instructure / **canvas-lms**

Production Start

Edit New Page

Shawn Iverson edited this page 9 days ago · 179 revisions

Production Start

These instructions will walk you through how to get a production-ready instance of Instructure's open source Canvas LMS running and serving your users. If you just want to play around with Canvas and get the simplest server running possible, please see the Quick Start page.

If you need help installing Canvas or troubleshooting your installation, your best bet is to join the community mailing list or the IRC channel (see the wiki) and ask specific questions there. It's likely that somebody else has already tackled the same problem.

Prerequisites

We assume you are minimally familiar with website configuration and administration - specifically, Apache and/or generic Ruby on Rails setups. It also doesn't hurt to have a small working knowledge of Git, Postgres, and Passenger. We'll point out in this tutorial places where you may need to go learn about these components.

Secondly, this tutorial is targeting POSIX-based systems (like Mac OS X and Linux). This tutorial was written and tested using Ubuntu 14.04 and 16.04 LTS. If you have a different system, consider setting up a server or virtual machine running the latest Ubuntu LTS. We'll assume you've either done so or are familiar with these working parts enough to do translations yourself.

Finally, Canvas likes RAM. While it will run on smaller configurations, we recommend a server with at least 8GB RAM, especially if everything is being run on one server.

Choose your server configuration

You can choose to run Canvas on one or many servers, hosted by a database. You can install the database on the same server as the one Canvas is hosted from, or you can install it separately. No matter what you choose, you will simply need to ensure that all Canvas instances can communicate with your database server, wherever it is.

Additionally, you will need one Canvas app server to run automated jobs. Again, this can be one of your web servers, or it can be a dedicated node. While there is no downside to running the automated job daemon alongside your webserver, if you plan on having much traffic it is recommended to keep the job traffic and user traffic partitioned onto different nodes for best performance.

For the purposes of this tutorial, we will be referring to the server (possibly one of many) that is running Canvas as *appserver*, whereas we'll be referring to the server running your database as *dbserver*. An *appserver* node can either be hosting the website, be processing automated jobs, or both, depending on whether or not you set up a webserver or the automated jobs daemon.

Database installation and configuration

Installing Postgres

Rails, the library Canvas uses, supports many database adapters, but we primarily use Postgres and SQLite (for testing). Since this tutorial is for setting up a production environment, we recommend Postgres.

You can run Postgres on the same server you're going to run Canvas on, or not. It really doesn't matter. Just make sure the server you're running Canvas on can talk to the Postgres database.

If Postgres isn't already on the host you are planning on running your database on, if the host is Debian/Ubuntu, then this is as easy as

Show 2 more pages…

+ Add a custom sidebar

Clone this wiki locally

`https://github.com/instru`

Clone in Desktop

```
sysadmin@dbserver:~$ sudo apt-get install postgresql-9.5
```

N.B., if you're running MacOS X and using the excellent Homebrew tool, then you can just run `brew install postgresql`. Note that you need Xcode though.

Be sure you're running at least Postgres version 9.5.

Running Postgres on a different server

If you are running Postgres on a different server than the server that Canvas will be running on, you'll need to make sure Postgres is listening to connections from foreign clients. You can do this by editing `postgresql.conf` and `pg_hba.conf` as described in [the Postgres documentation] (http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html).

Configuring Postgres

You'll want to set up a Canvas user inside of Postgres. Note that in the below commands, you'll want to replace *localhost* with the hostname of the server Canvas is running on, if Canvas is running on a different server than Postgres.

```
# createuser will prompt you for a password for database user
sysadmin@dbserver:~$ sudo -u postgres createuser canvas --no-createdb \
   --no-superuser --no-createrole --pwprompt
sysadmin@dbserver:~$ sudo -u postgres createdb canvas_production --owner=canvas
```

Getting the code

There's two primary ways to get a copy of Canvas

Using Git

You can install Git on Debian/Ubuntu by running

```
sysadmin@appserver:~$ sudo apt-get install git-core
```

Once you have a copy of Git installed on your system, getting the latest source for Canvas is as simple as checking out code from the repo, like so:

```
sysadmin@appserver:~$ git clone https://github.com/instructure/canvas-lms.git
canvas
sysadmin@appserver:~$ cd canvas
sysadmin@appserver:~$ git checkout stable
```

Using a Tarball or a Zip

You can also download a tarball or zipfile.

- Canvas Tarball
- Canvas Zip

Code installation

We need to put the Canvas code in the location where it will run from. On a Unix machine, choosing something like the following is a good choice:

```
/var/canvas
```

Take your tarball or your checkout and make sure you move the contents to this directory you've chosen such that all of the directories inside the canvas directory (app, config, db, doc, public, etc). all exist inside this new directory you chose.

We'll be referring to */var/canvas* (or whatever you chose) as your Rails application root.

As an example:

```
sysadmin@appserver:~$ sudo mkdir -p /var/canvas
sysadmin@appserver:~$ sudo chown -R sysadmin /var/canvas
sysadmin@appserver:~$ cd canvas
sysadmin@appserver:~/canvas$ ls
app     db   Gemfile  log     Rakefile  spec  tmp
config  doc  lib      public  script    test  vendor
sysadmin@appserver:~/canvas$ cp -av . /var/canvas
sysadmin@appserver:~/canvas$ cd /var/canvas
sysadmin@appserver:/var/canvas$ ls
app     db   Gemfile  log     Rakefile  spec  tmp
config  doc  lib      public  script    test  vendor
sysadmin@appserver:/var/canvas$
```

Dependency Installation

Canvas now requires Ruby 2.4.0, as of the 2017-04-22 release.

External dependencies

Debian/Ubuntu

We now need to install the Ruby libraries and packages that Canvas needs. On Debian/Ubuntu, there are a few packages you're going to need to install. If you're running Ubuntu, you'll need to add a PPA in order to get the required Ruby version, by using the following commands:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:brightbox/ruby-ng
$ sudo apt-get update
```

Now, We install Ruby 2.4 via the following command:

```
$ sudo apt-get install ruby2.4 ruby2.4-dev zlib1g-dev libxml2-dev \
                       libsqlite3-dev postgresql libpq-dev \
                       libxmlsec1-dev curl make g++
```

Node.js installation:

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install nodejs
```

After installing Postgres, you will need to set your system username as a postgres superuser. You can do so by running the following commands:

```
sudo -u postgres createuser $USER
sudo -u postgres psql -c "alter user $USER with superuser" postgres
```

Mac OS X

For OS X, you'll need to install the Command Line Tools for Xcode, and make sure you have Ruby 2.1. You can find out what version of Ruby your Mac came with by running:

```
$ ruby -v
```

You also need Postgres and the xmlsec library installed. The easiest way to get these is via homebrew. Once you have homebrew installed, just run:

```
$ brew install postgresql nodejs xmlsec1
```

Ruby Gems

Most of Canvas' dependencies are Ruby Gems. Ruby Gems are a Ruby-specific package management system that operates orthogonally to operating-system package management systems.

Bundler and Canvas dependencies

Canvas uses Bundler as an additional layer on top of Ruby Gems to manage versioned dependencies. Bundler is great!

```
sysadmin@appserver:/var/canvas$ sudo gem install bundler --version 1.13.6
sysadmin@appserver:/var/canvas$ bundle install --path vendor/bundle
```

Note on Mac OS X Mavericks

If your on Mac OS X Mavericks, the thrift gem may fail to build due to a bug concerning xcode. This can be resolved by running the following:

```
sysadmin@appserver:/var/canvas$ sudo gem install bundler --version 1.13.6
sysadmin@appserver:/var/canvas$ bundle config build.thrift --with-cppflags='-
D_FORTIFY_SOURCE=0'
sysadmin@appserver:/var/canvas$ bundle install --path vendor/bundle
```

Yarn Installation

Canvas now prefers yarn instead of npm.

```
sysadmin@appserver:/var/canvas$ curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg |
sudo apt-key add -
sysadmin@appserver:/var/canvas$ echo "deb https://dl.yarnpkg.com/debian/ stable
main" | sudo tee /etc/apt/sources.list.d/yarn.list
sysadmin@appserver:/var/canvas$ sudo apt-get update && sudo apt-get install
yarn=1.6.0-1
```

Also, make sure python is installed (needed for contextify package)

```
sysadmin@appserver:/var/canvas$ sudo apt-get install python
```

Then install the node modules:

```
sysadmin@appserver:/var/canvas$ yarn install
```

Canvas default configuration

Before we set up all the tables in your database, our Rails code depends on a small few configuration files, which ship with good example settings, so, we'll want to set those up quickly. We'll be examining them more shortly. From the root of your Canvas tree, you can pull in the default configuration values like so:

```
sysadmin@appserver:/var/canvas$ for config in amazon_s3 database \
  delayed_jobs domain file_store outgoing_mail security external_migration; \
  do cp config/$config.yml.example config/$config.yml; done
```

Dynamic settings configuration

This config file is useful if you don't want to run a consul cluster with canvas. Just provide the config data you would like for the DynamicSettings class to find, and it will use it whenever a call for consul data is issued. Data should be shaped like the example below, one key for the related set of data, and a hash of key/value pairs (no nesting)

```
sysadmin@appserver:/var/canvas$ cp config/dynamic_settings.yml.example
config/dynamic_settings.yml
```

```
sysadmin@appserver:/var/canvas$ nano config/dynamic_settings.yml
```

Database configuration

Now we need to set up your database configuration to point to your Postgres server and your production databases. Open the file *config/database.yml*, and find the **production** environment section. You can open this file with an editor like this:

```
sysadmin@appserver:/var/canvas$ cp config/database.yml.example config/database.yml
sysadmin@appserver:/var/canvas$ nano config/database.yml
```

Update this section to reflect your Postgres server's location and authentication credentials. This is the place you will put the password and database name, along with anything else you set up, from the Postgres setup steps.

Outgoing mail configuration

For Canvas to work properly, you need an outgoing SMTP mail server. All you need to do is get valid outgoing SMTP settings. Open *config/outgoing_mail.yml*:

```
sysadmin@appserver:/var/canvas$ cp config/outgoing_mail.yml.example
config/outgoing_mail.yml
sysadmin@appserver:/var/canvas$ nano config/outgoing_mail.yml
```

Find the **production** section and configure it to match your SMTP provider's settings. Note that the *domain* and *outgoing_address* fields are not for SMTP, but are for Canvas. *domain* is required, and is the domain name that outgoing emails are expected to come from. *outgoing_address* is optional, and if provided, will show up as the address in the *From* field of emails Canvas sends.

If you don't want to use authentication, simply comment out the lines for *user_name*, *password*, and *authentication*.

URL configuration

In many notification emails, and other events that aren't triggered by a web request, Canvas needs to know the URL that it is visible from. For now, these are all constructed based off a domain name. Please edit the **production** section of *config/domain.yml* to be the appropriate domain name for your Canvas installation. For the *domain* field, this will be the part between `http://` and the next `/`. Instructure uses *canvas.instructure.com*.

```
sysadmin@appserver:cp config/domain.yml.example config/domain.yml
sysadmin@appserver:/var/canvas$ nano config/domain.yml
```

Note that the optional *files_domain* field is required if you plan to host user-uploaded files and wish to be secure. *files_domain* must be a different hostname from the browser's perspective, even though it can be the same Apache server, and even the same IP address.

Security configuration

You must insert randomized strings of at least 20 characters in this file:

```
sysadmin@appserver:/var/canvas$ cp config/security.yml.example config/security.yml

sysadmin@appserver:/var/canvas$ nano config/security.yml
```

Generate Assets

Canvas needs to build a number of assets before it will work correctly. First, create the directories that will store the generated files. See the Canvas ownership section below in case you want to plan to assign ownership to canvasuser and the user does not exist yet.

```
sysadmin@appserver:~$ cd /var/canvas
sysadmin@appserver:/var/canvas$ mkdir -p log tmp/pids public/assets
app/stylesheets/brandable_css_brands
sysadmin@appserver:/var/canvas$ touch
app/stylesheets/_brandable_variables_defaults_autogenerated.scss
sysadmin@appserver:/var/canvas$ touch Gemfile.lock
sysadmin@appserver:/var/canvas$ touch log/production.log
sysadmin@appserver:/var/canvas$ sudo chown -R canvasuser config/environment.rb log
tmp public/assets \

app/stylesheets/_brandable_variables_defaults_autogenerated.scss \
                              app/stylesheets/brandable_css_brands Gemfile.lock
config.ru
```

Then will need to run:

```
sysadmin@appserver:/var/canvas$ yarn install
sysadmin@appserver:/var/canvas$ RAILS_ENV=production bundle exec rake
canvas:compile_assets
sysadmin@appserver:/var/canvas$ sudo chown -R canvasuser public/dist/brandable_css
```

Subsequent Updates (NOT NEEDED FOR INITIAL DEPLOY)

If you are updating code, and you run canvas:compile_assets in a place that does not have a database connection, then you'll also want to run the following once code is in place and an active DB connection exists, in order to full update existing themes:

```
sysadmin@appserver:/var/canvas$ RAILS_ENV=production bundle exec rake
brand_configs:generate_and_upload_all
```

Database population

Once your database is configured, and assets are installed, we need to actually fill the database with tables and initial data. You can do this by running our *rake* migration and initialization tasks from your application's root:

```
sysadmin@appserver:/var/canvas$ RAILS_ENV=production bundle exec rake
db:initial_setup
```

Note that this initial setup will interactively prompt you to create an administrator account, the name for the default account, and whether to submit usage data to Instructure. The prompts can be "pre-filled" by setting the following environment variables:

Environment Variable	Value(s) supported
CANVAS_LMS_ADMIN_EMAIL	E-mail address used for default administrator login
CANVAS_LMS_ADMIN_PASSWORD	Password for default administrator login
CANVAS_LMS_ACCOUNT_NAME	Account name seen by users, usually your organization name
CANVAS_LMS_STATS_COLLECTION	opt_in, opt_out, or anonymized

Canvas ownership

Making sure Canvas can't write to more things than it should.

Set up or choose a user you want the Canvas Rails application to run as. This can be the same user as your webserver (*www-data* on Debian/Ubuntu), your personal user account, or something else. Once you've chosen or created a new user, you need to change the ownership of key files in your application root to that user.

```
sysadmin@appserver:/var/canvas$ sudo adduser --disabled-password --gecos canvas
canvasuser
```

Making sure other users can't read private Canvas files

There are a number of files in your configuration directory (`/var/canvas/config`) that contain passwords, encryption keys, and other private data that would compromise the security of your Canvas installation if it became public. These are the *.yml* files inside the *config* directory, and we want to make them readable only by the *canvasuser* user.

```
sysadmin@appserver:/var/canvas$ sudo chown canvasuser config/*.yml
sysadmin@appserver:/var/canvas$ sudo chmod 400 config/*.yml
```

Note that once you change these settings, to modify the configuration files henceforth, you will have to use `sudo` .

Making sure to use the "most restrictive" permissions

Passenger will choose the user to run the application based on the ownership settings of config/environment.rb (you can view the ownership settings via the `ls -l` command). Note that it is probably wise to ensure that the ownership settings of all other files besides the ones with permissions set just above are restrictive, and only allow your canvasuser user account to read the rest of the files.

Apache configuration

Installation

You're now going to need to set up the webserver. We're going to use Apache and Passenger to serve the Canvas content. Before proceeding, you'll need to add the Phusion Passenger APT repository, which contains the `passenger` package. After you've installed the repository, you'll need to install the apache and passenger packages. If you are on Debian/Ubuntu, you can do this quickly by typing:

```
sysadmin@appserver:/var/canvas$ sudo apt-get install passenger libapache2-mod-
passenger apache2
```

We'll be using mod_rewrite, so you'll want to enable that.

```
sysadmin@appserver:/var/canvas$ sudo a2enmod rewrite
```

On Mac OS X you can simple use:

```
sysadmin@appserver:/var/canvas$ brew install passenger
```

and follow the instructions.

Once you have Apache and Passenger installed, we're going to need to set up Apache, Passenger, and your Rails app to all know about each other. This will be a brief overview, and for more detail, you should check out the Passenger documentation for setting up Apache.

Configure Passenger with Apache

First, make sure Passenger is enabled for your Apache configuration. In Debian/Ubuntu, the *libapache2-mod-passenger* package should have put symlinks inside of */etc/apache2/mods-enabled/* called *passenger.conf* and *passenger.load*. If it didn't or they are disabled somehow, you can enable passenger by running:

```
sysadmin@appserver:/var/canvas$ sudo a2enmod passenger
```

In other setups, you just need to make sure you add the following lines to your Apache configuration, changing paths to appropriate values if necessary:

```
LoadModule passenger_module /usr/lib/apache2/modules/mod_passenger.so
PassengerRoot /usr
PassengerRuby /usr/bin/ruby
```

If you have trouble starting the application because of permissions problems, you might need to add this line to your passenger.conf, site configuration file, or httpd.conf (where canvasuser is the user that Canvas runs as, *www-data* on Debian/Ubuntu systems for example):

```
PassengerDefaultUser canvasuser
```

Configure SSL with Apache

Next, we need to make sure your Apache configuration supports SSL. Debian/Ubuntu doesn't ship Apache with the SSL module enabled by default, so you will need to create the appropriate symlinks to enable it.

```
sysadmin@appserver:/var/canvas$ sudo a2enmod ssl
```

On other systems, you need to make sure something like below is in your config:

```
LoadModule ssl_module /usr/lib/apache2/modules/mod_ssl.so
SSLRandomSeed startup builtin
SSLRandomSeed startup file:/dev/urandom 512
SSLRandomSeed connect builtin
SSLRandomSeed connect file:/dev/urandom 512
SSLSessionCache        shmcb:/var/run/apache2/ssl_scache(512000)
SSLSessionCacheTimeout  300
SSLMutex  file:/var/run/apache2/ssl_mutex
```

```
SSLCipherSuite HIGH:MEDIUM:!ADH
SSLProtocol all -SSLv2
```

Configure Canvas with Apache

Now we need to tell Passenger about your particular Rails application. First, disable any Apache VirtualHosts you don't want running. On Debian/Ubuntu, you can simply unlink any of the symlinks in the */etc/apache2/sites-enabled* subdirectory you aren't interested in. In other set-ups, you can remove or comment out VirtualHosts you don't want.

```
sysadmin@appserver:/var/canvas$ sudo unlink /etc/apache2/sites-enabled/000-
default.conf
```

Now, we need to make a VirtualHost for your app. On Debian/Ubuntu, we are going to need to make a new file called */etc/apache2/sites-available/canvas*. On other setups, find where you put VirtualHosts definitions. You can open this file like so:

```
sysadmin@appserver:/etc/apache2/sites-enabled$ sudo nano /etc/apache2/sites-
available/canvas.conf
```

In the new file, or new spot, depending, you want to place the following snippet. **You will want to modify** the lines designated *ServerName*(2), *ServerAdmin*(2), *DocumentRoot*(2), *SetEnv*(2), *Directory*(2), and probably *SSLCertificateFile*(1) and *SSLCertificateKeyFile*(1), discussed below in the "Note about SSL Certificates".

```
<VirtualHost *:80>
  ServerName canvas.example.com
  ServerAlias canvasfiles.example.com
  ServerAdmin youremail@example.com
  DocumentRoot /var/canvas/public
  RewriteEngine On
  RewriteCond %{HTTP:X-Forwarded-Proto} !=https
```

```
    RewriteCond %{REQUEST_URI} !^/health_check
    RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI} [L]
    ErrorLog /var/log/apache2/canvas_errors.log
    LogLevel warn
    CustomLog /var/log/apache2/canvas_access.log combined
    SetEnv RAILS_ENV production
    <Directory /var/canvas/public>
      Allow from all
      Options -MultiViews
    </Directory>
  </VirtualHost>
  <VirtualHost *:443>
    ServerName canvas.example.com
    ServerAlias canvasfiles.example.com
    ServerAdmin youremail@example.com
    DocumentRoot /var/canvas/public
    ErrorLog /var/log/apache2/canvas_errors.log
    LogLevel warn
    CustomLog /var/log/apache2/canvas_ssl_access.log combined
    SSLEngine on
    BrowserMatch "MSIE [2-6]" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-
  response-1.0
    BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    # the following ssl certificate files are generated for you from the ssl-cert
  package.
    SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
    SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
    SetEnv RAILS_ENV production
    <Directory /var/canvas/public>
      Allow from all
      Options -MultiViews
    </Directory>
  </VirtualHost>
```

Apache 2.4 users: the allow/options configuration inside the `<Directory /var/canvas/public>` have changed in Apache 2.4. You'll likely want something like this:

```
  <Directory /var/canvas/public>
    Options All
    AllowOverride All
    Require all granted
  </Directory>
```

And finally, if you created this as its own file inside */etc/apache2/sites-available*, we'll need to make it an enabled site.

```
sysadmin@appserver:/etc/apache2/sites-enabled$ sudo a2ensite canvas
```

A Note about SSL Certificates

You'll notice in the above Canvas configuration file that we provided directives to an *SSLCertificateFile* and an *SSLCertificateKeyFile*. The files specified are **self-signed** certificates that come with your operating system.

Browsers, by default, are configured not to accept self-signed certificates without complaining. The reason for this is because otherwise a server using a self-signed certificate can risk what's called a man-in-the-middle attack.

If you want to get a certificate for your Canvas installation that will be accepted automatically by your user's browsers, you will need to contact a *certificate authority* and generate one. For the sake of example, Verisign is a commonly used certificate authority.

For more information on setting up Apache with SSL, please see O'Reilly OnLamp.com's instructions, Apache's official SSL documentation, or any one of many certificate authority's websites.

Optimizing File Downloads

If you are storing uploaded files locally, rather than in S3, you can optimize the downloading of files using the X-Sendfile header (X-Accel-Redirect in nginx). First make sure that apache has mod_xsendfile installed and enabled. In `config/environments/production.rb` you'll find the necessary `config.action_dispatch.x_sendfile_header` line, but commented out. We recommend that you create a `config/environments/production-local.rb` file and add the uncommented line to that file, to avoid future merge conflicts.

In your canvas virtual host at /etc/apache2/sites-available/canvas, add the following two directives:

```
    XSendFile On
    XSendFilePath /var/canvas
```

Cache configuration

Canvas supports two different methods of caching: Memcache and redis. However, there are some features of Canvas that require redis to use, such as OAuth2, so it's recommended that you use redis for caching as well to keep things simple.

Below are instructions for setting up redis.

Redis

Required version: redis 2.6.x or above.

Note: Ubuntu installs an older version by default. See http://redis.io/download for instructions on how to manually install redis 2.6.x or above manually or use the PPA below.

If you're using Homebrew on Mac OS X, you can install redis by running the command: `brew install redis`.

For Ubuntu, you can use the redis-server package. However, on trusty, it's not new enough, so you'll want to use a backport PPA to provide it: https://launchpad.net/~chris-lea/+archive/redis-server.

```
sysadmin@appserver:/var/canvas$ sudo add-apt-repository ppa:chris-lea/redis-server
sysadmin@appserver:/var/canvas$ sudo apt-get update
sysadmin@appserver:/var/canvas$ sudo apt-get install redis-server
```

After installing redis, start the server. There are multiple options for doing this. You can set it up so it runs automatically when the server boots, or you can run it manually.

To run it manually from a Homebrew installation, run the command: `redis-server /usr/local/etc/redis.conf` .

Now we need to go back to your canvas-lms directory and edit the configuration. Inside the config folder, we're going to copy cache_store.yml.example and edit it:

```
sysadmin@appserver:/var/canvas$ cd /var/canvas/
sysadmin@appserver:/var/canvas$ cp config/cache_store.yml.example
config/cache_store.yml
sysadmin@appserver:/var/canvas$ nano config/cache_store.yml
sysadmin@appserver:/var/canvas$ sudo chown canvasuser config/cache_store.yml
sysadmin@appserver:/var/canvas$ sudo chmod 400 config/cache_store.yml
```

The file may start with all caching methods commented out. Match your config file to the entries below:

```
test:
    cache_store: redis_store
development:
    cache_store: redis_store
production:
    cache_store: redis_store
```

Then specify your redis instance information in `redis.yml` , by coping and editing redis.yml.example:

```
sysadmin@appserver:/var/canvas$ cd /var/canvas/
sysadmin@appserver:/var/canvas$ cp config/redis.yml.example config/redis.yml
sysadmin@appserver:/var/canvas$ nano config/redis.yml
sysadmin@appserver:/var/canvas$ sudo chown canvasuser config/redis.yml
sysadmin@appserver:/var/canvas$ sudo chmod 400 config/redis.yml
```

```
production:
  servers:
    - redis://localhost
```

In our example, redis is running on the same server as Canvas. That's not ideal in a production setup, since Rails and redis are both memory-hungry. Just change 'localhost' to the address of your redis instance server.

Canvas has the option of using a different redis instance for cache and for other data. The simplest option is to use the same redis instance for both. If you would like to split them up, keep the redis.yml config for data redis, but add another separate server list to cache_store.yml to specify which instance to use for caching.

Save the file and restart Canvas.

QTIMigrationTool

The QTIMigrationTool needs to be installed to enable copying or importing quiz content. Instructions are at https://github.com/instructure/QTIMigrationTool/wiki. After installation, ensure the plugin is active in Site Admin -> Plugins -> QTI Converter (it should detect the QTIMigrationTool and auto-activate).

Automated jobs

Canvas has some automated jobs that need to run at occasional intervals, such as email reports, statistics gathering, and a few other things. Your Canvas installation will not function properly without support for automated jobs, so we'll need to set that up as well.

Canvas comes with a daemon process that will monitor and manage any automated jobs that need to happen. If your application root is */var/canvas*, this daemon process manager can be found at */var/canvas/script/canvas_init*.

You'll need to run these job daemons on at least one server. Canvas supports running the background jobs on multiple servers for capacity/redundancy, as well.

Because Canvas has so many jobs to run, it is advisable to dedicate one of your app servers to be just a job server. You can do this by simply skipping the Apache steps on one of your app servers, and then only on that server follow these automated jobs setup instructions.

Installation

If you're on Debian/Ubuntu, you can install this daemon process very easily, first by making a symlink from */var/canvas/script/canvas_init* to */etc/init.d/canvas_init*, and then by configuring this script to run at valid runlevels (we'll be making an *upstart* script soon):

```
sysadmin@appserver:/var/canvas$ sudo ln -s /var/canvas/script/canvas_init
/etc/init.d/canvas_init
sysadmin@appserver:/var/canvas$ sudo update-rc.d canvas_init defaults
sysadmin@appserver:/var/canvas$ sudo /etc/init.d/canvas_init start
```

Rich Content Editor

Canvas includes a new rich content editor component to support a consistent editor experience across multiple applications in the Canvas ecosystem. To make use of this component you need to run a supporting API server. See the Canvas RCE API Documentation for information on running the service and configuring Canvas to make use of it. Starting July 14, 2018, the `stable` branch of `canvas-lms` will require this service to be running and configured for full rich content editing functionality.

Ready, set, go!

Restart Apache (`sudo /etc/init.d/apache2 restart`), and point your browser to your new Canvas installation! Log in with the administrator credentials you set up during database configuration, and you should be ready to use Canvas.

Troubleshooting

We have a full page of frequently asked questions about troubleshooting your Canvas installation. See our Troubleshooting page.

Common configuration options

There are many other aspects of Canvas that you can now configure, having a working production environment. Please see Canvas Integration for more information.

+ Add a custom footer