Notes for taler.net admins and developers

Version 0.2.0
31 May 2017

Marcello Stanisci (marcello.stanisci@inria.fr)

Table of Contents

1 Taler.net

1.1 Git

Git at `taler.net` is managed by *gitolite*. Creation and deletion of repositories, as well as users management, is done entirely by editing the `gitolite.conf` file within the `gitolite-admin` repository.

Please refer to gitolite official documentation, if more information is needed: `http://gitolite.com/gitolite/`.

This section documents the set-up of our main server `taler.net`.

1.2 Buildbot

Note: 'worker' and 'slave' are used interchangeably

The user running the buildbot master is *containers*.

1.2.1 Master

To start the master, log in as *containers*, and run:

```
$ ~/buildbot/start.sh

# To stop it, run:
$ ~/buildbot/stop.sh
```

There is also a "restart" script, runnable as follows:

```
$ ~/buildbot/restart.sh
```

NOTE: the following command reloads the configuration without restarting the bot.

```
$ source ~/buildbot/venv/bin/activate
$ buildbot reconfig ~/buildbot/master/
```

1.2.2 Documentation worker

This worker is responsible for building all the documentation on `https://docs.taler.net`. It is run by the user `docbuilder`, whose home directory must have all the repositories with documentation to be built. Namely,

- *exchange*
- *merchant*
- *merchant-frontend-examples*
- *deployment* - because of "onboarding" documentation
- *api*

It can be start with the following command:

```
# First log-in as the 'docbuilder' user.
$ source ~/buildbot/venv/activate
$ buildbot-worker start ~/buildbot/worker

# stop it with:
```

```
$ buildbor-worker stop ~/buildbot/worker

# deactivate the virtual env with:
$ deactivate
```

> **Note:** use the `--umask=022` option when creating this worker, because Buildbot gives default `077` umask to its processes and this makes generated files unreadable.

1.2.3 Wallet worker

This worker is responsible for running wallet testcases. It is run by the *containers* user. Manage this worker with the following commands:

```
$ source ~/buildbot/venv/bin/activate
$ buildbot-worker start ~/buildbot/wallet_worker/

# stop it with:
$ buildbot-worker stop ~/buildbot/wallet_worker/

# deactivate the virtual env with:
$ deactivate
```

1.2.4 Selenium worker

This worker is responsible for running the Selenium wallet test: an automatic clicker that performs the cycle withdraw-and-spend.

The *containers* user is also responsible for running the Selenium buildbot worker.

Start it with:

```
$ source ~/buildbot/venv/bin/activate
$ buildbot-worker start ~/buildbot/selenium_worker/

# stop it with:
$ buildbot-worker stop ~/buildbot/selenium_worker/

# deactivate the virtual env with:
$ deactivate
```

1.2.5 Lcov worker

The worker is implemented by the *lcovslave* user and is responsible for generating the HTML showing the coverage of our tests, then available on *https://lcov.taler.net*.

To start the worker, log in as *lcovslave* and run:

```
$ source ~/activate
$ taler-deployment-bbstart

# To stop it:
$ taler-deployment-bbstop
```

NOTE: a Postgres "per-user" database service should always be running, as *lcovslave* is deployed in a standalone environment.

Start the database service in the following way:

```
# Assuming you already sourced ~/activate
$ taler-deployment-arm -s
$ taler-deployment-arm -i taler-postgres-standalone
# Stop it this way:
$ taler-deployment-arm -k taler-postgres-standalone
# To stop 'arm', issue:
$ taler-deployment-arm -e
```

1.2.6 Switcher worker

Taler.net uses a "blue/green" fashion to update the code it uses in demos. Practically, there are two users: *test-green* and *test-blue*, and only one of them is "active" at any time.

Being *active* means that whenever nginx receives a HTTP request for one of the Taler services (at our demo), it routes the request to either test-blue or test-green via unix domain sockets.

Upon any push to any of the Taler's subprojects, this worker is responsible for building the code hosted at the inactive user and, if all tests succeed, switching the active user to the one whose code has just been compiled and tested.

The worker is implemented by the *testswitcher* user. This user has some additional "sudo" rights, since it has to act as *test-blue*, 'test-gree*$1*est' user in order to accompish its task. Note that the "sudo file" is tracked in this (*deployment*) repository, under the *sudoers* directory.

To start the worker, log in as *testswitcher* and run:

```
$ source ~/venv/bin/activate
$ buildbot-worker start ~/buildbot/slave

# To stop it:
$ buildbot-worker stop ~/buildbot/slave

# To exit the virtual env
$ deactivate
```

1.2.7 Manual switch

After the desired blue/green party has been compiled, it is possible to log-in as *test* and run the script `$HOME/.ln-<COLOR>.sh`, in order to make `test-<COLOR>` active.

1.2.8 Website lcov.taler.net

The directory `/var/www/lcov.taler.net` contains the following two symlinks

- exchange –> `/home/lcovslave/exchange/doc/coverage`
- merchant –> `/home/lcovslave/merchant/doc/coverage`

The pointed locations are updated by the *lcovslave*.

2 Standalone deployment

This tecnique aims to set a thorough Taler installation up on a machine whose nginx configuration is configured by config files from *https://git.taler.net/deployment.git/tree/etc/nginx*.

This installation assumes that all the steps are run with `$HOME` as `$CWD`.

The first step is to fetch the *deployment* repository, which hosts all the needed scripts.

```
# Adapt the repository's URI to your needs.
$ git clone /var/git/deployment.git/
```

The next step is to fetch all the codebases from all the components.

```
$ ./deployment/bootstrap-standalone
```

If the previous step succeeded, a file named `activate` should be now in the `$CWD`. It contains environmental definitions for `$PATH` and database names.

Note: Please *ignore* the output from the previous script when it succeeds, which is

```
WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -
A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

/usr/lib/postgresql/9.5/bin/pg_ctl -D talerdb -l logfile start
```

The reason is that this message is generated by Postgresql's utilities and you never need to start your database manually; it will be started by the init script that launches all the Taler processes.

Now we need to compile and install all the downloaded codebases.

```
# We first update ``$PATH'', in order to make all the compilation
# and configuration utilities available.
$ source activate

# Double check if the previous step worked: $PATH should
# contain $HOME/local/bin.
$ echo $PATH

# The actual compilation:
$ taler-deployment-build
```

The following step will generate config files for all the components. Please **note** that although a default currency will be picked up by the script, it is possible to have a custom currency by setting the environment variable `TALER_CONFIG_CURRENCY` to the wanted currency, and then running the config generator.

```
$ taler-deployment-config-generate
```

whereas the following one will place signatures inside wireformat JSON files.

```
$ taler-deployment-config-sign
```

The next step is to generate *signkeys* and *denomkeys*. Note that it will also get the denomkeys signed by the (local mock) auditor.

```
$ taler-deployment-keyup
```

> **Note:**
>
> A database error about non existent auditor-related tables might be returned while generating keys. Fix it by running:
>
> ```
> taler-auditor -m $(taler-config -s exchange -o master_public_key) -r
> ```
>
> This is likely to happen after database resets, and `taler-auditor` is responsible for creating all auditor-related tables.

It may be necessary to define database tables for the exchange. The following command does that.

Note that you have to manually start the database, with the following command.

```
taler-deployment-arm -s
taler-deployment-arm -i taler-postrges-standalone
# Erase all the data!
$ taler-exchange-dbinit -r
```

As of the merchant backend, it creates tables at launch time, so it is not required to define tables before launching it. *However*, if some table's definition changed over the time, and there is a need to force a redefinition of tables, then the following command accomplishes that for the merchant:

```
# Erase all the data!
$ taler-merchant-dbinit -r
```

If all previous steps succeeded, it is now possible to launch all the processes. That is accomplished by the following command:

```
$ taler-deployment-start
```

> **Note:** Please make sure your nginx works correctly with its configuration at `<DEPLOYMENT-REPO>/etc/nginx`.

3 Deployment on demo.taler.net

This section describes how to upgrade the whole Taler setup on the `taler.net` Web site. Here, the deployment scripts include a "stable" setup at `demo.taler.net` and an "experimental" setup at `test.taler.net`. This section documents the steps for moving the "experimental" logic to the "stable" site. It is mostly useful for administrators of `taler.net`, but given that all of the configuration files are public, it may also make a good starting point for others.

First, make sure that the deployment *AND* the deployment scripts work on the *test.taler.net* deployment.

For all repositories that have a separate stable branch (currently exchange.git, merchant.git, merchant-frontends.git, bank.git, landing.git) do:

```
$ cd $REPO
$ git pull origin master stable
$ git checkout stable

# option a: resolve conflicts resulting from hotfixes
$ git merge master
$ ...

# option b: force stable to master
$ git update-ref refs/heads/stable master

$ git push # possibly with --force

# continue development
$ git checkout master
```

Log into taler.net with the account that is *not* active by looking at the *sockets* symlink of the *demo* account.

The following instructions wipe out the old deployment completely.

```
$ ls -l ~demo/sockets

[...] sockets -> /home/demo-green/sockets/
```

In this case, *demo-green* is the active deployment, and *demo-blue* should be updated. After the update is over, the */home/demo/sockets* symlink will be pointed to *demo-blue*.

```
# Remove all existing files
$ find $HOME -exec rm -fr {} \;

$ git clone /var/git/deployment.git
# Pick color depending on which one is inactive and being rebuilt.
$ ./deployment/bootstrap-bluegreen demo [blue|green]

# set environment appropriately
$ . activate
$ taler-deployment-build
```

```
# (re)generate configuration
$ taler-deployment-config-generate

# generate signatures
$ taler-deployment-config-sign

# upgrade the database!  this
# process depends on the specific version

# generate denomination keys: this is OPTIONAL,
# as the keys under ~/shared-data might be okay
# to use.
$ taler-deployment-keyup

$ taler-deployment-start

# look at the logs, verify that everything is okay
```

Now the symlink can be updated.

4 Releases

4.1 Release Process and Checklists

This document describes the process for releasing a new version of the various Taler components to the official GNU mirrors.

The following components are published on the GNU mirrors

- taler-exchange (exchange.git)
- taler-merchant (merchant.git)
- talerfrontends (merchant-frontends.git)
- taler-bank (bank.git)
- taler-wallet-webex (wallet-webex.git)

4.2 Tagging

Tag releases with an **annotated** commit, like

```
git tag -a v0.1.0 -m "Official release v0.1.0"
git push origin v0.1.0
```

4.3 Database for tests

For tests in the exchange and merchant to run, make sure that a database *talertest* is accessible by *$USER*. Otherwise tests involving the database logic are skipped.

4.4 Exchange, merchant

Set the version in `configure.ac`. The commit being tagged should be the change of the version.

For the exchange test cases to pass, `make install` must be run first. Without it, test cases will fail because plugins can't be located.

```
./bootstrap
./configure # add required options for your system
make dist
tar -xf taler-$COMPONENT-$VERSION.tar.gz
cd taler-$COMPONENT-$VERSION
make install check
```

4.5 Wallet WebExtension

The version of the wallet is in *manifest.json*. The `version_name` should be adjusted, and *version* should be increased independently on every upload to the WebStore.

```
./configure
make dist
```

4.6 Upload to GNU mirrors

See *https://www.gnu.org/prep/maintain/maintain.html#Automated-FTP-Uploads*

Directive file:

```
version: 1.2
directory: taler
filename: taler-exchange-0.1.0.tar.gz
```

Upload the files in **binary mode** to the ftp servers.