

Talk Proposal - Scottish Ruby Conference 2010

has_many :client_databases

If you have an application you offer as a service to others (think Google Docs and such), you're going to run into the problem of needing to keep data separate between your customers. You'll want to do this as efficiently as you can, with as little code as possible.

I'll discuss methods you can adopt to solve this problem and prove with real-world examples that it's not quite as difficult a problem as it seems to be.

Background

We had an application we were developing which we knew would be used by many customers. Each of these customers needed to have their own database and because of the myriad of security regulations we had to conform to with the application, we needed to ensure data integrity for each customer and restrict access as far as possible.

We were also mindful of resources and planned from the start to use as few instances of our application as possible.

What needed to be done:

- Detect (somehow) the customer making a request
- Connect that customer to their own database
 - Ensure it's not possible to connect to another's databases
- Do so with as little overhead as possible

We came up with a fairly elegant solution for a Rails 1.2 application which will be explained in full detail, and have been successfully using this since the application went live in 2007.

I'll also present a proof-of-concept method which uses Rack::Middleware and a more generic way of communicating database information to your application, potentially allowing the same methodology to be used on any Rack compatible application.

Target Audience

I cover generally how such a problem should be solved and give specific implementation direction in Rails and also in Rack making this talk suitable for anyone with an interest who is working with a Ruby web framework. The theory content of the talk to just as easily be applied to any other framework in any language.

Talk Outline

I propose the talk be structured as such:

- Problem Overview
 - Many customers, each with a database
 - No code differences between application instances
 - Data integrity important, data security a must
- Possible Solutions:
 - Multiple application instances
 - Wrong answer:
 - Costly on resources
 - Costly to maintain deployment
 - Only really works for a tiny number of clients
 - Current working Solution (Rails 1.2.3 to 2.3.4)
 - `before_fiter` + Hostname Sniffing
 - Extensions to ActiveRecord
 - Client-Specific DB Config Files
 - DB Connection Enforcement
 - Can't switch between requests
 - Second Approach - Rack Compatible
 - `Rack::Middleware` gets connection information
 - Passes to application
 - Smaller `before_fiter` on Application controller
 - Merits / Drawbacks of both approaches
- Questions & Answers

Outcome

Afterwards, attendees will hopefully be able to apply the topic covered in the presentation to their own projects and be able to reap the rewards of being able to provide the same application to a great many clients without the need to spawn multiple instances.