Appsembler Documentation

Release 1.0

Colin Su

August 27, 2013

CONTENTS

CHAPTER

ONE

INDEX

1.1 Setting up Flower monitoring on OpenShift

1. **Install Flower**

   ```
   echo "flower" >> requirements.txt
   ```

2. **Run Flower during your deploy**

   Put the following lines to your *.openshift/action_hooks/post_deploy* file and redeploy the app:

   ```
   echo "Starting flower"
   python "$OPENSHIFT_REPO_DIR"appsembler-launch/openshift_deploy/manage.py celery flower --setting
   ```

3. **Forward ports**

   On your local machine, run the following command to forward ports:

   ```
   rhc port-forward <app_name>
   ```

   Flower will be available at the adress 127.0.0.1:5445

1.2 Deploy procedure for Mediathread

The whole deploy process is organized around 3 repositories:

1. Mediathread Django app (https://github.com/appsembler/mediathread)
2. Mediathread Openshift wrapper (https://github.com/appsembler/mediathread-openshift-quickstart)
3. Openshift app repo

The deploy process will be explained on an example. For instance, you need to make a change to a file in the Mediathread app:

1. Make the change to the file in the Mediathread Django app repo
2. Commit changes and push to Github
3. Go to your `mediathread-openshift-quickstart` repo and do:

   ```
   cd mediathread
   git pull
   cd ..
   ```

```
git commit -am "Updated submodule"
git push
```

4. Next, go to your Openshift app repo (the one where you do a git push to production) and do

```
git pull -X theirs upstream master
git push
```

1.3 Adding New Relic monitoring to OpenShift

1. **Install New Relic package**

   You'll want to add the New Relic package to you requirements.txt file:

```
echo "newrelic" >> requirements.txt
```

2. **Initialize the config file**

   Once the New Relic package is installed, either during the deploy or manually using pip, SSH into your OpenShift gear, activate the virtual environment and initialize the config file:

```
rhc ssh <app_name>
source python/virtenv/bin/activate
cd app-root/data
newrelic-admin generate-config <your-license-key> newrelic.ini
```

3. **Change the app name in the config file**

   The app name defines how the app is going to show in the New Relic site:

```
# The appplication name. Set this to be the name of your
# application as you would like it to show up in New Relic UI.
# The UI will then auto-map instances of your application into a
# entry on your home dashboard page.
app_name = YourAppName
```

4. **Initialize New Relic library in the WSGI script**

   Add the following code to your wsgi script in `wsgi/application` in your repo above the part where WSGI application is created:

```
try:
    import newrelic.agent

    config_file = os.path.join(os.environ['OPENSHIFT_DATA_DIR'], 'newrelic.ini')
    if os.path.exists(config_file):
        newrelic.agent.initialize(config_file, 'production')
except IOError:
    pass
```

5. **(Optional) add New Relic support for Celery**

   Add the following code to your `.openshift/action_hooks/post_deploy` script:

```
NEW_RELIC_CONFIG_FILE=${OPENSHIFT_DATA_DIR}newrelic.ini newrelic-admin run-python "$OPENSHIFT_RE
```

1.4 Setting up a periodic PostgreSQL DB dump

1. **Add cron cartridge**

   In order to run periodic tasks, be sure to add the cron cartridge to your app:

   ```
   rhc cartridge add cron-1.4 -a <app_name>
   ```

   Note: you currently can't add the cron cartridge to scaled apps

2. **Download and copy s3cmd to your OpenShift data folder**

   Download s3cmd from http://s3tools.org/s3cmd, copy it to OpenShift and unpack it:

   ```
   scp s3cmd-1.5.0-alpha3.tar.gz openshiftusername@address:app-root/data
   rhc ssh <app_name>
   cd app-root/data
   tar xzvf s3cmd-1.5.0-alpha3.tar.gz
   rm s3cmd-1.5.0-alpha3.tar.gz
   mv s3cmd-1.5.0-alpha3 s3cmd
   cd s3cmd
   ```

   Now run the following command and put in your Amazon S3 secret keys:

   ```
   s3cmd --configure -c .s3cfg
   ```

3. **Set up cron script**

   Add the following code to your `9.openshift/cron/daily` folder (or hourly if you want more frequent backups):

   ```
   #!/bin/bash
   # Backs up the OpenShift PostgreSQL database for this application
   # by Skye Book <skye.book@gmail.com>

   NOW="$(date +"%Y-%m-%d")"
   FILENAME="$OPENSHIFT_DATA_DIR/db_backups/$OPENSHIFT_APP_NAME.$NOW.backup.sql.gz"
   /opt/rh/postgresql92/root/usr/bin/pg_dump -h $OPENSHIFT_POSTGRESQL_DB_HOST -p $OPENSHIFT_POSTGRE
   cd $OPENSHIFT_DATA_DIR/s3cmd
   s3cmd -c .s3cfg put $FILENAME s3://mediathreadbackups
   ```

1.5 How to add Redis to your OpenShift app

1. **Add the Redis cartridge to your app**

   ```
   rhc add-cartridge http://cartreflect-claytondev.rhcloud.com/reflect\?github\=smarterclayton/open
   ```

2. **Get Redis env variables**

   When you SSH into your gear, you can see Redis env variables such as host, port, username and password that you can use in your applications:

   ```
   rhc ssh <app_name>
   env | grep -i redis | sort
   ```

1.6 Handling secret settings in Django

When deploying apps somewhere, often there's a need to remove secret data from the settings in the repo you keep on GitHub or somewhere else. This document will show you how to handle secret settings when deploying to OpenShift.

1. **Create a secret_keys file**

   You'll need to create a secret_keys file in which you'll set and export the secret settings as shell enviromental variables. The file should look similar to this one:

   ```
   #!/bin/bash

   # Django settings
   export SECRET_KEY='yyyyyyy'

   # Email settings
   export MANDRILL_API_KEY='xxxxxxxx'

   # Sentry settings
   export SENTRY_DSN='https://xxxx:yyyy@domain.com/2'

   # Pusher settings
   export PUSHER_APP_ID='xxxxx'
   export PUSHER_APP_KEY='yyyyy'
   export PUSHER_APP_SECRET='zzzzz'

   # OpenShift settings
   export OPENSHIFT_USER='user@domain.com'
   export OPENSHIFT_PASSWORD='secretpassword'

   # New Relic settings
   export NEW_RELIC_CONFIG_FILE='newrelic.ini'
   export NEW_RELIC_ENVIRONMENT='production'
   ```

   Put that file in the root of your OpenShift app repository (the one you see when you do ``rhc app show``).

2. **Copy the secret_keys file to the server**

   Run the following commands to add the secret_keys to the OpenShift app repo.

   ```
   git add secret_keys
   git commit -m "Added secret_keys file"
   git push
   ```

3. **Source secret_keys during deploy**

   Be sure to source the secret_keys file in every action hook on OpenShift where you have to run something related to Django. To be safe, it would be best to simply source it in all action hooks like this:

   ```
   source ${OPENSHIFT_REPO_DIR}secret_keys
   ```

1.7 How to set up Sentry on OpenShift

1. **Start a Sentry instance**

```
rhc app create --scaling <app_name> python-2.6 mongodb-2.2 postgresql-8.4
cd <app_name>
git remote add upstream -m master git://github.com/zemanel/openshift-sentry-quickstart.git
git pull -s recursive -X theirs upstream master
git push origin
```

2. **Get the API key**

   Log in into your deployed Sentry instance, create a project and find the Client configuration->Python option in the settings and copy the `RAVEN_CONFIG` setting. It should look something like this:

```
# Set your DSN value
RAVEN_CONFIG = {
    'dsn': 'https://sdasfdgfdjng342rewfsd@sentryweb-namespace.rhcloud.com/2',
}
```

3. **Install raven library**

   Add the raven library to your requirements.txt:

```
echo "raven" >> requirements.txt
```

4. **Set the DSN value in settings.py**

   Paste the `RAVEN_CONFIG` setting you got from the Sentry site to your settings.py file

5. **Add raven to installed apps**

```
# Add raven to the list of installed apps
INSTALLED_APPS = INSTALLED_APPS + (
    # ...
    'raven.contrib.django.raven_compat',
)
```

6. **(Optional) Integrate django logging with Sentry**

   Paste this in your settings.py and tweak it to your requirements:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'root': {
        'level': 'WARNING',
        'handlers': ['sentry'],
    },
    'formatters': {
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s %(process)d %(thread)d %(message)s'
        },
    },
    'handlers': {
        'sentry': {
            'level': 'ERROR',
            'class': 'raven.contrib.django.raven_compat.handlers.SentryHandler',
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose'
        }
    },
```

```
        'loggers': {
            'django.db.backends': {
                'level': 'ERROR',
                'handlers': ['console'],
                'propagate': False,
            },
            'raven': {
                'level': 'DEBUG',
                'handlers': ['console'],
                'propagate': False,
            },
            'sentry.errors': {
                'level': 'DEBUG',
                'handlers': ['console'],
                'propagate': False,
            },
        },
    }
```

1.8 Saving and restoring applications on OpenShift

1. **Taking a snapshot**

   First, you need to take an archived snapshot of the current running application:

   ```
   rhc snapshot save -a <app_name>
   ```

2. **Restoring a snapshot**

   You have to create a new application using the same cartridges as the saved app, either on a different account or on the same after you delete your app using `rhc app delete <app_name>`. Do so by using the following:

   ```
   rhc snapshot restore -a <app_name> -f <path/to/your/archive.tar.gz>
   ```

Note: the app name must be the same on both the old and new apps for the process to work

Note #2: you can run into problems restoring except with the most simple apps so be sure to have some other way of backing up the data on the app

1.9 SSL for Django apps on OpenShift

1. **Enable secure cookies**

   Add the following lines to your settings.py file:

   ```
   SESSION_COOKIE_SECURE = True
   CSRF_COOKIE_SECURE = True
   ```

2. **Enable HTTPS on the server**

   Add the following lines to your `wsgi/.htaccess` file:

   ```
   RewriteEngine on

   RewriteCond %{HTTP:X-Forwarded-Proto} !https
   RewriteRule .* https://%{HTTP_HOST}%{REQUEST_URI} [R,L]
   ```

3. **Enable HTTPS in the WSGI script**

   Add the following lines to your `wsgi/application` file:

   ```
   # make django aware that SSL is turned on
   os.environ['HTTPS'] = "on"
   ```

1.10 Serving Django static assets on from OpenShift

1. **Create a collected_static folder during build**

   Add the following to your `.openshift/action_hooks/build` script:

   ```
   if [ ! -d $OPENSHIFT_REPO_DIR/wsgi/static/collected_static ]; then
       mkdir $OPENSHIFT_REPO_DIR/wsgi/static/collected_static
   fi
   ```

2. **Correctly set up Django settings**

   Make sure your `STATIC_ROOT`, `STATIC_URL` and `STATICFILES_DIR` settings are correctly set. They should look something like this:

   ```
   STATIC_ROOT = os.path.join(get_env_variable('OPENSHIFT_REPO_DIR'), 'wsgi', 'static', 'collected_
   STATIC_URL = '/static/'
   STATICFILES_DIRS = (
       os.path.join(PROJECT_ROOT, 'static'),
   )
   ```

3. **Set up apache mod_rewrite**

   Add the following lines to your `wsgi/.htaccess` file:

   ```
   RewriteEngine on
   RewriteRule ^application/site_media/(.+)$ /static/collected_static/$1 [L]
   ```

4. **Run collectstatic in deploy script**

   Add the following somewhere at the end of your `.openshift/action_hooks/deploy` script:

   ```
   export DJANGO_SETTINGS_MODULE=your.settings.module
   echo "Executing 'django-admin.py collectstatic'"
   django-admin.py collectstatic --noinput --settings=$DJANGO_SETTINGS_MODULE
   ```

Note that the `get_env_variable` mentioned above is defined in the settings.py as:

```
from django.core.exceptions import ImproperlyConfigured


def get_env_variable(var_name):
    """ Get the environment variable or return exception """
    try:
        return os.environ[var_name]
    except KeyError:
        error_msg = "Set the %s environment variable" % var_name
        raise ImproperlyConfigured(error_msg)
```

1.11 Git-related Conventions for appsembler-mediathread

1.11.1 Branch Naming

The branch should be named as `<trello-card-#>-<branch-name>`.

e.g. 15-new-form, 57-landing-page

1.11.2 Git flow

refer to A successful Git branching model.

Installation

Using your Package Manager to install package which is called *git-flow*.

Linux `sudo apt-get install git-flow`

Mac OS X `brew install git-flow`

Configuration

- **branches**
  - production branch: appsembler_master
  - "next release" branch: appsembler_develop
- **prefix**
  - feature: feature/ (Default)
  - release: release/ (Default)
  - hotfix: hotfix/ (Default)
  - support: support/ (Default)
  - version tag: None (Default)

Commands

initialization `git flow init` and answer the prompt by the order of following configuration

start a flow `git flow <prefix> start <branch name>` this will create a new branch and checkout it

finish a flow `git flow <prefix> finish <branch name>` this will commit your change in that branch and merge it back to "the next release" branch

Setup

For the first time you need to run `git flow init` to initialize git-flow for specified repo, or you can direct modify git config file.

```
[gitflow "branch"]
    master = appsembler_master
    develop = appsembler_develop
[gitflow "prefix"]
    feature = feature/
    release = release/
    hotfix = hotfix/
    support = support/
    versiontag =
```

You can also copy and paste above text into your `path/to/mediathread/.git/config` (replace original one or append in the bottom)

CHAPTER

TWO

USEFUL LINKS FOR RESTRUCTUREDTEXT

- Syntax
- Quick Reference
- Quick Start
- Sphinx Documentation

CHAPTER

THREE

HOW TO UPDATE THIS WIKI

1. clone from this Documentation Repository
2. `cd docs`
3. `vim index.rst`
4. `make html`
5. `git commit -am 'some msg'`
6. `git push origin master`

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*