YANGcore Installation Guide

Watsen Networks

July 8, 2025

Abstract

This documentation is for the YANGcore product by Watsen Networks.

This documentation is still a work-in-progress. Some sections clearly indicate when material is pending, but there are also some missing sections, and other sections may not have enough detail.

YANGcore is currently in its "pre-alpha" stage and details captured in this document may change.

Contents

1 Introduction

YANGcore is an application server that uses YANG to define an API contract and RESTCONF to present a REST API.

YANGcore is provided as software, providing both a CLI command called “yangcore” as well as a Python package that could be used in Python “import statements.

In any case, YANGcore presents a northbound API for configuration and monitoring, and West/East-bound hooks for integration with deployment-specific infrastructure.

This documentation is in preparation for a “1.0.0” release, using the common “major.minor.patch” semantic versioning convention. The use of versions like “0.0.N”, where ‘N’ is an integer, should be read as “the Nth Pre-Alpha, with no statement about if it contains”major” or “minor” changes.

2 Installation

YANGcore is installed using the command:

```
$ pip install yangcore
```

or, if Python 3 is installed separately:

```
$ pip3 install yangcore
```

On some systems it may be necessary to install the dependency pacakges first[1]. For the Ubuntu and MacOS platforms, the project’s GitHub Action workflow shows commands used in CI/CD..

2.1 Machine Resources

YANGcore has resource requirements as described in the following sections.

2.1.1 Processors

YANGcore runs as a single asynchronous I/O event driven process.

It is ideal to dedicate a whole core to the ‘yangcore’ process, thus, in addition to the operating system, the machine should have at least two processors.

It is additionally recommended for the processor to have a fast clock speed (e.g., in the 3 to 4 GHz range).

[1] The SQLite package in particular can be tricky to get installed.

2.1.2 Memory

YANGcore uses significant memory resources[2]. Sizing numbers haven't been measured yet. The following sections consider YANGcore's constant and transitive memory demands.

2.1.2.1 Constant Memory Demand

YANGcore has a constant memory demand that grows in proportion to configuration data.

YANGcore reduces configuration data both passively and actively:

- Passively, YANGcore's data model is highly normalized[3], thus elimanating the need for redundant definitions.
- Actively, binary/base64 values, not needed for YANG-validation, are converted to zero-byte values in memory. This is notable as the binary values tend to be large to very large in size.

2.1.2.2 Transitive Memory Demand

YANGcore has transitive memory demands when processing requests from clients. The highest memory-impacting request is a "PUT" request used to replace the entire YANGcore configuration. In this case, the transitive memory demand is proportional to the sum of the old and new configuration sizes.

2.1.3 Filesystem

Unless YANGcore is configured to use a file-based database, YANGcore has effectively no interaction with the filesystem, all configuration and logs are stored in the database. That said, here are some exceptions:

- YANGcore accesses the directories created when the 'yangcore' packages was installed (i.e., 'pip install'). This in the Python virtual environment ('pip show' location).
- YANGcore creates temporary files in the '/tmp' directory.
- It is expected that YANGcore's stdout/stderr will be piped to a file (e.g., /var/log/yangcore.log).

When using a file-based database, the database file grows proportionally to the size of the configuration plus operational state (e.g., audit logs, bootstrapping logs, etc.). Sizing numbers for this haven't been measured yet.

2.1.4 Network Interfaces

YANGcore's networking demands vary by interface, as different behavior is presented to Northbound clients and West/East-bound infrastructure.

While the Northbound API is exclusively a programmatic interface, it is expected to be used to drive human-facing interfaces and thus the speed of the network interface may matter.

The West/East-bound interfaces (e.g., used for logging, authenticating users, etc.) scale with the number of northbound interactions.

[2] And doubly so when using an in-memory database, though doing so in a production environment doesn't make sense.
[3] Normalized in that objects are defined once and thereafter can be referenced many times.

2.2 Operating System

YANGcore is offered as software that depends on Python 3.11 or 3.12. Any operating system that supports Python 3.11 or 3.12 above should be able to run YANGcore.

YANGcore is currently tested on Ubuntu 20.04, 22.04, and 24.04, and MacOS 14 and 15^[These operating systems are the GitHub-hosted "runners" (not including Windows). CentOS and OpenBSD worked once before, but have not been tried in awhile.

It is recommended to install YANGcore into a dedicated Python virtual environment, as opposed to using any Operating System specific preinstalled version of Python. That said, when YANGcore is installed on an ephmeral system, lasting mere minutes, using the preinstalled Python can be expediant[4].

2.3 Networking

YANGcore's networking usage is described in the following two sections. The first section regards the networking needs for inbound connections. The second section regards the networking needs for outbound connections.

2.3.1 Inbound Connections

YANGcore opens listening ports for only one reason:

1. To listen for connections from "northbound" clients (i.e., users) perhaps via an external web-based interface or orchestration/controller application[5].

[4]The project's GitHub Action workflow does this, creating first a runner to test coverage and generate a "matrix" of other runners, testing the many variations of operating system, Python version, database, etc.

[5]By default, as described in Defaults, YANGcore opens a single listening port for "northbound" connections. But is possible to configure YANGcore to open more than one listening port for Northbound access (e.g., one without TLS for internal use and one with TLS for external use.

2.3.1.1 APIs

Each listening port must specify which “interface” it presents. YANGcore defines interface as follows:

1. The “native” Interface

   The native interface presents an API enabling “northbound” clients to configure everything and anything that can be configured.

   There must always be extacly one “native” interface configured. The default listening port described in Defaults illustrates the “native” interface.

2.3.1.2 HTTP vs HTTPS

Each listening port must be configured to present the “HTTP” or “HTTPS” protocol. The default listening port described in Defaults presents the “HTTP” protocol.

When configured to use “HTTPS”, the YANGcore process terminates the TLS connections itself. By contrast, when configured to use “HTTP”, it is expected that an external endpoint (e.g., an HTTP proxy, such as NGINX) positioned in front of YANGcore will terminate the TLS connections.

Terminating TLS connections outside of YANGcore offloads some CPU load, which may improve performance.

When “HTTP” is used, YANGcore may be additionally configured with information about the external endpoint so that, e.g., when YANGcore sends callback information to remote peers, such as the hyperlinks in account-activation emails, it can address the URLs to the external endpoint instead of YANGcore’s local endpoint.

When “HTTP” is used, it is necessary for the external endpoint to send the client’s certificate in an HTTP header field called “X-Client-Cert” containing a url-encoded PEM. For instance, using NGINX, this is accomplished using the configuration line:

```
proxy_set_header X-Client-Cert $ssl_client_escaped_cert;"
```

2.3.2 Outbound Connections

YANGcore initiates outbound connections for the reasons discussed in this section.

2.3.2.1 Verifying certificate paths

YANGcore may need to verify certificate paths for the following reasons:

- Verifying a remote server’s certificate.
- Verifying certificate-based client credentials.

In order to properly verify certificate paths, it is sometimes necessary for YANGcore to check the current revocation status of certificate paths. Each certificate may specify URIs for where the CRL or OCSP can be obtained and, when needed, YANGcore will initiate a connection to the URI specified in the certificate.

2.3.2.2 Accessing a remote database

When configured to use a remote database (see Persistence Selection), YANGcore will initiate connections to the remote database, specified by the URI given on the command line.

While the database is “remote” to the ‘yangcore’ process it could be running on the same machine.

The remote port accessed varies by database and TLS configuration.

2.3.2.3 DNS resolution

When initiating any of the previously discussed outbound connections, the configurations for the remote host may be given as either an IP address or a hostname. When a hostname is provided, YANGcore resolves the hostname to IP addresses using the system configured DNS resolver. Depending on how

the DNS resolver is configured, the host system may initiate a connection to a network-based resolver. Traditionally this is a TCP-based connection destined to port 53.

2.4 Security

YANGcore includes a host of security features for both data in motion as well as data at rest.

2.4.1 Data in Motion

All of YANGcore's APIs are presented as mutually-authenticated HTTPS connections There is an ability to offload TLS termination to a TLS terminator (e.g., NGINX). In either case, the traffic to client endpoints is always protected by TLS.

Each request sent from clients is authorized against configured policy and a corresponding record is added to the audit log.

2.4.2 Data at Rest

YANGcore hashes passwords used to authenticate clients, but otherwise recommends database-level encryption, which is only possible when using an RDBMS-based database, to protect secret data stored as cleartext in the database. Such secret data includes unencrypted private keys and unencrypted passwords.

As mentioned in the previous paragrah, YANGcore hashes client passwords, specifically passwords used by 'users'. Passwords are hashed as described by the "iana-crypt-hash" module defined in RFC 7317. YANGcore uses the SHA-256 algorithm to hash passwords. For instance, when a clear-text input password value (e.g., "0<secret>") is configured, the value stored in the database is "5rounds=<rounds><salt><hash>".

2.5 The yangcore Executable

When the installation completes, the executable “yangcore” is installed in your shell’s path.

To test running YANGcore and see its “help” page:

```
$ yangcore --help
```

Which produces:

```
usage: yangcore [-h] [-v] [-C CACERT] [-c CERT] [-k KEY] database-url

YANGcore implements the "bootstrap server" defined in RFC 8572.

positional arguments:
  database-url          see below for details.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show version number and exit.
  -C CACERT, --cacert CACERT
                        path to certificates used to authenticate the database
                        (see below for details).
  -c CERT, --cert CERT  path to cert used to authenticate YANGcore to the
                        database (see below for details).
  -k KEY, --key KEY     path to key used to authenticate YANGcore to the database
                        (see below for details).

Exit status code: 0 on success, non-0 on error.  Error output goes to stderr.

The "cacert" argument is a filepath to a PEM file that contains one or more X.509
CA certificates used to authenticate the RDBMS's TLS certificate.

The "key" and "cert" arguments are each a filepath to a PEM file that contains
the key and certificate that YANGcore should use to authenticate itself to the
RDBMS.  These parameters must be specified together, and must be specified
in conjunction with the "cacert" parameter.

The "database-url" argument has the form "<dialect>:<dialect-specific-path>".
Three dialects are supported: "sqlite", "postgresql", and "mysql+pymysql".
The dialect-specific-path for each of these is described below.

For the "sqlite" dialect, <dialect-specific-path> follows the format
"///<sqlite-path>", where <sqlite-path> can be one of:

  :memory:    - an in-memory database (only useful for testing)
  <filepath>  - an OS-specific filepath to a persisted database file

  Examples:

    $ yangcore sqlite:///:memory:                          (memory)
    $ yangcore sqlite:///relative/path/to/yangcore.db      (unix)
    $ yangcore sqlite:////absolute/path/to/yangcore.db     (unix)
    $ yangcore sqlite:///C:\path\to\yangcore.db            (windows)

For both the "postgresql" and "mysql+pymysql" dialects, <dialect-specific-path>
follows the format "//<user>[:<passwd>]@<host>:<port>/<database-name>".

  Examples:

    The following two examples assume the database is called "yangcore" and
    that the database server listens on the loopback address with no TLS.

      $ yangcore mysql+pymysql://user:pass@localhost:3306/yangcore
      $ yangcore postgresql://user:pass@localhost:5432/yangcore

Please see the documentation for more information.
```


2.5.1 Persistence Selection

YANGcore persists all data into the database provided on the command line (i.e., the database URL passed into the 'yangcore' command)[6]. Varying the database URL enables use of different databases. Currently tested to work databases are SQLite and MySQL. Databases that haven't been tested in awhile include Postgres and MariaDB.

2.5.1.1 In-memory Database

The in-memory database type "persists" all data in memory, which is automatically lost when the 'yangcore' process ends. In-memory databases are fast, but demand additional process memory usage proportional to the size of the data stored.

An in-memory database is an excellent choice for development and test efforts, as it automatically disappears when the 'yangcore' process ends.

An in-memory database may also have use in a production environment (e.g., in an SDN context) whereby the intent is for the YANGcore instance to be ephemeral, lasting only as long as needed to bootstrap a specific set of devices.

Use of the in-memory database type is specified by the database URL provided on the command line having the form:

```
sqlite:///:memory:
```

2.5.1.2 File-based Database

The file-based database storage persists all the data into a single file located by the path-component in the database URL specified on the command line. File-based databases may be desirable in cases where persistence across power cycles is important and yet dependency on a remote RDBMS is not desirable.

The specified file may reside on a filesystem mounted from a RAID system providing resiliency against disk failures. Care should be to ensure fast and secure access to a remote system (e.g., a NAS or SAN).

The file itself may be backed-up and restored as needed to ensure disaster recovery. If the filesystem does not support taking snapshots while a system is running, it is recommended that the 'yangcore' process is either stopped or suspended during the backup operation.

Use of the file-based database type is specified by the database URL provided on the command line having one of the following form:

```
sqlite:///relative/path/to/yangcore.db      (unix)
sqlite:////absolute/path/to/yangcore.db     (unix)
sqlite:///C:\path\to\yangcore.db          (windows)
```

[6] YANGcore does not have or use any configuration files on the host filesystem.

2.5.1.3 RDBMS Database

The RDBMS database engine persists all the data into one of several well-known database systems (e.g., MariaDB, Postgres, Oracle, etc.). In theory, YANGcore can use any RDBMS supported by SQLAlchemy, though it is recommended to only use those that YANGcore has been tested against.

Use of an RDBMS database enables large data sets with fast-access to keyed data. Scaling of the database tier supports a variety of performance and availability targets. It is recommended to use database-level encryption to ensure to protection of the YANGcore data at rest.

Use of the RDBMS database type is specified by the database URL provided on the command line having the form:

```
<engine>://<user>:<passwd>@<host>:<port>/<database-name>
```

For instance:

```
postgresql://yangcore-admin:secret@db.example.com:5432/yangcore-db
mysql+pymysql://yangcore-admin:secret@db.example.com:3306/yangcore-db
```

The RDBMS database may run on the same machine as YANGcore or on another machine. If the database runs on the same machine, it could listen on the loopback address (e.g., 127.0.0.1) on thus not require network protection. However, if the database runs on another machine, even if in a "secure" or private network, it is recommended that the connection to a remote RDBMS is protected by transport level security (TLS), which entails both configuring the RDBMS server to listen for TLS connections and configuring YANGcore to connect to the the RDBMS using TLS.

To direct YANGcore to establish a TLS connection to the RDBMS, the "–cacert" must be specified. This parameter specifies the trust anchor certificate(s) that YANGcore must use to authenticate the RDBMS's end entity certificate. For example:

```
$ yangcore --cacert db-cacert.pem postgresql://user:pass@localhost:5432/yangcore
```

To direct YANGcore to authenticate itself to the RDBMS using a client certificate, the "–key" and "–cert" parameters must be specified. Note that these parameters must be specified in conjunction with the previously mentioned "–cacert" parameter. For instance:

```
[Note: '\' line endings per RFC 8792]

$ yangcore --cacert db-cacert.pem --key db-client-key.pem -cert db-client-cert.pem \
postgresql://user:pass@localhost:5432/yangcore
```

RDBMS-specific details are provided in the RDBMS Configuration section.

2.5.2 First-time Initialization

The first time YANGcore runs[7] the server will detect that the database is uninitialized and hence prompt for Contract acceptance.

2.5.2.1 Prompting for Contract Acceptance

Example output:

```
<Non-Production Use Contract>

First time initialization. Please accept the license terms.

By entering "Yes" below, you agree to be bound to the terms and conditions contained on this screen with Watso

Please enter "Yes" or "No": <YANGcore waits for input here>
```

[7] Really each time it runs against an uninitialized database, in case the database is ever reinitialized.

Entering "Yes" allows YANGcore to proceed. YANGcore will exit for any other entered response, not just "No".

Alternatively, the "YANGCORE_ACCEPT_CONTRACT" environment variable may be used to accept the contract. For instance:

```
$ export YANGCORE_ACCEPT_CONTRACT="Yes"; yangcore sqlite:///:memory:
```

2.5.3 Defaults

The follow sections describe defaults the freshly installed system uses.

2.5.3.1 Listening Ports

A freshly installed version will open a dual-stack (v4/v6) loopback port as follows:

- local address: 127.0.0.1 and ::1
- local port: 8080

Whilst the IP address bindings cannot be changed, due to security reasons[8], the port value may be changed using environment variable "YANGCORE_INIT_PORT". For instance:

```
[Note: '\' line endings per RFC 8792]

$ export YANGCORE_INIT_PORT="9090"; \
  yangcore sqlite:///:memory:
```

The default port presents the HTTP (without TLS) protocol over which the RESTCONF API is presented. A YANGcore-client must use this port to provide an initial configuration, including configuring YANGcore to open other listening ports and/or use TLS.

2.5.3.2 User Account

The default server (i.e., one that has never been configured) does not have any user accounts defined. A "user" here regards YANGcore's northbound interface.

Prior to configuring a user, no client authentication needs to be supplied. It's not possible, as none exists.

However, the very first configuration *write* request to a freshly installed YANGcore instance must configure at least one user account having unrestricted access to the system.

Additional users may be added/removed over time, but the software will ensure that there is at least one user having unrestricted access.

2.5.3.3 Validation

The default server will enforce validation of each NBI request that is a write-request (e.g., POST, PUT, DELETE, etc.). In some environments, if may be desirable to disable the validation checks in production code to optimize performance, knowing that the client-code never sends invalid data.

It is recommended to leave validation on while developing the client code.

To disable validation, the environment variable "YANGCORE_DISABLE_VAL" is used. It can be set to any value. Only its existance is checked. For instance:

```
[Note: '\' line endings per RFC 8792]

$ export YANGCORE_DISABLE_VAL="true"; \
  yangcore sqlite:///:memory:
```

[8]The default endpoints present the NBI with no user authentication required. Forcing the initial configuration to occur over the loopback interface ensures that the client must have access to the host operating system, which protects the NBI until authentication can be configured.

2.5.4 Daemonize

The 'yangcore' executable should be executed as a daemon, gracefully starting and stopping with the host system. Many systems use "rc" scripts located in '/etc/rc.d' or the like for this purpose.

2.5.5 Signals

The 'yangcore' executable responds the SIGNALS as follows:

- 'SIGHUP': YANGcore shuts down all ports and restarts.
- any other signal: YANGcore shuts down and exits.

Notably, YANGcore automatically sends itself a 'SIGHUP' signal whenever YANGcore's configuration for its NBI transport connection is updated (see the Administrator's Guide for the "RESTCONF Server" section under the "NBI Overview" section).

2.5.6 User Privileges

Unless YANGcore is asked to open a listening port below port number 1024, the 'yangcore' does not require any special user privileges (other than be able to read any input file and write any output files), and hence it is recommended to run the 'yangcore' executable using an unprivileged user account. Note that the 'yangcore' executable does not itself drop user privileges.

2.5.7 Dedicated Cores

YANGcore is a single-threaded asynchronous event-driven executable. While the CPU demand has yet to be measured, the demand is not expected to be excessive under realitic loads.

It is ideal if the 'yangcore' process is locked down to a core, thus avoiding delays introduced from process swapping. This means that the machine should have at least two cores, with the other core being for the underlying operating system and all of its applications.

2.5.8 Output

The 'yangcore' process does not produce any on-disk log files, all logs and other persisted data are held within the database (see Persistence Selection).

The 'yangcore' process itself may emit output to STDOUT and/or STDERR. For instance, if an unexpected condition is encountered, a Python stack trace is generated.

It is recommended to pipe the 'yangcore' process's output to a file. For instance:

```
yangcore sqlite:///:memory: >> /var/logs/yangcore.log 2>&1
```

2.6 High-Availability

It is recommended to have a cold standby instance on the ready in another geographic location, with active database replication configured between the two sites.

The "cold standby" system may be booted (i.e., the operating system is running), but the 'yangcore' process must not be started until it is intended to be the active system. This is necessary as YANGcore maintains a cache derived from the database contents that would be out-of-synch if the database contents were updated out-of-band. Note that YANGcore takes a couple seconds to startup on a fast CPU with a small database, how long it takes to startup with a large database has yet to be measured.

Database-level replication varies by database. The File-based Database type entails copying a file to the remote system. Replication using an RDBMS entails using the replication mechanism provided by that RDBMS.

Another form of high-availability can be had through RDBMS-based database clustering, which must be configured using the clustering mechanism provided by the RDBMS.

The cold-standby system must be running the same version of YANGcore to ensure seemless transfer.

2.7 Scaling Guidelines

Scale testing numbers have not been measured yet.

YANGcore by itself is not that interesting to garner much client use.

2.8 Upgrades

Disclaimer: a pip upgrade has never been attempted before. It is unknown if the text below is accurate. But the gist is on point regarding the potential need for the upgrade process to do an ETL (extract-transform-load) operation.

Upgrades will be distributed using 'pip' and hence an upgrade can be applied using the command:

```
pip install --upgrade yangcore
```

Before running this command, the 'yangcore' process should be shutdown[9] and a database backup is taken.

The first time YANGcore is run after an upgrade is installed, there may be a need for the database to be migrated[10], which may take some time depending on the nature of the upgrade and how much data there is in the database. If no database migration is required (typical case) then the firsttime running the upgraded code should take mere seconds to start.

[9] For instance, by sending the 'SIGTERM' signal to the 'yangcore' process. Alternatively, if using a process manager, via a command such as 'stop yangcore'.

[10] Data migration will never be needed for a patch release, or for an in-memory database.

3 RDBMS Configuration

This section provides details to set up a persistent relational data store for the YANGcore server. This document does not cover database configuration options in full detail and is not meant to replace a database server guide.

Generally you will want to configure the database server to have as much RAM as possible in order maximize the data and indexes cached in memory. This guide only regards initial database setup, but be advised that ongoing database tuning, optimization, and/or parameter resizing may be necessary as a database grows.

3.1 MySQL and MariaDB

YANGcore has been tested against MySQL 8.0.27 .

This document section uses "MySQL" interchangeably for MySQL on AWS EC2, AWS Aurora, AWS RDS MySQL, RDS MariaDB, MariaDB, Percona XtraDB and various MySQL forks and flavors.

3.1.1 Getting Started

3.1.1.1 Using a MySQL Instance Installed on the Same Machine

Assuming a freshly installed MySQL instance running on the same machine as YANGcore, with an admin account called "root" with no password set, the following commands might be used to initialize a user account for YANGcore:

```
$ mysql -u root -e "CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'secret';"
$ mysql -u root -e "GRANT ALL ON yangcore.* TO 'testuser'@'localhost';"
$ mysql -u root -e "FLUSH PRIVILEGES;"
```

Note: the grant above is to the yet-to-be-created "yangcore" database.

At which point YANGcore can be run as follows:

```
$ yangcore mysql+pymysql://testuser:secret@localhost:3306/yangcore
```

Note that YANGcore will itself create a database called "yangcore"[11] and initialize all the tables in it[12].

The following command returns the database to the state before YANGcore ran:

```
$ mysql -u root -e "DROP DATABASE yangcore;"
```

And this command returns the database to the state before creating the user:

```
mysql -u root -e "DROP USER 'testuser'@'localhost';"
```

Note that, in all of the commands above, the string "localhost" could have been replaced with the string "127.0.0.1". However, doing so switches from using a UNIX socket file to the TCP/IP stack. This may require setting the "bind-address" variable (e.g., in the "my.cnf" file) to "127.0.0.1" and restarting the MySQL server (i.e., 'mysql.server restart').

3.1.1.2 Using a MySQL Instance Installed on Another Machine

Assuming the MySQL instance is installed on another machine, the commands replace "localhost" with the appropriate IP addresses for the two machines.

For instance, assuming YANGcore is running on "1.1.1.1" and the MySQL instance is running on "2.2.2.2", the MySQL user-creation commands would be:

[11] The name of the database YANGcore creates/uses is specified at the end of the database URL string passed into the 'yangcore' command.

[12] YANGcore initializing the database only happens the firsttime it is run (i.e., when the database doesn't exist). Upon subsequent invocations, YANGcore will simply use the already created database, which is then presumably populated with data.


```
$ mysql -u root -e "CREATE USER 'testuser'@'1.1.1.1' IDENTIFIED BY 'secret';"
$ mysql -u root -e "GRANT ALL ON yangcore.* TO 'testuser'@'1.1.1.1';"
$ mysql -u root -e "FLUSH PRIVILEGES;"
```

And for YANGcore to connect to it, the command would be:

```
$ yangcore mysql+pymysql://testuser:secret@2.2.2.2:3306/yangcore
```

Note that, in order for the MySQL server to support accepting connections from remote systems, its "bind-address" variable (e.g., in the "my.cnf" file) must either be unset, or set to either a specific IP address (i.e., "2.2.2.2") or the wildcard address appropriate for the IP stack (e.g., "0.0.0.0" for IPv4). Be sure to restart the MySQL server afterwards (i.e., 'mysql.server restart').

3.1.2 Client Authenticating the Database's TLS Certificate

Assume the following certificate chain with associated keys and certificates for the MySQL server's certificate:

```
   Root CA        ->   Intermediate 1   ->   Intermediate 2   ->   Intermediate 3   ->   End Entity
(mysql-root.pem)       (mysql-int1.pem)       (mysql-int2.pem)       (mysql-int3.pem)       (mysql-ee.pem)
(mysql-root.key)       (mysql-int1.key)       (mysql-int2.key)       (mysql-int3.key)       (mysql-ee.key)
```

The chain is split as follows:

- Chain presented by MySQL: 'Intermediate 2 + Intermediate 3 + End Entity'
- Chain used by YANGcore to authenticate the MySQL server: 'Root CA + Intermediate 1'

The following commands setup some additionally needed files:

```
# cat mysql-root.pem mysql-int1.pem > mysqld-trust-anchor-certs.pem
# cat mysql-int2.pem mysql-int3.pem > mysqld-intermediate-cacerts.pem
```

3.1.2.1 Configure MySQL to Present the TLS Certificate

While it is possible to configure MySQL dyanmically, editing its configuration file (e.g., in the "my.cnf" file) is more understandable. Building on top of the previous example to bind the local address as well, the following is set:

```
[mysqld]
bind-address = 2.2.2.2

ssl_ca=mysqld-intermediate-cacerts.pem
ssl_cert=mysql-ee.pem
ssl_key=mysql-ee.key

require_secure_transport=ON
```

Be sure to restart the MySQL server afterwards (i.e., 'mysql.server restart').

3.1.2.1.1 Regarding AWS Aurora

AWS Aurora configures TLS by default, but it does NOT set the "require_secure_transport" variable. An equivalent setting is to use the "REQUIRE SSL" parameter on a per-user basis via in the "CREATE USER" clause 5.7 or "GRANT USAGE" clause in 5.6.

3.1.2.2 Configure YANGcore to Authenticate the MySQL Server's TLS Certifcate

YANGcore is "configured" to authenticate the server's TLS certificate solely on the command line[13]. Specifically:

```
$ yangcore --cacert mysqld-trust-anchor-certs.pem mysql+pymysql://testuser:secret@2.2.2.2:3306/yangcore
```

3.1.2.2.1 Regarding AWS Aurora

AWS Aurora instances use AWS-created certificates. It is thus necessary to pass the AWS-specific CA certificates file (e.g., rds-ca-2019-root.pem) via the "cacert" parameters. Please refer to this document to obtain the PEM file appropriate for your region.

3.1.3 Database Authenticating the Client's TLS Certificate

Assume the following certificate chain with associated keys and certificates for the YANGcore client's identity certificate:

```
 Root CA              ->     Intermediate 1   ->   End Entity
(yangcore-root.pem)        (yangcore-int1.pem)     (yangcore-ee.pem)
(yangcore-root.key)        (yangcore-int1.key)     (yangcore-ee.key)
```

The chain is split as follows:

- Certificate presented by YANGcore: 'End Entity'
- Chain used by MySQL to authenticate YANGcore: 'Root CA + Intermediate 1'

Note that MySQL client does not support sending a partial chain for the client certificate.

The following commands setup some additionally needed files:

```
# cat yangcore-root.pem yangcore-int1.pem > yangcore-trust-anchor-cacerts.pem

# cat yangcore-trust-anchor-cacerts.pem mysqld-intermediate-cacerts.pem > mysqld-all-cacerts.pem
```

Note that MySQL does not maintain separate files for its intermediate CA certificates and the CA certificates used to authenticate clients. It is for this reason that the two sets of certificates are merged above into the one file "mysqld-all-cacerts.pem".

[13] These command line parameters cannot be stored in the database as they are needed in order to establish a connection to the database, a catch-22

3.1.3.1 Configure MySQL to Authenicate TLS Clients

Building on top of the previous example, the "my.cnf" file is updated to:

```
[mysqld]
bind-address = 2.2.2.2

ssl_ca=mysqld-all-cacerts.pem
ssl_cert=mysql-ee.pem
ssl_key=mysql-ee.key

require_secure_transport=ON
```

Be sure to restart the MySQL server afterwards (i.e., 'mysql.server restart').

It is additionally necessary to change the "user" definition from before.

```
$ mysql -u root -e "CREATE USER 'testuser'@'1.1.1.1' REQUIRE X509;"
$ mysql -u root -e "GRANT ALL ON yangcore.* TO 'testuser'@'1.1.1.1';"
$ mysql -u root -e "FLUSH PRIVILEGES;"
```

Note that additional "REQUIRE" options exist to ensure that the client certificate contains a specific 'SUBJECT' and/or is signed by a specific 'ISSUER'. These tests are in addition to the MySQL server testing that the client certificate has a chain of trust to any of the CA certificates contained in the file specified by the "ssl_ca" variable.

It is also possible to create a user that requires both a client certificate and a password:

```
$ mysql -u root -e "CREATE USER 'testuser'@'1.1.1.1' IDENTIFIED BY 'secret' REQUIRE X509;"
$ mysql -u root -e "GRANT ALL ON yangcore.* TO 'testuser'@'1.1.1.1';"
$ mysql -u root -e "FLUSH PRIVILEGES;"
```

3.1.3.1.1 Regarding AWS Aurora

Client-certificate based authentication to an Aurora MySQL instance is not well supported.

AWS Aurora MySQL databases do not enable filesystem access to update the CA file pointed at by the "ssl_ca" variable set in the "my.cnf" file.

The command "SHOW VARIABLES LIKE '%SSL%';" suggests the that files might be found in "/rdsdbdata/rds-metadata/ca-cert.pem", but this folder doesn't appear to be accessble.

This AWS page states that the CA file may be updatable via an API call, but it is only available for MySQL 5.6.

This MySQL page states that it may be dynamically (i.e., via API) updated starting in release 8.0.16 that, at the time of this writing, is not yet supported by AWS Aurora.

3.1.3.2 Configure YANGcore to Send a Client Certificate to MySQL

YANGcore is "configured" to send a client certificate solely on the command line[14]. Specifically:

```
# NOTE: '\' line wrapping per RFC 8792

$ yangcore --cacert mysqld-trust-anchor-certs.pem --cert=yangcore-ee.pem \
       --key=yangcore-ee.key mysql+pymysql://testuser@2.2.2.2:3306/yangcore
```

Or, if also needing to pass a password:

```
# NOTE: '\' line wrapping per RFC 8792

$ yangcore --cacert mysqld-trust-anchor-certs.pem --cert=yangcore-ee.pem \
       --key=yangcore-ee.key mysql+pymysql://testuser:secret@2.2.2.2:3306/yangcore
```

[14] These command line parameters cannot be stored in the database as they are needed in order to establish a connection to the database.

3.2 PostgreSQL

3.2.1 Install and Start Postgres

SZTPD has been tested with Postgres 16.8.

3.2.1.1 Ubuntu

```
sudo apt install postgresql
sudo systemctl start postgresql
sudo systemctl status postgresql
```

3.2.1.2 MacOS

```
brew install postgresql@16
brew services start postgresql@16
brew services info postgresql@16
```

3.2.2 Create a user and grant permissions

The username can be any string, so long as the same value is used in the URL passed to the 'yangcore' executable. Here the username "testuser" with password "secret" is used.

3.2.2.1 Ubuntu

```
sudo -u postgres psql -c "CREATE ROLE testuser WITH LOGIN PASSWORD 'secret';"
sudo -u postgres psql -c "ALTER ROLE testuser CREATEDB;"
```

3.2.2.2 MacOS

```
export PATH="/opt/homebrew/opt/postgresql@16/bin:$PATH"
psql -d postgres -c "CREATE ROLE testuser WITH LOGIN PASSWORD 'secret';"
psql -d postgres -c "ALTER ROLE testuser CREATEDB;"
```

3.2.3 Install psycopg2

The psycopg2 utility is used by SQLAlchemy.

```
pip install psycopg2
```

The above command needs to access the 'pg_config' executable. On MacOS, set the 'PATH' as follows:

```
export PATH="/opt/homebrew/opt/postgresql@16/bin:$PATH"
```

3.2.4 Postgres Encryption Options

- https://www.postgresql.org/docs/current/encryption-options.html

3.2.5 Enabling TLS communication between PostgreSQL database and YANGcore server (THIS SECTION IS OLD!)

To enable TLS 1.2 with server certificate validation, edit Postgres server configuration parameters in the postgresql.conf file, which specifies server behavior with regards to auditing, authentication, encryption, and other behaviors. The postgresql.conf file usually resides in the data directory under your installation:

```
password_encryption = scram-sha-256     # md5 prior to 10 or scram-sha-256 post version 10

# - SSL -
ssl = on
ssl_ca_file = ''
ssl_cert_file = 'server.crt'
ssl_crl_file = ''
ssl_key_file = 'server.key'
ssl_ciphers = 'TLSv1.2:!aNULL' #'HIGH:MEDIUM:+3DES' or TLSv1.3
              or a list of ciphers but TLSv1.2 is a safe bet
ssl_prefer_server_ciphers = on
pg_ctl reload
```


If you have a Postgres release 11.4, set ssl_ciphers to TLSv1.2.

PostgreSQL release 12 contains two new server settings (ssl_min_protocol_version and ssl_max_protocol_version) that are are used to control the oldest (minimum) and newest (maximum) version of the SSL and TLS protocol family that the server will accept. Set these to TLSv1.2 and TLSv1.3 respectively.

Client authentication is controlled by a configuration file, which is named pg_hba.conf and is stored in the database data directory. Make sure tcp localhost connections are enabled in pg_hba.conf and modify your pg_hba.conf file to use scram-sha-256 algorithm. For more information, see

```
- https://www.postgresql.org/docs/12/auth-pg-hba-conf.html
```

```
TYPE  DATABASE    USER    ADDRESS    METHOD
local   all         all                  scram-sha-256
```

Procure the Certificate Authority (CA) signed certificate for the PostgreSQL database from the system administrator of your organization. Ensure that the certificate is in x509 format. For example, postgres.crt. Save the procured certificate file in the following locations:

YANGcore Server: /secure PostgreSQL Engine Server: /secure

- https://info.crunchydata.com/blog/how-to-upgrade-postgresql-passwords-to-scram
- https://github.com/MagicStack/asyncpg/blob/master/asyncpg/protocol/scram.pyx#L263

To start a PostgreSQL shell client connecting to the default Postgres database, type:

```
$ psql postgres
psql (12)
Type "help" for help.
```

You can start the server from a specific directory. To do this use the command and substitute in for the specified values:

```
pg_ctl -D [Data Directory] -l [Log file] start
```

The “Data Directory” refers to the directory that was just initialized. The “Log file” is a file that will record server events for later analysis. Generally log files are formatted to contain the date in the file name (e.g. “2018-05-27.log” or “myData-logfile-2018-05-27.log”) and should be stored outside of the database that they are logging so as to avoid unnecessary risks.

The server will only start if the port is free. If the default server is running it must first be stopped using command:

```
pg_ctl -D /usr/local/var/postgres stop
```

Once started, the database instance can be connected using an open source admin and development tool such as pgAdmin or simply a PostgreSQL shell:

```
$ psql --help
Connection options:
  -h, --host=HOSTNAME      database server host or socket directory (default: "local socket")
  -p, --port=PORT          database server port (default: "5432")
  -U, --username=USERNAME  database user name (default: "kristen")
  -w, --no-password        never prompt for password
  -W, --password           force password prompt (should happen automatically)

$ psql postgres
```

Once inside PostgreSQL shell, create a user account with a password and permission to create a database, as follow:

```
postgres=> CREATE USER my_user WITH LOGIN ENCRYPTED PASSWORD ''my_pass CREATEDB;
```


For more details, see https://www.postgresql.org/docs/9.1/sql-createrole.html

Exit out and login back in to verify that the user was created successfully

```
postgres=>exit
$ psql yangcore -U my_user
postgres=conninfo
You are connected to database "yangcore" as user "my_user" via socket in "/tmp" at port "5432".
```

Then you can start the YANGcore server, which creates a schema and populate seed data for the tables. While the YANGcore server is running and waiting for client connections, you will want to verify that the schema and tables have been created successfully.

```
yangcore=> dn
  List of schemas
  Name   |   Owner
---------+-----------
 public | kristen
 yangcore  | my_user
```

3.2.6 Enabling TLS communication between PostgreSQL database hosted on AWS and YANGcore server

The easiest way to set up the Postgres database is by using the AWS RDS service. This has been tested with Aurora RDS on a replicated database across east/west regions. When defining the AWS instance, you will need to use a parameter group that will specify that TLS must be used:

```
resource "aws_rds_cluster_parameter_group" "my_sztpd_secure_param_group" {
  name        = "sztpd-cluster-${data.aws_region.current.name}-param-group"
  family      = "${var.aurora_engine}${local.engine_major_version}"
  description = "SZTPD Aurora PostgreSQL Custom Parameter Group"

  parameter {
    name  = "rds.force_ssl"
    value = "1"
  }

  lifecycle {
    create_before_destroy = false
  }
}
```

Or in case you are using the AWS GUI directly, just set the force_ssl value to true.

AWS will then create the database with TLS encryption enabled, using one of the general rootCA for the corresponding region. The certificate bundle that will need to be passed to YANGcore will be the concatenation of all the general certs for the regions involved found in:

- https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/UsingWithRDS.SSL.html

Since the key for these certificates is not available, the the sslmode option for Postgres should be verify-ca, which is the default and it is hardcoded at today's date. If a custom root CA is used and the key is available, the server client cert and key could be generated and signed by that rootCA, and the sslmode option for Postgres could be set to verify-full. The different configuration options and parameters to configure TLS for the Postgres queries can be found here:

- https://www.postgresql.org/docs/current/libpq-ssl.html

The current available options for YANGcore are:

- sslrootcert : path to PEM file containing a list of X.509 certificate. This is going to be the concatenated certificate bundle for the AWS RDS instance with the different regions involved.
- sslcert : path to PEM file containing a X.509 certificate. This is going to be the certificate for the client.

- sslkey: path to PEM file containing a X.509 private key. This is going to be the private key for the client.
- sslmode: currently set to verify−ca, and at the moment, cannot be changed.

