Multi-Tenancy on Rails

An Ann Arbor Ruby Brigade Feature Length Presentation

you can go home when the number in the bottom right is 56ish

i know you cant see the numbers now but wait a few slides

John DeSilva

April 25

Revela, Inc.

What is Multi-Tennancy?

What is Multi-Tennancy?

> *The term “software multitenancy” refers to a software architecture in which a single instance of software runs on a server and serves multiple tenants. A tenant is a group of users who share a common access with specific privileges to the software instance.*

- Wikipedia

What is Multi-Tennancy?

In short, we want to scale our customer base by sliding the heroku slider to the right instead of creating new heroku apps with a new db per customer.

if we need to add databases to scale to our data size or for customer reasons,

thats fine,

but only one app cluster

(or a multi zone federated cluster)

Types of Tennancy

‘Multi User’ Applications

Some applications are used by many individual users. Often B2C.

- Twitter
- Gmail
- Spotify

‘Tenantless’ Applications

Many applications do not have tenancy at all. Lots of low touch news, marketing, and cooking sites here.

- Allrecipes
- 4chan
- Darksky
- Buzzfeed
- Wikipedia

‘Organization’ Applications

Often, enterprise applications have a ‘shared dataset’ amongst users of different organizations. B2B SaaS products tend to fall into this bucket.

- Basecamp
- Slack
- Salesforce

‘Gray Area’ Applications

Some applications do not clearly fall into one of the previous three categories.

- Github
- Shopify

Demo 1 / 3

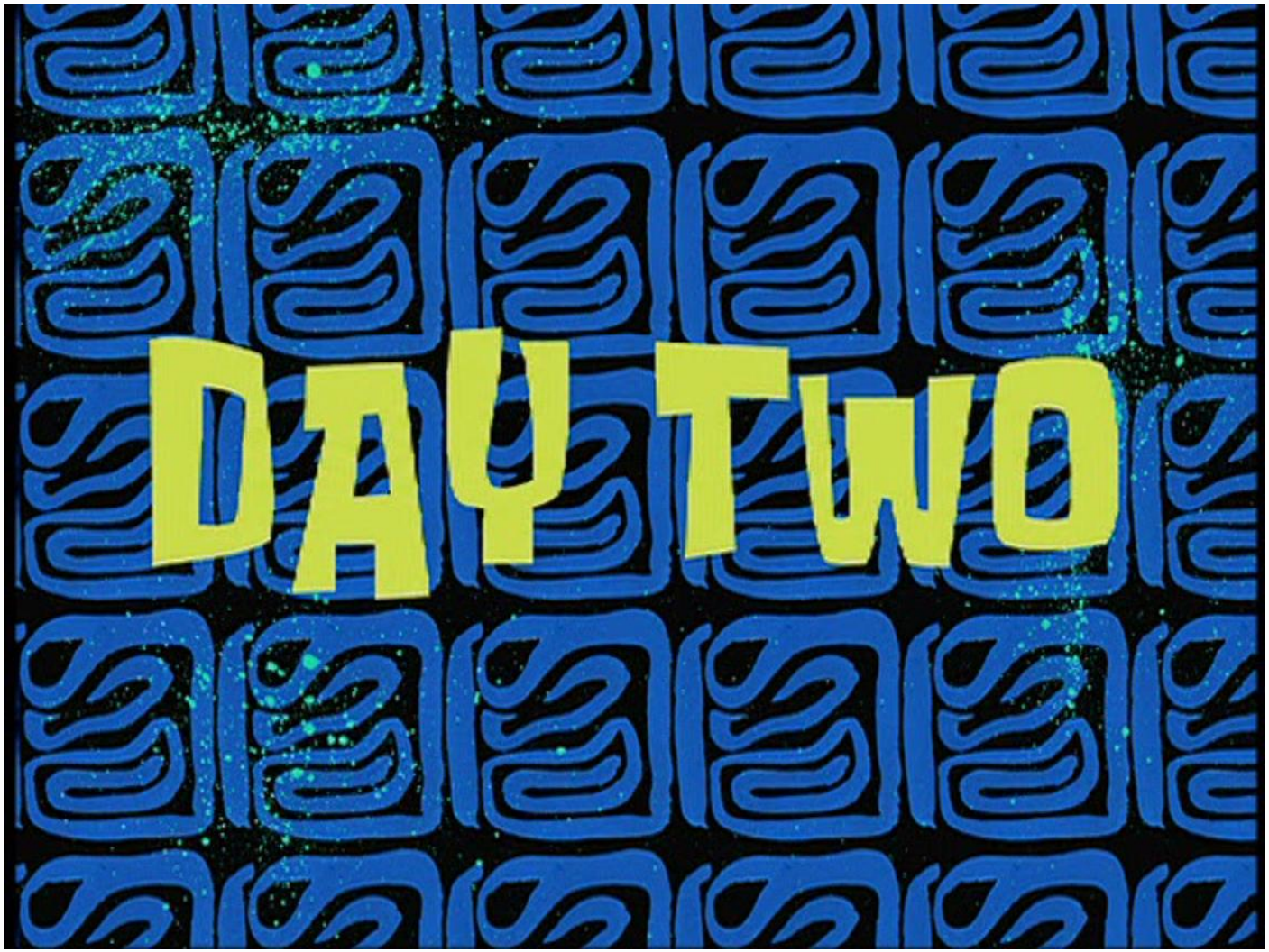
```
$ git checkout no_tennancy
```

(1/2): Query Scoping Tennancy

Straightforward

```
class RecipesController
  def show
    @recipe = Recipe.find(params[:id])
  end
end
```

Straightforward



Pretty Straightforward

```
class RecipesController
  def show
    @recipe = current_user.recipes
                          .find(params[:id])
  end
end
```

Kinda Straightforward

```
class RecipesController
  def show
    @recipe = current_user
                .company
                .recipes
                .find(params[:id])
  end
end
```

Somewhat Straightforward

```
class RecipesController
  before_action :set_recipe # Rails-it-up-a-notch!

  def show; end

  def set_recipe
    @recipe = recipes.find(params[:id])
  end

  delegate :company, to: :current_user
  delegate :recipes, to: :company
end
```

Demo 2 / 3

```
$ git checkout scoping
```


MUCH
LATER

has_mariana_trench

```
Companies
\-> Buildings
    \-> Units
        \-> Leases
            \-> Lease Memberships
                \-> Tenants
                    \-> Payments
```

‘Display a list of Payments’

```
select * from payments
join tenants on tenants.id = payments.tenant_id
join lms on lms.tenant_idid = tenants.id
join leases on leases.id = lms.lease_id
join units on units.id = leases.unit_id
join buildings on buildings.id = units.building_id
join companies on companies.id = buildings.company_
where companies.id = $1;

$1 => current_user.company_id;
```


y tho

Why not denormalize `customer_id`?

- The database schema should (probably?) represent your application's data, not your application's data performance needs or access paterns.
  - *This is CRUD, not Twitter*
- Now you have to have a `Customer` at arms reach at the point of `Payment` creation.
  - *You might, but hey - think of the children. Erm, I mean, background jobs and one off scripts*
- Hey, what else could use a `customer_id`?
  - *It will start to get sprinkled on everywhere, I promise*

Why not put `customer_id?` on every table?

I’m pretty sure Citus Data’s commercial pg_shard evolution actually does this or something similar


2
SCHEMA BASED
MULTI
TENNANCY

Lingo

Schema

Lingo - Database Schema

Database Schema

> *The term ‘schema’ refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database.*

- Wikipedia

Lingo - Database Schema

```ruby
# db/schema.rb

ActiveRecord::Schema.define(
  version: 20170424000157
) do

  create_table "companies", force: :cascade do |t|
    t.string   "name"

    # ...
```

Lingo - Database Schema

```
soda $ psql soda_development

[local] john@soda_development=# \dt

                List of relations

 Schema |           Name           | Type  | Owner
--------+--------------------------+-------+-------
 public | ar_internal_metadata     | table | john
 public | companies                | table | john
 public | recipes                  | table | john
 public | schema_migrations        | table | john
(4 rows)
```

Lingo - Database Schema

```
[local] john@soda_development=# \d recipes

            Table "public.recipes"

    Column     |                 Type                 | Modifi
---------------+--------------------------------------+------
 id            | integer                              | not n
 title         | character varying                    |
 instructions  | character varying                    |
 created_at    | timestamp without time zone          | not n
 updated_at    | timestamp without time zone          | not n
 company_id    | integer                              |
```

PostgreSQL Schema

> *A database contains one or more named schemas, which in turn contain tables. Schemas also contain other kinds of named objects, including data types, functions, and operators. The same object name can be used in different schemas without conflict; for example, both schema1 and myschema can contain tables named mytable. Unlike databases, schemas are not rigidly separated: a user can access objects in any of the schemas in the database they are connected to, if they have privileges to do so.*

- https://www.postgresql.org/docs/9.6/static/ddl-schemas.html

Lingo - PostgreSQL Schema

There are several reasons why one might want to use schemas:

- To allow many users to use one database without interfering with each other.
- To organize database objects into logical groups to make them more manageable.
- Third-party applications can be put into separate schemas so they do not collide with the names of other objects.

Lingo - PostgreSQL Schema

```
[local] john@soda_development=# \dn

  List of schemas

  Name   |  Owner
---------+----------
 public | postgres

(1 row )
```

Lingo - PostgreSQL Schema

```
[local] john@soda_development=# show search_path;

   search_path
-----------------
 "$user", public
(1 row)

Time: 0.114 ms
```

Mr. Krabs I Have An Idea

Lingo - PostgreSQL Schema

```
[local] john@soda_development=# \dn

  List of schemas

  Name   |  Owner
---------+----------
 coke    | john
 pepsi   | john
 public  | postgres
(3 rows)
```

Lingo - PostgreSQL Schema

```
[local] john@soda_development=# \dt *.*

              List of relations

 Schema    | Name       | Type   | Owner
-----------+------------+--------+--------
 coke      | recipes    | table | john
 pepsi     | recipes    | table | john
 public    | recipes    | table | john

    ...

(74 rows)
```

Public Recipes

```
[local] john@soda_development

# select title from recipes;

 title
-------
(0 rows)

Time: 0.405 ms
```

Coke Recipes

```
[local] john@soda_development

# select title from coke.recipes;

     title
---------------
 Coca-Cola
 Sprite
 Mellow Yellow
(3 rows)

Time: 0.355 ms
```

Pepsi Recipes

```
[local] john@soda_development

# select title from pepsi.recipes;

     title
---------------
 Pepsi
 Mountain Dew
 Mug Root Beer
(3 rows)

Time: 0.292 ms
```

Pepsi Recipes, Magically

```
[local] john@soda_development
# set search path to pepsi;
# select title from recipes;

     title
---------------
 Pepsi
 Mountain Dew
 Mug Root Beer
(3 rows)

Time: 0.292 ms
```

The `apartment` Gem

apartment

`influitive/apartment`

Database multi-tenancy for Rack (and Rails) applications

apartment's API

```
Apartment::Tenant.create('pepsi')
# => create schema pepsi;

Apartment::Tenant.switch!('pepsi')
# => set search_path to pepsi;

Apartment::Tenant.drop('pepsi')
# => drop schema pepsi cascade;
```

Installing apartment

```
# Gemfile

gem 'apartment'


# and then do a

rails g apartment:install
```

Configuring apartment - Elevator

```
# config/initializers/apartment.rb

require 'apartment/elevators/subdomain'

Rails.application.config.middleware.use \
  'Apartment::Elevators::Subdomain'

# (this is already the default config)
```

Configuring `apartment` - Subdomain Elevator

```
GET pepsi.recipevault.com/recipes/1
  => set search_path to 'pepsi';

Recipe.find(1)
  => select * from recipes
     where recipes.id = 1;

  (aka)

  => select * from pepsi.recipes
     where recipes.id = 1;
```

Configuring `apartment` - Public Models

```ruby
# config/initializers/apartment.rb
Apartment.configure do |config|

  # Keep companies in the public schema
  config.excluded_models = %w(Company)

end
```

Configuring apartment - Public Models

```
Apartment::Tenant.switch!('coke')

Company.all
  => select * from public.companies;
                   ^^^^^^

Recipe.all
  => select * from recipes;

  (aka)

  => select * from coke.recipes;
                   ^^^^
```

Configuring apartment - Schemas

```ruby
# config/initializers/apartment.rb
Apartment.configure do |config|

  # Find subdomains from Companies
  config.tenant_names = lambda do
    Company.pluck(:subdomain)
  end

end
```

Accessing Local Tenants

```
pepsi.localhost:3000/recipes
```

Accessing Local Tenants - lvh.me

```
~ % dig +noall +answer lvh.me @8.8.8.8

lvh.me.   1544   IN   A   127.0.0.1
```

Accessing Local Tenants

`pepsi.lvh.me:3000/recipes`

instead of

`localhost:3000/recipes`

Demo 3 / 3

```
$ git checkout apartment
```

Caveats

Migrations

```
# Every schema has to be migrated individually.
# Apartment does this transparently,
# but it could be problematic with
# thousands of tenants.
#
# It's probably fine.
```

Rails & Heroku Console

```
# It seems annoying at first
Company.find_by(subdomain: 'pepsi').activate!
# but then you realize its worth it
```

Background Jobs

```
# For normal jobs: #solved
#
# This adds metadata containing the current
# tenant to the job payload, then transparently
# switch!s upon running the job
gem 'apartment-sidekiq'
```

Background Jobs

```ruby
# For system jobs:
class MuhJob
  def perform(*args)
    Company.find_each do |company|
      company.activate!

      # Normal job stuff
      Post.old.archive!
    end
  end
end
```

Action Cable

```ruby
# It's a pretty quick fix.
class ApplicationCable::Connection < ActionCable::C
  identified_by :current_user, :tenant
  #                             ^^^^^^^

  def connect
    self.tenant = request.subdomain
    Apartment::Tenant.switch!(tenant)

    # As usual
    self.current_user = find_current_user
  end
```

Considerations

Multi-Tennancy And You

Think about what kind of tennancy model your application will have to support. This seems obvious, but its actually so obvious that you might overlook it and mess up. (It took me three tries to get this right)

Multi-Tennancy And You

Different types of tennancy do not necessarily correspond to distinct architecture solutions. Although Salesforce and Basecamp are both B2B SaaS organizational products, they have very different use cases, business goals, and data requirements.

Multi-Tennancy And You

Thinking about user behaviour should tell you what kind of tennancy you should support, but your data complexity and business requirements should guide you when implementing it.