[Data Stream Tutorial for Real-Time Web Apps with Ruby on Rails | PubNub](http://www.pubnub.com/docs/ruby/tutorial/data-push.html#_ruby_data_stream_tutorial_for_real_time_apps_pubnub)

You when you are finished with this tutorial, you will know how to send or receive “Hello!” to the PubNub Real-Time Network with the Ruby on Rails.

The PubNub supports the gem called pubnub written by Ruby, you can use the data stream APIS of the pubnub gem directly.

Tutorial Contents:

* [Getting Started](http://www.pubnub.com/docs/ruby/tutorial/data-push.html#_getting_started)
* Create the Project
* Configuration
* Write the Controller
* Push Server
* Application architecture and server communications
* How to run the project from the git

### [Getting Started](http://www.pubnub.com/docs/ruby/tutorial/data-push.html#_getting_started)

To complete the exercises in this tutorial you’ll need to have installed Ruby on Rails.

Ruby version required: 2.1.1

Rails version required: 4.1.0

### Step 1: Create the Project

Once you already installed the Ruby on Rails on your computer, you can create a project with the generate command.

$ rails g pubnub\_demos

### Add the PubNub gem

In order to use the pubnub APIs, you need to add the pubnub gem onto the Gemfile.

### Add the Unicorn gem

We use unicorn as the app server.

Then you can run the bundle command.

bundle install

### Step 2: Configuration

### Now we need to set up the appropriate credentials to access the PubNub network. We do this by initializing our PubNub API. Here we can define the publish\_key and subscribe\_key we will need to publish and subscribe to a channel.

Please open the config/pubnub.yml and define the publish\_key and subscribe\_key of your accounts on the pubnub as follows.

### Access the API

subscribe\_key: sub-c-xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx

publish\_key: pub-c-yyyyyyy-yyyy-yyyyy-yyyy-yyyyyyyyyy

We defined the class called PubnubListener and Notify for access the pubnub API. You can check this class in the github(bitbucket url).

### PubnubListener and Notify Classes

require ‘pubnub’

class PubnubListener

CONFIG = ::Psych.load\_file “#{::Rails.root}/conf ig/pubnub.yml”

@pubnub = Pubnub.new(

:publish\_key => CONFIG[’publish\_key’],

:subscribe\_key => CONFIG[’subscribe\_key’]

)

MAIN\_CHANNEL = ‘main\_channel’

class << self

attr\_reader :subscribed

def subscribe channel = MAIN\_CHANNEL, &block

@pubnub.subscribe :channel => channel, &block

end

def publish message, channel = MAIN\_CHANNEL

@pubnub.publish :channel => channel, :message => message do

puts ‘Message sended’

end

end

def unsubscribe channel = MAIN\_CHANNEL, &block

@pubnub.unsubscribe :channel => channel, &block

end

end

end

require ‘httparty’

module Notifier

class << self

def notify channel, message

request Hash[channel: channel, message: message].to\_json

end

private

CONFIG = ::Psych.load\_file “#{::Rails.root}/config/push\_server.yml”

HOST\_WITH\_PORT = “http://#{CONFIG['host']}:#{CONFIG['port']}”

REQUEST\_TIMEOUT = 1

def request\_url

“#{HOST\_WITH\_PORT}”

end

def request message

::HTTParty.post request\_url, body: message, timeout: REQUEST\_TIMEOUT

rescue ::Exception => error

logger = ::Rails.logger

logger.error “Tried to send message to push server but: #{error.inspect}”

end

end

end

### Step 3: Write the Controller

Now you need to create a controller, and define the methods

### Send Messages

It’s time to send a message to everyone around the world subscribed to channel. Because we have defined the sending message method in the PublishListener class so that you can send the message.

Once we are subscribed to the channel all we need to do is use the Publish API, specify the channelname and message we’d like to send. We’ll also include code to capture the response from PubNub.

Subscribe & Publish to channel *demo*

def message

msg = {author: params[:author], message: params[:message]}

PubnubListener.publish msg

render json: {status: 'ok'}

end

def subscribe

@name = params[:user][:nickname]

if @name.present?

cookies[:name] = @name

redirect\_to root\_url

else

flash[:notice] = 'Invalid name'

render :login

end

end

Register the route

Please add the routes into the config/route.rb file.

resource :chat, path: '/', only: [:index] do

collection do

post :message, to: 'chat#message'

end

end

That’s it, as soon as you make the publish call your message is replicated around the world and received by anyone subscribed to the channel.

### Step 4: Create the Chatting Form

Please put the text input box for send message and list for message history, send button as follows.

= hidden\_field\_tag :nickname, @name

.well.col-md-3

.panel.panel-primary

.panel-heading

%h3.panel-title Online

.panel-body

%ul.list-group#online\_users

.well.col-md-9

%ul.nav.nav-tabs#channel\_tabs

%li.active

%a{href: '#main\_channel', role: 'tab', data: {toggle: 'tab'} } Main Channel

.tab\_content

%ul.tab-pane.list-group.message\_list#main\_channel

.form-horizontal.col-md-12

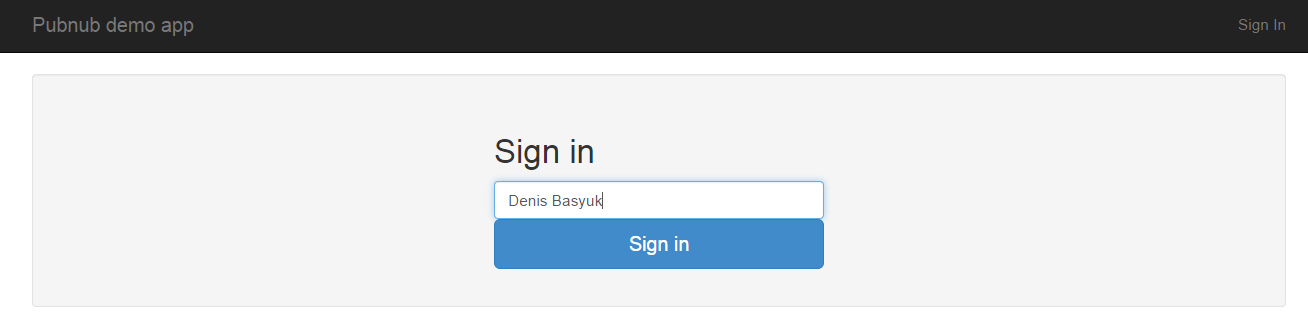
.form-group.col-md-10

%input#chat\_input.form-control

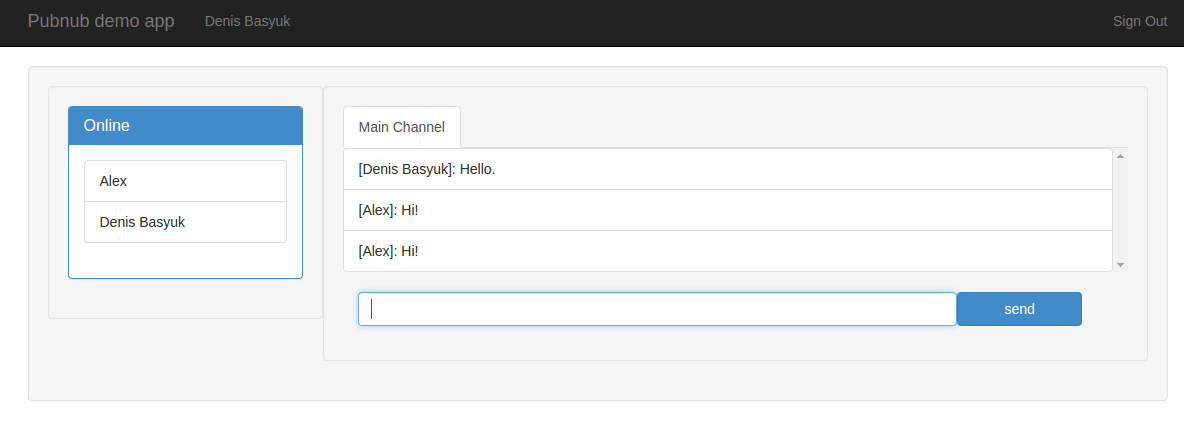
%button#submit.btn.btn-primary.col-md-2

send

Login Page



Chatting Room Page



### Step 5: Push Server

Please create the push\_server.yml with content in the config directory of the project.

Host: localhost

Port: 8082

Next, you need to import the push server to the project from the git.

### Application architecture and server communications

Rails server

Push Server

Web Browsers

Notify

Publish

Message

Post Message

Message

Subscribe by Web Socket



### How to run the project from the git

1. Please install rvm on your os.

2. Then you can clone repo locally.

3. please go to the project and run this command: cd app/ && bundle install

4. create config/push\_server.yml with content

host: localhost

port: 8082

5. run rails server

rails s

6. run push server

cd push\_server && npm install

node server

7. configure nginx

upstream application {

server 127.0.0.1:3000 max\_fails=0 fail\_timeout=20s;

}

upstream push\_server {

server 127.0.0.1:3333 max\_fails=0 fail\_timeout=20s;

}

server {

listen 80;

client\_max\_body\_size 4G;

root /path/to/app/demo\_app;

location @unicorn {

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

include blocked.conf;

proxy\_pass http://application;

}

location ~ ^/broadcast? {

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_http\_version 1.1;

chunked\_transfer\_encoding off;

proxy\_set\_header Connection keep-alive;

proxy\_buffering off;

proxy\_send\_timeout 600s;

proxy\_pass http://push\_server;

}

try\_files /public/$uri @unicorn;

error\_page 500 502 503 504 /500.html;

}

8. run application in browser by http://localhost/