Praekelt Ways of Working Documentation

Release 1

Praekelt Foundation devs and individual contributors

January 12, 2016

Hey, hopefully you're here because you're about to embark on a project with us.

Here are some bits & pieces of documentation to help you get up to speed with our ways of working, the tools we use, processes follow and the things we would be expecting from you when doing projects.

If anything is unclear or plain wrong please do notify us or better yet, submit a pull request to the repository!

Enjoy!

CHAPTER 1

Our Platforms

Praekelt uses two main platforms for the bulk of engineering work:

1. Django

   We use Django for websites, mobi sites, responsive sites, mobi HTML5 apps.

2. Vumi

   We use our Vumi platform for SMS, USSD and other messaging protocols.

Use of any other platform must be approved by our engineering management, and may need to be hosted separately from our usual environments, so please get this sorted out prior to commencing development on a project.

1.1 Django

You should use the latest stable release of Django unless otherwise specified.

We deploy Django in the following stack:

- Ubuntu Server (current LTS release)
- haproxy for load balancing where appropriate
- nginx
- gunicorn
- supervisord
- postgresql

For development, you can simplify this, and for QA we won't bother about haproxy but the rest of the stack will be required for QA so we recommend you keep your dev environment as close to this as you can.

Notes:

- We manage hostnames in nginx, because there may be multiple QA and live hostnames so don't use Django's ALLOWED_HOSTS.

1.2 Vumi

Vumi is a scalable messaging engine which we use for SMS, USSD and other messaging protocols. Vumi Go is a hosted version of Vumi. Where Vumi gives you the tools to build large scale messaging applications, Vumi Go provides you with a working environment that is already integrated into numerous countries.

Apps can be written for Vumi Go, to power messaging campaigns or information systems. These apps can be written in Javascript to run in a sandboxed environment (which is our preferred option) or in Python.

See the Vumi Go documentation regarding writing apps, and there is documentation for a state machine for USSD as well.

CHAPTER 2

Tools we use

The following are tools we use on a regular basis and which we expect you to either use or at least be very familiar with.

2.1 IRC

IRC is our team's communication tool of choice. Join us in `#prk-dev` for general developer support, or `#vumi` or `#jmbo` for development of those platforms, on irc://irc.freenode.net/.

Various tools report into these channels and provide insight into what is going on.

2.2 Git

We use Git. If you work with us, you will use Git for revision control, and GitHub. There is no exception.

Provide us with your GitHub username and we will provide you with a repository to work on. All repositories are to be hosted under the Praekelt Organization on GitHub.

Please read What's in a Good Commit? for a good introduction to effective use of version control commits.

2.3 Git Flow

We use the Git Flow branching model as part of our development. It's a convenient way to manage your branches. You are not required to use Git Flow but you are required to follow naming conventions it sets with regard to branch-names and prefixes.

Have a read through the blog post describing the general idea and follow the installation instructions in the repository to install it for your development platform of choice.

Unless you've explicitly been told otherwise, we require our team to review your code before landing it in the develop branch. Please provide pull requests for our review, the command line tool Hub (see below) is a convenient way of turning GitHub issues into pull-requests.

The pull-request requirement still remains when using Jira. You can still use Hub - however your Jira ticket's status will not automatically change when the feature branch lands, so you will need to update this yourself.

Please read Useful Github Patterns to see ways of working with branches and pull requests that we like.

2.4 Hub

For projects with issues tracked in Github issues, We use Hub to interface with GitHub's API. It allows one to turn issues on GitHub into pull-requests. If that is done then once the pull-request is merged into the main branch the issue is automatically closed.

2.5 Issues & Tickets

For project work we use Jira. Only our core open-source platforms maintain their issues in the GitHub repository.

You will be given an account to use which will have access to the relevant projects.

For development, if there is no ticket it does not exist. Make sure the work you are doing has a ticket and is being tracked. Stop working and protest immediately if people are treating your mailbox as their ticketing system. We've tried that, it does not work.

If a Jira project has a workflow, you need to update your tickets appropriately: New -> Open -> Fixed in dev (when pushed to github) -> Deployed to QA

Our QA team will move the ticket to QA Passed, and our DevOps team will be responsible for the production deployment before the ticket is resolved.

If a ticket is QA Failed then it's back into your section of the workflow.

A ticket should represent a solid piece of work you intend to do. Make an effort to keep the work you are trying to do in one ticket to no more than 16 hours.

Any estimate you make for actual work done beyond 16 hours is assumed to be

1. largely thumb-suck.
2. going to be very hard to review.

Make an effort to keep it to 16 hours or break it up unto multiple tickets each representing 16 hours of work.

2.6 Sentry

We have a dedicated Sentry instance for our projects. You are expected to configure your application to make use of this for error reporting.

You will be given access to your Sentry project and access tokens to will be made available for you to configure your application's client with.

2.7 Puppet

We try and automate as much as possible, this includes our hosting environment. You will need to give us your SSH key so we can provision a machine for your project. Generally you will be given access to a machine that is to be used for QA. Since our DevOps team do the production deployments, and you will get access to production error reports via Sentry, you won't get access to production without a valid need for troubleshooting, and then it will be without sudo access.

These machines are provisioned using *Puppet*. You will not get access to our puppet repository. If you need specific software installed on your machine that it was not provisioned with then please ask for it to be added. Do not install it yourself without notifying us. This would break our assumption that every machine can be provisioned from scratch with puppet.

If the machine you've been working on needs to be rebuilt and you've made changes that are not in puppet then it'll be provisioned without those changes.

2.8 Databases / data stores

We use the following services to store our data. Not all projects will use all of them but generally a number of these will be involved.

1. PostgreSQL
2. Riak
3. Memcached
4. Redis
5. Neo4J

These will be made available to you on a per project basis. Puppet ensures that each of these are backed up.

2.9 Django Applications

For Django applications, some applications are mandatory:

1. Sentry for application reporting.
2. South for managing database schema changes.
3. Nose for running tests.
4. Haystack for search.
5. Memcached for caching.

2.10 Translations

We use Gettext or translations in shell scripts, applications and web pages. Read more about Gettext along with some examples on Wikipedia: http://en.wikipedia.org/wiki/Gettext

In Django, Gettext is used by default for translations, utilizing ugettext_lazy for models.py and ugettext in other places. We like {% trans %} and {% blocktrans %} tags and enforce these for our open source products.

2.11 Graphite

We use Graphite for the majority of our metric publishing for dashboards. If appropriate, you will be given details for the Graphite server and how metrics are to be published to it.

CHAPTER 3

Our project process

The lifecycle of our projects is typically as follows:

1. We produce a Scope of Work for a project, which might not have all the technical details, but should be comprehensive enough to list all the features so that you can quote on the project's development.
2. We work on a fixed cost basis for a fixed scope. If the scope changes, we ask you for a costing for the delta or new work.
3. The authorisation to proceed with work consists of a Purchase Order, without which you cannot invoice us - so never start work without the PO.
4. Development commences - see below. **If you don't have a github repo by this point, please bug us until we provide it - please do not use your own repo.**
5. We provide you with one QA server, with the same OS setup that we'll use in production - for all the projects you do for us, unless a project has special needs which justify its own QA server. **Please bug us for a QA URL for this project to be pointed to your QA server.** It must be on our domain for client UAT. Please note that QA may need sample or real data to be populated. Often, the QA data gets migrated to the production site when finally deploying that, so please ensure that dummy data can be cleaned up, and use **CMS credentials** on QA that are *suitable for production*.
6. You are responsible for deploying your code to this QA server, so that you can support the fixing of bugs found during our QA testing. You should **always** deploy to QA from the github repo, to avoid any side effects of uncommitted code.
7. We'll deploy to production so that we can support it - see below.

CHAPTER 4

Our development process

The process involved in how we work is fairly straight forward and we expect you to follow this as well.

1. We use Git Flow's convention with regard to branch names.
2. All work requires a ticket with a unique number or name.
3. Work happens in a feature branch. Feature branches names are composed of the ticket / issue number along with a one-line description of the issue.
4. Write tests for the new features you are developing.
5. Your schema changes are expected to be handled by a schema migration script.
6. When work in a feature branch is ready for review then we create a pull-request.
7. All collaborators on the GitHub repository are notified of the pull-request and will start the process of reviewing the changes.
8. Any issues, concerns or changes raised or recommended are expected to be attended to. Once done please notify the reviewers of the changes and ask for the changes to be re-reviewed.
9. Once all the changes are approved and one or more of the collaborators has left a *:+1:* in the pull-request's comments it can be merged into the main branch and is ready for a deploy.

For your code to be ready for review we have the following expectations:

1. It is to be pep8 compliant and pyflakes is not raising any issues.
2. It is to have tests.
3. The tests have to pass.
4. There are no commented lines of code.
5. There is adequate amount of documentation.

4.1 Example flow

```
$ virtualenv ve
$ source ve/bin/activate
(ve)$ git flow feature start issue-1-update-documentation
(ve)$ git flow feature publish issue-1-update-documentation
..// hack hack hack // ..
(ve)$ nosetests
.............
```

```
----------------------------------------------------------------------
Ran 13 tests in 0.194s
OK
(ve)$ git push
(ve)$ hub pull-request -b develop -i 1
https://github.com/praekelt/some-repository/pulls/1
..// review, update, re-eview, update, re-review ... +1 // ..
(ve)$ git flow feature finish issue-1-update-documentation
..// changes merged to develop by git flow // ..
(ve)$ git push
```

CHAPTER 5

Production Deployments

Our DevOps team are responsible for all production deployments. This enables us to support the live sites and systems after hours, and ensure that infrastructural requirements like backups and monitoring are standardised.

Please note that production deployments need to be booked with the DevOps team by the appropriate Praekelt project manager, and that we deploy on Mondays through Thursdays.

CHAPTER 6

Things you need to know when starting a project

Sometimes there's a rush to get a project started. To spare yourself future trouble here's a checklist of things you need to have before starting any work.

1. You need to have been issued a purchase order.
2. You need to have been given a Scope of Work describing the thing you are to be building.
3. You need to have agreed to the timelines, estimates and deliverables described in the Scope of Work. If there are any ambiguities in any of the wording they need to be resolved before you start.
4. You need to have a clear picture of which stats need to be collected for the project and how those are to be stored to enable the people wanting access to those stats do the type of analysis they need to do. This differs per project so make sure you take the time to do this properly.