Bringing closer Debian and Rails: Bridging apparently incompatible cultures

Gunnar Wolf — gwolf@debian.org

DebConf 8
August 2008, Mar del Plata, Argentina
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
Gems

- Ruby's own package management system
- Integrates the semantics of module inclusion and package downloading in one single, comfortable(?) step: `require_gem`
- One of its main points: Managing different installed versions of the same gem
Gems

- Way to easily installed unsupported (by Debian) third-party libs
- Source-intrusive we need to patch a gemified lib so that it works without rubygems
- Ignores non-rubygems packages when trying to satisfy deps
- Some ruby libs are only distributed as gems
- De-gemification of complex ruby apps is very complex
- Encourages strict deps on a specific versions: you end up with many versions installed
- Doesn’t encourage stable APIs at all (people just install the latest upstream, then cry)
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
Plugins

- Very similar to Gems-based modules, but more tightly integrated into Rails' workflow, modifying or adding to Rails' normal behaviour.
- Usually carry an initialization file (init.rb), and require only being dropped in their place (RAILS_ROOT/vendor/plugins/); no need for explicitly including them.
Some plugins have started migrating to become Gems

Some plugins... Completely despise the concept of *versions* — The only way it is distributed is as a git clone URL. And often, they don’t even tag in their git trees!
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
Rails’ API is still evolving at a fast speed — often introducing backwards incompatibility

It is common for application developers to *vendorize* Rails — To copy a whole version of Rails into RAILS_ROOT/vendor/rails, in order to ensure his system will work regardless if the systemwide version changes
In fact, when creating the skeleton for a new Rails app, the Debian-packaged Rails creates symlinks from `RAILS_ROOT/vendor/rails` to `/usr/share/rails` (for the currently installed Rails). Of course, this leads to breakage on updates :-/

Supporting Rails installed inside application directories is far from The Debian Way. And, of course, supporting multiple old versions of Rails in the system... is also bad.
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
Deployment

- One of the most unstable areas in Rails
- Favored deployment has went through Apache+Pen+FCGI, Apache+SCGI, any_web_server+Mongrel, Apache+mod_rails (Passenger), and there is a newcomer somewhere around there... bah.
- Many people have complained about Rails’ unfriendliness for deployment
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
Rails applications packaging

- No Rails applications are currently packaged for Debian... But there will be. At least, there should be.
- And all of the before mentioned points have to be integrated.
- We must aim for the best ease of deployment possible.
Temas

1. Gems
2. Plugins
3. Framework vendorization
4. Deployment
5. Rails applications packaging
6. General developer community culture
General developer community culture

- There is a big cultural difference between us Debianers and most of the Rails community.
- Rails developers tend to look for the newest features for their (production!) systems, and tend to value less long-term stability/maintainability. Interaction with upstream regarding older versions can be quite problematic.
- Of course, I expect the previous statement to raise rushes of indignation among Rails (and other Extreme Programming converts), but... In Debian we commit to supporting our stable versions!