query is the heart of the magic. For each field, it setups a join on the translations table, trying to find a translation in the current language (using `translation.get_language()`) and then in the language returned by `get_fallback()` on the instance (for addons, that's `default_locale`; if the `get_fallback()` method doesn't exist, it will use `settings.LANGUAGE_CODE`, which should be `en-US` in zamboni).

Only those 2 languages are considered, and a double join + `IF` / `ELSE` is done every time, for each field.

This query is then ran on the slave (`get_trans()` gets a cursor using `connections[multidb.get_slave()]`) to fetch the translations, and some Translation objects are instantiated from the results and set on the instance(s) of the original query.

To complete the mechanism, `TranslationDescriptor.__get__` returns the `Translation`, and `Translations.__unicode__` returns the translated string as you'd expect, making the whole thing transparent.

When setting

Everytime you set a translated field to a string value, `TranslationDescriptor __set__` method is called. It determines which method to call (because you can also assign a dict with multiple translations in multiple languages at the same time). In this case, it calls `translation_from_string()` method, still on the "hidden" `TranslationDescriptor` instance. The current language is passed at this point, using `translation_utils.get_language()`.

From there, `translation_from_string()` figures out whether it's a new translation of a field we had no translation for, a new translation of a field we already had but in a new language, or an update to an existing translation.

It instantiates a new `Translation` object with the correct values if necessary, or just updates the correct one. It then places that object in a queue of Translation instances to be saved later.

When you eventually call `obj.save()`, the `pre_save` signal is sent. If you followed the example above, that means `translations.fields.save_signal` is then called, and it unqueues all Translation objects and saves them. It's important to do this on `pre_save` to prevent foreign key constraint errors.

When deleting

Deleting all translations for a field is done using `delete_translation()`. It sets the field to `NULL` and then deletes all the attached translations.

Deleting a *specific* translation (like a translation in spanish, but keeping the english one intact) is implemented but not recommended at the moment. The reason why is twofold:

1. MySQL doesn't let you delete something that still has a FK pointing to it, even if there are other rows that match the FK. When you call `delete()` on a translation, if it was the last translation for that field, we set the FK to `NULL` and delete the translation normally. However, if there were any other translations, instead we temporarily disable the constraints to let you delete just the one you want.
2. Remember how fetching works? If you deleted a translation that is part of the fallback, then when you fetch that object, depending on your locale you'll get an empty string for that field, even if there are `Translation` objects in other languages available!

For additional discussion on this topic, see https://bugzilla.mozilla.org/show_bug.cgi?id=902435

Additional tricks

In addition to the above, `apps/translations/__init__.py` monkeypatches Django to bypass errors thrown because we have a `ForeignKey` pointing to multiple rows.

Also, you might be interested in `translations.query.order_by_translation`. Like the name suggests, it allows you to order a `QuerySet` by a translated field, honoring the current and fallback locales like it's done when querying.

How to build these docs

To simply build the docs:

```
make docs
```

If you're working on the docs, use `make loop` to keep your built pages up-to-date:

```
cd docs && make loop
```

/api

```
GET /api/v1/reviewers/canned-responses/,
      ??
GET /api/v1/reviewers/canned-responses/(int:id)/,
      ??
GET /api/v1/reviewers/scores/,??
GET /api/v1/reviewers/scores/(int:id)/,
      ??
GET /api/v2/account/installed/mine/,??
GET /api/v2/account/operators/,??
GET /api/v2/account/permissions/mine/,
      ??
GET /api/v2/account/settings/mine/,??
GET /api/v2/account/shelves/,??
GET /api/v2/apps/(int:id)|(string:slug)/privacy/,
      ??
GET /api/v2/apps/app/,??
GET /api/v2/apps/app/(int:id)/payments/,
      ??
GET /api/v2/apps/app/(int:id)/payments/debug/,
      ??
GET /api/v2/apps/app/(int:id)|(string:slug)/,
      ??
GET /api/v2/apps/app/(int:id|string:app_slug)/content_ratings,
      ??
GET /api/v2/apps/category/,??
GET /api/v2/apps/category/(string:slug)/,
      ??
GET /api/v2/apps/features/,??
GET /api/v2/apps/rating/,??
GET /api/v2/apps/rating/(int:id)/,??
GET /api/v2/apps/search/,??
GET /api/v2/apps/search/?tag=featured-game,
      ??
GET /api/v2/apps/search/?tag=featured-game-[adventure,
    action, puzzle, strategy],??
GET /api/v2/apps/versions/(int:id)/,??
GET /api/v2/comm/app/(int:id|string:slug)/,
      ??
GET /api/v2/comm/thread/,??
GET /api/v2/comm/thread/(int:id)/,??
GET /api/v2/comm/thread/(int:thread_id)/note/,
      ??
GET /api/v2/comm/thread/(int:thread_id)/note/(int:i
      ??
GET /api/v2/extensions/extension/,??
GET /api/v2/extensions/extension/(int:id)|(string:s
      ??
GET /api/v2/extensions/extension/(int:id)|(string:s
      ??
GET /api/v2/extensions/extension/(int:id)|(string:s
      ??
GET /api/v2/extensions/queue/,??
GET /api/v2/extensions/search/,??
GET /api/v2/extensions/validation/(string:id)/,
      ??
GET /api/v2/feed/apps/,??
GET /api/v2/feed/apps/(int:id)/,??
GET /api/v2/feed/apps/(int:id|string:slug)/image/,
      ??
GET /api/v2/feed/brands/,??
GET /api/v2/feed/brands/(int:id)/,??
GET /api/v2/feed/collections/,??
GET /api/v2/feed/collections/(int:id)/,
      ??
GET /api/v2/feed/elements/search?q=(str:q),
      ??
GET /api/v2/feed/get/?carrier=(str:carrier)&region=
      ??
GET /api/v2/feed/items/,??
GET /api/v2/feed/items/(int:id)/,??
GET /api/v2/feed/shelves/,??
GET /api/v2/feed/shelves/(int:id|string:slug)/,
      ??
GET /api/v2/feed/shelves/(int:id|string:slug)/image
      ??
GET /api/v2/fireplace/app/,??
GET /api/v2/fireplace/consumer-info/,
      ??
GET /api/v2/fireplace/multi-search/,??
GET /api/v2/fireplace/search/,??
```

```
GET /api/v2/fireplace/search/featured/, ??
GET /api/v2/games/daily/, ??
GET /api/v2/langpacks/, ??
GET /api/v2/langpacks/(string:uuid)/, ??
GET /api/v2/latecustomization/?carrier=(string:carrier)&region=(string:region), ??
GET /api/v2/multi-search/, ??
GET /api/v2/payments/(string:origin)/in-app/, ??
GET /api/v2/payments/(string:origin)/in-app/(string:id)/, ??
GET /api/v2/payments/account/, ??
GET /api/v2/payments/account/(int:id)/, ??
GET /api/v2/payments/stub-in-app-products/, ??
GET /api/v2/payments/stub-in-app-products/(string:guid)/, ??
GET /api/v2/payments/upsell/(int:id)/, ??
GET /api/v2/reviewers/reviewing/, ??
GET /api/v2/reviewers/search/, ??
GET /api/v2/rocketfuel/collections/, ??
GET /api/v2/rocketfuel/collections/(int:id)/, ??
GET /api/v2/rocketfuel/collections/(int:id)/curation/, ??
GET /api/v2/rocketfuel/collections/(int:id)/image/, ??
GET /api/v2/services/carrier/, ??
GET /api/v2/services/carrier/<slug>/, ??
GET /api/v2/services/config/site/, ??
GET /api/v2/services/price-currency/, ??
GET /api/v2/services/price-currency/(int:id)/, ??
GET /api/v2/services/price-tier/, ??
GET /api/v2/services/price-tier/(int:id)/, ??
GET /api/v2/services/region/, ??
GET /api/v2/services/region/<slug>/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/abuse_reports/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/average_rating/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/installs/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/revenue/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/totals/, ??
GET /api/v2/stats/app/(int:id)|(string:slug)/visits/, ??
GET /api/v2/stats/global/abuse_reports/, ??
GET /api/v2/stats/global/apps_added_by_package/, ??
GET /api/v2/stats/global/apps_added_by_premium/, ??
GET /api/v2/stats/global/apps_available_by_package/, ??
GET /api/v2/stats/global/apps_available_by_premium/, ??
GET /api/v2/stats/global/apps_installed/, ??
GET /api/v2/stats/global/ratings/, ??
GET /api/v2/stats/global/revenue/, ??
GET /api/v2/stats/global/total_developers/, ??
GET /api/v2/stats/global/total_visits/, ??
GET /api/v2/stats/global/totals/, ??
GET /api/v2/transactions/(string:transaction_id)/, ??
GET /api/v2/tv/app/, ??
GET /api/v2/tv/multi-search/, ??
GET /api/v2/tv/website/, ??
GET /api/v2/webpay/prices/, ??
GET /api/v2/webpay/prices/(int:id)/, ??
GET /api/v2/webpay/product/icon/, ??
GET /api/v2/webpay/product/icon/(int:id)/, ??
GET /api/v2/webpay/status/(string:uuid)/, ??
GET /api/v2/websites/website/(int:id)/, ??
POST /api/v1/reviewers/canned-responses/, ??
POST /api/v1/reviewers/scores/, ??
POST /api/v2/abuse/app/, ??
POST /api/v2/abuse/user/, ??
POST /api/v2/abuse/website/, ??
POST /api/v2/account/dev-agreement/read/, ??
POST /api/v2/account/dev-agreement/show/, ??
POST /api/v2/account/feedback/, ??
POST /api/v2/account/installed/mine/remove_app/, ??
POST /api/v2/account/login/, ??
POST /api/v2/account/newsletter/, ??
POST /api/v2/apps/app/, ??
POST /api/v2/apps/app/(int:id)/payments/status/,
```

```
      ??                                        ??
POST /api/v2/apps/app/(int:id|string:slug)/refresh-manifest/,  POST /api/v2/services/price-currency/,
      ??                                        ??
POST /api/v2/apps/rating/,??                    POST /api/v2/services/price-tier/,??
POST /api/v2/apps/rating/(int:rating_id)/flag,  POST /api/v2/webpay/inapp/prepare/,??
      ??                                        POST /api/v2/webpay/prepare/,??
POST /api/v2/comm/note/(int:note_id)/attachment/ POST /api/v2/webpay/product/icon/,??
      ??                                        POST /api/v2/webpay/sig_check/,??
POST /api/v2/comm/thread/,??                    POST /api/v2/websites/website/submit/,
POST /api/v2/comm/thread/(int:thread_id)/note/,??  PUT /api/v1/reviewers/canned-responses/(int:id)/,
      ??                                        ??
POST /api/v2/extensions/extension/,??           PUT /api/v1/reviewers/scores/(int:id)/,
POST /api/v2/extensions/extension/(int:id)/,    PUT /api/v2/account/,
      ??                                        ??
POST /api/v2/extensions/extension/(int:id)|(string:slug)/,  PUT /api/v2/apps/app/(int:id)/,??
      ??                                        PUT /api/v2/apps/app/(int:id|string:app_slug)/icon/,
POST /api/v2/extensions/extension/(int:id)|(string:slug)/versions/,  ??
      ??                                        PUT /api/v2/apps/rating/(int:rating_id)/,
POST /api/v2/extensions/extension/(int:id)|(string:slug)/versions/(int:id)/publish/,  ??
      ??                                        PUT /api/v2/feed/apps/(int:id|string:slug)/image/,
POST /api/v2/extensions/extension/(int:id)|(string:slug)/versions/(int:id)/reject/,  ??
      ??                                        PUT /api/v2/feed/builder/,??
POST /api/v2/extensions/validation/,??          PUT /api/v2/feed/shelves/(int:id|string:slug)/image
POST /api/v2/feed/apps/,??                      ??
POST /api/v2/feed/brands/,??                    PUT /api/v2/langpacks/(string:uuid)/,
POST /api/v2/feed/collections/,??               ??
POST /api/v2/feed/items/,??                     PUT /api/v2/payments/(string:origin)/in-app/(string
POST /api/v2/feed/shelves/,??                   ??
POST /api/v2/fireplace/report_error,??          PUT /api/v2/payments/account/(int:id)/,
POST /api/v2/installs/record/,??                ??
POST /api/v2/langpacks/,??                      PUT /api/v2/rocketfuel/collections/(int:id|string:s
POST /api/v2/latecustomization/,??              ??
POST /api/v2/payments/(string:origin)/in-app/,  PUT /api/v2/services/price-currency/(int:id)/,
      ??                                        ??
POST /api/v2/payments/account/,??               PUT /api/v2/services/price-tier/(int:id)/,
POST /api/v2/payments/upsell/,??                ??
POST /api/v2/receipts/install/,??               DELETE /api/v1/reviewers/canned-responses/(int:id),
POST /api/v2/receipts/reissue/,??               ??
POST /api/v2/receipts/test/,??                  DELETE /api/v1/reviewers/scores/(int:id)/,
POST /api/v2/reviewers/app/(int:id)|(string:slug)/token,  ??
      ??                                        DELETE /api/v2/account/logout/,??
POST /api/v2/rocketfuel/collections/,           DELETE /api/v2/apps/app/(int:id)/,??
      ??                                        DELETE /api/v2/apps/rating/(int:rating_id)/,
POST /api/v2/rocketfuel/collections/(int:id)/duplicate/,  ??
      ??                                        DELETE /api/v2/extensions/extension/(int:id)|(strin
POST /api/v2/rocketfuel/collections/(int:id|string:slug)/add_app/,  ??
      ??                                        DELETE /api/v2/extensions/extension/(int:id)|(strin
POST /api/v2/rocketfuel/collections/(int:id|string:slug)/add_curator/,  ??
      ??                                        DELETE /api/v2/feed/apps/(int:id)/,??
POST /api/v2/rocketfuel/collections/(int:id|string:slug)/remove_app/,  DELETE /api/v2/feed/apps/(int:id|string:slug)/image
      ??                                        ??
POST /api/v2/rocketfuel/collections/(int:id|string:slug)/remove_brand/,  DELETE /api/v2/feed/brands/(int:id)/,
      ??                                        ??
POST /api/v2/rocketfuel/collections/(int:id|string:slug)/reorder/,  DELETE /api/v2/feed/collections/(int:id)/,
```

```
       ??
DELETE /api/v2/feed/items/(int:id)/, ??
DELETE /api/v2/feed/shelves/(int:id|string:slug)/,
       ??
DELETE /api/v2/feed/shelves/(int:id|string:slug)/image/,
       ??
DELETE /api/v2/langpacks/(string:uuid)/,
       ??
DELETE /api/v2/latecustomization/(int:id)/,
       ??
DELETE /api/v2/payments/account/(int:id)/,
       ??
DELETE /api/v2/payments/upsell/(int:id)/,
       ??
DELETE /api/v2/rocketfuel/collections/(int:id|string:slug)/,
       ??
DELETE /api/v2/rocketfuel/collections/(int:id|string:slug)/image/,
       ??
DELETE /api/v2/services/price-currency/(int:id)/,
       ??
DELETE /api/v2/services/price-tier/(int:id)/,
       ??
PATCH /api/v1/reviewers/canned-responses/(int:id)/,
       ??
PATCH /api/v1/reviewers/scores/(int:id)/,
       ??
PATCH /api/v2/account/settings/mine/,
       ??
PATCH /api/v2/apps/versions/(int:id)/,
       ??
PATCH /api/v2/extensions/extension/(int:id)|(string:slug)/,
       ??
PATCH /api/v2/feed/apps/(int:id)/, ??
PATCH /api/v2/feed/brands/(int:id)/, ??
PATCH /api/v2/feed/collections/(int:id)/,
       ??
PATCH /api/v2/feed/items/(int:id)/, ??
PATCH /api/v2/feed/shelves/(int:id|string:slug)/,
       ??
PATCH /api/v2/langpacks/(string:uuid)/,
       ??
PATCH /api/v2/payments/upsell/(int:id)/,
       ??
PATCH /api/v2/rocketfuel/collections/(int:id|string:slug)/,
       ??
PATCH /api/v2/webpay/failure/(int:transaction_id)/,
       ??
```