Optimization Guide

intel.

3rd Generation Intel® Xeon® Scalable Processors

Nginx* HTTPs with Intel® QAT Tuning Guide on 3rd Generation Intel® Xeon® Scalable Processors

Revision Record

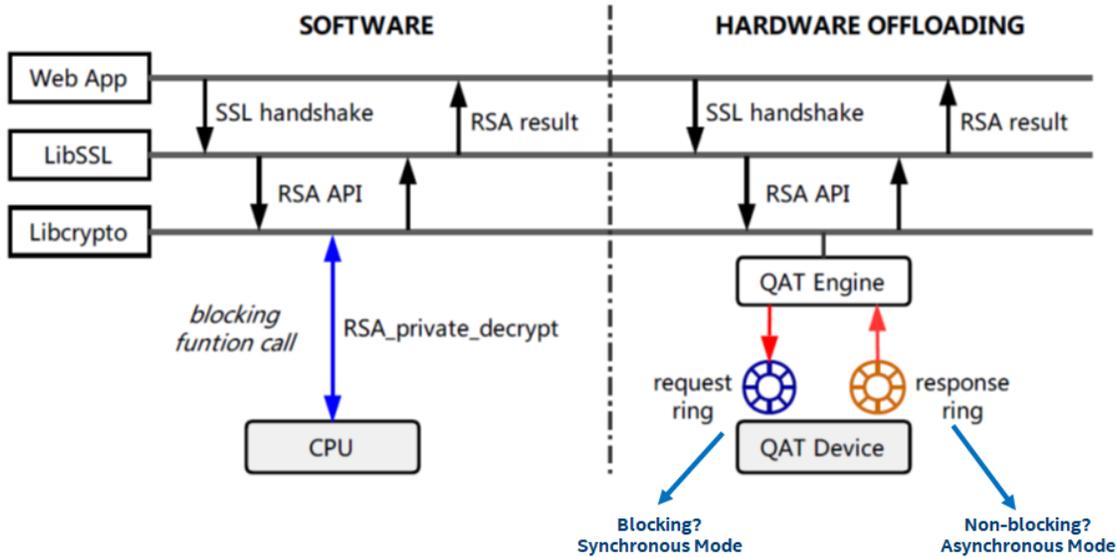| Date	Rev.	Description
05/27/21	1.1	Added conclusion and feedback sections
4/22/2021	1.0	Initial public release

1. Nginx* Introduction

Developed by Igor Sysoev, a Russian software engineer, Nginx* is a high-performance HTTP and reverse proxy web server based on a BSD-like license. It also provides IMAP/POP3/SMTP services in the meantime. Since the release of the first version in 2004, its market penetration rate has increased year by year, and it has been widely applied in many front-line Internet companies and IT enterprises. Nginx's architecture design is very flexible, with a very small and simple kernel containing core modules, basic modules, and tripartite modules. It collaborates with modules through file static mapping and configurable instructions, highlighting significant advantages of high performance, high concurrency, and low memory in various application scenarios such as HTTP proxy, static and dynamic separation, load balancing, virtual host, reverse proxy, cache acceleration, authorized access, and others.
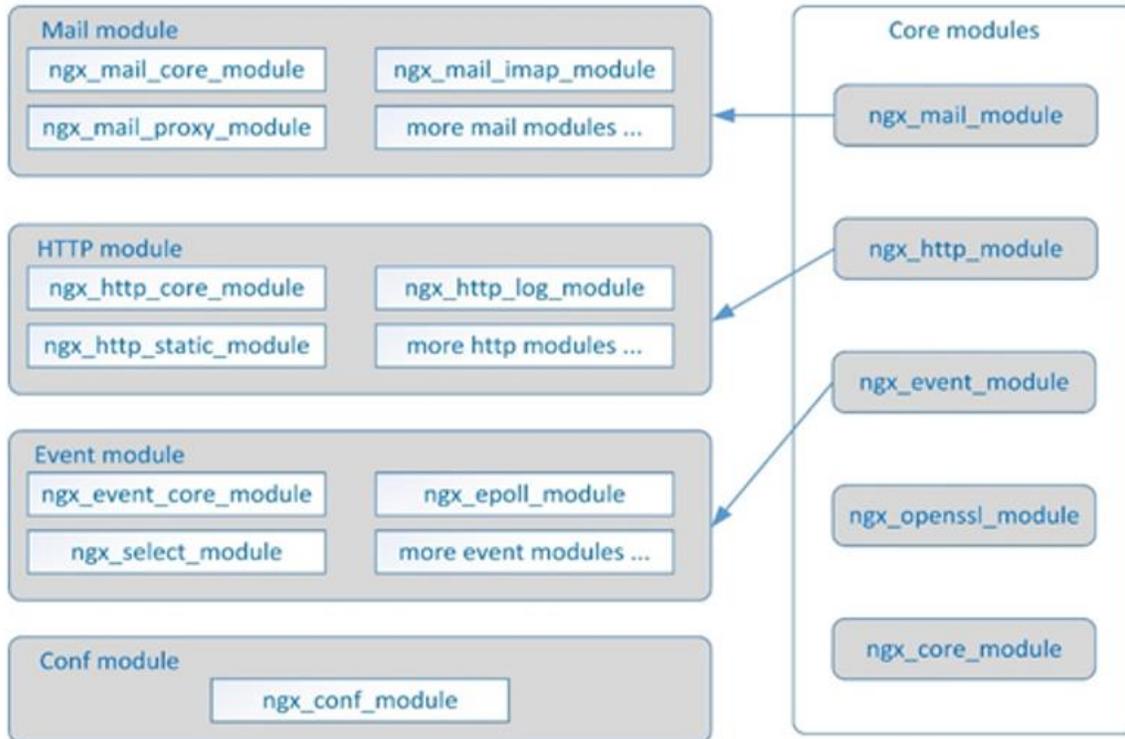
For more information about Nginx, please visit: **http://nginx.org**.

Nginx uses SSL/TLS to enhance web access security. Intel has introduced the Crypto-NI software solution which is based on 3rd generation Intel® Xeon® Scalable Processors (Codename Ice Lake/Whitley). It can effectively improve the security of web access
.
Crypto-NI (NI stands for New Instruction) is a new instruction set in the field of encryption and decryption for 3rd generation Intel® Xeon® Scalable Processors and it adds new instructions such as Vectorized AES and Integer Fused Multiply Add on the basis of the Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI) that the Intel® Xeon® Scalable Processors already has. The main software used in this solution are IPP Cryptography Library, Intel Multi-Buffer Crypto for IPsec Library (**intel-ipsec-mb**) and Intel® QuickAssist Technology (Intel® QAT), which provide batch submission of multiple SSL requests and parallel asynchronous processing mechanism based on the new instruction set, greatly improving the performance.

For details on proprietary Intel® QAT hardware, please see: https://01.org/sites/default/files/downloads/intelr-quickassist-technology/337003-001-intelquickassisttechnologyandopenssl-110.pdf

Tested hardware and software environment for this tuning guide:

Server Configuration	Hardware	CPU	Intel® Xeon® PLATINUM 8360Y CPU @ 2.20GHz
		Memory	16*32 GB DDR4, 3200 MT/s
		Hard Drive	Intel SSD S4610, 960G
		NIC	Intel® Ethernet Controller XXV700 25GbE SFP28
		Intel QAT card	Intel® QuickAssist Accelerator8970
	Software	Operating System	CentOS* 7.8
		Kernel	3.10.0-1127.el7.x86_64
		Nginx	asynch_mode_nginx v0.4.4
		OpenSSL	v1.1.1i
		Intel QAT driver	QAT v1.7L4.12.0. 00011
		Intel QAT Engine for OpenSSL	v0.6.4

Notes:

- The configuration described in this article is based on Intel® QuickAssist Accelerator hardware. Memory, hard drives, network interface cards can be determined according to customer usage requirements.

- For the software listed in configuration, you can refer to **https://github.com/intel/QAT_Engine.**   The oldest versions of each software supported are described on the page the link. The software versions listed in this article meet the above requirements and have been verified and tested.
- **Asynch_mode_nginx** is an optimized version of Nginx, used by Intel to support Nginx hardware and software acceleration. It can be compiled directly after download. For related descriptions, please refer to the following link: **https://github.com/intel/asynch_mode_nginx**
- **Intel QAT Engine for OpenSSL** link: *https://github.com/intel/QAT_Engine.*

2. BIOS Settings

The BIOS configuration items that can be optimized and their recommended values are as follows:

Configuration item	Recommended value
Hyper-Threading	Enable
CPU C6 report	Auto
SpeedStep (Pstates)	Enable
Turbo Mode	Enable
PCIe Link Speed	Gen4
Energy Efficient Turbo	Disable
Boot Performance Mode	Max Performance

3. Linux* Optimization

1. Set the corresponding parameters for system startup

```
intel_iommu=off
processor.max_cstates=1 idle=poll pcie_aspm=off
```

2. Stop **cpupower** service

```
systemctl stop cpupower.service
```

3. Disable the firewall

```
systemctl disable firewalld.service
```

4. User process settings

```
ulimit -c unlimited #generate core dump
ulimit -n 1000000 #set the maximum number of file open handles
```

4. Intel® QuickAssist Accelerator installation and Settings

Intel® QuickAssist Accelerator is a PCIe card that needs to be inserted into the PCIe slot in the server at the start. Below is a picture of the Intel® QuickAssist Accelerator card.

After the card is installed, you need to go to the BIOS settings and set the **PCIe link speed** to **Gen4 and X16** for highest possible performance.

You can use the following link to download the Intel® QuickAssist Accelerator driver, before you extract, configure and install it. In this article, **/QAT** is used as the installation directory for the driver: **https://01.org/intel-quickassist-technology/intel-quick-assist-technology/downloads/intel%C2%AE-quickassist-technology-driver-linux-hw-version-1.7**

```
$cd /
$mkdir QAT
Copy qat1.7.l.4.12.0-00011.tar.gz to the /QAT directory
$cd /QAT
$ar xvzf ./qat1.7.l.4.12.0-00011.tar.gz
$./configure
$./make install
```

After the driver installation is completed, you will see at the command output that three QAT acceleration devices (**qat_dev**) are in the startup mode.
Next, use the following command to start the QAT service.

```
$service qat_service start
```

5. OpenSSL* and Intel QAT Engine for OpenSSL installation

5.1. Download **OpenSSL** Source Code

```
$ cd /usr/local/src
$ wget https://www.openssl.org/source/openssl-1.1.1i.tar.gz
```

5.2. Build and Install OpenSSL

```
$ tar xvzf openssl-1_1_1j.tar.gz
$ cd ./openssl-1_1_1_1i
$./config --prefix=/usr/local/ssl -Wl,-rpath, /usr/local/ssl/lib
#In this article, openssl is installed in /usr/local/ssl
$ make
$ make install
```

5.3. Download Intel QAT Engine for OpenSSL Source Code

```
$ export OPENSSL_ENGINES=/usr/local/ssl/lib/engines-1.1
$ cd /usr/local/src
$ git clone https://github.com/intel/QAT_Engine.git
```

5.4. Build and Install the Intel QAT Engine for OpenSSL

```
$ cd ./QAT_Engine
$./autogen.sh
$./configure \
         -- with-qat_dir=/QAT \
         -- with-openssl_install_dir=/usr/local/ssl
$ make
$ make install
```

5.5. Replace the QAT Service Configuration File.

To enable QAT Engine to work with the QAT service, it is also necessary to copy the instance configuration file in the QAT Engine directory to the **/etc** directory and replace the original QAT configuration file.

```
$ cd /usr/local/src/QAT_Engine/qat/config/c6xx/multi_process_event-driven_optimized
$ cp * /etc/ (before performing this operation, the original QAT configuration file can be
renamed. For example, c6xx_dev0.conf can be renamed as c6xx_dev0.conf.bak)
```

After replacing the configuration file, restart the QAT_service to load the new configuration file:

```
$ service qat_service restart
```

Use OpenSSL to load QAT Engine:

```
/usr/local/ssl/bin/openssl engine -t -vvv qatengine
```

5.6. Using the Openssl Speed Command

With the **Openssl speed** command, user can verify view and verify information before or after using the **QAT Engine** .

```
$/usr/local/ssl/bin/openssl speed rsa2048
$/usr/local/ssl/bin openssl speed -engine qatengine -async_jobs 8 rsa2048
```

6. Nginx installation and Optimization Settings

6.1. Nginx Installation

1. Complete the installation and configuration of Intel® QAT, OpenSSL and Intel® QATEngine for OpenSSL.

2. Download Nginx source code

```
$ git clone https://github.com/intel/asynch_mode_nginx.git
```

3. Build and install Nginx

```
$ cd asynch_mode_nginx
$ export NGINX_INSTALL_DIR=<Nginx installation directory>
$ export OPENSSL_LIB=<SSL installation directory>
$./configure \
        --prefix=$NGINX_INSTALL_DIR \
        --with-http_ssl_module \
        --add-dynamic-module=modules/nginx_qat_module \
            --with-cc-opt="-DNGX_SECURE_MEM -I$OPENSSL_LIB/include -Wno-error=deprecated-
declarations" \
            --with-ld-opt="-Wl,-rpath=$OPENSSL_LIB/lib -L$OPENSSL_LIB/lib -lz"
$ make
$ make install
```

4. Start Nginx services

After completing the Nginx settings, execute the **$ ./sbin/nginx** command in the Nginx installation directory to start the Nginx service.

6.2. Optimized Settings for Nginx

1. Generate RSA self-signed certificate and key file.

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out
server.crt -subj "/C=CN/ST=Beijing/L=Beijing/O=Example Inc./OU=Web Security/CN=example.com"
```

Create a new cert directory under the Nginx installation directory and copy **server.crt** and **server.key** to that directory.

2. Key Nginx configuration. Set the following parameters in the **nginx.conf** file:
   - Set the number of worker processes and event mode

```
worker_processes <number of worker processes>; #It is recommended to have the same number
of worker processes as the number of CPU cores in server
events {
      use epoll; #epoll is an enhanced version of select/poll, and it improves the
processing of a large number of file descriptors
      …
}
```

   - Load and set up the SSL Engine module

```
load_module modules/ngx_ssl_engine_qat_module.so;
ssl_engine {
      use_engine qatengine;
            default_algorithms ALL;
            qat_engine {
                  qat_offload_mode async;
                  qat_notify_mode poll;
                  qat_poll_mode heuristic;
                  qat_shutting_down_release off;
            }
}
```

- Set HTTPS Server

```
server {
            Listen 443 ssl reuseport so_keepalive=on... ; #if you want to verify the SSL
handshake performance, then set it to off.
            sendfile on;
            Server_name example1.com; #consistent with certificate CN
            ssl_asynch on;
            ssl_session_tickets on; #set it to off, if you want to verify the SSL
handshake performance
            ssl_certificate <Nginx installation directory>/cert/server.crt;
            ssl_certificate_key <Nginx installation directory>/cert/server.key;
            ssl_ciphers         AES256-SHA:AES128-SHA;
    #You can set other cipher suites according to customer needs, to verify the performance
of different authentication, encryption and decryption protocols
            ssl_prefer_server_ciphers on;
            …
}
```

3. A complete sample configuration of nginx.conf, which is suitable for RSA2K handshake performance test:

```
user root;
worker_processes 8;
load_module modules/ngx_ssl_engine_qat_module.so;
events {
    use epoll;
    worker_connections 8192;
    multi_accept on;
    accept_mutex on;
}

ssl_engine {
    use_engine qatengine;
    default_algorithms ALL;
    qat_engine {
        qat_offload_mode async;
        qat_notify_mode poll;
        qat_poll_mode heuristic;
        qat_shutting_down_release off;
    }
}
http {
    server {
        listen example:443 ssl reuseport backlog=131072 so_keepalive=off rcvbuf=65536
sndbuf=65536;
        keepalive_timeout 0s;
        ssl_verify_client off;
        ssl_session_tickets off;
        access_log  off;
        ssl_asynch on;
        ssl_session_timeout 300s;
        ssl_protocols TLSv1.2;
        ssl_ciphers  AES128-SHA;
        ssl_prefer_server_ciphers on;
        ssl_certificate     server2048.crt;
        ssl_certificate_key server2048.key;
        location / {
            root html;
            index  index.html index.htm;
        }
    }
```

```
}
```

7. Introduction to Performance testing tools

In this section, we will introduce two web server stress testing tools: **wrk** and **ab**.

- **Wrk** is a benchmark testing tool for **HTTP/HTTPS** protocols. On a single machine powered by multi-core CPUs, it can generate a huge workload on the target machine in multi-threading and event I/O modes, using the system's high-performance I/O mechanisms, such as **epoll** and **kqueue**.

  Usage Example: *(where users would replace the URL with their own Nginx URL)*
  ```
  $ wrk -t 10 -c 1000 -d 30S https://example1.com:443/index.html
  ```
  Description: the command uses 10 threads and 1,000 concurrent connections to fetch files and stress test the server for 30 seconds. For more information, please visit **https://github.com/wg/wrk**.

- The **ab** tool is a stress testing tool that comes with Apache*. It can be used not only for website access stress testing of Apache servers, but also for stress testing of other types of servers. For example, **Nginx**, **Apache Tomcat***, and **IIS**. If the user has installed Apache, the **ab** tool has already been installed with Apache. If the user has not installed Apache, then it can be installed easily with the following command (**CentOS***):
  ```
  $ yum -y install httpd-tools
  ```

  Usage Example: *(where users would replace the URL with their own Nginx URL)*
  ```
  $ ab -n 100000 -c 100 -Z AES128-SHA -f TLS1.2 https://example1.com:443/index.html
  ```
  Description: The command uses 100 concurrent connections to send 100,000 requests for "get index" files to perform stress testing on the server. For more information, please visit: **https://httpd.apache.org/docs/2.4/programs/ab.html.**

8. Conclusion

We understand every application is unique. We shared many of our experiences with NGINX HTTPs with QAT hoping that some of our learnings could be applied to your specific application. NGINX HPPTS with QAT has been well tested on Intel platforms. With 3rd Generation Intel® Xeon® Scalable processor, Intel takes it even further by optimizing the platform as a whole -- CPU, memory, storage, and networking working together for the best user experience.

9.   Feedback

We value your feedback. If you have comments (positive or negative) on this guide or are seeking something that is not part of this guide, please reach out to us here:
https://community.intel.com/t5/Software-Tuning-Performance/bd-p/software-tuning-perf-optimization