Contents

1 Important Note:

- It is recommended that before deploying the app you set up an SSL certificate for the application, as the security and privacy of this app was based around using HTTPS.

2 Installing Ruby with RVM

Before you can deploy your app on the production server, you need to install Ruby. In this tutorial we recommend that you use Ruby Version Manager (RVM) for this purpose. RVM is a tool for installing and managing multiple Ruby versions.

There are other ways to install Ruby, e.g. through yum, apt-get, source tarball, rbenv and chruby. You can use one of those other installation methods if you so wish, and this tutorial will work fine even if you installed Ruby using one of those other installation methods. But the one that we recommend in this tutorial is RVM, because in our opinion it is the easiest option.

If you have already installed Ruby, then you can skip to the next section

Ensure that curl and gpg are installed, as well as a compiler toolchain. Curl and gpg are needed for further installation steps, while the compiler toolchain is needed for installing common Ruby gems.

Debian, Ubuntu

```
sudo apt-get update
sudo apt-get install -y curl gnupg build-essential
```

3 Install RVM

Run the following commands on your production server to install RVM:

```
$ sudo gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB8
$ curl -sSL https://get.rvm.io | sudo bash -s stable
$ sudo usermod -a -G rvm ‘whoami‘
```

You may need to use gpg2 instead of gpg on some systems. On systems where sudo is configured with secure _path, the shell environment needs to be modified to set rvmsudo _secure _path=1. $\text{secure}_{\text{path}}$ is set on most Linux systems, but not on OS X. The following command tries to autodetect whether it is necessary to install $\text{rvmsudo}_{\text{securepath}}$=1, and only installs the environment variable if it is the code.

```
$ if sudo grep -q secure_path /etc/sudoers; then sudo sh -c "echo export rvmsudo_secure
```

When you are done with all this, relogin to your server to activate RVM. This is important: if you don't relogin, RVM doesn't work. Also if you use gnu screen or another terminal multiplexer, RVM also won't work; you must use a plain ssh session.

4 Install the Ruby version you want

Usually, installing the latest Ruby version will suffice. If you are deploying the example app from the quickstart, then that example application works with all Ruby versions.

However, if you are deploying your own app, then your app may have a specific Ruby version requirement.

4.1 To install the latest version of Ruby, run:

```
$ rvm install ruby
$ rvm --default use ruby
```

4.2 To install a specific version of Ruby, run:

```
$ rvm install ruby-X.X.X
$ rvm --default use ruby-X.X.X
```

Replace X.X.X with the Ruby version you want.

5 Install Bundler

Bundler is a popular tool for managing application gem dependencies. We will use Bundler in this tutorial, so let us install it:

```
$ gem install bundler --no-rdoc --no-ri
```

install Node.js if you're using Rails

If you are using Rails, then you must install Node.js. This is because Rails's asset pipeline compiler requires a Javascript runtime. The Node.js version does not matter.

If you do not use Rails then you can skip to the next step.

To install Node.js:

Ubuntu

```
sudo apt-get install -y nodejs &&
sudo ln -sf /usr/bin/nodejs /usr/local/bin/node
sudo node install -g ember
```

5.1 Heads-up: sudo vs rvmsudo

One thing you should be aware of when using RVM, is that you should use rvmsudo instead of sudo when executing Ruby-related commands. This is because RVM works by manipulating environment variables. However, sudo nukes all environment variables for security reasons, which intereferes with RVM.

6 Install Passenger

6.1 Step 1: install Passenger packages

These commands will install Passenger + Nginx through Phusion's APT repository. If you already had Nginx installed, then these commands will upgrade Nginx to Phusion's version (with Passenger compiled in).

Copy# Install our PGP key and add HTTPS support for APT

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 561F9B9CAC40B2F7
sudo apt-get install -y apt-transport-https ca-certificates

# Add our APT repository
sudo sh -c 'echo deb https://oss-binaries.phusionpassenger.com/apt/passenger xenial ma
sudo apt-get update

# Install Passenger + Nginx
sudo apt-get install -y nginx-extras passenger
```

6.2 Step 2: enable the Passenger Nginx module and restart

Edit /etc/nginx/nginx.conf and uncomment include /etc/nginx/passenger.conf;.
For example, you may see this:

```
# include /etc/nginx/passenger.conf;
```

Remove the '#' characters, like this:

```
include /etc/nginx/passenger.conf;
```

If you don't see a commented version of include /etc/nginx/passenger.conf; inside nginx.conf, then you need to insert it yourself. Insert it into /etc/nginx/nginx.conf inside the http block. For example:

Copy...

```
http {
    include /etc/nginx/passenger.conf;
    ...
}
```

When you are finished with this step, restart Nginx:

```
$ sudo service nginx restart
```

6.3 Step 3: check installation

After installation, please validate the install by running sudo /usr/bin/passenger-config validate-install. For example:

```
$ sudo /usr/bin/passenger-config validate-install
 * Checking whether this Phusion Passenger install is in PATH...
 * Checking whether there are no other Phusion Passenger installations...
```

All checks should pass. If any of the checks do not pass, please follow the suggestions on screen.

Finally, check whether Nginx has started the Passenger core processes. Run sudo /usr/sbin/passenger-memory-stats. You should see Nginx processes as well as Passenger processes. For example:

```
$ sudo /usr/sbin/passenger-memory-stats
Version: 5.0.8
Date   : 2015-05-28 08:46:20 +0200
...

---------- Nginx processes ----------
PID    PPID   VMSize   Private  Name
-------------------------------------
12443  4814   60.8 MB  0.2 MB   nginx: master process /usr/sbin/nginx
12538  12443  64.9 MB  5.0 MB   nginx: worker process
### Processes: 3
### Total private dirty RSS: 5.56 MB
```

```
----- Passenger processes ------
PID    VMSize    Private   Name
--------------------------------
12517  83.2 MB   0.6 MB    PassengerAgent watchdog
12520  266.0 MB  3.4 MB    PassengerAgent server
12531  149.5 MB  1.4 MB    PassengerAgent logger
...
```

If you do not see any Nginx processes or Passenger processes, then you probably have some kind of installation problem or configuration problem. Please refer to the troubleshooting guide.

6.4 Step update regularly

Nginx updates, Passenger updates and system updates are delivered through the APT package manager regularly. You should run the following command regularly to keep them up to date:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

You do not need to restart Nginx or Passenger after an update, and you also do not need to modify any configuration files after an update. That is all taken care of automatically for you by APT.

7 PostgreSQL Server Installation

To install the server locally use the command line and type:

```
$ sudo apt-get install postgresql postgresql-contrib
```

This will install the latest version available in your Ubuntu release and the commonly used add-ons for it.

See "External Links" below for options for getting newer releases.

7.1 Server Setup

If you don't intend to connect to the database from other machines, this alternative setup may be simpler.

By default in Ubuntu, Postgresql is configured to use 'ident sameuser' authentication for any connections from the same machine. Check out the

excellent Postgresql documentation for more information, but essentially this means that if your Ubuntu username is 'foo' and you add 'foo' as a Postgresql user then you can connect to the database without requiring a password.

Since the only user who can connect to a fresh install is the postgres user, here is how to create yourself a database account (which is in this case also a database superuser) with the same name as your login name and then create a password for the user:

```
sudo -u postgres createuser --superuser $USER
sudo -u postgres psql

postgres=# \password $USER
```

Client programs, by default, connect to the local host using your Ubuntu login name and expect to find a database with that name too. So to make things REALLY easy, use your new superuser privileges granted above to create a database with the same name as your login name:

```
sudo -u postgres createdb $USER
```

Connecting to your own database to try out some SQL should now be as easy as:

```
psql
```

Creating additional database is just as easy, so for example, after running this:

```
create database amarokdb;
```

You can go right ahead and tell Amarok to use postgresql to store its music catalog. The database name would be amarokdb, the username would be your own login name, and you don't even need a password thanks to 'ident sameuser' so you can leave that blank.

8 Transferring the app code to the server

8.0.1 Login to your server, create a user for the app

Login to your server with SSH:

```
$ ssh adminuser@yourserver.com
```

Replace adminuser with the name of an account with administrator privileges or sudo privileges.

Starting from this point, unless stated otherwise, all commands that we instruct you to run should be run on the server, not on your local computer! Now that you have logged in, you should create an operating system user account for your app. For security reasons, it is a good idea to run each app under its own user account, in order to limit the damage that security vulnerabilities in the app can do. Passenger will automatically run your app under this user account as part of its user account sandboxing feature.

```
$ sudo adduser ahasuser
```

We also ensure that that user has your SSH key installed:

```
$ sudo mkdir -p ~myappuser/.ssh
$ touch $HOME/.ssh/authorized_keys
$ sudo sh -c "cat $HOME/.ssh/authorized_keys >> ~myappuser/.ssh/authorized_keys"
$ sudo chown -R myappuser: ~myappuser/.ssh
$ sudo chmod 700 ~myappuser/.ssh
$ sudo sh -c "chmod 600 ~myappuser/.ssh/*"
```

8.0.2 Install Git on the server

```
$ sudo apt-get install -y git
```

8.0.3 Pull code

You need to pick a location in which to permanently store your application's code. A good location is /var/www/APP _NAME. Let us create that directory.

```
$ sudo mkdir -p /var/www/ahas
$ sudo chown ahasuser: /var/www/ahas
```

Now let us pull the code from Git:

```
$ cd /var/www/myapp
$ sudo -u myappuser -H git clone https://github.com/CMPUT401/ahasServer.git code
```

Your app's code now lives on the server at /var/www/myapp/code.

8.1 Preparing the app's environment

8.1.1 Login as the app's user

All subsequent instructions must be run under the application's user account. While logged into your server, login under the application's user account as follows:

```
$ sudo -u ahasuser -H bash -l
```

Since you are using RVM, make sure that you activate the Ruby version that you want to run your app under. For example:

```
$ rvm use ruby-2.3.3
```

8.1.2 Install app dependencies

Your application has various dependencies. They must be installed. Most of these dependencies are gems in your Gemfile, managed by Bundler. You can install them by running bundle install –deployment –without development test -j 2 in your app's directory:

```
$ cd /var/www/myapp/code
$ bundle install --deployment --without development test
```

Your app may also depend on services, such as PostgreSQL, Redis, etc. Installing services that your app depends on is outside of this tutorial's scope.

8.1.3 Configure database.yml and secrets.yml

Since your Rails app probably needs a database, you need to edit config/database.yml. For demonstration purposes, we will setup your app with an SQLite database because that is the easiest.

Open the file:

```
$ nano config/database.yml
Ensure that the production section looks like this:
```

#+END$_{\text{SRC}}$ Rails also needs a unique secret key with which to encrypt its sessions. Starting from Rails 4, this secret key is stored in config/secrets.yml. But first, we need to generate a secret key. Run:

```
$ bundle exec rake secret
...
```

This command will output a secret key. Copy that value to your clipboard. Next, open config/secrets.yml:

```
$ nano config/secrets.yml
```

If the file already exists, look for this:

```
production:
  secret_key_base: <%=ENV["SECRET_KEY_BASE"]%>
```

Then replace it with the following. If the file didn't already exist, simply insert the following.

```
production:
  secret_key_base: the value that you copied from 'rake secret'
```

To prevent other users on the system from reading sensitive information belonging to your app, let's tighten the security on the configuration directory and the database directory:

```
$ chmod 700 config db
$ chmod 600 config/database.yml config/secrets.yml
```

1. Email

   The application uses the postmark service, `http://www.postmarkapp.com`. You will need to create an account on Postmark, go to an account page and copy the "Account Api Token" that they provide you. Next, go to config/secrets.yml:

   ```
   $ nano config/secrets.yml
   ```

   Look for this in the file.

   ```
   production:
     postmark_api_key: <%= ENV["POSTMARK_API_KEY"] %>
   ```

   Then replace it with the following.

   ```
   production:
     postmark_api_key: the value that you copied from 'http://www.postmarkapp.com'
   ```

2. Setting Domain

   We need to set the domain of the application, so that our email client can properly link users to the correct web pages. Next, go to config/application.rb

   ```
   $ nano config/application.rb
   ```

   Find the following line in the file.

   ```
   $ config.domain = ENV["DOMAIN"]
   ```

   Replace that line with the following.

   ```
   $ config.domain = https://www.yourdomain.com
   ```

8.1.4 Compile Rails assets and run database migrations

From the root directory run the node build script (this downloads and builds the client side code)

```
$ ./build.sh
```

Run the following command to compile assets for the Rails asset pipeline, and to run database migrations:

```
$ bundle exec rake assets:precompile db:migrate RAILS_ENV=production
```

8.2 Configuring Nginx and Passenger

Now that you are done with transferring your app's code to the server and setting up an environment for your app, it is time to configure Nginx so that Passenger knows how to serve your app.

8.2.1 Determine the Ruby command that Passenger should use

We need to tell Passenger which Ruby command it should use to run your app, just in case there are multiple Ruby interpreters on your system. Please run passenger-config about ruby-command to find out which Ruby interpreter you are using. For example:

```
$ passenger-config about ruby-command
passenger-config was invoked through the following Ruby interpreter:
  Command: /usr/local/rvm/gems/ruby-2.4.0/wrappers/ruby
  ...
```

Please take note of the path after "Command" (in this example, /usr/local/rvm/gems/ruby-2.4.0/wrappers/ruby). You will need it in one of the next steps.

8.2.2 Go back to the admin account

You have previously logged into your app's user account in order to prepare the app's environment. That user does not have sudo access. In the next steps, you need to edit configuration files, for which sudo access is needed. So you need to switch back to the admin account.

This can be done by simply exiting the shell that was logged into the app's user account. You will then be dropped back to the admin account. For example:

```
# This is what you previously ran:
admin$ sudo -u ahasuser -H bash -l
ahasuser$ ...

# Type ‘exit‘ to go back to the account you were before
ahasuser$ exit
admin$ _
```

8.2.3 Edit Nginx configuration file

We need to create an Nginx configuration file and setup a virtual host entry that points to your app. This virtual host entry tells Nginx (and Passenger) where your app is located.

```
$ sudo nano /etc/nginx/sites-enabled/ahas.conf
```

Replace myapp with your app's name.

Put this inside the file:

```
server {
    listen 80;
    server_name yourserver.com;

    # Tell Nginx and Passenger where your app's 'public' directory is
```

```
    root /var/www/myapp/code/public;

    # Turn on Passenger
    passenger_enabled on;
    passenger_ruby /path-to-ruby;
}
```

Replace yourserver.com with your server's host name, and replace /var/www/myapp/code with your application's code directory path. However, make sure that Nginx is configured to point to the public subdirectory inside it!

Replace /path-to-ruby with the Ruby command that you obtained in step 3.1.

When you are done, restart Nginx:

```
$ sudo service nginx restart
```

1. Setting your SSL certificate

   Because this app handles sensitive information, it is strongly suggested you follow the below guide on setting up an SSL certificate for a ruby on rails app.

   SSL Setup

8.2.4 Test drive

You should now be able to access your app through the server's host name! Try running this from your local computer. Replace yourserver.com with your server's hostname, exactly as it appears in the Nginx config file's $\text{server}_{\text{name}}$ directive.

```
$ curl https://yourserver.com/
...your app's front page HTML...
```

If you do not see your app's front page HTML, then these are the most likely causes:

You did not correctly configure your server _name directive. The server _name must exactly match the host name in the URL. For example, if you use the command curl `http://45.55.91.235/` to access your app, then the $\text{server}_{\text{name}}$ must be 45.55.91.235. You did not setup DNS records. Setting up DNS is outside the scope of this tutorial. In the mean time, we recommend that you use your server's IP address as the server name.

9 Alternate Documentation

- If you're looking for alternate documentation for setting up a ruby on rails server on a linux a. `https://www.phusionpassenger.com/library/walkthroughs/deploy/ruby/` b. `https://gorails.com/setup/ubuntu/16.04`