How Do I Configure SSL (HTTPS) in CouchDB?

Background

Secure Socket Layer (SSL) is used in conjunction with HTTP to secure web traffic. The resulting protocol is known as HTTPS. In order to utilize SSL, you must generate a key and cert. Additionally, if you want your web traffic to be safely accepted by most web browsers, you will need the cert to be signed by a CA (Certificate Authority). Otherwise, if you bypass the CA, you have the option of self signing your certificate.

Production Security

Apache CouchDB leverages Erlang/OTP's SSL, which is usually linked against a system-provided OpenSSL installation. The security, performance & compatibility with other browsers and operating systems therefore varies heavily depending on how the underlying OpenSSL library was set up. It is strongly recommended that for production deployments, a dedicated well-known SSL/TLS terminator is used instead. There is nothing fundamentally wrong with Erlang's crypto libraries, however a dedicated TLS application is generally a better choice, and allows tuning and configuring your TLS settings directly rather than relying on whatever Erlang/OTP release is provided by your operating system.

The following have been used at times within the CouchDB community:

- http://www.haproxy.org/
- https://github.com/varnish/hitch
- https://www.stunnel.org/
- http://nginx.org/
- https://github.com/bumptech/stud since deprecated
- https://httpd.apache.org/
- http://varnish-software.com/

Key & CSR Procedure using OpenSSL

OpenSSL is an open source SSL utility and library. It comes standard with many UNIX/LINUX distributions. We will use OpenSSL to generate our private key and generate our certificate signing request (CSR).

Generate Key with OpenSSL

The private key is a special file that must never be revealed to the outside world. It holds a secret that only our CouchDB server should see. CouchDB uses this secret to encrypt traffic and prove to clients that it is the rightful owner to the cert it holds.

```
# Performed on Ubuntu 14.04 with OpenSSL 1.0.1f
openssl genrsa -out server.key 2048
```

Generate Certificate Signing Request with OpenSSL

```
# Performed on Ubuntu 14.04 with OpenSSL 1.0.1f
$ openssl req -new -key server.key -out server.csr
```

During the signing process, you will be asked a number of questions. Most CA's will only require you to answer the following fields: country, state, locality, organizational name, common name. Other fields should be left blank

```
# Performed on Ubuntu 14.04 with OpenSSL 1.0.1f
$ openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Maryland
Locality Name (eg, city) []:Forest Hill
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Apache CouchDB Project
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:apache.couchdb.org
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Self Signed Certificates

If you don't require the SSL to be signed by a CA, you can self sign. This is a quick and dirty method that doesn't provide a lot of trust between different parties, but it is great for isolated systems.

```
# Performed on Ubuntu 14.04 with OpenSSL 1.0.1f
openssl x509 -req -sha256 -days 1095 -in server.csr -signkey server.key -out server.crt
```

Many browsers are now requiring SHA2 signed certs, which is controlled by the **-sha256** option in the command.

Requesting CA Signed Cert with Free Service (Comodo)

The most "official" way of obtaining a certificate is to find a recognized CA and request your CSR to be signed. For the sake of this tutorial, we will use a free service from Comodo: https://www.comodo.com/landing/ssl-certificate/free-ssl/

Go to this page: https://secure.instantssl.com/products/SSLIdASignup1a

Fill the form as follows:

1. Copy and paste your CSR into this box: **<PASTE CONTENTS OF SERVER.CSR HERE>**
2. Select the server software used to generate the CSR: **OTHER**
3. Select the hash algorithm you would prefer us to use when signing your Certificate: **SHA-2**

Click the "Next" button.

Continue follow the prompts so that email authorization is sent to the owner of the domain specified by the CSR (in our example, the domain was couchdb.apache.org, so the official contact for that domain will be looked up in DNS).

Once you finish proving that you have permission to represent the domain in question, you will eventually be emailed the certificate, along with the intermediate and root certificates.

Saving the Server Certificate

When your CA (in this case Comodo) emails you your certificate, it will sometimes appear in the body of the email. Copy this certificate, starting with "-----BEGIN CERTIFICATE-----" and ending with "-----END CERTIFICATE-----" and paste it into a file named "server.crt". This file represents a public signature derived from the private key we generated earlier (i.e. "server.key").

Creating the Chained CA Cert File

Some clients will require a chain of certs to clear a server cert (in our case, the server cert is the server.crt file). The Comodo CA will email you these files in an attached zip file. These files are the following:

1. COMODORSAAddTrustCA.crt

2. COMODORSADomainValidationSecureServerCA.crt
3. AddTrustExternalCARoot.crt

Combine the contents of these files into a single file, called "comodo.crt". NOTE: This is the only file in this tutorial that is not specific to our server. Other servers from Comodo will be using the same chained certificates to show the chain of authorization all the way up to the root server.

Making Accessible by CouchDB

In order for CouchDB to be able to access and use the certificates, you will need to make sure they are owned by the CouchDB user. Additionally, you will want to restrict other users from accessing these files by modifying the permissions to 600 (only owner can read and write).

Configuring CouchDB To Use The Certificates

Finally, we get back to our CouchDB server. Open up your local.ini file. On Ubuntu, this file can be found in /usr/local/etc/couchdb or /etc/couchdb/local.ini.

Add the following key value pairs to the specified sections:

```
[daemons]
httpsd = {couch_httpd, start_link, [https]}
[ssl]
port = 6984
key_file = /path/to/ssl/certs/server.key
cert_file = /path/to/ssl/certs/server.crt
# If self signed, the following file is not needed:
cacert_file = /path/to/ssl/certs/comodo.crt
```

Disabling CouchDB HTTP layer

It is not possible to disable the HTTP layer completely in CouchDB 2.3.1, so only allowing HTTPS access requires a firewall rule.

In some older versions of CouchDB 2.x (that have OTHER security issues, and we do NOT recommend you run these) you can disable port 5984 by amending `/usr/local/etc/couchdb/default.ini [daemons]` section accordingly:

/usr/local/etc/couchdb/default.ini

```
[daemons]
; find the following line and prepend a ; to comment it out
;httpd={couch_httpd, start_link, []}
```

Currently it is not possible to disable it in the more common `/usr/local/etc/couchdb/local.ini [daemons]` file.

Please watch this ticket for progress on restoring this functionality: https://github.com/apache/couchdb/issues/2106

Accessing and Verifying SSL

By default, CouchDB uses port 6984 for HTTPS. If our server lives locally, we can verify SSL works properly by accessing this URL:

https://localhost:6984/

Your web browser should show a lock icon next to the URL. If you click this lock icon, you should be able to see information about your certificate and the CA that signed it.

To verify your SSL is configured correctly, use one of the following sites:

- https://www.sslshopper.com/ssl-checker.html
- https://dev.ssllabs.com/ssltest/analyze.html
- https://www.howsmyssl.com/ will help you check what your browser is capable of doing
- https://github.com/drwetter/testssl.sh is also of use for automated or local testing

Issues and Problems

As CouchDB's TLS support is directly related to whatever is bundled with the OS (Erlang and OpenSSL library), at times these versions are too old to support the latest crypto deployed by web browsers and other tools.

For example, the process described in this tutorial has some issues when using CouchDB 1.6.1 on Ubuntu 14.04 LTS. When a CouchDB HTTPS page is viewed in Safari, it works fine. The sslshopper.com check works fine. However, when viewed in the latest builds of Firefox and Chrome, the page will not load. Firefox gives the following error:

> *Secure Connection Failed*
>
> *An error occurred during a connection to mydomainname.example.com:6984. The key does not support the requested operation. (Error code: sec_error_invalid_key)*
>
> *The page you are trying to view cannot be shown because the authenticity of the received data could not be verified. Please contact the website owners to inform them of this problem.*

On Chrome, the following issue is given:

> # *This webpage is not available*

> *The webpage at https://mydomainname.example.com:6984/ might be temporarily down or it may have moved permanently to a new web address.*
> *Error code: ERR_SSL_CLIENT_AUTH_SIGNATURE_FAILED*

This SSL problem does not occur in CouchDB 1.6.1 on Ubuntu 14.10.