SSL/TLS cert on HTTPS://go.genivi.org

Documenting SSL/TLS certificate installation on **go.genivi.org**

The steps to copy the certificates from the host to the container, and convert them to go-server preferred format, have been encoded in a shell script which is also attached to this page

The rest of this page only documents the manual steps and more information.

First Nicholas Contino created the certificates with Letsencrypt, by closing down the go-server and starting a vanilla apache install, and then going through the standard Letsencrypt procedures to confirm ownership of the site, and all that stuff.

This created

/etc/letsencrypt/{accounts,archive,csr,keys,live,renewal}...

The rest was done by Gunnar Andersson. First I transferred the files into the go-server container as such:

This little trick transfers it into the container and unpacks it again. It pipes the tarball contents into the docker command that runs in a terminal (-i) and executes tar extract with STDIN as input (-).

```
$ sudo tar -l -cf - /etc/letsencrypt | docker exec -i go-server /bin/tar xvf -
```

Since docker exec defaults to the root directory / as working directory, and it executes the command with root privileges, this tar command successfully extracts the files into /etc/letsencrypt inside the container.

The next steps are documented on the Go.CD documentation but I'm just repeating them here with our specific details.

The first steps could be done outside the container but for simplicity we do everything inside:

Let's get a shell inside the container:

```
$ docker exec -ti go-server bash
```

("docker attach go-server" also works in fact, because there is already a shell session to attach to, but in the future the container might be set up differently, and in that case if there's no shell session in the terminal there is a risk that you try to CTRL-C to get out of the attached terminal, or exit the shell session. Depending on the container setup, this might stop the process and ultimately the container. The safest way is therefore to start a new shell that you can later exit out of without problems.)

NOTE: After the tar command the **/live** directory should now point to the right files (for example after an update), so let's simply work inside of there.

```
$ cd /etc/letsencrypt/live/go.genivi.org
```

It is apparently required to convert format, and set a passphrase on the key. For simplicity I'm reusing the same password throughout and the Go.CD documentation says it must be the following:
serverKeystorepa55w0rd

```
$ openssl rsa -des3 -in privkey.pem -out privkey.key.new -passout pass:serverKeystorepa55w0rd
```

The PEM passphrase is no longer requested since it is given on the command line.

Then to put the certificate into a Java compatible keystore it first needs to be converted to a PKCS12 format.

(From Go.CD documentation): openssl pkcs12 -inkey privkey.key.new -in <example.com.crt> -export -out cert1.crt.pkcs12
*I failed at the first attempt I assumed example.crt meant our own cert file only. But this will make the Go server output the cert as self-signed as usual. It turns out that the **fullchain** file should be used, which includes both our cert, and the trust chain. Ref: [2] (ignore the first answer which is wrong, and see further down)*

So we run:

```
$ openssl pkcs12 -inkey privkey.key.new -in fullchain.pem -export -out fullchain.pkcs12 -passin pass:
serverKeystorepa55w0rd -passout pass:serverKeystorepa55w0rd
```

Againthe passwords are given on the command line.

In this step the new privkey we created of course needs be decrypted again. Input the previously used password to decrypt, then use the same again, for the output stage.

Finally, Java prefers a so called "keystore" format. The **keytool** comes with Java.

Although Java was not installed on the host, it is used inside the go-agent container, so it's all good. We keep working inside the container.

NOTE: **Still inside container.**

```
  $ cd /etc/letsencrypt/live/go.genivi.org
  $ keytool -importkeystore -srckeystore fullchain.pkcs12 -srcstoretype PKCS12 -srcstorepass
'serverKeystorepa55w0rd' -srcalias 1 -destalias cruise -deststorepass 'serverKeystorepa55w0rd' -destkeypass
'serverKeystorepa55w0rd' -destkeystore ./keystore -noprompt
```

During update, we're recreating a keystore that existed. You may be asked to overwrite "alias 1". **Answer yes**.

Continuing inside container I now set up a symlink from /etc/go/keystore pointing to the file stored in /etc/letsencrypt/live.

⚠️ NOTE: If the symlink is already there - of course you will get a complaint. You can skip this step if doing an upgrade because the symlink already exists.

```
  $ cd /godata/config
```

Verify that keystore file exists already.
Import into the already existing keystore file:

OLD:

```
$ keytool -importkeystore -srckeystore fullchain.pkcs12 -srcstoretype PKCS12 -srcstorepass
'serverKeystorepa55w0rd' -srcalias 1 -destalias cruise -deststorepass 'serverKeystorepa55w0rd' -destkeypass
'serverKeystorepa55w0rd' -destkeystore ./keystore -noprompt
```

NEW:

```
$ keytool -importkeystore -srckeystore /etc/letsencrypt/live/go.genivi.org/fullchain.pkcs12 \
-srcstoretype PKCS12 -destkeystore keystore -srcalias 1 -destalias cruise -srcstorepass 'serverKeystorepa55w0rd'-
deststorepass serverKeystorepa55w0rd -destkeypass serverKeystorepa55w0rd

$ ln -s /etc/letsencrypt/live/go.genivi.org/keystore
```

Finally I made the keystore file readable by the go user that runs the server. Let's do all the permissions to be sure:

Permissions: All directories leading up to /etc/letsencrypt/archive/go.genivi.org/keystore need to be readable by the **go** user.
I.e. if they are **root** owned, then rwx-r-xr-x. But all others we lock down for good measure.

Only **keystore** needs to be readable by Go Server (go user). All others can be chmoded to be **NOT** readable as a security precaution.

```
  $ chown -R root /etc/letsencrypt
  $ chmod -R 700 /etc/letsencrypt
  $ chmod 755 /etc/letsencrypt/ /etc/letsencrypt/live /etc/letsencrypt/live/go.genivi.org

  $ cd /etc/letsencrypt/live/go.genivi.org
  $ chmod 400 privkey* fullchain* chain* cert*

  $ chown go:go /etc/letsencrypt/live/go.genivi.org/keystore
  $ chmod 600 /etc/letsencrypt/live/go.genivi.org/keystore
```

Restarting go-server from inside the container is basically just to stop it because there is a monitor process "runsv" that restarts it right away if it stops.

Still inside container:

The service command was hanging for me sometimes. If that happens, kill the process instead. Start nicely using -15 signal

```
$ ps aux
```

Take note of the PID for the java process

Try:

```
$ service go-server stop
```

The service command was hanging for me sometimes. Give it time - e.g. 10 seconds. Otherwise try CTRL-C

Again check if the the process restarted (PID has changed).

If for some reason that did not work, kill the java process instead. Start nicely using -15 signal:

```
$ ps aux
```

```
$ kill -15 <pid>
$ kill -9 <pid>  (should NOT be needed)
```

Shortly after this, runsv has probably already restarted the server. Run ps again, and check that that PID has changed. If it has, then the server has restarted. Now give it some time, then try to access https://go.genivi.org

HTTP to HTTPS redirection

The following is just documenting what has already been done. Check that apache2 is still running however (see commands at bottom)

The last step was to disable HTTP usage on go.genivi.org but provide a friendly redirect.
1. Stop go-server from listening on port 80 (or actually, avoid forwarding the port - see below).
2. Run a server on port 80 that will redirect all HTTP requests to HTTPS.

To be specific actually I didn't modify Go.CD to avoid HTTP. Since this runs in a docker container, there is explicit port forwarding from host to container. Simply don't forward port 80 and the problem is solved.

1. This can be done in many ways but the easiest I found was to persist the state of the docker container, stop it and start it again from the snapshot that was created. Something like:

```
$ docker commit go-server go-server-snapshot
$ docker stop go-server
$ docker rename go-server go-server-old
```

and then run a **new instance** from the snapshot, **but we do not forward port 80** (to 8153) any longer. Only 443 to 8154.

I'll split the line with \ for easier reading:

```
$  docker run -ti --name go-server -v /gousers:/gousers \
   -v /mnt/go/go-server-artifacts:/var/lib/go-server/artifacts \
   -v /mnt/go/go-server-pipelines:/var/lib/go-server/pipelines \
   -p 8153-8154:8153-8154 -p <exposed host IP>:443:8154 go-server-snapshot
```

exposed host IP is in this case an internal address of the VM, which is later exposed to the internet, but that's beside the point.

The old docker container can now be removed.

2. (Redirect) : Apache was already installed on the host so I might as well use it:

- In **/etc/apache2/sites-enabled** removed the symlink to 000-default-le-ssl.conf (or maybe it was default-ssl.conf). This makes it... well... not enabled.
- Commented all lines that enable port **443** in ports.conf. Thus, apache only listens on port **80**
- Added the following lines to /etc/apache2/apache2.conf. Ref [3]

```
NameVirtualHost *:80
<VirtualHost *:80>
   ServerName go.genivi.org
   Redirect / https://go.genivi.org/
</VirtualHost>
```

(A cleaner way is likely to create a separate file in sites-*, but this worked).

This **redirects** all HTTP requests to their corresponding HTTPS, which the go-server will answer on.

Finally:

```
$ systemctl enable apache2
$ systemctl restart apache2
```