Varnish + Redis

redis

Redis is a key value store.

```
key => value
set key value
get key # value
```

Redis can also store hashes as a value.

```
key => hash
hset user:12345 access active
hget user:12345 access # active
```

Varnish is a high-performance HTTP accelerator.

Why use Varnish?

Fast, stable, low overhead.
20k+ requests per second.
Stop hitting your slow backend.

Why put them together?

Paywalls

Varnish uses a hash key to store request objects.

```
/path/to/content.html
/path/to/content.html + myhost.com + user_access_level
```

Multiple responses for a single url.

Generalize Users

The simple idea is to put your users into different groups based on certain business logic in your application.

Putting it all together

```ruby
Warden::Manager.after_set_user do |user, auth, opts|
  auth.cookies[:user] = user.id
  auth.cookies[:railsapp_token] =
    OpenSSL::HMAC.hexdigest(
      ‘sha256’,
      RailsApp::Application.config.secret_token,
      user.id.to_s + RailsApp::Application.config.secret_token)
end

Warden::Manager.after_authentication do |user, proxy, opts|
  $redis.hset("user:#{user.id}", 'access', user.access_level)
end

Warden::Manager.before_logout do |user, auth, opts|
  auth.cookies[:user] = nil
  auth.cookies[:railsapp_token] = nil
end
```

```
import redis;
import digest;

sub vcl_init {
  redis.init_redis('127.0.0.1', 6379, 200);
}

sub vcl_hash {
  hash_data(req.url);
  hash_data(req.http.host);

  if (req.request != "GET") {
    return(hash);
  } else {
    if (req.request == "GET" && req.http.Cookie ~ "rails_app_token=") {
      # Grab the user id from cookies
      set req.http.X-RailsApp-User = regsub(req.http.Cookie, "^.*?user=([^;]*);*.*$", "\1");

      # Grab the token from cookies
      set req.http.X-RailsApp-Token = "0x" + regsub(req.http.Cookie, "^.*?rails_app_token=([^;]*);*.*$",
"\1");

      # Sign the secret token with the user id
      set req.http.X-RailsApp-Signed = digest.hmac_sha256("<%= RailsApp::Application.config.secret_token %>”,
req.http.X-RailsApp-User + "<%= RailsApp::Application.config.secret_token %>");

      # Check if the signed request equals the cookie token
      # If so, we have a valid request
      if (req.http.X-RailsApp-Signed == req.http.X-RailsApp-Token) {
        set req.http.X-RailsApp-Auth = "<%= RailsApp::Application.config.secret_token %>";

        # Grab the access level from redis and use that as a part of the hash
        set req.http.X-RailsApp-Access = redis.call("HGET user:" + req.http.X-RailsApp-User + " access");
        hash_data(req.http.X-RailsApp-Access);
      }
    }
  }

  unset req.http.Cookie;
  unset req.http.Authorization;

  return(hash);
}
```

```
import redis;
import digest;

sub vcl_init {
  redis.init_redis('127.0.0.1', 6379, 200);
}
```

```
sub vcl_hash {
  hash_data(req.url);
  hash_data(req.http.host);

  if (req.request != "GET") {
    return(hash);
  } else {
    …
  }

  unset req.http.Cookie;
  unset req.http.Authorization;

  return(hash);
}
```

```
if (req.request == "GET" && req.http.Cookie ~ "rails_app_token=") {
  # Grab the user id from cookies
  set req.http.X-RailsApp-User = regsub(req.http.Cookie, "^.*?
user=([^;]*);*.*$", "\1");

  # Grab the token from cookies
  set req.http.X-RailsApp-Token = "0x" + regsub(req.http.Cookie,
"^.*?rails_app_token=([^;]*);*.*$", "\1");

  # Sign the secret token with the user id
  set req.http.X-RailsApp-Signed = digest.hmac_sha256("<%=
RailsApp::Application.config.secret_token %>”, req.http.X-RailsApp-
User + "<%= RailsApp::Application.config.secret_token %>");

  # Check if the signed request equals the cookie token
  # If so, we have a valid request
  if (req.http.X-RailsApp-Signed == req.http.X-RailsApp-Token) {
    set req.http.X-RailsApp-Auth = "<%=
RailsApp::Application.config.secret_token %>";

    # Grab the access level from redis and use that as a part of
the hash
    set req.http.X-RailsApp-Access = redis.call("HGET user:" +
req.http.X-RailsApp-User + " access");
    hash_data(req.http.X-RailsApp-Access);
  }
}
```

How this looks in Rails

```ruby
class ArticleController < ApplicationController
  def show
    @article = Article.find(params[:id])
    render(template_for_show)
  end

  def template_for_show
    # Not signed in, so show them a signup page or something.
    return :show_excerpt unless user_signed_in?

    # Getting a request from Varnish, use it's access level.
    if request.headers.key?('HTTP_X_RAILSAPP_ACCESS')
      return request.headers['HTTP_X_RAILSAPP_ACCESS'].to_sym
    end

    # Fallback and for development.
    "show_#{current_user.access_level}".to_sym
  end
end
```