Technical Manual of the `Universal Webserver`

Organisation: Copyright (C) 2019-2025 Olivier Boudeville

Contact: about (dash) universal-webserver (at) esperide (dot) com

Creation date: Saturday, May 2, 2020

Lastly updated: Saturday, April 19, 2025

Version: 1.1.6

Status: Stable

Dedication: Users and maintainers of the `Universal Webserver`.

Abstract: The Universal Webserver, part of the Universal Server umbrella project, provides a multi-domain, multi-virtualhost webserver integrating various web-related services.

We present here a short overview of these services, to introduce them to newcomers.

The next level of information is either to browse the US-Web API documentation or simply to read the corresponding source files, which are intensely commented and generally straightforward.

Table of Contents

Overview

We present here a short overview of the services offered by the *Universal Webserver*, to introduce them to newcomers.

The goal of **US-Web** is to provide an integrated, extensible web framework in order:

- to better operate websites based on virtual hosting, so that a networked computer can serve as many websites corresponding to as many domains as wanted; this involves reading and interpreting vhost and other configuration information, handling properly 404 errors, producing access logs that are adequate for web analytics, rotating all logs, using/generating/renewing automatically SSL certificates, etc.
- to link to the Universal Server optionally (i.e. if available, knowing both should be able to run in the absence of the other), in order to offer a web front-end for it

Beyond this document, the next level of information about US-Web is to read the corresponding source files, which are intensely commented and generally straightforward.

The project repository is located here.

Easy Testing of US-Web

Provided that no server already runs at TCP port #8080, just downloading the get-us-web-from-sources.sh script and running it with no specific parameter (e.g. `sh get-us-web-from-sources.sh`, as a normal user) should suffice.

This should result in US-Web being cloned and built, and a test server (if requested, see the `--help` script option for usage) should be configured and run. It should be then available at http://localhost:8080.

Another way of testing, this time from a US-Web Git compiled with its prerequisites, is simply to execute `make debug` from its root.

See also the Server Deployment section for, beyond a mere testing of US-Web, a setting up of it in a production context.

Layer Stack

From the highest level to the lowest, as summarised here, a (free software) stack involving the Universal Webserver usually comprises:

- the *Universal Webserver* services themselves (i.e. this US-Web layer)
- Cowboy (for a small, fast and modern web framework)
- [optional] LEEC (for the management of Let's Encrypt certificates)
- [optional] Awstats (for the analysis of access log files)
- US-Common (for US base facilities)
- Ceylan-Traces (for advanced runtime traces)

- Ceylan-WOOPER (for OOP)
- Ceylan-Myriad (as a general-purpose Erlang toolbox)
- Erlang/OTP (for the compiler and runtime)
- GNU/Linux (for a suitable, reliable operating system of course)

The shorthand for the `Universal Webserver` (a.k.a. `US-Web`) is `uw`.

Server Deployment

> **Note**
>
> Please ensure that no prior instance of US-Web is already running, otherwise the new one will not be able to auto-launch, with an error message like: `Protocol 'inet_tcp': the name us_web@YOUR_SERVER seems to be in use by another Erlang node`. Refer to the Troubleshooting section for further guidance.

In all cases, if opting for the generation of certificates, in order to avoid hitting, in case of problem, any rate limit enforced by the ACME servers, we recommend to perform first a "dry run" by setting in your US-Web configuration file `{certificate_mode, development}` so that the staging ACME infrastructure is targeted first, leading only to temporary certificates.

If the whole deployment procedure went smoothly, then only switch back to `{certificate_mode, production}` and restart US-Web, in order to rely on the actual ACME servers and thus obtain the intended final certificates.

Native Deployment (recommended)

This procedure relies on our *native* build/run system, which is the only one that is officially supported in the context of US-Web, and involves only two steps.

First step is to:

- create a `/etc/xdg/universal-server` directory in which a relevant US configuration file (`us.config`) is added, referencing a suitable US-Web configuration file (e.g. `foobar-us-web.config`) located in the same directory; refer to the Configuring the Universal-Webserver section
- transfer to the target server our deploy-us-web-native-build.sh script (in a US-Web clone, it is located in `priv/bin`), possibly simply at the root of one's normal user
- ensure that no prior webserver is running at the target port(s) (otherwise specify the `--no-launch` option below)

After deployment, the installation itself will be located by default in the `/opt/universal-server/us_web-native` directory.

Step #2 exists in two versions, either based on a direct launching or, preferably, on one managed by Systemd.

Direct, Ad Hoc Deployment

This **second step** boils down to executing (as a normal user - the sudo permissions will be requested only whenever necessary) our deploy-us-web-native-build.sh script, with no specific argument.

Deployment for Systemd Integration (recommended)

If relying on `systemctl` (hence on our `us-web-as-native-build.service` file), the following procedure shall be preferred for this **second step**:

```
# Get ready first (as a regular user; sudo permissions to be
# requested only whenever necessary):
#
$ ./deploy-us-web-native-build.sh --no-launch

# Useful, as Systemd shall detect the update of the US-Web service file:
$ sudo systemctl daemon-reload

# Then stop properly any prior US-Web instance:
$ sudo systemctl stop us-web-as-native-build.service

# If EPMD still thinks a previous US-Web instance is running:
# (unlikely to be needed, and could impact any other running
# Erlang program, so be cautious...)
#
#$ sudo killall epmd

# Relaunch US-Web, and initiate the certificate renewal process:
$ sudo systemctl start us-web-as-native-build.service
```

Then, as a first check of that launch, one may execute:

```
$ sudo systemctl status us-web-as-native-build.service
```

See the next section for more in-depth checking and monitoring.

Checking Launch Outcome

If having requested the renewal of SSL certificates (the `certificate_support` entry of the US-Web configuration file being set to `renew_certificates`), incoming HTTP requests will be directly served yet will be redirected to HTTPS ones that will complete successfully *only* once proper certificates will have been obtained (hence after a few dozens of seconds after server startup).

These certificates can be checked (e.g. in terms of timestamps) in the `certificates` subdirectory of the one designated by the `us_web_data_dir` entry (typically in `/opt/universal-server/us_web-data/certificates`).

Finally, from a client host, one may also check the availability of the target webserver(s); refer to the Remote Monitoring section for that.

Deployment based on rebar3

Note

With rebar3 we encountered *a lot* of difficulties regarding build and release. So, at least for the moment, we dropped the use of rebar3 and focused instead on our native build/run system, which is perfectly suitable and fully sufficient. We do not plan to restore the rebar3 build anymore (contributions are welcome though - but be aware that the dependency management is bound to be tricky).

If wanting nevertheless to rely on rebar3, the `prod` profile defined in the context of `rebar3` shall be used.

Currently we prefer re-using the (supposedly already installed) local Erlang environment on the server (to be shared across multiple services), so by default ERTS is *not* included in a US-Web release.

Sources are not included either, as we prefer re-rolling a release to editing and compiling code directly on a server.

To generate from scratch such a (mostly) standalone release, one may use:

```
$ make release-prod
```

It should generate a tarball such as `us_web-x.y.z.tar.gz`

The `export-release` make target allows in the same movement to lightly update a pre-existing release and also to transfer it to any target server, designated by setting the `WEB_SRV` (make or environment) variable to the FQDN of this server.

So we recommend running:

```
$ make export-release
 Generating rebar3 us_web.app file
 Compiling us_web from XXX/us_web
 ===> Verifying dependencies...
 ===> Compiling myriad
 Hiding unsupported sources
 Populating the include symlinks
 [...]
```

We recommend installing a release in `REL_BASE_ROOT=/opt/universal-server`:

```
$ mv /tmp/us_web-x.y.z.tar.gz ${REL_BASE_ROOT}
$ cd ${REL_BASE_ROOT}
$ tar xvf us_web-x.y.z.tar.gz
```

Then various steps are necessary in order to have a functional release running satisfactorily.

We automated the full deployment process of US-Web on a server for that: once the release has been transferred to that server (possibly thanks to the aforementioned `export-release` target, possibly to the `/tmp` directory of that server), one may rely on our deploy-us-web-release.sh script. One may also just take inspiration from it in order to devise one's deployment scheme.

Let's say from now on that the UNIX name chosen for the US user is `us-user`, the one of the US-web user is `us-web-user` and the US group (containing both users, and possibly only them) is `us-group` (one should keep in mind that `US-Web` is a specialization of the `US` framework).

Using that script boils down to running, as root:

```
$ /tmp/deploy-us-web-release.sh
Detected US-Web version: 'x.y.z'.
Trying to stop gracefully any prior release in us_web-x.y.z.
Removing already-existing us_web-x.y.z.
US-Web release ready in '/opt/universal-server/us_web-x.y.z/bin/us_web'.
```

```
Changing, from '/opt/universal-server/us_web-x.y.z', the owner of release
files to 'us-web-user' and their group to 'us-group'.
(no auto-launch enabled)
```

Note that the goal of that deployment phase is to start from a clean state, and as such it will try to stop any already running US-Web instance (for all possible versions thereof).

Then US-Web is fully *deployed*. Once properly *configured*, it will be able to be *launched* for good.

Some related information are specified below.

Configuring the Universal-Webserver

As explained in start-us-web.sh and in class_USWebConfigServer.erl, the US configuration files will be searched through various locations.

The main, overall US configuration file, `us.config`, is found based on a series of directories that are gone through in an orderly manner; the first directory to host such a file becomes *the* (single) US configuration directory.

The other US-related configuration files (e.g. any `us-web.config`) are referenced directly from the main one (`us.config`), designated there through specific keys (e.g. `us_web_config_filename`); they may be either absolutely defined, or relatively to the aforementioned US configuration directory.

Let's name `US_CFG_ROOT` the actual directory in which they all lie; it is typically either `~/.config/universal-server/` (in development mode), or `/etc/xdg/universal-server/` (in production mode).

Note that, as these files might contain sensitive information (e.g. Erlang cookies), they shall be duly protected. Indeed, in terms of permissions, we should have `640`, supposing that the `us.config` designates, in its `us_web_config_filename` entry, `foobar-us-web-for-production.config` as the name of the US-Web configuration file[1]:

```
-rw-r----- 1 us-user     us-group [...] us.config
-rw-r----- 1 us-web-user us-group [...] foobar-us-web-for-production.config
```

Finally, if being able to rely on multiple, different US configuration directories is necessary in order to be able to launch multiple US-Web instances (these configurations can then register the managers involved - typically the US-Web Configuration server - under different names), only a single node can be used per EPMD instance as, by default, such a node is named `us_web` (two US-Web instances would then clash at this level).

So, in order to run multiple US-Web instances (e.g. one in production, one for testing) on a given host, they shall be multiplexed on different EPMD ports (see the `epmd_port` key of the US and US-Web configuration files; as a result, different `universal-server` configuration directories shall then be defined and used).

[1] They shall be in the same `US_CFG_ROOT` directory (discussed below), and may be symbolic links.

In a Development Setting

The US main configuration file, us.config, is in a directory (US_CFG_ROOT) that is ~/.config/universal-server/ here. This US-level configuration file will reference (through its us_web_config_filename entry) a US-Web counterpart configuration file, probably in the same directory.

The US-Web configuration file may define a us_web_app_base_dir entry. If not, this application directory will then be found thanks to the US_WEB_APP_BASE_DIR environment variable (if defined, typically through one's ~/.bashrc); otherwise, as a last resort, an attempt to guess it will be done.

The US webserver may be then run thanks to make debug, from the relevant us_web directory (typically the root of a Git clone located in the user's directory).

In such a development context, in us_web/conf/sys.config, we recommend to leave the batch mode disabled (just let the default {is_batch,false}), so that a direct, graphical trace supervision is enabled (provided that a relevant trace supervisor is available, see Ceylan-Traces for that).

In a Production Setting

The start/stop management scripts will be run initially as root (possibly through systemd) and must access the us.config file. Then, once run, us_web will most probably switch to a dedicated user (see the us_web_username entry in the US-Web configuration file), who will need in turn to be able to read the us.config file and any related one (e.g. for US-Web, here supposed to be named foobar-us-web-for-production.config).

As a result, a relevant configuration directory (denoted US_CFG_ROOT in this document), in that shared setting, is the standard /etc/xdg one, resulting in the /etc/xdg/universal-server directory to be used.

As mentioned, care must be taken so that root and also the US and US-Web users can read the content of that directory - at least the US and US-Web configuration files in it - and that the other users cannot.

For that, a dedicated us-group group can be created, and any web user (e.g. us-web-user) shall belong to that group. For example:

```
$ id us-web-user
uid=1002(us-web-user) gid=1002(us-web-user) groups=1002(us-web-user),
 1007(us-group)
```

Then, in /etc/xdg/universal-server, for the US and US-Web configuration files:

```
$ chown us-user us.config
$ chown us-web-user foobar-us-web-for-production.config

$ us_files="us.config foobar-us-web-for-production.config"
$ chgrp us-group ${us_files}
$ chmod 640 ${us_files}

$ chgrp us-group /etc/xdg/universal-server
$ chmod 700 /etc/xdg/universal-server
```

We recommend directly setting the `us_web_app_base_dir` configuration entry to the relevant, absolute path.

Let's name here `US_WEB_REL_ROOT` the root of the US-Web release of interest (e.g. corresponding to `${REL_BASE_ROOT}/us_web-latest/`) and `US_WEB_APP_ROOT` the root of the corresponding US-Web application (e.g. corresponding to `${US_WEB_REL_ROOT}/lib/us_web-l`

A `systemd` service shall be declared for US-Web, in `/etc/systemd/system`; creating there, as root, a symbolic link to `${US_WEB_APP_ROOT}/priv/conf/us-web.service` will suffice.

This service requires `start-us-web.sh` and `stop-us-web.sh`. Adding for user convenience `get-us-web-status.sh`, they should all be symlinked that way, still as root:

```
$ cd /usr/local/bin
$ for f in start-us-web.sh stop-us-web.sh get-us-web-status.sh; \
  do ln -s ${US_WEB_APP_ROOT}/priv/bin/$f ; done
```

The log base directory (see the `log_base_directory` entry) shall be created and writable; for example:

```
$ LOG_DIR=/var/log/universal-server
$ mkdir -p ${LOG_DIR}
$ chown us-user ${LOG_DIR}
$ chgrp us-group ${LOG_DIR}
$ chmod 770 ${LOG_DIR}
```

In such a production context, in `sys.config` (typically located in `${US_WEB_REL_ROOT}/releases/latest` we recommend to enable batch mode (just set `{is_batch,true}`), so that by default no direct, graphical trace supervision is triggered (a server usually does not have a X server anyway).

Instead the traces may then be supervised and browsed remotely (at any time, and any number of times), from a graphical client (provided that a relevant trace supervisor is available locally, see Ceylan-Traces for that), by running the monitor-us-web.sh script.

For that the relevant settings (notably which server host shall be targeted, with what cookie) shall be stored in that client, in a `us-monitor.config` file that is typically located in the `~/.config/universal-server` directory.

Configuration Files

A `us.config` file *referencing* a suitable US-Web configuration file will be needed; most of the behaviour of the US-Web server is determined by this last configuration file.

As the US-related configuration files are heavily commented, proceeding based on examples in the simplest approach[2].

Refer for that at this (US) us.config example and at this US-Web counterpart example.

[2] The second best approach being to have directly a look at the code reading them, see class_USConfigServer.erl for us.config and class_USWebConfigServer.erl for its US-Web counterpart.

In a nutshell, this US-Web configuration files allows unsurprisingly to specify all web-related settings, regarding users, directory locations, options such as ports, log analysis or certificate management and, most importantly, the routes.

Defining routes allows to tell what it the web root directory[3] to serve for each given client-specified URL (in general, only the hostname part is of interest here), based on pattern-matching.

Such a route definition consists mostly on enumerating each hostname that is to manage (e.g. `"foobar.org"` - knowing that the `default_domain_catch_all` atom designates all hostnames that could not be matched specifically otherwise) and enumerating then all the virtual hosts to manage for this domain (e.g. the `"hurricane"` host, whose FQDN is thus `hurricane.foobar.org` - knowing that the `default_vhost_catch_all` atom designates all hostnames that could not be matched specifically otherwise, in the form `*.foobar.org`, and that the `without_vhost` atom designates the hostname itself, i.e. `foobar.org`).

For example:

```
{ default_web_root, "/var/web-storage/www" }.

%{ log_analysis, awstats }.

{ certificate_support, use_existing_certificates }.
{ certificate_mode, production }.

% Use 'dns-01' if wanting wildcard certificates:
{ challenge_type, 'http-01' }.

% With 'dns-01', the DNS provider must be specified:
{ dns_provider, ovh }.

{ routes, [

  % Note that the first hostname listed will be, regarding
  % https (SNI), the main, default one:

  { "foobar.org", [

      % So any content requested regarding the www.foobar.org
      % FQDN (e.g. http://www.foobar.org/index.html) will be
      % looked-up relatively to the
      % /var/web-storage/www/Foobar-main directory:
      %
      { "www", "Foobar-main" },
      { "club", "Foobar-club" },
      { "archives", "/var/web-storage/Archives-for-Foobar" },

      % If just the domain is specified:
      { without_vhost, "Foobar" }
```

[3] A web root directory is either specified as an absolute path, otherwise it is relative to a `default_web_root` entry to be specified in the US-Web configuration file.

```
      % With https, catch-all shall be best avoided:
      %{ default_vhost_catch_all, "Foobar" }

                  ] },

  { "foobar.net", [
      % (similar rules may be defined)
                  ] }

  % Best avoided as well with https:
  %{ default_domain_catch_all, [
  %
  %           { default_vhost_catch_all, "Foobar" }
  %
  %                ] }

        ]

}.
```

US-Web Application Base Directory

This directory allows US-Web to select the various runtime elements it is to use.

This **US-Web Application Base Directory** may be set in the US-Web configuration file thanks to its (optional) `us_web_app_base_dir` entry.

If not set there, US-Web will try to use the `US_WEB_APP_BASE_DIR` environment variable instead.

If not usable either, US-Web will try to determine this directory automatically, based on how the server is launched (with our native deployment approach or as an OTP release).

Of course failing to resolve a proper application base directory will result in a fatal launch error to be reported at start-up.

Let's designate this directory from now on with the `US_WEB_APP_BASE_DIR` pseudo-variable.

Separating the US-Web Software from its data

It is useful to regularly upgrade the US-Web code while maintaining the data it uses as it is (not losing it because of a mere software update).

This notably includes the information such as the state of the log analysis tool (i.e. the synthesis aggregated after having processed a potentially longer history of access logs when rotating them) or the currently generated certificates (which are not always easy to renew frequently, if having to respect rate limits)[4].

These various states are stored in the **US-Web data directory**, which can be set, by decreasing order of priority:

[4] Access/error logs and the own US-Web traces are not considered belonging to the application data; their storage location is discussed in the Webserver Logs section.

- in the us_web_data_dir entry of the US-Web configuration file, as an absolute directory, or as one that is relative to the US-Web application base directory (see US-Web Application Base Directory), then as US_WEB_APP_BASE_DIR/data

- otherwise in the US_WEB_DATA_DIR environment variable

If the default path for this location is us-web-data (relatively to the US-Web instance base directory, typically deployed in /opt/universal-server/us_web-native), a recommended practice is nevertheless to set this location explicitly *outside* of that instance so that the instance can be upgraded/replaced without any loss of state.

So the us_web_data_dir entry might be set in the US-Web configuration file for example to /opt/universal-server/us_web-data. Of course, as always, proper permissions shall be set in this data tree as well.

Let's designate this directory from now on with the US_WEB_DATA_DIR pseudo-variable.

This directory may notably gather following elements:

- the log-analysis-state subdirectory (if log analysis is enabled), containing the working state of the tool for web access analysis, typically Awstat, thus containing awstats*.baz.foobar.org.txt files for the preprocessing of its reports

- the certificates subdirectory (if certificate support is enabled), containing the related keys and certificates for the domains of interest:

  - if managing domains D in ["foobar.org", "example.com"], then D.csr, D.key and D.crt will be stored or produced there (refer to this section of the documentation of LEEC for further explanations)
  - LEEC-related transverse security elements may be found there (if using it to renew certificates) as well, notably (see this LEEC documentation section):

    * us-web-leec-agent-private.key, the RSA private key of the US-Web LEEC agent so that it can safely authenticate to the ACME servers it is interacting with and afterwards validate https handshakes with clients
    * lets-encrypt-r3-cross-signed.pem, the certificate associated to the *Certificate Authority* (Let's Encrypt here) that is relied upon

  - dh-params.pem, a key generated by LEEC to allow for safer *Ephemeral Diffie-Helman key exchanges*

One may create from this certificates subdirectory a GNUmakefile symlink pointing to GNUmakefile-for-certificates to easy any checking or backup of these certificates.

Catch-alls, HTTPS, Challenge Types and Wildcard Certificates

Having wildcard certificates (which is the default now) - thus relying on the dns-01 challenge - allows in turn to define catch-alls (host-level and/or domain-level) routes and wildcard DNS entries (such as * IN CNAME foobar.org.).

Conversely, if setting another challenge type (refer to the challenge_type configuration entry, typically then set to the http-01 challenge), defining catch-all routes is not recommended if using https, as these unanticipated hostnames are by design absent from the corresponding certificates, and thus any user requesting them (possibly due to a typo) will have their browser report a security risk (such as SSL_ERROR_BAD_CERT_DOMAIN).

Similarly, wildcard DNS entries should be then avoided (on the other hand, they allow not to advertise what are the precise virtual hostnames supported; note though that with https, these virtual hostnames will be visible in the SANs).

Other Configuration Files

Credentials for One's DNS Provider If wanting wildcard certificates, the dns-01 challenge will be used, and this will require credentials to be specified in order to update DNS entries from one's DNS provider.

US-Web expect credential files to be available in the base directory where all US configuration is to be found (typically /etc/xdg/universal-server). Note that such credentials files (e.g. leec-ovh-credentials-for-foobar.org.txt) must be well-protected; refer to this LEEC section for more information.

Operating System Settings

Regarding authbind

Many distributions will parameter authbind so that only the TCP port 80 will be available to "authbound" programs.

If wanting to run an HTTPS server, the TCP port 443 will be most probably needed and thus must be specifically enabled.

For that, relying on the same user/group conventions as before, one may enter, as root:

```
$ cd /etc/authbind/byport
$ cp 80 443
$ chown us-web-user:us-group 443
```

Otherwise an exception will be raised (about eperm / not owner) when US-Web will try to create its HTTPS listening socket.

Running the Universal-Webserver

In this section we suppose that a native deployment is being used, with uniform conventions between the development and server settings; if using a release-based deployment, note that the Erlang versions used to produce the release (typically in a development computer) and run it (typically in a production server) must match (we prefer using *exactly* the same version).

Supposing a vhost to be served by US-Web is `baz.foobar.org`, to avoid being confused by your browser, a better way is to test whether a US-Web instance is already running thanks to `wget` or `links`:

```
$ wget http://baz.foobar.org -O -
```

This will display the fetched content directly on the console (not creating a file).

Indeed, when testing a website, fully-fledged browsers such as Firefox may be quite misleading as they may attempt to hide issues, and tend to cache a lot of information (not actually reloading the current page), even if the user held Shift and clicked on "Reload". Do not trust browsers!

One may disable temporarily the cache by opening the developer toolbox (`Ctrl+Shift+I` or `Cmd+Opt+I` on Mac), clicking on the settings button (near the top right), scrolling down to the `Advanced settings` (on the bottom right), checking the option `Disable Cache (when toolbox is open)` and refreshing the page. `wget` may still be a better, simpler, more reliable solution.

Stopping any prior instance first

From now on, we will suppose the current directory is `US_WEB_APP_ROOT`.

The `stop-us-web-native-build.sh` script can be used for stopping a US-Web instance, typically simply as:

```
$ priv/bin/stop-us-web-native-build.sh
```

In development mode, still from the root of US-Web, one might use `make stop-brutal` to operate regardless of dynamically-changed cookies, while in a production setting one would go preferably for:

```
$ systemctl stop us-web-as-native-build.service
```

Launching the US-Web Server

In development mode, from the root of US-Web, one may use `make debug`, while, in production mode, the US-Web server can be launched either with its dedicated script `start-us-web-native-build.sh` or, even better, directly through:

```
$ systemctl start us-web-as-native-build.service
```

Monitoring the US-Web Server

Local Monitoring

Here operations will be done directly on the server, and supposing a native build.

Checking EPMD

If using the default US-Web EPMD port, checking whether an instance is running is as simple as:

```
$ export ERL_EPMD_PORT=4508; epmd -names
epmd: up and running on port 4508 with data:
name us_web at port 50002
```

Then executing `kill-us-web.sh` will kill any live US-Web instance and unregister it from its EPMD (without killing any EPMD daemon).

Overall Local Inquiry

The `get-us-web-native-build-status.sh` script (still in `priv/bin`, as all US-Web shell scripts) may then be used to investigate any problem in a streamlined, integrated way.

Alternate (yet often less convenient) solutions are to run `systemctl status us-web-as-native-build.service` or, often with better results, `journalctl -xe --unit us-web-as-native-build.service --no-pager` to get some insights.

General Logs

A deeper level of detail can be obtained by inspecting the *general* logs (as opposed to the *webserver* ones, discussed in next section), which regroup the VM-level, technical ones and/or the applicative ones.

VM logs are written in the `${REL_BASE_ROOT}/log` directory (e.g. `/opt/universal-server/us_web-lat` which is created when the release is started first. Generally, `run_erl.log` (if launched as a daemon) will not help much, whereas the latest Erlang log file (`ls -lrt erlang.log.*`) is likely to contain relevant information.

So one may consult them for example with:

```
$ tail -n 50 -f /opt/universal-server/us_web-latest/us_web/log/erlang.log.1
```

As for our higher-level, **applicative traces**, they are stored in the the `us_web.traces` file, in the directory defined in the `us_web_log_dir` entry of the US-Web configuration file. This last file is specified in turn in the relevant `us.config` configuration file (see the `us_web_config_filename` key for that), which, in development mode, is itself typically found in `~/.config/universal-server` while, in production mode, is likely located in `/etc/xdg/universal-server`.

In practice, this trace file is generally found:

- in development mode, if testing, in `priv/for-testing/log`, relatively to the base directory specified in `us_app_base_dir`

- in production mode, generally in `/var/log/universal-server` or `/opt/universal-server`

This path is updated at runtime once the US-Web configuration file is read; so typically it is renamed from `/opt/universal-server/us_web-x.y.z/traces_via_otp.traces` to `/var/log/universal-server/us-web/us_web.traces`. If available, it is surely the most complete source of information.

Webserver Logs

These logs, which are maybe less useful for troubleshooting, designate access and error logs, per virtual host (e.g. for a virtual host `VH`, `access-for-VH.log` and `error-for-VH.log`). Their previous versions are archived in timestamped, compressed files (e.g. `error-for-VH.Y-M-D-at-H-M-S.xz`).

These files will be written in the directory designated by the `us_web_log_dir` entry of the US-Web configuration file. Refer to the previous section to locate this directory (for example it may be `/var/log/universal-server/us-web`).

Remote Monitoring

Here the state of a US-Web instance will be inspected remotely, with no need for a shell connection to its server.

HTTP Client Check

First of all, is this US-Web instance available at all? Check with:

```
$ wget http://baz.foobar.org -O -
```

One may define one's dedicated convenience script for that, like a `check-baz-website-availability.sh` script whose content would be:

```
#!/bin/sh
target_url="http://baz.foobar.org"
echo "  Checking whether '${target_url}' is available:"
wget ${target_url} -O - | head
```

More elaborate scripts may be devised, testing various virtual hosts with various protocols (http/https), looping continously should a failure be detected, like in:

```
#!/bin/sh
test_url()
{
   target_url="$1"
   #echo "  Checking whether '${target_url}' is available:"
   if wget -q "${target_url}" -O - 1>/dev/null; then
       echo " ('${target_url}' is available)"
       return 0
   else
       echo "######### Problem: '${target_url}' is NOT available! #########" 1>&2
       return 1
   fi
}

# Testing base domains, some virtual hosts, and HTTP/HTTPS availability as
# well:
com_test_url="http://foobar.com"
# Testing also a virtual host:
com_vh_test_url="http://hello.foobar.com"
```

```
# As HTTPS:
com_vh_https_url="https://other.foobar.com"
com_urls="${com_test_url} ${com_vh_test_url} ${com_vh_https_url}"
# Testing also the other domains:
net_test_url=[...]
all_urls="${com_urls} ${net_urls} ${org_urls} ${info_urls} ${biz_urls} [...]"
all_good=1
while [ $all_good -eq 1 ]; do
   all_good=0
   for u in ${all_urls}; do
       if ! test_url $u; then
           all_good=1
       fi
   done
   echo
   if [ $all_good -eq 1 ]; then
       echo "At least an URL could not be reached, testing again..."
   fi
   sleep 2
done
echo "Good news, all tested domains are available from this local host!"
echo
```

Trace Listener Check

As for the applicative traces, they may be monitored remotely as well, thanks to the monitor-us-web.sh script.

This script operates based on a configuration file, `us-web-for-remote-access.config` (typically located in `~/.config/universal-server`), that allows to designate the US-Web instance to access for the client host, and the various settings for that.

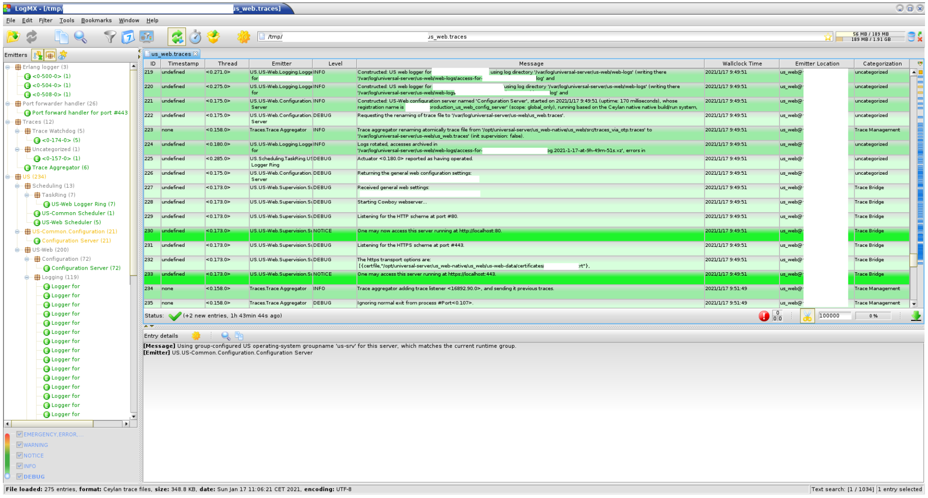
For example:

```
% This file is used by the US-Web framework.
%
% This is a configuration file to enable the interaction with a remote US-Web
% instance (e.g. a production server to be reached from a client host), typically
% to monitor traces, to trigger the generation of log access reports, etc.

% The hostname where the US-Web instance of interest runs:
{us_web_hostname, "baz.foobar.org"}.

% Its cookie, as set in its configuration file:
{remote_vm_cookie, 'foobar-8800a3f8-7545-4a83-c925-ca115a7b014b'}.

% If specifically overridden for thay server:
{us_web_config_server_registration_name, foobar_prod_us_web_cfg_srv}.

% A range of TCP ports that may be used for out-of-band inter-VM communication
% (not using the Erlang carrier; e.g. for send_file):
```

```
%
% (this range must be typically in-line with any one set at the level of
% firewalls)
%
{tcp_port_range, {50000, 55000}}.
```

Then running the aforementioned `monitor-us-web.sh` script should launch a trace browser synchronised to the trace aggregator of the targeted US-Web instance, displaying all traces (i.e. both all that have been emitted since the start of that instance and all newer ones, being then displayed in direct).

Here is a user-submitted screenshot (edited for confidentiality) of the LogMX-based trace browser corresponding to the native US-Web trace interface (refer to Ceylan-Traces for further details):



ID	Timestamp	Thread	Emitter	Level	Message	Wallclock Time	Emitter Location	Categorization
219	undefined	<0.271.0>	US.US-Web.Logging.Logge for	INFO	Constructed: US web logger for using log directory '/var/log/universal-server/us-web/web-logs' (writing there '/var/log/universal-server/us-web/web-logs/access-for- log' and	2021/1/17 9:49:51	us_web@	uncategorized
220	undefined	<0.275.0>	US.US-Web.Logging.Logge for	INFO	Constructed: US web logger for using log directory '/var/log/universal-server/us-web/web-logs' (writing there '/var/log/universal-server/us-web/web-logs log' and	2021/1/17 9:49:51	us_web@	uncategorized
221	undefined	<0.175.0>	US.US-Web.Configuration. Server	INFO	Constructed: US-Web configuration server named 'Configuration Server', started on 2021/1/17 9:49:51 (uptime: 170 milliseconds), whose registration name is production_us_web_config_server' (scope: global_only), running based on the Ceylan native native build/run system,	2021/1/17 9:49:51	us_web@	uncategorized
222	undefined	<0.175.0>	US.US-Web.Configuration. Server	DEBUG	Requesting the renaming of trace file to '/var/log/universal-server/us-web/us_web.traces'.	2021/1/17 9:49:51	us_web@	uncategorized
223	none	<0.158.0>	Traces.Trace Aggregator	INFO	Trace aggregator renaming atomically trace file from '/opt/universal-server/us_web-native/us_web/src/traces_via_otp.traces' to '/var/log/universal-server/us-web/us_web.traces' (init supervision: false).	2021/1/17 9:49:51	us_web@	Trace Management
224	undefined	<0.180.0>	US.US-Web.Logging.Logge for	INFO	Logs rotated, accesses archived in '/var/log/universal-server/us-web/web-logs/access-for- og.2021-1-17-at-9h-49m-51s.xz', errors in	2021/1/17 9:49:51	us_web@	uncategorized
225	undefined	<0.285.0>	US.Scheduling.TaskRing.US Logger Ring	DEBUG	Actuator <0.180.0> reported as having operated.	2021/1/17 9:49:51	us_web@	uncategorized
226	undefined	<0.175.0>	US.US-Web.Configuration. Server	DEBUG	Returning the general web configuration settings:	2021/1/17 9:49:51	us_web@	uncategorized
227	undefined	<0.173.0>	US.US-Web.Supervision.Su	DEBUG	Received general web settings:	2021/1/17 9:49:51	us_web@	Trace Bridge
228	undefined	<0.173.0>	US.US-Web.Supervision.Su	DEBUG	Starting Cowboy webserver...	2021/1/17 9:49:51	us_web@	Trace Bridge
229	undefined	<0.173.0>	US.US-Web.Supervision.Su	DEBUG	Listening for the HTTP scheme at port #80.	2021/1/17 9:49:51	us_web@	Trace Bridge
230	undefined	<0.173.0>	US.US-Web.Supervision.Su	NOTICE	One may now access this server running at http://localhost:80.	2021/1/17 9:49:51	us_web@	Trace Bridge
231	undefined	<0.173.0>	US.US-Web.Supervision.Su	DEBUG	Listening for the HTTPS scheme at port #443.	2021/1/17 9:49:51	us_web@	Trace Bridge
232	undefined	<0.173.0>	US.US-Web.Supervision.Su	DEBUG	The https transport options are: [{certfile,"/opt/universal-server/us_web-native/us_web/us-web-data/certificates/ rt"},	2021/1/17 9:49:51	us_web@	Trace Bridge
233	undefined	<0.173.0>	US.US-Web.Supervision.Su	NOTICE	One may access this server running at https://localhost:443.	2021/1/17 9:49:51	us_web@	Trace Bridge
234	none	<0.158.0>	Traces.Trace Aggregator	INFO	Trace aggregator adding trace listener <16892.90.0>, and sending it previous traces.	2021/1/17 9:51:49	us_web@	Trace Management
235	none	<0.158.0>	Traces.Trace Aggregator	DEBUG	Ignoring normal exit from process #Port<0.107>.	2021/1/17 9:51:49	us_web@	Trace Management

Status: ✓ (+2 new entries, 1h 43min 44s ago)

[Message] Using group-configured US operating-system groupname 'us-srv' for this server, which matches the current runtime group.
[Emitter] US.US-Common.Configuration.Configuration Server

File loaded: 275 entries, **format:** Ceylan trace files, **size:** 348.8 KB, **date:** Sun Jan 17 11:06:21 CET 2021, **encoding:** UTF-8 | Text search: [1 / 1034] | 1 entry selected

Remote Action

It is possible to trigger the generation of web access reports (refer to the Auto-generated Meta Website section for details) remotely, either for a single virtual host of a given domain, or for all virtual hosts of all the hosted domains.

For that one should execute the generate-us-web-log-report.sh script, which relies on the same `us-web-for-remote-access.config` configuration file already discussed in the Remote Monitoring section.

Nitrogen Support

A US-Web instance may host one[5] Nitrogen-based website.

[5] Only a single instance of Nitrogen is supported per US-Web instance, as otherwise the various pages of the Nitrogen websites would clash through the code path (e.g. there would be multiple `index.beam` files there; there would also be problems with the relative paths currently expected by `nitro_cache`, etc.).

So, if wanting to have multiple Nitrogen websites available from a common domain name, one shall rely on a reverse proxy. It is hardly a limitation, as web applications are better managed independently anyway, in order to be able to freely start / stop / update them, to isolate them with different users, to separate their resource uses, etc.

Deployment

Deployment of a Nitrogen-enabled US-Web

US-Web must be deployed accordingly, typically with the `--support-nitrogen` option of the `deploy-us-web-native-build.sh` script[6].

Deployment of the Nitrogen-based website / application

Then of course a Nitrogen-based website must have already been developed, an example of which being, in US-Main, the `us_main_nitrogen_app` web application.

Such a website is typically obtained by cloning the official Nitrogen repository (or possibly our fork thereof) and running:

```
$ make slim_cowboy PREFIX=$MY_NITROGEN_APP_ROOT PROJECT=my_nitrogen_app
```

(we prefer relying on a separate system-wide Erlang installation rather than on an full OTP release)

The root tree of interest then lies in `$MY_NITROGEN_APP_ROOT/my_nitrogen_app/site`.

The whole of it may be put in VCS, or at the very least its `include`, `src`, `static/{css,images}`, `templates` subtrees.

Configuration

Here we must declare to US-Web, through its configuration file (say, `my-us-web.config`, as listed under the `us_web_config_filename` key of one's `us.config` file), that, for a given domain name (say, `foo.org`), a given virtual host (say, `bar`, thus designated as `bar.foo.org`) is to be served by a Nitrogen instance whose content root is, relatively to the US-Web default web root, `./my-nitrogen-website`.

This is done simply thanks to a user-defined route, like in:

```
{ routes, [

 % These are implicitly static and relative to default_web_root:
 { "foo.org", [ { "project-a", "test-static-website-A" },
                { "project-b", "test-static-website-B" },
                { "project-c", "test-static-website-C" },
                { "bar",       "my-nitrogen-website", nitrogen },
                { default_vhost_catch_all, "test-static-website-D" } ] },

 [...] ] }
```

Nitrogen-related Information

Official Nitrogen sources:

- website

[6] This script relies on a few forks that we made for better control, namely of nitrogen_core and simple_bridge (the latter being installed by the former).

- insightful tutorial
- wiki
- Git repository
- the associated Google group
- the Build It With Nitrogen book / ebook

Extra information:

- "*It is strongly recommended to catch static files with the static_paths setting. simple_bridge does not serve large static files in an optimal way (it loads the files into memory completely before sending)*"
- Zotonic relied on Nitrogen
- How Nitrogen processes requests
- How to add Nitrogen and Cowboy as dependency libs to your erlang application

Extra Features

Auto-generated Meta Website

If requested, at server start-up, a "meta" website - i.e. a website sharing information about all other websites being hosted by that server - can be generated and made available through a dedicated virtual host and web root.

For that, in the US-Web configuration file, among the user-specified `routes`, one may add the following element in the list of virtual host entries associated to a given domain (e.g. `foobar.org`):

```
{"mymeta", "My-meta-generated", meta}
```

This will generate a suitable website in the `My-meta-generated` subdirectory of the default web root (as, here, the specified directory is not an absolute one), and this website will be available as `mymeta.foobar.org` (of course both `mymeta` and `My-meta-generated` are examples; these names can be freely chosen).

Currently no specific access control to this website is enforced (thus by default anyone knowing or able to guess its virtual hostname can access it).

Note that apparently, at least under some circumstances, some versions of Firefox may incorrectly render a Meta web page: despite its HTML source being correct, for some reason the entry page of websites may correspond to another virtual host (whereas other browsers display correctly the same Meta page).

Icon (favicon) Management

This designates the little image that is displayed by browsers on the top of the tab of the website being visited.

A default icon file can be defined, it is notably used in order to generate better-looking 404 pages.

To do so, the `icon_path` key in the US-Web handler state shall be set to the path of such file (possibly a symbolic link), relatively to the content root of the corresponding virtual host.

In the context of the (default) US-Web static web handler, if such a `common/default-icon.png` exists (e.g. obtained thanks to this kind of generator), it is automatically registered in `icon_path`.

CSS Management

A default CSS file can be defined, notably in order to generate better-looking 404 pages.

To do so, the `css_path` key in the US-Web handler state shall be set to the path of such file (possibly a symbolic link), relatively to the content root of the corresponding virtual host.

In the context of the (default) US-Web static web handler, if such a `common/default.css` exists, it is automatically registered in `css_path`.

Error 404 Management

Should some requested web page not be found:

- a suitable 404 page is automatically generated by the US-Web server, and returned to the caller
- the error is appropriately logged

A 404 image can be optionally displayed instead of the "0" of "404". To do so, the `image_404` key in the US-Web handler state shall be set to the path of such image (possibly a symbolic link), relatively to the content root of the corresponding virtual host.

In the context of the (default) US-Web static web handler, if such a `images/404.png` file exists, it is automatically registered in `image_404`.

Site Customisation

As a summary of the sections above, if there is a file (regular or symbolic link), from the content root of a hosted static website, in:

- `common/default.css`, then this CSS will be used for all generated pages for that website (e.g. the one for errors such as 404 ones)
- `images/default-icon.png`, then this image will be used as an icon (the small image put on browser tabs next to their labels) for all generated pages
- `images/404.png`, then this image will be used for the "0" in the main "404" label of the page denoting a content not found

Each content root is expected to contain at least a (proper) `index.html` file (possibly as a symbolic link).

Usage Recommendations

In terms of security, we would advise:

- to stick to the **latest stable version** of all software involved (including US-Web and all its stack including Cowboy and LEEC, Erlang, and the operating system itself)

- depending on the preferred trade-off between accessibility and security, to build US-Web either with the `us_web_security` flag set to `strict` (more security; the default) or to `relaxed` (more compatibility with various clients); typically they aim respectively a grade of A and at least B at the SSL Labs server test; this build flag determines the versions of TLS supported (e.g. only 1.3 and 1.2; or 1.1 and 1.0 as well), the accepted list of ciphers (including or not some that are becoming weak), possibly the RSA key length, Diffie-Hellman parameters, how Forward Secrecy is managed, etc.; as it is a bit fun, the US-Web server tries (yet with no luck) to spoof its server signature in the headers, pretending it is an Apache instance; DNS CAA (*Certificate Authority Authorization*) is not specifically supported (as LEEC uses throwaway accounts)

- to apply a streamlined, reproducible **deployment process**, preferably based on our deploy-us-web-release.sh script

- to rely on dedicated, **different, low-privileged users and groups** for US and US-Web, which both rely on `authbind`; refer to our start-us-web.sh script for that; see also the `us_username` key of US-Common's us.config, and the `us_web_username` key of the US-Web configuration file that it refers to

- still in `us.config`, to set:

  - a strong-enough Erlang **cookie**: set the `vm_cookie` key to a well-chosen value, possibly a random one deriving from an output of `uuidgen`
  - possibly a **limited TCP port range** (see the `tcp_port_range` key)
  - the **execution context** to `production` (see the `execution_context` key)

- to use also the stop-us-web.sh counterpart script, and to have them triggered through `systemd`; we provide a corresponding us-web.service **unit file** for that, typically to be placed in `/etc/systemd/system` and whose `ExecStart/ExecStop` paths shall preferably be symlinks pointing to the latest deployed US-Web release (e.g. `/opt/universal-server/us_web-latest`)

- to ensure that a **firewall** blocks everything from the Internet by default, including the EPMD port(s) (i.e. both the default Erlang one and any non-standard one specified through the `epmd_port` key defined in `us.config`); one may get inspiration from our iptables.rules-Gateway.sh script for that

- no need to advertise specifically a virtual host in your DNS; for example, so that a `baz.foobar.org` is available, only `foobar.org` has to be declared in the DNS records (even a `*.foobar.org` wildcard is not necessary) for the corresponding website to be available; as a result, unless the full name of that virtual host is disclosed or a (typically brute-force) guessing succeeds, that virtual host will remain private by default (useful as a first level of protection, notably for any meta website)
- to **monitor regularly** both:
  - the US-Web server itself (see our monitor-us-web.sh script for that, relying on the trace supervisor provided by the Ceylan-Traces layer)
  - the remote, browser-based, accesses made to the hosted websites, typically by enabling the US-Web "meta" feature, generating and updating automatically a dedicated website displaying in one page all hosted websites and linking to their web analysis report; refer to the `log_analysis` key of the US-Web configuration file (e.g. see us-web-for-tests.config as an example thereof)

Licence

The `Universal Webserver` is licensed by its author (Olivier Boudeville) under the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of this license, or (at your option) any later version.

This allows the use of the Universal Webserver code in a wide a variety of software projects, while still maintaining copyleft on this code, ensuring improvements are shared.

We hope indeed that enhancements will be back-contributed (e.g. thanks to merge requests), so that everyone will be able to benefit from them.

Current Stable Version & Download

As mentioned, the single, direct prerequisites of the Universal Webserver are:

- Cowboy (version 2.8 or above)
- LEEC as an optional, runtime-only dependency
- Awstats as an optional, runtime-only dependency (version 7.8 or above)
- US-Common

The latter relies on Ceylan-Traces, which implies in turn Ceylan-WOOPER, then Ceylan-Myriad and Erlang.

We prefer using GNU/Linux, sticking to the latest stable release of Erlang, and building it from sources, thanks to GNU `make`. We recommend indeed obtaining Erlang thanks to a manual installation; refer to the corresponding Myriad prerequisite section for more precise guidelines.

The build of the US-Web server is driven by rebar3, which can be obtained by following our guidelines.

If a tool for web analysis is needed (typically if enabling a meta website), this tool must be installed beforehand. Currently US-Web supports Awstats, which can be obtained thanks to your distribution of choice (ex for Arch Linux: `pacman -S awstats`[7]) .

If wanting to be able to operate on the source code of the Ceylan and/or US dependencies, you may define appropriate symbolic links in a `_checkouts` directory created at the root one's `US-Web` clone, these links pointing to relevant Git repositories (see the `create-us-web-checkout` make target for that).

Using Cutting-Edge Git

This is the installation method that we use and recommend; the Universal Webserver `master` branch is meant to stick to the latest stable version: we try to ensure that this main line always stays functional (sorry for the pun). Evolutions are to take place in feature branches and to be merged only when ready.

Once Erlang (see here), rebar3 (see here) and possibly LEEC (see here) or Awstats (see here) are available, it should be just a matter of executing our get-us-web-from-sources.sh script for downloading and building all dependencies at once, and run a test server (use its `--help` option for more information).

For example:

```
$ cd /tmp
$ wget https://raw.githubusercontent.com/Olivier-Boudeville/us-web/master/priv/bin/get
$ sh ./get-us-web-from-sources.sh --checkout
Switching to checkout mode.

 Installing US-Web in /tmp...

 Cloning into 'us_web'...
 [...]
 ===> Compiling us_web
 Starting the us_web release (EPMD port: 4526):
 [...]
 US-Web launched, please point a browser to http://localhost:8080 to
  check test sites.

$ firefox http://localhost:8080 &
```

One shall then see a text-only page such as:

```
This is static website D. This is the one you should see if pointing
to the default virtual host corresponding to the local host. This
shows that the US-Web server is up and running.
```

Understanding the role of the main US configuration file and of the corresponding US-Web configuration file for this test should be fairly straightforward.

Based on that, devising one's version of them should allow to have one's US-Web server running at the cost of very little efforts.

[7] To avoid a future reading access error, execute after installation: `chmod -R +r /usr/share/webapps/awstats/icon`.

OTP Considerations

The default build system of US-Web is the native Ceylan one (triggered by `make all`), yet a rebar3-based one is also available (triggered by `make all-rebar3`).

In this last context, as discussed in these sections of Myriad, WOOPER, Traces and US-Common, the Universal Webserver *OTP application* is generated out of the build tree, ready to result directly in an *(OTP) release*. For that we rely on rebar3, relx and (possibly) hex.

Then we benefit from a standalone, complete Universal Webserver able to host as many virtual hosts on as many number of domains as needed.

As for Myriad, WOOPER, Traces, LEEC and US-Common, most versions of the Universal Webserver will be also published as Hex packages.

For more details, one may have a look at rebar.config.template, the general rebar configuration file used when generating the Universal Webserver OTP application and release (implying the automatic management of all its dependencies).

Troubleshooting

Start-up Failure

Many operations take place when launching a US-Web instance - especially when requesting the generation of certificates; so most of the issues are expected to happen then.

If having deployed a (here, native) release (typically by running deploy-us-web-native-build.sh) and if `systemctl restart us-web-as-native-build.service` failed, start by executing:

```
$ systemctl status us-web-as-native-build.service
```

It should return some appropriate information.

Most common sources of failures are:

- there is **already a program listening at the target TCP port** (typically port 80) designated for US-Web; one may check for example with `lsof -i:80`, otherwise with `netstat -ltpn | grep ':80'`; of course use the same procedures for TCP port `443` if having enabled `https` on its standard port
- check that no firewall is in the way (e.g. thanks to `iptables -nL | grep ':80'`)
- there may be a **prior, lingering US-Web** installation that is still running in the background; one may check for example with `ps -edf | grep us_web`; in incrementally brutal ways, one may rely on:
  - `systemctl stop us-web-as-native-build.service`; note that, if any prior US-Web instance will be shutdown indeed, at least one some cases EPMD will still believe it is running and will thus block any next launch; so EPMD should be taken care of as well (at least by forcibly unregistering a vanished US-Web node), which is done automatically by the our stop/kill scripts discussed just next

  - or on stop-us-web-native-build.sh
  - or even on kill-us-web.sh

- as mentioned above, the **EPMD daemon** of interest (possibly running on a non-standard TCP port) may wrongly believe that a prior US-Web is running, and thus prevent a new one to be launched; a simple solution could be to run `killall epmd`, yet unfortunately this solution is absolutely not selective enough: any other running Erlang application could be impacted as well; so now our EPMD daemons are *explicitly* launched, on different ports (default one for US-Web being 4508), with appropriate settings (`-relaxed_command_check`) that allow acting on these daemons independently
- check your `authbind` local configuration (refer to this section)

If the problem remains, time to perform some investigation, refer to our Local Monitoring section.

Certification Generation Issues

One may check first that a webserver can indeed be reached for the target domain or subdomain (e.g. do not forget to define a wildcard CNAME like `*.foobar.org` to point on `foobar.org`, so that virtual hosts can be reached by the ACME servers).

Beware also not to trigger too many / too frequent certificate requests, as this could lead to reaching the rate limits implemented by the ACME servers: then even perfectly legit, correct operations would fail.

Hints

Various keys (e.g. `us_app_base_dir`) may be defined in the US configuration files (e.g. in a `{us_app_base_dir, "/opt/some_dir"}` entry), from which other elements may derive (e.g. paths). To denote the value associated to a key, we surround in this documentation the key with `@` characters (this is just a reading convention).

For example `@us_app_base_dir@/hello.txt` would correspond here to `/opt/some_dir/hello.txt`.

Development vs Production Mode

Should a mismatch be detected between the compile-time execution target and the runtime, user-configured, execution context, a warning will be issued in the traces.

When building a fresh release thanks to `make release-dev`, the corresponding action (`rebar3 release`) will build that release with the base settings, hence in development mode (so not "as prod" - i.e. not with the `prod` profile, hence not selecting our `production` execution target).

Note that all dependencies (thus of course including Myriad, WOOPER, Traces and LEEC) are built by rebar3 with their `prod` settings. As a result, relying on `basic_utils:get_execution_target/0` will only tell us about `Myriad` settings (thus always built in production mode), not about any other layer (including `US-Web`). A US-Web trace allows to check all relevant modes, notably that, in production, the production settings apply indeed.

Development Hints

Operating directly from within the rebar build tree

(not recommended in a development phase)

If having modified and recompiled a Ceylan prerequisite (e.g. WOOPER), then, before generating the release, run from its root (e.g. `us_web/_build/default/lib/wooper`):

```
$ make rebar3-copy-beams REBAR_BUILD_DIR=../../
```

Or, simply run `make rebar3-compile REBAR_BUILD_DIR=../../` (and no need to run `make release` afterwards).

Operating from `_checkouts` build trees

(recommended in a development phase, as a lot more flexible/unified than the previous method)

Create a `us_web/_ckeckouts` directory containing symbolic links to repositories of dependencies (e.g. `myriad`) that may be updated locally; or, preferably, execute the `create-us-web-checkout` make target to automate and streamline this step.

Configuration Hints

Batch Mode

One can update `us_web/conf/sys.config` in order to toggle batch mode (e.g. to enable/disable a graphical trace supervision) after build time.

It deals with the configuration of the `Traces` application, so it comes very early at start-up (at application boot, hence before US-Web specific settings can be read, so this option would be difficult to put in the US configuration files).

Location of Applications

The location of the US-Web application is bound to differ depending on the context of use (development/production deployments, host-wise, etc.), whereas it is needed to be known at the very least by its start/stop scripts.

So its path must be specified (most preferably as an absolute directory) in the US-Web configuration file. However, at least for test configuration files, relying on an absolute directory would not be satisfactory, as the user may install the US-Web framework at any place and testing should not require a manual configuration update.

As a result, should an application location (e.g. US-Web) not be specified in the configuration, it will be looked-up in the shell environment (using the `US_WEB_APP_BASE_DIR` variable) for that and, should this variable not be set, as a last-resort an attempt to guess that location will be done.

System-related Hints

Besides the use of specific, unprivileged user/group thanks to the use of authbind (set with a minimal depth level), the system resources (notably the maximum number of file descriptors) are reported (among the traces; search for `System description`) at the startup of the US-Web server.

Moreover the launch scripts (namely `start-us-web-native-build.sh` and `start-us-web-release.sh`) raise the limit in terms of maximum number of file descriptors opened by the US-Web process, so that the connection acceptor does not have to reduce its accept rate (typically because of rogue bots performing pseudo-denials of service).

Finally, any webserver shall rely preferably on: - a sufficiently paranoid firewall (see section in Execution Hints) - an uninterruptible power supply (UPS), protecting the server and the telecom appliances involved

Web-related hints

- most paths (e.g. `default_web_root`, in the US-Web configuration) can be defined as **relative** ones (mostly useful for embedded tests; otherwise absolute paths shall be preferred); in this case they will be relative to the runtime current directory, typically `[...]/us_web/_build/default/rel/us_web/` in development mode
- the `default_domain_catch_all` atom allows to designate any **domain-level host** (e.g. `foobar.org`) that did not match previous host rules
- in the context of a given host (e.g. `foobar.org`), the `default_vhost_catch_all` atom allows to designate any of its **virtual hosts** (e.g. `bar`, to be understood as `bar.foobar.org`) that did not match previous path rules[8]
- refer to `us_web/priv/for-testing` for an example setup and configuration files
- the web roots shall be owned by the user running US-Web (e.g. `chown -R us-web-user:us-group /opt/www`)

Execution Hints

- the current working directory of a US-Web instance deployed thanks to `deploy-us-web-release.sh` is `/opt/universal-server/us_web-x.y.z`
- if unable to connect, the firewall (e.g. `iptables -L`) might be the culprit! Note that the whole US framework tends to rely on a specific TCP range (e.g. `50000-55000`) for inter-VM communications; for HTTP, TCP port 80 is expected to be opened, and this is TCP port 443 for HTTPS (see also our iptables.rules-Gateway.sh script)

[8] When using https, configuring a catch-all might be ill-advised, as whenever an unanticipated virtual host is requested, a certificate warning (e.g. `SSL_ERROR_BAD_CERT_DOMAIN`) is triggered - as by design no certificate (direct or through SANs) will exist for this host (a "connection failed" error is then more desirable).

- to debug (once batch mode has been enabled/disabled), one may use the `debug` make target, from the tree root
- to test server-side errors, one may create, in a web root, a directory that cannot be traversed (e.g. `chmod 000 my-dir`) and direct one's browser to a content supposedly located in this directory[9]; note that, if requesting instead that faulty directory itself (not an element within), then (whether or not a trailing `/` is specified), an error 403 (`ERROR 403: Forbidden`) is returned instead (corresponds to the `case A` in the corresponding sources)
- to test host matching (e.g. for a `baz` virtual host), including the default catch-all even on a computer having no specific public DNS entries, one may simply add in `/etc/hosts` entries like:

```
127.0.0.1 foobar-test.net baz.foobar-test.net other.foobar-test.net
```

- log rotation results in timestamped, compressed files such as `access-for-bar.localhost.log.2019-12` note that the timestamp corresponds to the moment at which the rotation took place (hence not the time range of these logs; more an upper bound thereof)
- to test whether a combination of EPMD port and cookie is legit, one may use for example:

```
$ ERL_EPMD_PORT=44000 /usr/local/lib/erlang/erts-x.y/bin/erl_call
   -name us_web@baz.foobar.org -R -c 'MyCookie' -timeout 60
   -a 'erlang is_alive'
```

You then expect `true` to be returned - not:

```
erl_call: failed to connect to node us_web@baz.foobar.org
```

Monitoring Hints

In terms of (UNIX) Processes

A running US-Web server will not be found by looking up `beam` or `beam.smp` through `ps`; as an OTP release, it relies first on the `run_erl` launcher, like shown, based on `ps -edf`, in:

```
UID         PID    PPID  C STIME TTY TIME     CMD
us-web-user 767067    1  0 Feb15 ?   00:00:00 /usr/local/lib/erlang/erts-x.y/bin/run_e
  -daemon /tmp/erl_pipes/us_web@MY_FQDN/ /opt/universal-server/us_web-US_WEB_VERSION/l
  exec "/opt/universal-server/us_web-US_WEB_VERSION/bin/us_web" "console" ''
  --relx-disable-hooks
```

This can be interpreted as:

[9] See `priv/for-testing/test-static-website-A/to-test-for-errors`, which was created precisely for that. Note that its permissions have been restored to sensible values, as otherwise that directory was blocking the OTP release generation.

- not running as root, but as a dedicated, weakly privileged user
- its parent process (PPID) is the first overall process (as a daemon)
- STIME is the time when the process started
- no associated TTY (runs detached in the background)

This launcher created the main, central, `us_web` (UNIX) process, parent of all the VM worker (system) threads.

`pstree -u` (or `ps -e --forest`) tells us about the underlying process hierarchy:

```
[...]
  |-run_erl(us-web-user)---beam.smp---erl_child_setup---inet_gethost---inet_gethost
  |                                |-158*[{beam.smp}]
[...]
```

The 158 threads must correspond to:

- 128 async ones (`-A 128`)
- 30 "normal" threads (on a computer having a single CPU with 8 cores with Hyperthreading, hence 16 logical cores)

Using `htop`, one can see that the `run_erl` process spawned a `us_web` one (namely `/opt/universal-server/us_web-US_WEB_VERSION/bin/us_web`) that is far larger in (VIRT) memory (e.g. 5214MB, instead of 5832KB for the former).

`us_web` in turn created the numerous threads.

`RSS/RSZ` (*Resident Set Size*) is a far better metric than `VIRT/VSZ` (*Virtual Memory Size*); indeed `VIRT = RSS + SWP` and:

- `RSS` shows how much memory is allocated to that process and is in RAM; it does not include memory that is swapped out; it includes memory from shared libraries (as long as the pages from those libraries are actually in memory), and all stack and heap memory used
- `VIRT` includes all memory that the process can access, including memory that is swapped out, memory that is allocated but not used, and memory that is from shared libraries

Knowing that, with `ps`, one may add `-L` to display thread information and `-f` to have full-format listing, a base command to monitor the US-Web processes is: `ps -eww -o rss,pid,args | grep us_web`, with:

- `-e`: select all processes
- `-w` (once or twice): request wide output
- `-o rss,pid,args`: display RSS memory used (in kilobytes), PID and full command line

(apparently there is no direct way of displaying human-readable sizes)

See also our list-processes-by-size.sh script; typical use:

```
$ list-processes-by-size.sh
   Listing running processes by decreasing resident size in RAM (total size in KiB):
 RSS  PID      COMMAND
[...]
1204  1695242  /usr/local/lib/erlang/erts-11/bin/run_erl -daemon /tmp/erl_pipes/us_wel
67776 1695243 /opt/universal-server/us_web-x.y.z/bin/us_web -A 128 -- -root /opt/unive
[...]
```

This confirms that, even if higher VIRT sizes can be reported (e.g. 5214M, hence roughly 5GB), the RSS size may be 67776 (KB, hence 66 MB), i.e. very little, and does not vary much.

Indeed, in terms of RSS use (for a few domains, each with a few low-traffic websites, if that matters), we found:

- at start-up: only about 67MB
- after 5 days: just 68 MB
- after 24 days: 76 MB
- after 55 days: 80-88 MB

A more recent server instance is using, after more than 70 days, 60 MB of RSS memory, whereas after 30 days another ones was 108 MB. Anyway quite low.

Trace Monitoring

Use the `us_web/priv/bin/monitor-us-web.sh` script in order to monitor the traces of an already running, possibly remote, US-Web instance.

Note that the monitored US-Web instance will be by default the one specified in any `us-monitor.config` file located in the US configuration directory.

One may specify on the command-line another configuration file if needed, such as `us-monitor-for-development.config`.

Node Test & Connection

If desperate enough, one may also run, possibly from another host, and based on the settings to be found in the configuration files:

```
$ ERL_EPMD_PORT=XXXX erl -name tester -setcookie CCCC -kernel inet_dist_listen_min MII

1> net_adm:ping('us_web@foobar.org').
pong
```

Then one may switch to the *Job control mode* (JCL) by pressing `Ctrl-G` then `r` to start a remote job on the US-Web node.

Forcing Certificate Renewal Without Restarting the Server

The renewal of certificates is a rather error-prone operation, as it involves third-party (ACME) servers, which additionally enforce various rate limits.

Moreover this operation will be triggered automatically a long time[10] after the server has been launched, in the background, so its silent failure shall be considered (even though the traces should duly keep track of it).

So the goal is to trigger the `renewCertificate/1` oneway of the certificate manager of each website. This can be done simply by triggering the `renewCertificates/1` request of the US-Web configuration server - which happens to be a registered process.

To determine the name and scope of this server and the various Erlang settings (EPMD port and Erlang cookie) that will be needed in order to connect to the US node, one may read one's US configuration, typically in `/etc/xdg/universal-server/us.config`, for: `vm_cookie`, `epmd_port` and `us_web_config_filename`, the configuration file from which `us_web_config_server_registration_name` can be obtained.

It may be safer to monitor remotely from then on the traces of that server, using for that the `priv/bin/monitor-us-web.sh` script.

Then, when logged on the server, or from any client able to connect to it:

```
# We are using short names here:
$ ERL_EPMD_PORT=4506 erl -sname renewer -setcookie 'MY_VM_COOKIE'
(renewer@tempest)1> net_adm:ping('us_web@hurricane').
pong
(renewer@tempest)2> CfgSrvPid = global:whereis_name('my_name_for_the_us_web_config_ser
<9385.172.0>
(renewer@tempest)3> CfgSrvPid ! {renewCertificates, [], self()}.
{renewCertificates,[],<0.94.0>}
% Expecting certificate_renewals_over:
(renewer@tempest)4> receive {wooper_result,R} -> io:format("Received: ~p~n", [R]) end
```

Safer US-Web Updates: Testing a New Instance First Out of the Loop

Objective

When running a reference server for a set of given domains, one should avoid stopping US-Web, updating its code and attempting to relaunch it: not only because this may imply a non-negligible downtime, but also, more importantly, because, should a failure happen, it would have to be fixed live, in the hurry, whereas no one can access anymore any of the hosted sites. Not a convenient situation.

A better, safer (but not simpler) approach is instead to install one's newer server instance as a test one on the target host, build it and test it on a temporary setting (on an auxiliary TCP port) and, if and only if the operation is fully satisfactory, to switch the same setting on the actual port (once the prior reference instance has been shut down); if on the contrary something goes

[10] Refer to the `dhms_cert_renewal_period_{development,production}` defines in `class_USCertificateManager`; typically, in production mode, 70 days are to elapse between two renewals.

wrong with the test, it can be safely fixed out of the critical path, not implying a specific downtime.

> **Note**
>
> Note though that running US-Web on non-standard ports will not permit the http-01 challenges to succeed (as the ACME servers will expect to find the answers to their challenges by accessing standard ports).

Test Installation

For such a test bed to exist, the usual installation step shall be followed by a specific, separate configuration step.

Installing is merely a matter of transferring the `deploy-us-web-native-build.sh` script to one's server and to run it (as a normal user), as `deploy-us-web-native-build.sh --no-launch` so that it cannot interfere with any reference version running.

A `check-foobar-website-availability.sh` script may be useful to efficiently check the state of that reference version.

Test Configuration

A first action is to devise a corresponding configuration file and to launch a corresponding instance on dedicated TCP ports.

It will require a dedicated US-Web configuration file, let's name it `foobar-us-web-for-testing.config`. It will have to be referenced by a corresponding US configuration file (namely `us-config`), both of them expected to be in a `universal-server` directory in a test-specific root directory, e.g. in `TEST_DIR=/opt/universal-server/us_web-data/configuration-for-t`
- thus as `/opt/universal-server/us_web-data/configuration-for-testing/universal-server/{fooba`
- rather than in a standard root directory such as `/etc/xdg`.

The most direct/reliable approach is to copy the reference versions of these two files and modify them as little as possible.

The test `us-config` file should specify:

- a cookie of its own (e.g. `foobar-testing-be3be613-96d8-4203-a502-73cc42a5e5e2`)
- (the EPMD port may be set only in its US-Web counterpart configuration file)
- possibly different users (preferably not, as wanting to stick as close as possible to the target settings)
- updated directories (possibly all - like `us_log_dir` - pointing to `TEST_DIR`)
- the updated `us_web_config_filename`, like `foobar-us-web-for-testing.config`

This `foobar-us-web-for-testing.config` associated file could then specify:

- an EPMD port of its own (e.g. 4515); if using our `iptables.rules-Gateway.sh` script (refer to this section for more details), one may check in `/etc/iptables.settings-Gateway.sh` the `us_{low,high}est_epmd_port` variables in order to pick an EPMD port in the LAN-allowed input range, so that hosts from the LAN may connect to the test US-Web instance on the gateway if needed

- `us_web_data_dir` and `us_web_log_dir` entries pointing to the aforementioned TEST_DIR
- updated, non-clashing TCP ports, for example `http_tcp_port` being 8000 and `https_tcp_port` being 8443
- for extra caution, `certificate_mode` may be set to `development` (rather than `production`)
- `challenge_type` can be set, knowing its default is `dns-01`
- for routes, for the sake of testing, a single domain may be kept

Finally, if relying on a `dns-01` challenge, the credentials for the DNS provider will have to be available. Creating a symlink in the test-specific root directory to, say, `/etc/xdg/universal-server/leec-ovh-credentials-for-foobar.org.txt` should be sufficient for that.

Firewall Configuration

A second step is, firewall-wise, to enable (temporarily) the access to the test instance.

The target EPMD port must already fall in a LAN-supported range, yet we also want to be able to properly test as if we were a random Internet user, thus connecting from the WAN.

So:

```
$ iptables -A INPUT -p tcp --dport 8000 -m state --state NEW,ESTABLISHED -j ACCEPT
$ iptables -A INPUT -p tcp --dport 8443 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Test Code Updates

Tests may show that some modules have to be fixed, and thus that the server installation has to be updated accordingly (rather than be deployed again from scratch). The `sync-all-to-server` make target is useful to synchronise the code base of a given layer on the server from a development computer; for that the `CEYLAN_SYNC_TARGET_ROOT` must be set to the root path on the server; for example:

```
$ make sync-all-to-server CEYLAN_SYNC_TARGET_ROOT=/opt/universal-server/us_web-native
```

Launching the Test Server

For this use, `systemctl` would not be convenient. A better approach is to start one's test server explicitly, based on the configuration files that were just devised.

First of all, no prior test instance should exist, otherwise they would clash EPMD-wise. `ps -edf|grep -i epmd` may help.

Then the target test instance can be launched, as root:

```
$ cd /opt/universal-server/us_web-latest/us_web/priv/bin
$ ./start-us-web-native-build.sh /opt/universal-server/us_web-data/configuration-for-t
```

Testing it

Either the launch was reported as succeeded or not.

In the latter case, a launch time-out was reported. The log of the Erlang VM (typically `/opt/universal-server/us_web-native/us_web/log/erlang.log.1`) will probably give relevant information. May a prior test instance is on the way, clashing in the same EPMD instance, or regarding the target TCP ports?

In the former case, a success is assumed. The best way to confirm this is to check the overall result by ourself, with `wget http://foobar.org:8000 -O -`.

Further insight to be obtained based on `/opt/universal-server/us_web-data/configuration-for-tes`

Test Teardown

Finally, now that the test is over, we can close up the mountain with:

```
# The -n option is used to have port numbers rather than aliases:
$ iptables -L --line-numbers -n

$ iptables -D INPUT X

# Again, as the previous deletion decremented the number of the last line:
$ iptables -D INPUT X
```

Standard Log Generation & Analysis

The objective in this section is to have US-Web generate access logs like the standard webservers do, so that standard tools able to produce log analyses can be used afterwards. They may generate HTML pages, which can in turn be appropriately hosted directly by the US-Web framework itself.

US-Web (rather than a `crontab`) takes control of log writing, for full control onto the start, stop and rotation behaviours.

For write efficiency, It is done by an (Erlang) process specific to a given virtual host (see `class_USWebLogger`), and a task ring is used for synchronicity and load balancing with regard to log report generation.

For each virtual host (e.g. `baz.foobar.org`), following log files shall be generated:

- `access-for-baz.foobar.org.log`
- `error-for-baz.foobar.org.log`

They are to be stored by default in the `/var/log` directory, which can be overridden in the US-Web configuration file thanks to the `us_web_log_dir` key.

At the Cowboy level, logging could be done as a middleware, yet we preferred to perform it directly in the handler (typically the static one, `us_web_static`), presumably for a better control.

Once access logs are produced, a specific tool is to generate reports out of them, as discussed in the next section.

Web Analytics Software: Choice of Tool

The desired properties for such a tool are mainly: available as free software, trustable, standard, actively and well-maintained, running locally (as opposed to remote services such as Google Analytics), standalone, not resource-demanding, controllable, generating a local analysis (static) website, based only on basic webserver logs (not on a dedicated database, not on markers to be added to all webpages such as 1-pixel images), virtual-host compliant.

Various tools can be considered, including (best spotted candidates, by increasing level of interest):

- The Webalizer: simple reports only, not maintained since 2013?
- OWA: for professional, store-like business
- Matomo: interesting, yet a bit too complete / integrated; requires a dedicated database
- GoAccess: a good candidate, almost no dependency, actively maintained, beautiful reports, supports GeoLite2, but more real-time oriented (more like a web monitor) and with less in-depth metrics
- AWStats: old yet still maintained, real community-based open-source software, very complete, probably the most relevant in the list, whose code is apparently now maintained here

So we finally retained AWStats.

Log Format

The most suitable log format that we spotted (see [1] and [2] for more information) is named "*NCSA combined with several virtualhostname sharing same log file*".

Its correct definition is exactly:

```
LogFormat="%virtualname %host %other %logname %time2 %methodurl %code %bytesd %referer
```

For example:

```
virtualserver1 62.161.78.73 - - 2020-02-02 01:59:02 "GET /page.html HTTP/1.1" \
200 1234 "http://www.from.com/from.htm" "Mozilla/4.0 (compatible; MSIE 5.01; Windows N
```

This format is better than the "Apache combined logs" (combined, not common) log format, as containing the virtual host (important); note that for this second format, precisely named "*Apache or Lotus Notes/Domino native combined log format (NCSA combined/XLF/ELF log format)*" would be defined as:

```
LogFormat="%host %other %logname %time1 %methodurl %code %bytesd %refererquot %uaquot
```

For example:

```
62.161.78.73 - - [dd/mmm/yyyy:hh:mm:ss +0x00] "GET /page.html HTTP/1.1" \
200 1234 "http://www.from.com/from.htm" "Mozilla/4.0 (compatible; MSIE 5.01; Windows N
```

Field descriptions for this last format: [1], [2], [3], [4].

See also regarding Awstats log formats: [1], [2], [3].

In all these cases, the log separator is a single space (hence LogSeparator=" ").

Awstats Management

Awstats Installation

On Arch Linux, one should follow these guidelines (example for version 7.8):

```
$ pacman -Sy --needed awstats
```

Awstats will then be installed in /usr/share, including the /usr/share/webapps/awstats/cgi-bin/awst script and the /usr/share/webapps/awstats/icon/ directory.

Some permission fixes (to be done as root) might be needed first:

```
$ chmod +r /usr/share/webapps/awstats/icon/os/*
```

Awstats Configuration

Log analysis will be triggered periodically by the US-Web server rather than on-demand via CGI Perl scripts, and its result, i.e. the web pages generated from the access logs, will be available in the meta website (e.g. mymeta.foobar.org; refer to Auto-generated Meta Website for more information).

More precisely, and as already mentioned, in the US-Web log directory (see us_web_log_dir), dedicated access and error log files will be generated for each known virtual host. For example the accesses to a baz.foobar.org virtual host will be written by the US-Web server in a corresponding access-for-baz.foobar.org.log file.

At server start-up, the US-Web meta module (us_web_meta) will have generated a suitable Awstats configuration file (namely awstats.baz.foobar.org.conf) that will trigger the generation of the corresponding static web pages (awstats.baz.foobar.org.*, notably awstats.baz.foobar.org.html) in the web root of the meta website.

These configuration files are now placed in /usr/local/etc/awstats (they were previously in the conf subdirectory of the root specified in us_web_app_base_dir).

Indeed, if starting from version 7.8, Awstats allows these configuration files to be specified as absolute paths, its previous versions:

- either required such configuration files to be in /etc/awstats, /usr/local/etc/awstats, /etc or in the same directory as the awstats.pl script file
- or, if the configuration files could be specified as absolute paths, the generated pages would then include some faulty links because of that

US-Web retained the most controllable, less "system" directory, /usr/local/etc/awstats. All these locations are mostly root-only, whereas the US-Web server is designed to run as a normal, non-privileged user and is to generate there these Awstats configuration files.

Such a target directory shall thus be created beforehand, and made writable by the user specified in us_web_username.

Each virtual host (say: baz.foobar.org) will have its configuration file deriving from priv/conf/awstats.template.conf, where the following patterns will be replaced by relevant ones (through keyword-based replacements):

- `US_WEB_VHOST_LOG_FILE` to become the full path to the corresponding access log (e.g. `access-for-baz.foobar.org.log`, in `us_web_log_dir`)
- `US_WEB_VHOST_DOMAIN` to become the virtual host domain (e.g. `baz.foobar.org`)
- `US_WEB_LOG_ANALYSIS_DATA_DIR` to become the directory in which the working data (e.g. state files) of the web analyzer (here Awstats) shall be written

Awstats icons are copied to the `icon` directory at the root of the meta website.

The Awstats database, typically located in `/var/local/us-web/data`, will be updated once an access log file will be rotated; just after, this log file will be compressed and archived under a relevant filename, such as `access-for-baz.foobar.org.log.2020-2-1-at-`

Awstats Troubleshooting

Various issues may prevent log reports to be available.

Let's try with a real US-Web uncompressed log file first (e.g. `xz -d access-vhost-catchall.log.test.x` supposing that it corresponds to a `my-test` virtual host).

Then configure Awstats (e.g. through a `/usr/local/etc/awstats/awstats.my-test.conf` file) to process that log file; for that, run on that host:

```
$ perl /usr/share/awstats/tools/awstats_configure.pl
```

Then, to debug the whole process, use, as root:

```
$ rm -f /usr/share/webapps/awstats/cgi-bin/awstats*.txt ; echo ;
   LANG=C /usr/share/webapps/awstats/cgi-bin/awstats.pl
     -config=my-test -showdropped
```

Most problems should become visible then.

To do the same for a series of web logs in the context of US-Web, one can have them analysed first thanks to:

```
$ for f in /usr/local/etc/awstats/awstats-*conf; do echo ;
   LANG=C /usr/share/webapps/awstats/cgi-bin/awstats.pl
     -config=$f -update ; done
```

Then all web reports can be generated manually with:

```
$ for f in /usr/local/etc/awstats/awstats-*conf; do echo ;
   LANG=C /usr/share/webapps/awstats/cgi-bin/awstats.pl
     -config=$f -output ; done
```

Geolocation with Awstats

Multiple plugins exist for that.

Apparently, none is able to load the new GeoIP2 format (see also this).

As a consequence: topic dropped for the moment.

Managing Public-Key Certificates

The goal here is to benefit from suitable certificates, notably (but not only) for https (typically running on TCP port 443, multiplexed thanks to SNI, i.e. Server Name Indication[11]), by automatically (and freely) generating, using and renewing them appropriately, for each of the virtual hosts managed by the US-Web server.

X.509 Certificates

The certificates managed here are X.509 TLS certificates, which can be seen as standard containers of a public key together with an identity and a hierarchical, potentially trusted *Certificate Authority* (CA) that signed them[12].

Such certificates can be used for any protocol or standard, and many do so - including of course TLS and, to some extent, SSH. Being necessary to the https scheme, they are used here.

Let's Encrypt

US-Web relies on Let's Encrypt, a non-profit certificate authority from which one can obtain mono-domain (possibly with SANs - *Subject Alternative Names*; and now wildcard domains) X.509 certificates[13] for free, valid for 90 days and that can be renewed as long as needed.

Let's Encrypt follows the ACME (*Automatic Certificate Management Environment*) protocol. For mono-domain validation, it relies on an `http-01` challenge, with an agent running on the server bound to the domain for which a certificate is requested.

This agent generates first a RSA key pair in order to interact with the Let's Encrypt certificate authority, so that it can prove through received challenge(s) that it is bound to the claimed domain / virtual host (e.g. `baz.foobar.org`) and has the control to the private key corresponding to the public one that it transmitted to the CA.

Generally this involves for that agent to receive a "random" piece of data from the CA (the nonce), to sign it with said private key, and to make the resulting answer to the challenges available (as tokens) through the webserver at a relevant URL that corresponds to the target virtual host and to a well-known path (e.g. `http://baz.foobar.org/.well-known/acme-challenge/xxx`).

Refer to this page for more information; the overall process is explained here and in this RFC.

As for wildcard certificates, the ACME protocol relies on `dns-01` challenges, which require the certificate requester to prove that it controls the DNS of the target domain, by updating its zone entries according to information chosen by the ACME server of the certificate issuer.

[11] As a consequence, the specific visited virtual hostname (e.g. `baz`, in `baz.foobar.org`) is *not* encrypted, and thus might be known of an eavesdropper.

[12] The X.509 standard also includes certificate revocation lists and the algorithm to sign recursively certificates from a trust anchor.

[13] `Let's Encrypt` provides *Domain Validation* (DV) certificates, but neither more general *Organization Validation* (OV) nor *Extended Validation* (EV).

US-Web Mode of Operation

For mono-domain certificates

Rather than using a standalone ACME agent such as the standard one, `certbot`, we prefer driving everything from Erlang, for a better control and periodical renewal (see the scheduler provided by US-Common).

Various libraries exist for that in Erlang, the most popular one being probably letsencrypt-erlang; we forked it (and named it LEEC, for *Let's Encrypt Erlang with Ceylan*, to tell them apart), in order notably to support newer Erlang versions and to allow for *concurrent* certificate renewals (knowing that one certificate per virtual host will be needed).

Three action modes can be considered to interact with the Let's Encrypt infrastructure and to solve its challenges. As the US-Web server is itself a webserver, the `slave` mode is the most relevant here.

For that, the us_web_letsencrypt_handler has been introduced by US-Web.

By default, thanks to the US-Web scheduler, certificates (which last for up to 90 days and cannot be renewed before 60 days are elapsed) will be renewed every 75 days, with some random jitter added to avoid synchronising too many certificate requests when having multiple virtual hosts - as they are done concurrently, if not SANs are used.

For wildcard certificates

US-Web relies here also as much as possible on LEEC (see this section), even if, at least currently, the standard `certbot` is used internally; here US-Web, thanks to its US-Common Scheduler, acts mostly like an integrated crontab on steroids.

Settings

Various types of files are involved in the process:

- a `.key` file contains any type of key, here this is a RSA private key; typically `letsencrypt-agent.key-I`, where `I` is an increasing integer, will contain the PEM RSA private key generated by the certificate agent `I` on behalf of the US-Webserver (so that it can sign the nonces provided by Let's Encrypt, and thus prove that it controls the corresponding key pair); for a `baz.foobar.org` virtual host, the `baz.foobar.org.key` file will be generated and used (another PEM RSA private key)

- a `.pem` (*Privacy-enhanced Electronic Mail*) file just designates a Base64-encoded content with header and footer lines; here it stores an ASN.1 (precisely a Base64-encoded DER) certificate

- `.csr` corresponds to a PKCS#10 *Certificate Signing Request*; it contains information (encoded as PEM or DER) such as the public key and common name required by a Certificate Authority to create and sign a certificate for the requester (e.g. `baz.foobar.org.csr` will be a PEM certificate request)

- `.crt` is the actual certificate (encoded as PEM or DER as well), usually a X509v3 one (e.g. `baz.foobar.org.crt`); it contains the public key and also much more information (most importantly the signature by the Certificate Authority over the data and public key, of course)

We must determine:

- the size of the RSA key of the agent; the higher the better, hence: 4096
- where the certificate-related files will be stored: in the `certificates` subdirectory of the US-Web data directory, i.e. the one designated by the `us_web_data_dir` key in US-Web's configuration file (hence it is generally the `/var/local/us-web/us-web-data` or `/opt/universal-server/us_web-x.y.z/us-web-data` directory)

The precise support for X.509 certificates (use, and possibly generation and renewal) is determined by the `certificate_support` key of the US-Web configuration file:

- if not specified, or set to `no_certificates`, then no certificate will be used, and thus no https support will be available
- if set to `use_existing_certificates`, then relevant certificates are supposed to pre-exist, and will be used as are (with no automatic renewal thereof done by US-Web)
- if set to `renew_certificates`, then relevant certificates will be generated at start-up (none re-used), used since then, and will be automatically renewed whenever appropriate

Another setting applies, determined this time by the `certificate_mode` key, whose associated value (which matters iff `certificate_support` has been set to `renew_certificates`) is either `development` or `production` (the default). In the former case, the staging ACME parameters will apply (implying relaxed limits, yet resulting in the obtained certificates to be issued by a fake, test CA), whereas the latter case will imply the use of the production ones.

When a proper certificate is available and enabled, the US webserver promotes automatically any HTTP request into a HTTPS one, see the us_web_port_forwarder module for that (based on relevant routing rules).

Standard, basic firewall settings are sufficient to enable interactions of US-Web (through LEEC) with Let's Encrypt, as it is the US-Web agent that initiates the TCP connection to Let's Encrypt, which is to check the challenge(s) through regular http accesses done to the webserver expected to be available at the domain of interest.

The US-Web server must be able to write in the web content tree, precisely to write files in the `well-known/acme-challenge/` subdirectory of the web root.

HTTPS Troubleshooting

If trying to connect with the https scheme whereas it has not been enabled, `wget https://baz.foobar.org/ -O -` is to report `Connection refused`.

Planned Enhancements

Reverse Proxy

Purpose

The goal of a reverse proxy is to have a (proxy) server seamlessly (read: without doing a HTTP forward - thus with unchanged URLs from the point of view of the client) redirect incoming traffic to other "proxied" webservers (possibly other US-Web instances) running on other hosts and/or ports.

A typical use case here[14] is having an US-Web instance serve a domain name of interest (e.g. `foo.org`) whereas more than one of its virtual hosts are Nitrogen-based, knowing that no two of such virtual hosts shall be executed by the same (Erlang) VM (as notably their Nitrogen page modules would clash).

For example a reverse proxy could ensure that connecting to the `https://my-server.foo.org` (virtual) host is transparently forwarded to `http://foo.org:8150` (a port which could be filtered by the local firewall - thus with no possible direct access from the outside network).

So this latter, autonomous webserver `my-server` instance would run thanks to its own VM on TCP port 8150, yet would only by reachable by connecting to the former, centralising webserver, which would act as a reverse proxy. As many other virtual hosts as wanted could be declared (e.g. as `https://my-other-server.foo.org`) to the reverse proxy, each pointing then to its own actual webserver instance (e.g. `http://foo.org:8151`).

Challenges

The HTTP/HTTPS routing shall be fully transparent, notably not breaking any connectivity (e.g. regarding Comet, AJAX, Websockets) and complying with the proper use of the underlying protocols (e.g. so that browsers do not interpret a proxying as the spoofing of SSL certificates).

Implementations

There are many solutions in order to obtain a reverse proxy; it may be done:

- directly at the firewall level, typically by configuring iptables accordingly
- thanks to nginx, as a NGINX Reverse Proxy (probably the most popular option)
- at the Erlang-level: see for example cowboy_revproxy, Surrogate or yaws_revproxy

US-Web Reverse Proxy

For better control US-Web will implement its own reverse proxy.

For that it will automatically modify header fields (e.g. `Host`, `Connection`, `X-Real-IP`; the source IP could also by set) in the proxied requests.

[14] The main other uses cases happen when wanting to:

- run a webserver on a non-privileged port
- balance the load between multiple actual servers

The responses from proxied servers could be buffered until they are fully received - to better manage slower clients - yet, as here the proxied servers may offer rather interactive uses, buffering may not be desirable.

Support

Bugs, questions, remarks, patches, requests for enhancements, etc. are to be reported to the project interface (typically issues) or directly at the email address mentioned at the beginning of this document.

Please React!

If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, follow the Support guidelines).

Ending Word

Have fun with the Universal Webserver!

Universal Webserver