Fire Documentation

Release 2.0.1

Arnar Birgisson

Dec 04, 2017

User documentation

1 Frequently Asked Questions 3

2 Introduction 5

3 Setup and system components 7

4 Usage 9

5 Automated Checks 11

6 Git branches 13

7 Databse model 15

8 Manual testing 17

9 Automated testing 21

10 Creating a new release 23

11 The fire server (a.k.a. The Firehouse) 25

12 API documentation 27

13 Indices and tables 29

Contents:

CHAPTER 1

Frequently Asked Questions

1.1 How do I change group in the middle of a course?

It happens that one needs to change group in the middle of a course, that is after having already completed some assignments. To change group you must first leave your current group by clicking on the "no entry" sign () beside your group's number on the index page. Once this is done, you can join an existing group or create a new one.

Caution: After changing group, you won't be able to submit to assignments that you already submitted in the old group. E.g. in a course with two labs A and B, if you submitted lab A as a member of group 1 and then leave group 1 and create a new group (group 2), you will not be able to submit to lab A anymore (lab B is fine).

If you have labs that have not yet been accepted when changing group, we recommend that you **note the join code for your original group** to be able to switch back in case you need to re-submit.

CHAPTER 2

Introduction

CHAPTER 3

Setup and system components

This document describes the prerequisites on a Fire server. Fire setup is divided in two parts, *course setup* which specifies a particular release version of Fire and associated data files, and the *system setup* which includes server requirements and the setup of a main webserver that distributes requests to individual course instances.

3.1 Course setup

3.1.1 Stating a vew course

The web server is nginx. However, each fire instance runs as a fastcgi python server, each under a separate user. The nginx config is under /etc/nginx. The fire.conf file is for the old version of fire, so just ignore that. In nginx.conf there are two places to change, look for "fafl". You can see in two places very similar regexes that match paths like /(?<course>lbs|fafl|...).../. They proxy requests to the right socket file for that instance, stored under /var/run/fire/coursename.sock. Add your course slug to the two regex.

Then you need to create a user called fire_X where X is the course name:

```
useradd -m -r fire_X
```

Under that user, install virtualenv and use it to create a virtual env directory called .fire-virtualenv in the home directory. In it, install fire and its dependencies with pip:

```
sudo -u fire_X -i
virtualenv -p python2.7 ~/.fire-virtualenv
. ~/.fire-virtualenv/bin/activate
git clone https://bitbucket.org/cse-fire/fire
pip install ./fire
```

Then create a fire.conf file in the home directory, again look at fire_fafl for the example. Create also a directory for the data, e.g. data-2014 or whatever path you use in fire.conf and for the logs:

```
mkdir ~/data ~/logs
```

Make sure the virtualenv is active and run "fireadmin -c fire.conf initdb". This will populate the data directory and create the sqlite file:

```
fireadmin -c fire.conf initdb
```

Start the new fire instance:

```
systemctl start fire@cp
```

This should start the python server and create the socket file. If you want it to start when the server is restarted, you need to enable it:

```
systemctl enable fire@cp
```

If you didn't already, reload the nginx config (systemctl reload nginx, or restart if reload doesn't work). Now you should be able to visit http://xdat09.ce.chalmers.se/X/ and see your new instance.To create users etc, for now, log in as the user, activate the virtualenv and run pshell. Then you have a python shell where you can do stuff like:

```
import transaction
from fire.models import *
with transaction.manager:
  u = User('someone@somewhere.com')
  u.password = 't0ps3cr3t'
  u.role = User.Roles.admin
  DBSession().add(u)
  #   to close the with block
```

This should commit the transaction if there are no errors. Otherwise, refer to the docs on SQLAlchemy and the transaction package.

Setup crontab, that is used to send mail asynchronously. Run `crontab -e`. Sample config:

```
* * * * * ~/.fire-virtualenv/bin/qp --hostname mail.medic.chalmers.se ~/XXXX/mailq 1>␣
↪~/logs/mail.log 2>&1
```

where `XXXX` is a directory with data (the same as `fire.data_dir` in `fire.conf`). To make sure, that changes went though, run `crontab -l`.

TODO: nginx config

3.2 System setup

CHAPTER 4

Usage

4.1 Entering course information

4.2 Adding graders

4.3 Adding labs

CHAPTER 5

Automated Checks

Automated checks are a way for a course to run automated verifications on student submissions. For instance it can:

- build the student code
- run a linter (e.g. `hlint`) to make sure that the code is minimally readable
- run automated tests on the student code
- etc.

The results of automated checks are shown in the submission page to both students and graders.

Note: fire will **not** automatically accept/reject student submission based on automated checks, this is left at the discretion of the graders.

5.1 Overview

Here is how automated checks works

- Students create a submission on fire
- The fire server notify the test server by requesting a particular URL and pass it two addresses: one where the test server can download the students' submission and one where it can submit feedback
- Once the test server has finished running the tests, it notifies the fire server using the passed URL and sends the check status (pass or fail) and optionally an address where more information are displayed (test log, generated report, etc...)
- The fires server displays the check status on the submission page to both the grader and the students.

5.2 Use case: using Jenkins as a test server

In this example we will see how to use Jenkins as a test server for student submissions. We assume that you already have a fire instance and a sever with jenkins installed.

5.2.1 Create a Jenkins job

- Click on New Item, give your job a name and select "Freestyle project".
- Tick the check-box "This build is parameterized"
- Add two *Password Parameter*: `submission_url` and `feedback_url`
- Bellow, under *Build Triggers*, check *Trigger builds remotely (e.g. from scripts)*
- Create a good authentication token (you may want to use **`pwgen`**). In this example we'll use *s3cr3t* for brevity.
- Under *Build Environment*, check *Delete workspace befor build starts*
- Under *Build*, add an *Execute shell* build step with the following script:

```
#!/bin/sh -eux

curl --silent $submission_url | tar zxf -

if [ $[RANDOM % 2] = 0 ]; then
  status=pass
else
  status=fail
fi

curl --silent -X POST \
  --data status=$status \
  --data details_url=$BUILD_URL/console \
  $feedback_url
```

  (of course a real check will not choose a status at random)

- Click *Save*

Congratulations, your check job is now ready!

5.2.2 Set-up fire to use the Jenkins job

- In fire, go to the lab for which you want to add an automated check and click the *Checks* tab.
- Give a name to your check and the following url:

```
<url to your jenkins server/job/<job name>/buildWithParameters?token=<your token>&
↪submission_url=$sumbission_url&feedback_url=$feedback_url
```

- Click *Add*

Now your check is ready. You might want to create a phony student to make a test submission to make sure that everything works as expected.

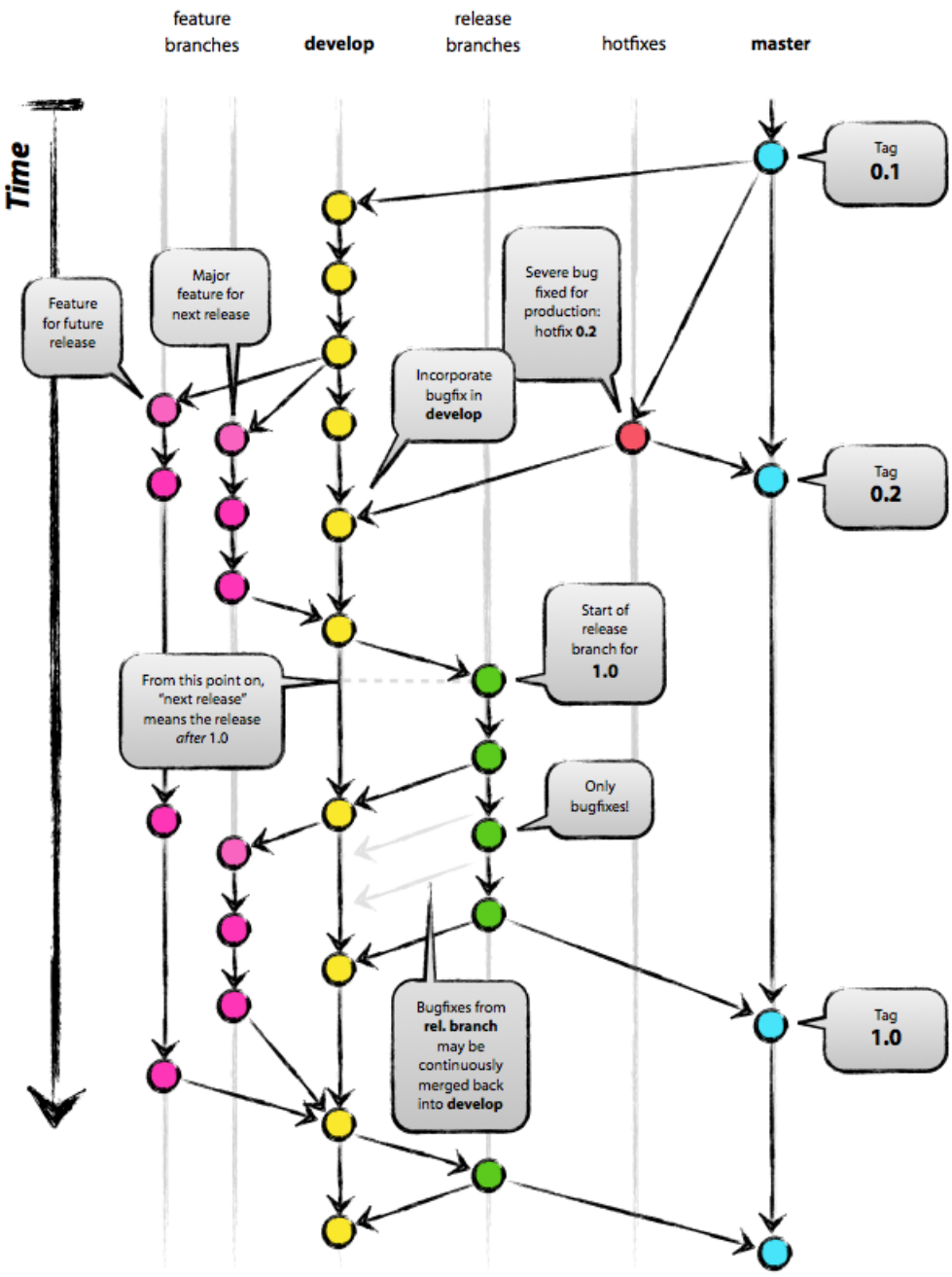
CHAPTER 6

Git branches

The development model used for fire development is a model introduced by Vincent Driessen in A successful Git branching model.

All work is done on the `develop` branch, or a feature branch (which is then merged into `develop`). `master` should *always* be clean. To make a release, You make a release branch form `develop`, bumg the version and commit any necessary bugfixes then you merge the release branch into master and tag a release. Usually the only commit on the release branch is actually bumping the version numbers. Have a look at the git tree view of the history.

Here is a graphical representation, courtesy Vincent Driessen (Creative Commons BY-SA)


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

CHAPTER 7

Databse model

CHAPTER 8

Manual testing

8.1 Test cases

8.1.1 Course initialization

Pre-conditions: none

#	Step details	Expected results
1	Add a test course to `courses.ini`	
2	Run `fab deploy`	Get an email
3	Click on the URL in the email	Site should open
4	Log-in with the password provided in the email	Log-in successful

8.1.2 Lab creation

Pre-conditions: *Course initialization*

#	Step details	Expected results
1	Log-in as a grader	
2	Add an individual lab (only `.pdf`, allow physical submissions)	Lab created
3	Add a group lab (Any file extension, min/max 2 members, no physical submissions)	Lab created
4	Click on "Graders" (top), add yourself to all all labs	

8.1.3 Student registration

Pre-conditions: *Course initialization*

#	Step details	Expected results
1	Enter an email	Get an email
2	Click link in email	Registration form
3	Fill-in information	Registration successful
4	Login with the newly created password	Success

Do this twice

8.1.4 Submit (Student 1)

Pre-conditions: *Lab creation Student registration*

#	Step details	Expected results
1	Log-in as student 1	
2	Go to lab 1	
3	Upload a pdf file Success	
4	Upload a file which is not a pdf	Failure
5	Submit	Success
6	Go to lab 2	Can't submit without group
7	Go to home	
8	Create group, record the join code	
9	Go to lab 2	Can't submit: not enough members in gruop

8.1.5 Submit (Student 2)

Pre-conditions: *Lab creation Student registration*

#	Step details	Expected results
1	Log-in as student 2	
2	Go to lab 1	
3	Check the physical submission box	
4	Submit	Success
5	Go to home	
6	Join student 1's group	Success
7	Go to lab 2	
8	submit	Success

8.1.6 Review

Pre-conditions: *Submit (Student 1) Submit (Student 2)*

#	Step details	Expected results
1	Log-in a sgrader	Success, the bar shows 3 pending submissions
2	Go to lab 2	1 pending submission assigned to you
3	Click on first	
4	Reject	Both students should get a mail
5	Go to lab 1	2 pending submissions assigned to you
6	Click on the first	
7	Accept	Student should get mail
8	Click on “register physical submissions”	
9	Register physical submission for student 2	Student should get mail
10	Go to lab 1	
11	Click on pending submission	
12	Reject	Student should get mail
13	Go to lab 1	
14	Click on register physical...	
15	Register submission for student 2	Student should get mail
16	Got to lab 1	
17	Click on pending submission	
18	Accept	Student should get mail

CHAPTER 9

Automated testing

Automated tests are in `tests` which is divided in three sub-directories:

- `unit`: unit tests, test small amount of code in isolation. Good for algorithms, string generation, or generally everything that can be easily isolated. Shouldn't do IO (database, files...) so they should be be fast.
- `integration`: integration tests, get a scratch-dir and an (in memory) database. Given the architecture of pyramid web apps, this is where most test will probably end-up. In particular, this is probably the place for view and model tests.
- `end2end`: End-to-end tests (or system tests). Those test a live server through a real browser using selenium. Good coverage but slow and hard to debug.

9.1 Running the test-suite

There are three ways to run the test suite:

- **`./scripts/citest`**: this is the script that is used by the continuous integration server. It runs the whole test suite, generate a report in Xunit format and hide the browser window in a frame buffer (probably only works in linux).
- **`tox`**: will set-up a virtualenv for the test run and install test dependencies.
- **`py.test`**: you need to install the test dependencies yourself, most flexible.

For instance, to run only the end-to-end tests:

```
py.test tests/end2end/
```

CHAPTER 10

Creating a new release

This is how to make a release:

```
git checkout -b release-VERSION develop
## bump version, commit
## Test, fix-bugs, repeat
## ...
git checkout master
git merge --no-ff release-VERSION
git tag -a VERSION
```

After that, you want to merge it in develop:

```
git checkout develop
git merge --no-ff release-VERSION
```

And delete the branch:

```
git branch -d release-VERSION
```

10.1 Versionning

The following is an adaptation of http://semver.org/ adapted to our particular use case: given a version number `MAJOR.MINOR.PATCH`, increment:

1. MAJOR version when you make backward incompatible API changes (here we are talking about an eventual web API, so it will probably no be incremented now). Use 2.
2. MINOR version when you make a change to the database schema
3. PATCH version when you make changes that don't touch the database schema

The idea being that ongoing courses can always be upgraded to a new patch version (changes that don't touch the database) but new minor versions can only be used for new instances.

CHAPTER 11

The fire server (a.k.a. The Firehouse)

11.1 Server information

OS Centos 7

Processor Quad-core AMD Opteron

Memory 16GB

Disk 80G

11.2 DNS configuration

To change the dns record, contact Chalmers NIC <cth-nic@chalmers.se>.

To avoid confusion with the old fire server (fire.cs.chalmers.se), we use the domain frs.cse.chalmers.se for this server.

The current DNS configuration should be something like this:

```
frs.cse.chalmers.se.    IN      TXT     "UINFO: alt-fire-reporting-system-on-web-
↪frameworks [en]"
frs.cse.chalmers.se.    IN      A       129.16.22.11
*.frs.cse.chalmers.se.  IN      A       129.16.22.11
```

Which means that any address of the form <something>.frs.cse.chalmers.se will go to our server.

11.3 Instances configuration

Configuration files are generated by a script in fire-config. See deployment for more information on how it works.

Each fire instance has its own user, a data directory, a systemd service and two nginx virtual servers (one for the app itself and one to serve uploaded files).

For instance, for the course `database-vt16` there will be

- a user named `fire-database-vt16`
- a directory `/srv/courses/database-vt16`
- a systemd service `fire-database-vt16` configured in `/etc/systemd/system/fire-database-vt16.service`
- Two nginx server config, one for https://database-vt16.frs.cse.chalmers.se and one for https://database-vt16-files.frs.cse.chalmers.se, configured in `/etc/nginx/conf.d/database-vt16.conf`.

Certificates are obtained from Let's Encrypt which is an automated certification authority. The configuration to generate and renew the certificates automatically is included in the fire-config repository.

CHAPTER 12

API documentation

This section documents the codebase, and is generated automatically from docstrings inside the `fire` module.

12.1 fire

12.1.1 fire.models

12.1.2 fire.routes

12.1.3 fire.security

12.1.4 fire.templating

12.1.5 fire.views

CHAPTER 13

Indices and tables

- genindex
- modindex
- search