

Trac with FastCGI

The `FastCGI` interface allows Trac to remain resident much like with `mod_python` or `mod_wsgi`. It is faster than external CGI interfaces which must start a new process for each request. Additionally, it is supported by a much wider variety of web servers.

Note that unlike `mod_python`, FastCGI supports `Apache SuEXEC`, ie run with different permissions than the web server runs with. `mod_wsgi` supports the `WSGIDaemonProcess` with `user / group` parameters to achieve the same effect.

Note for Windows: Trac's FastCGI does not run under Windows, as Windows does not implement `Socket.fromfd`, which is used by `_fcgi.py`. If you want to connect to IIS, you may want to try `AJP/ISAPI`.

Apache configuration

There are two FastCGI modules commonly available for Apache: `mod_fastcgi` and `mod_fcgid` (preferred). The latter is more up-to-date.

The following sections focus on the FCGI specific setup, see also `TracModWSGI` for configuring the authentication in Apache.

Regardless of which `cgi` module is used, be sure the web server has executable permissions on the `cgi-bin` folder. While FastCGI will throw specific permissions errors, `mod_fcgid` will throw an ambiguous error if this has not been done: `Connection reset by peer: mod_fcgid: error reading data from FastCGI server.`

Set up with `mod_fastcgi`

`mod_fastcgi` uses `FastCgiIpcDir` and `FastCgiConfig` directives that should be added to an appropriate Apache configuration file:

```
fastcgi-script .fcgi

fastcgi_module
```

Setting `FastCgiIpcDir` is optional if the default is suitable. Note that the `LoadModule` line must be after the `IfModule` group.

Configure `ScriptAlias` or similar options as described in `TracCgi`, but calling `trac.fcgi` instead of `trac.cgi`.

Add the following to the Apache configuration file (below the `FastCgiIpcDir` line) if you intend to set up the `TRAC_ENV` as an overall default:

```
-initial-env TRAC_ENV=/path/to/env/trac
```

Alternatively, you can serve multiple Trac projects in a directory by adding this:

```
-initial-env TRAC_ENV_PARENT_DIR=/parent/dir/of/projects
```

Set up with `mod_fcgid`

Configure `ScriptAlias` (see [TracCgi](#) for details), but call `trac.fcgi` instead of `trac.cgi`:

Note the slash at the end.

To set up Trac environment for `mod_fcgid` it is necessary to use `DefaultInitEnv` directive. It cannot be used in `Directory` or `Location` context, so if you need to support multiple projects, try the alternative environment setup below:

```
TRAC_ENV
```

Alternative environment setup

A better method to specify the path to the Trac environment is to embed the path into `trac.fcgi` script itself. That doesn't require configuration of the server environment variables, works for both [?FastCgi](#) modules as well as for [?lighttpd](#) and CGI:

```
osenviron
```

or:

```
osenviron
```

With this method different projects can be supported by using different `.fcgi` scripts with different `ScriptAliases`.

See [?this fcgid example config](#) which uses a `ScriptAlias` directive with `trac.fcgi` with a trailing `/` like this:

```
/
```

Cherokee Configuration

Configuring [?Cherokee](#) with Trac is straightforward, if you spawn Trac as an SCGI process. You can either start it manually, or better yet, automatically by letting Cherokee spawn the server whenever it is down.

First set up an information source in `cherokee-admin` with a local interpreter:

```
Host:
localhost:4433

Interpreter:
/usr/bin/tracd ?single-env ?daemonize ?protocol=scgi ?hostname=localhost ?port=4433 /path/to/project/
```

If the port was not reachable, the interpreter command would be launched. Note that, in the definition of the information source, you will have to manually launch the spawner if you use a *Remote host* as *Information source* instead of a *Local interpreter*.

After doing this, we will just have to create a new rule managed by the SCGI handler to access Trac. It can be created in a new virtual server, `trac.example.net` for instance, and will only need two rules. The **default** one will use the SCGI handler associated to the previously created information source. The second rule will be there to serve the few static files needed to correctly display the Trac interface. Create it as *Directory rule* for `/common` and just set it to the *Static files* handler and with a *Document root* that points to the appropriate files: `$TRAC_LOCAL/htdocs/` (where `$TRAC_LOCAL` is a directory defined by the user or the system administrator to place local Trac resources).

Note: If the `tracd` process fails to start up, and Cherokee displays a 503 error page, you might be missing the `?python-flup` package ([?#9903](#)). Python-flup is a dependency which provides Trac with SCGI capability. You can install it on Debian based systems with:

```
sudo apt-get install python-flup
```

Lighttpd Configuration

The FastCGI front-end was developed primarily for use with alternative webservers, such as [?Lighttpd](#).

Lighttpd is a secure, fast, compliant and very flexible web-server that has been optimized for high-performance environments. It has a very low memory footprint compared to other web servers and takes care of CPU load.

For using `trac.fcgi` (prior to 0.11) / `fcgi_frontend.py` (0.11) with Lighttpd add the following to your `lighttpd.conf`:

```
fcgi_binary="/usr/bin/python /path/to/fcgi_frontend.py" # 0.11 if installed with easy_setup, it is inside the e
fcgi_binary="/path/to/cgi-bin/trac.fcgi" # 0.10 name of prior fcgi executable
fcgi.server = ("/trac" =>

    ("trac" =>
        ("socket" => "/tmp/trac-fastcgi.sock",
         "bin-path" => fcgi_binary,
         "check-local" => "disable",
         "bin-environment" =>
             ("TRAC_ENV" => "/path/to/projenv")
        )
    )
)
```

Note that you will need to add a new entry to `fastcgi.server` for each separate Trac instance that you wish to run. Alternatively, you may use the `TRAC_ENV_PARENT_DIR` variable instead of `TRAC_ENV` as described above, and you may set one of the two in `trac.fcgi` instead of in `lighttpd.conf` using `bin-environment`, as in the section above on Apache configuration.

Note that Lighttpd has a bug related to `'SCRIPT_NAME'` and `'PATH_INFO'` when the uri of `fastcgi.server` is `'/'` instead of `'/trac'` in this example (see [?#2418](#)). This is fixed in Lighttpd 1.5, and under Lighttpd 1.4.23 or later the workaround is to add `"fix-root-scriptname" => "enable"` as a parameter of `fastcgi.server`.

For using two projects with `lighttpd` add the following to your `lighttpd.conf`:

```
fastcgi.server = ("/first" =>
    ("first" =>
        ("socket" => "/tmp/trac-fastcgi-first.sock",
         "bin-path" => fcgi_binary,
         "check-local" => "disable",
         "bin-environment" =>
             ("TRAC_ENV" => "/path/to/projenv-first")
        )
    ),
    "/second" =>
        ("second" =>
            ("socket" => "/tmp/trac-fastcgi-second.sock",
             "bin-path" => fcgi_binary,
             "check-local" => "disable",
             "bin-environment" =>
                 ("TRAC_ENV" => "/path/to/projenv-second")
            )
        )
    )
)
```

Note that the field values are different. If you prefer setting the environment variables in the `.fcgi` scripts, then copy/rename `trac.fcgi`, eg to `first.fcgi` and `second.fcgi`, and reference them in the above settings. Note that the above will result in different processes in any event, even if both are running from the same `trac.fcgi` script.

Note: The order in which the `server.modules` are loaded is very important: if `mod_auth` is not loaded **before** `mod_fastcgi`, then the server will fail to authenticate the user.

For authentication you should enable `mod_auth` in `lighttpd.conf` 'server.modules', select `auth.backend` and `auth.rules`:

```
server.modules          = (
...
    "mod_auth",
...
)

auth.backend            = "htpasswd"

# Separated password files for each project
# See "Conditional Configuration" in
# http://trac.lighttpd.net/trac/file/branches/lighttpd-merge-1.4.x/doc/configuration.txt

$HTTP["url"] =~ "^/first/" {
    auth.backend.htpasswd.userfile = "/path/to/projenv-first/htpasswd.htaccess"
}
$HTTP["url"] =~ "^/second/" {
    auth.backend.htpasswd.userfile = "/path/to/projenv-second/htpasswd.htaccess"
}

# Enable auth on trac URLs, see
# http://trac.lighttpd.net/trac/file/branches/lighttpd-merge-1.4.x/doc/authentication.txt

auth.require = ("/first/login" =>
    ("method" => "basic",
     "realm" => "First project",
     "require" => "valid-user"
```

```

    ),
    "/second/login" =>
    ("method" => "basic",
     "realm" => "Second project",
     "require" => "valid-user"
    )
  )
)

```

Note that Lighttpd (v1.4.3) stops if the password file doesn't exist.

Note that Lighttpd doesn't support 'valid-user' in versions prior to 1.3.16.

Conditional configuration is also useful for mapping static resources, ie serving out images and CSS directly instead of through FastCGI:

```

# Aliasing functionality is needed
server.modules += ("mod_alias")

# Set up an alias for the static resources
alias.url = ("/trac/chrome/common" => "/usr/share/trac/htdocs")

# Use negative lookahead, matching all requests that ask for any resource under /trac, EXCEPT in
# /trac/chrome/common, and use FastCGI for those
$http["url"] =~ "^/trac(?!/chrome/common)" {
# Even if you have other fastcgi.server declarations for applications other than Trac, do NOT use += here
fastcgi.server = ("/trac" =>
  ("trac" =>
    ("socket" => "/tmp/trac-fastcgi.sock",
     "bin-path" => fcgi_binary,
     "check-local" => "disable",
     "bin-environment" =>
      ("TRAC_ENV" => "/path/to/projenv")
    )
  )
)
}

```

The technique can be easily adapted for use with multiple projects by creating aliases for each of them, and wrapping the fastcgi.server declarations inside conditional configuration blocks.

Also there is another way to handle multiple projects and it uses TRAC_ENV_PARENT_DIR instead of TRAC_ENV as well as global authentication:

```

# This is for handling multiple projects
alias.url      = ( "/trac/" => "/path/to/trac/htdocs/" )

fastcgi.server += ("/projects" =>
  ("trac" =>
    (
      "socket" => "/tmp/trac.sock",
      "bin-path" => fcgi_binary,
      "check-local" => "disable",
      "bin-environment" =>
        ("TRAC_ENV_PARENT_DIR" => "/path/to/parent/dir/of/projects/" )
    )
  )
)

```

```
#And here starts the global auth configuration
auth.backend = "htpasswd"
auth.backend.htpasswd.userfile = "/path/to/unique/htpasswd/file/trac.htpasswd"
$HTTP["url"] =~ "^/projects/./login$" {
    auth.require = ("/" =>
        (
            "method" => "basic",
            "realm"   => "trac",
            "require" => "valid-user"
        )
    )
}
```

Changing date/time format also supported by lighttpd over environment variable LC_TIME:

```
fastcgi.server = ("/trac" =>
    ("trac" =>
        ("socket" => "/tmp/trac-fastcgi.sock",
            "bin-path" => fcgi_binary,
            "check-local" => "disable",
            "bin-environment" =>
                ("TRAC_ENV" => "/path/to/projenv",
                    "LC_TIME" => "ru_RU")
            )
        )
    )
)
```

For details about languages specification see [?TracFaq question 2.13](#).

Other important information like the [mapping static resources advices](#) are useful for non-fastcgi specific installation aspects.

Relaunch Lighttpd and browse to <http://yourhost.example.org/trac> to access Trac.

Note about running Lighttpd with reduced permissions: If nothing else helps and trac.fcgi doesn't start with Lighttpd settings `server.username = "www-data", server.groupname = "www-data"`, then in the `bin-environment` section set `PYTHON_EGG_CACHE` to the home directory of `www-data` or some other directory accessible to this account for writing.

LiteSpeed Configuration

The FastCGI front-end was developed primarily for use with alternative webservers, such as [?LiteSpeed](#).

LiteSpeed web server is an event-driven asynchronous Apache replacement designed from the ground-up to be secure, scalable, and operate with minimal resources. LiteSpeed can operate directly from an Apache config file and is targeted for business-critical environments.

1. Please make sure you have a working install of a Trac project. Test install with "tracd" first.
2. Create a Virtual Host for this setup. From now on we will refer to this vhost as TracVhost. For this tutorial we will be assuming that your Trac project will be accessible via:

```
http://yourdomain.com/trac/
```

3. Go "TracVhost ? External Apps" tab and create a new "External Application":

```
Name: MyTracFCGI
Address: uds://tmp/lshttpd/mytracfcgi.sock
Max Connections: 10
Environment: TRAC_ENV=/fullpathto/mytracproject/ <--- path to root folder of trac project
Initial Request Timeout (secs): 30
Retry Timeout (secs): 0
Persistent Connection Yes
Connection Keepalive Timeout: 30
Response Buffering: No
Auto Start: Yes
Command: /usr/share/trac/cgi-bin/trac.fcgi <--- path to trac.fcgi
Back Log: 50
Instances: 10
```

4. Optional: If you need to use htpasswd based authentication. Go to "TracVhost ? Security" tab and create a new security Realm:

```
DB Type: Password File
Realm Name: MyTracUserDB <--- any name you wish and referenced later
User DB Location: /fullpathto/htpasswd <--- path to your htpasswd file
```

If you don't have a htpasswd file or don't know how to create the entries within one, go to <http://sherylcarter.com/encrypt.php>, to generate the user:password combos.

5. Go to "PythonVhost ? Contexts" and create a new FCgi Context:

```
URI: /trac/ <--- URI path to bind to python fcgi app we created
Fast CGI App: [VHost Level] MyTracFCGI <--- select the Trac fcgi extapp we just created
Realm: TracUserDB <--- only if (4) is set. select realm created in (4)
```

6. Modify /fullpathto/mytracproject/conf/trac.ini:

```
#find/set base_rul, url, and link variables
base_url = http://yourdomain.com/trac/ <--- base url to generate correct links to
url = http://yourdomain.com/trac/ <--- link of project
link = http://yourdomain.com/trac/ <--- link of graphic logo
```

7. Restart LiteSpeed: `lswsctrl restart`, and access your new Trac project at

`http://yourdomain.com/trac/.`

Ngix Configuration

Ngix is able to communicate with FastCGI processes, but can not spawn them. So you need to start FastCGI server for Trac separately.

1. Ngix configuration with basic authentication handled by Ngix - confirmed to work on 0.6.32

```
10.9.8.7
```

2. Modified trac.fcgi:

```
sockaddr
osenviron

    dispatch_request

fcgiserv tracweb_fcgiWSGIServerdispatch_request
    bindAddress sockaddr umask
fcgiservrun

e

e

tb StringIOStringIO
tracebackprint_excfiletb
tbgetvalue
```

3. Reload nginx and launch trac.fcgi:

```
trac@trac.example ~ $ ./trac-standalone-fcgi.py
```


The above assumes that:

- There is a user named 'trac' for running Trac instances and keeping Trac environments in its home directory
- `/home/trac/instance` contains a Trac environment
- `/home/trac/htpasswd` contains authentication information
- `/home/trac/run` is owned by the same group the Nginx runs under
 - ◆ and if your system is Linux the `/home/trac/run` has setgid bit set (`chmod g+s run`)
 - ◆ and patch from [?#7239](#) is applied, or you'll have to fix the socket file permissions every time

Unfortunately Nginx does not support variable expansion in `fastcgi_pass` directive. Thus it is not possible to serve multiple Trac instances from one server block.

If you worry enough about security, run Trac instances under separate users.

Another way to run Trac as a FCGI external application is offered in [?#6224](#).

See also: [TracGuide](#), [TracInstall](#), [ModWSGI](#), [CGI](#), [ModPython](#), [?TracNginxRecipe](#)