SZTPD Installation Guide

Watsen Networks

August 27, 2020

Abstract

This documentation is for the "Secure ZTP Deamon (SZTPD)" 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.

SZTPD is currently in its "alpha" stage and details captured in this document may change.

Contents

1 Introduction

SZTPD is an implementation of a "bootstrap server", as defined in the Terminology section of RFC 8572, also known as an "SZTP server", as defined in the Terminology section of draft-kwatsen-netconf-sztp- csr".

SZTPD is provided as software, an asynchronous event-driven Python-based executable. SZTPD is an application (not a library) that includes a northbound API for configuration, a southbound API for device bootstrapping requests, and eastbound hooks for integration with other business systems.

This documentation is in preparation for a "1.0.0" release, using the common "major.minor.patch" semantic versioning convention.

2 Installation

SZTPD is installed using the command:

```
$ pip install sztpd
```

or, if Python 3 is installed separately:

```
$ pip3 install sztpd
```

On some systems it may be necessary to install the sqlite package first (e.g., 'yum install sqlite').

2.1 Machine Resources

SZTPD has resource requirements as described in the following sections.

2.1.1 Processors

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

It is ideal to dedicate a whole core to the 'sztpd' 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).

2.1.2 Memory

SZTPD uses significant memory resources[1]. The following sections consider SZTPD's constant and transitive memory demands.

2.1.2.1 Constant Memory Demand

SZTPD has a constant memory demand that grows in proportion to its configuration (minus all large base64 blobs stored in the configuration). Sizing numbers for this haven't been measured yet.

[1] And doubly so when using an in-memory database, though doing so in a production environment doesn't make sense.

2.1.2.2 Transitive Memory Demand

SZTPD has transitive memory demands when processing requests from clients. The highest memory-impacting request is a "PUT" request used to replace the entire SZTPD configuration. In this case, the transitive memory demand is proportional to the sum of the old and new configuration sizes. Sizing numbers for this haven't been measured yet.

2.1.3 Filesystem

Unless SZTPD is configured to use a file-based database, SZTPD has effectively no interaction with the filesystem, all configuration and logs are stored in the database.

That said, a filesystem must exist if the connection to the database is to be encrypted, as the requisit certificates are passed into the SZTPD executable as command line parameters as file paths. Additionally, it is expected that SZTPD's stdout/stderr will be piped to a file (e.g., /var/log/sztpd.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

SZTPD's networking demands vary by interface, as different APIs are presented for northbound administrators and southbound devices.

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 Southbound API is used exclusively for machine-to-machine interactions (i.e., SZTPD and the bootstrapping devices), and thus the speed of the network interface need only be fast enough to not cause timeouts. That said, the devices access the southbound interface when they are bootstrapping, which typically occurs immediately when being powered-up for the first time, which typically entails a human (the installer) manually plugging in a power cable. In some use cases, the installer will be instructed to wait for the device to provide some indication that the bootstrapping process has completed (e.g., an LED pattern, audible sound, etc.). From this perspective only, the speed of the network interface may matter[2].

2.2 Operating System

SZTPD is offered as software that depends on Python 3.7 or above. Any operating system that supports Python 3.7 or above should be able to run SZTPD.

Regardless the operating system selected[3], standard hardening guidelines should be observed (e.g., disable unnecessary processes, especially those that open listening ports of their own.

2.3 Networking

SZTPD'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] A profile of each kind of bootstrapping device is needed but, in general, device boot times may be an order of magnitude more time than any networking activity, thus the speed of the networking interface may be unimportant in the end.

[3] Some more so than others. For instance, OpenBSD is pretty secure out of the box.

2.3.1 Inbound Connections

SZTPD opens listening ports for only two reasons:

1. To listen for connections from "northbound" clients (i.e., administrators) perhaps via an external web-based interface or orchestration/controller application[4].
2. To listen for connections from "southbound" clients (i.e., the bootstrapping devices).

2.3.1.1 APIs

Each listening port must specify which "interface" it presents. SZTPD defines three interfaces as follows:

1. The "native" Interface

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

   The API the native interface presents varies slightly based on the product mode. That is, the configuration data model for "mode-1" and "mode-x" modes, as described in Prompting for Desired Mode Selection, are slightly different.

   There must always be extacly one "native" interface configured. The default listening port described in Defaults presents the "native" interface.

2. The "rfc8572" Interface

   The rfc8752 interface presents the "southbound" bootstrap server API defined in Section 7 of RFC 8572.

   In order for SZTPD to satisfy its business purpose, there must be at least one listening port presenting the "rfc8572" interface. More than one "rfc8572" interface may be configured if, e.g., distinct listening ports are needed to listen on different IP addresses and/or present different TLS certificates[5].

3. The "tenant" Interface

   The tenant interface is an additional "northbound" interface that is only useful for when the "mode-x" product mode is configured.

   The tenant interface presents a isolated view of each tenant's data. No tenant is aware of the existance of any other tenant, nor has access to any other tenant's data. The tenant interface presents an API that is almost exactly like the "mode-1" product mode's "native" interface.

   A single listening port presenting the "tenant" interface may be used to service a multiplicity of tenants. More then one "tenant" interface may be configured if, e.g., distinct listening ports are needed to listen on different IP addresses and/or present different TLS certificates[6].

[4] By default, as described in Defaults, SZTPD opens a single listening port for "northbound" connections.

[5] Additional listening ports may be used to present a distinct TLS end-entity certificate to different groups of devices, such as may be necessary in order to accommodate devices manufactured using different trust anchors.

[6] Additional listening ports may used to present a distinct TLS end-entity certificate to different groups of tenants. In particular, it is possible to use a certificate signed by a tenant-specific issuer.

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 SZTPD 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) positioned in front of SZTPD will terminate the TLS connections.

Terminating TLS connections outside of SZTPD offloads some CPU load, which may improve performance when the number of simultaneously bootstrapping devices grows to a large number.

When "HTTP" is used, SZTPD may be additionally configured with information about the external endpoint so that, e.g., when SZTPD 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 SZTPD'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

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

2.3.2.1 Sending Email

SZTPD sends email for the following purposes:

- Activating newly created administrator accounts.
- Sending account-deactivation warnings to account administrators.

In order to deliver the email, SZTPD, or the host operating system on its behalf, will initiate an outbound SMTP/S connection to a configured SMTP server. Traditionally this is a connection to port 25, 587, or 2525.

2.3.2.2 Sending Notifications

SZTPD sends notifications for the following reasons:

- Notification of device bootstrapping activity.
- Warning of pending expirations.
- Notification of an expiration.

In order to deliver notification, SZTPD uses mutually authenticated HTTPS connection to configured URIs. The default destination port for HTTPS is port 443, but the URIs may encode alternate port number values as needed.

2.3.2.3 Verifying Device Ownership

For deployments using the mode 'x' product mode, SZTPD enables the host system to ensure that the serial-numbers configured by tenants are rightfully owned by the tenant.

In order to verify device ownership, SZTPD initiates a mutually authenticated HTTPS connection to the URI configured when enabling this feature.

The default destination port for HTTPS is port 443, but the URIs may encode alternate port number values as needed.

2.3.2.4 Verifying certificate paths

SZTPD 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 SZTPD 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, SZTPD will initiate a connection to the URI specified in the certificate.

2.3.2.5 Accessing a remote database

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

While the database is "remote" to the 'sztpd' process it could be running on the same machine.

The remote port accessed varies by database and TLS configuration,

2.3.2.6 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 domain name. When a domain name is provided, SZTPD resolves the domain name 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

SZTPD 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 SZTPD's APIs are presented as mutually-authenticated HTTPS connections [7].

Two factor authentication (client certificate + password) may be configured.

Passwords, when used, may be required to have a minimal length[8].

Each request sent from clients is tested against access control and a corresponding entry is recorded in the audit log.

2.4.2 Data at Rest

SZTPD 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, SZTPD hashes client passwords, specifically the passwords used by SZTPD-administrators and the "authentication code" used by the bootstrapping devices, if configured. Passwords are hashed as described by the "iana-crypt-hash" module defined in RFC 7317. SZTPD 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 sztpd Executable

When the installation completes, the executable "sztpd" is installed in your shell's path.

To test running SZTPD and see its "help" page:

```
$ sztpd --help
```

Which produces:

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

SZTPD 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 SZTPD to the
                        database (see below for details).
  -k KEY, --key KEY     path to key used to authenticate SZTPD to the database
                        (see below for details).
```

[7] There is an ability to offload the TLS termination to a device fronting it (e.g., a firewall). In either case, the traffic to client endpoints is always protected by TLS.

[8] Note that long passwords, such as might better known as passphrases, are recommended. Setting the minimum length to, e.g., 25 characters encourages that best practice.


```
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 SZTPD 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:

    $ sztpd sqlite:///:memory:                      (memory)
    $ sztpd sqlite:///relative/path/to/sztpd.db     (unix)
    $ sztpd sqlite:////absolute/path/to/sztpd.db    (unix)
    $ sztpd sqlite:///C:\path\to\sztpd.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 "sztpd" and
    that the database server listens on the loopback address with no TLS.

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

Please see the documentation for more information.
```

2.5.1 Persistence Selection

SZTPD persists all data[9] into the database provided on the command line (i.e., the database URL passed into the 'sztpd' command). Varying the database URL enables use of different databases. Currently supported databases include Postgres, MySQL, MariaDB, and SQLite.

2.5.1.1 In-memory Database

The in-memory database type "persists" all data in memory, which is automatically lost when the 'sztpd' 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 'sztpd' process ends. The simulator (see the Administrator's Guide) uses the in-memory database for this reason.

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

[9] SZTPD does not use any configuration files of any sort.

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 'sztpd' 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/sztpd.db      (unix)
sqlite:////absolute/path/to/sztpd.db     (unix)
sqlite:///C:\path\to\sztpd.db          (windows)
```

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, SZTPD can use any RDBMS supported by SQLAlchemy, though it is recommended to only use those that SZTPD 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 SZTPD 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://sztpd-admin:secret@db.example.com:5432/sztpd-db
    mysql+pymysql://sztpd-admin:secret@db.example.com:3306/sztpd-db
```

The RDBMS database may run on the same machine as SZTPD 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 SZTPD to connect to the the RDBMS using TLS.

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


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

To direct SZTPD 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]

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

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

2.5.2 First-time Initialization

The first time SZTPD runs[10] the server will detect that the database is uninitialized and hence prompt for:

- Contract acceptance
- Desired mode selection.

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 Watsen N

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

Entering "Yes" allows SZTPD to proceed to prompting for the mode to run. SZTPD will exit for any other entered response, not just "No".

2.5.2.2 Prompting for Desired Mode Selection

SZTPD can run in one of two modes[11]:

Mode	Use Level	Purpose
1	Single Tenant	Ideal for single-domain deployments (e.g., enterprises).
x	Multi Tenant	Ideal to provide a self-service API for downstream customers.

The mode character represents, roughly, the number or tenants. For instance, '1' is for "one tenant" (i.e., you), while 'x' is for "many tenants" (i.e., you plus your customers).

```
Mode:
  1 - single-tenant
  x - multi-tenant

Mode: <SZTPD waits for input here>
```

Each of these modes are described in the following sections.

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

[11] Warning: As per the Release Notes, the 'product mode' concept may disappear in the future.

2.5.2.3 Mode '1'

Mode '1' enables a multiplicity of devices to be configured, but without an additional "tenants" layer. From an API perspective, the difference, in comparison to the mode 'x' tree diagram, is illustrated below.

Mode '1'

```
+--rw devices
   +--rw device* [serial-number]
      +--rw serial-number     string
      +-- ... // additional device parameters here.
```

2.5.2.4 Mode 'x'

Mode 'x' enables multi-tenancy, whereby each tenant effectively has their own Mode '1' instance, with the only aspect missing being the ability to configure system-level information (e.g., networking parameters).

The host view is not only a superset of all tenant views, but it also comes with the unique ability to configure plugins, define device-types, and define an "device ownership verification" callback function[12].

The difference in the APIs is illustrated below using tree diagrams (RFC 8340).

Mode 'x':

```
+--rw preferences
   +--rw device-ownership-verification!
   |  +-- ... // ownership verification parameters here
   +--rw plugins
      +--rw plugin* [name]
         +-- // plugin parameters here

+--rw device-types
  +--rw device-type* [name]
     +-- // device-type parameters here

+--rw tenants
   +--rw tenant* [name]
      +--rw name                    string
      +--rw device
         +--rw device* [serial-number]
            +--rw serial-number     string
            +-- ... // additional device parameters here.
```

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 single port by default.

This port uses the following parameters:

- local address: 127.0.0.1
- local port: 8080

[12]The "device ownership verification" callback function is used to ensure that the devices configured by tenants are rightfully owned by the tenants.

These values may be changed using environment variables:

- The default local address may be changed using the "SZTPD_DEFAULT_ADDR" environment variable.
- The default local port may be changed using the "SZTPD_DEFAULT_PORT" environment variable.

For instance:

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

$ export SZTPD_DEFAULT_ADDR="A.B.C.D"; \
  export SZTPD_DEFAULT_PORT="9090"; \
  sztpd sqlite:///:memory:
```

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

2.5.3.2 Admin Account

The default server (i.e., one that has never been configured) does not have any administrator accounts defined.

Prior to configuring an admin, no client authentication needs to be supplied.

The very first configuration write request must, at least or in addition to other changes, configure at least one administrator account and, further, this account must have the "unrestricted" access level.

2.5.4 Daemonize

The 'sztpd' 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 'sztpd' executable responds the SIGNALS as follows:

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

Notably, SZTPD automatically sends itself a 'SIGHUP' signal whenever the system-level "transport" configuration is updated (see the Administrator's Guide for information about "transport" configuration).

2.5.6 User Privileges

Unless SZTPD is asked to open a listening port below port number 1024, the 'sztpd' 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 'sztpd' executable using an unprivileged user account. Note that the 'sztpd' executable does not itself drop user privileges.

2.5.7 Dedicated Cores

SZTPD 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.

It is ideal if the 'sztpd' process is locked down to a core, thus avoiding delays introduced from process swapping. This means that the machine should have at least two core, at least on other being for the underlying operating system.

2.5.8 Output

The 'sztpd' process does not produce any log files, all audit logs and bootstrapping logs are held within the database layer (see Persistence Selection).

The 'sztpd' 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 'sztpd' process's output to a file. For instance:

```
sztpd sqlite:///:memory: >> /var/logs/sztpd.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 'sztpd' process must not be started until it is intended to be the active system. This is necessary as SZTPD 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 SZTPD 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.

2.7 Scaling Guidelines

Scale testing numbers have not been measured yet.

Prior deployment experience suggests low interaction rates on both the southbound and northbound interfaces, scaling proportionally to the number of bootstrapping devices. Prior deployment experience suggests that a high number of sustained interactions can be supported with SZTPD's current implementation strategy. Adjustments to the implementation will be made as needed to meet operator requirements.

This section will be updated when scale testing numbers become available.

2.8 Upgrades

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

```
pip install --upgrade sztpd
```

Before running this command, the 'sztpd' process should be shutdown[13] and a database backed up is taken.

When an upgrade is installed, there may be a need to migrate the database[14], which may take some time depending on the nature of the upgrade and how much data there is in the database. If not database migration is required (typical case) then the upgrade will take only seconds to complete.

3 RDBMS Configuration

This section provides details to set up a persistent relational data store for the SZTPD server. This document does not cover database configuration options in full detail and is not meant to replace a database server guide. This section uses the term "schema" to denote a set of database objects owned by a user account. This means a database may have more than one schemas similarly to Oracle RDBMS, where as in MySQL, a schema is referenced synonymously as a database.

Generally you will want to configure the database server to have as much RAM as possible and to fit as much data and indexes as possible in memory. We give some considerations towards an initial database set up but afterwards some database tuning, optimization, and parameter rezing would be necessary as a database grows.

3.1 MySQL and MariaDB

SZTPD has been tested against MySQL 8.0.19 .

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.

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

[14] Only occurring only on major SZTPD product version boundaries

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 SZTPD, with an admin account called "root" with no password set, the following commands might be used to initialize a user account for SZTPD:

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

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

At which point SZTPD can be run as follows:

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

Note that SZTPD will itself create a database called "sztpd"[15] and initialize all the tables in it[16].

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

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

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 SZTPD is running on "11.11.11.11" and the MySQL instance is running on "22.22.22.22", the MySQL user-creation commands would be:

```
$ mysql -u root -e "CREATE USER 'testuser'@'11.11.11.11' IDENTIFIED BY 'secret';"
$ mysql -u root -e "GRANT ALL ON sztpd.* TO 'testuser'@'11.11.11.11';"
$ mysql -u root -e "FLUSH PRIVILEGES;"
```

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

```
$ sztpd mysql+pymysql://testuser:secret@22.22.22.22:3306/sztpd
```

[15] The name of the database SZTPD creates/uses is specified at the end of the database URL string passed into the 'sztpd' command.

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

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., "22.22.22.22") 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 SZTPD 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 = 22.22.22.22

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 SZTPD to Authenticate the MySQL Server's TLS Certifcate

SZTPD is "configured" to authenticate the server's TLS certificate solely on the command line[17]. Specifically:

```
$ sztpd --cacert mysqld-trust-anchor-certs.pem mysql+pymysql://testuser:secret@22.22.22.22:3306/sztpd
```

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 SZTPD client's identity certificate:

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

The chain is split as follows:

- Certificate presented by SZTPD: 'End Entity'
- Chain used by MySQL to authenticate SZTPD: '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 sztpd-root.pem sztpd-int1.pem > sztpd-trust-anchor-cacerts.pem

# cat sztpd-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".

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 = 22.22.22.22

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').

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

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

```
$ mysql -u root -e "CREATE USER 'testuser'@'11.11.11.11' REQUIRE X509;"
$ mysql -u root -e "GRANT ALL ON sztpd.* TO 'testuser'@'11.11.11.11';"
$ 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'@'11.11.11.11' IDENTIFIED BY 'secret' REQUIRE X509;"
$ mysql -u root -e "GRANT ALL ON sztpd.* TO 'testuser'@'11.11.11.11';"
$ 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 may be dynamically (i.e., via API) updated stating in release 8.0.16 that, at this time, is not yet supported by AWS Aurora.

3.1.3.2 Configure SZTPD to Send a Client Certificate to MySQL

SZTPD is "configured" to send a client certificate solely on the command line[18]. Specifically:

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

$ sztpd --cacert mysqld-trust-anchor-certs.pem --cert=sztpd-ee.pem \
        --key=sztpd-ee.key mysql+pymysql://testuser@22.22.22.22:3306/sztpd
```

Or, if also needing to pass a password:

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

$ sztpd --cacert mysqld-trust-anchor-certs.pem --cert=sztpd-ee.pem \
        --key=sztpd-ee.key mysql+pymysql://testuser:secret@22.22.22.22:3306/sztpd
```

[18] 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

[Disclaimer: the section needs to be reviewed]

We installed and tested PostgresSQL release 11.4 or later. Once the installation is completed, start up a PostgresSQL instance, hosted locally on port 5432. The server will be run out of the directory /usr/local/-var/postgres.

By default, a Postgres installation has three databases defined template0, template1 and postgres. template0 and template1 are skeleton databases that are or can be used by the CREATE DATABASE command. postgres is the default database you will connect to before you have created any other databases. Once you have created another database you will want to switch to it in order to create tables and insert data. Often, when working with servers that manage multiple databases, you'll find the need to jump between databases frequently. This can be done with the connect meta-command or its shortcut c.

In the event that the postgres command is not found, you can locate it by issuing the 'locate bin/postgres' command:

```
$ locate bin/postgres
/usr/lib/postgresql/11.4/bin/postgres
```

Now with the direct path to the postgres utility, you can call it with the -V flag:

```
$ /usr/lib/postgresql/11.4/bin/postgres –V
```

To view the client version, again simply pass the -V flag to the psql client utility command:

```
psql –V
```

After verifying the PostgreSQL software, you can configure a different database server/data directory with a name of your choice - this example uses [Data Dirctory] as follow:

```
$ initdb [Data Directory] –E utf8
```

Start manually with the database directory in /usr/local/var/postgres:

```
$ pg_ctl –D /usr/local/var/postgres start
```

Stop manually:

```
$ pg_ctl –D /usr/local/var/postgres stop
```

Stop manually with launchd, which starts postgresql now and restarts at login:

```
$ brew services restart postgresql
```

Check for error messages in the server.log and verify that postgres is in /var/lib/pgsql and a running process with "ps -ef | grep postgres" (its on port 5432).

3.2.1 Postgres Encryption Options

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

3.2.2 Enabling TLS communication between PostgreSQL database and SZTPD server

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:

SZTPD Server: /secure PostgresSQL 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 PostgresSQL 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 PostgresSQL 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 sztpd -U my_user
postgres=conninfo
You are connected to database "sztpd" as user "my_user" via socket in "/tmp" at port "5432".
```

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

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

