Using Intel® QuickAssist Technology in Linux* Container and Docker*

Application Note

March 2021

Intel Confidential

Document Number: 337020-002

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/benchmarks.

*Other names and brands may be claimed as the property of others.

Contents

Figures

Tables

Revision History

Date	Revision	Description
March 2021	003	Update for unprivileged LXC
November 2020	002	Update for latest driver
January 2018	001	Initial release.

§

1 Introduction

1.1 About this Document

This document discusses the following topics related to using the Intel® QuickAssist Technology Software in Linux* container or Docker*:

- Software requirements
- Build and installation

Users of this document are expected to be familiar with virtualization technologies, like VT-D, SR-IOV, LXC (Linux* container) and Docker*.

For convenience, this document uses *acceleration drivers* as a generic term for the software that allows the QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator(s) integrated in the Intel® QuickAssist Technology.

1.2 Terminology

Table 1. Terminology

Term	Description
IOMMU	Input/Output Memory Management Unit
LXC	Linux* containers
PF	Physical Function
QAT	Intel® QuickAssist Technology
RHEL	RedHat* Enterprise Linux*
SR-IOV	Single-Root Input/Output Virtualization
UIO	Linux* User Space Input/Output System

1.3 Documentation

1.3.1 Where to Find Current Software and Documentation

Associated software and collateral can be found on the open source website: https://01.org/intel-quick-assist-technology

1.4 Reference Documents

Table 2 includes a list of related documentation.

Table 2. Reference Documents

Document Title	Document No./Location
Intel® QuickAssist Technology Software for Linux* Getting Started Guide - HW version 1.7	336212
Intel® QuickAssist Technology Software for Linux* Programmer's Guide - HW version 1.7	336210
Intel® QuickAssist Technology API Programmer's Guide	330684
Intel® QuickAssist Technology Cryptographic API Reference Manual	330685
Intel® QuickAssist Technology Data Compression API Reference Manual	330686
Intel® QuickAssist Technology Software for Linux* Release Notes - HW version 1.7	336211
Intel® Communications Chipset 89xx Series Datasheet	327879
Intel® C620 Series Chipset Datasheet	336067
Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note	330689
Intel® QuickAssist Technology Driver for Linux* - HW version 1.7	01.org

1.5 Documentation Conventions

The following conventions are used in this manual:

- `Courier font` - code examples, command line entries, API names, parameters, filenames, directory paths, and executables.
- Red text: Numbers related to system performance.

1.6 Software Requirements

Intel® QuickAssist Technology Software for Linux* - HW version 1.7 (l.4.11.0-00001) or later is required. Other software requirements will vary according to the particular use case.

Intel recommends using the latest version of the QuickAssist driver on your platform. Users might experience errors during installation or run-time use. Consult your Intel representative if you have a requirement to use another version of the driver.

SR-IOV may not work on GNU*/Linux* kernel versions older than 2.6.38.

These instructions were tested against the following Linux* distribution:

- CentOS* 7.8.2003 64-bit version, Kernel: GNU*/Linux* 3.10.0-1127.19.1.el7.x86_64

§

2 Installing Intel® QuickAssist Technology Software

To enable an Intel® QuickAssist Technology (QAT) acceleration device within a Linux* container or Docker*, the Intel® QAT software must be installed on the host. Single-Root I/O Virtualization (SR-IOV) can be enabled or disabled during installation.

SR-IOV enables the Linux* operating system to create multiple virtual functions on a single Intel® QAT acceleration device to support acceleration for multiple Linux* containers or Dockers*.

It is also possible to share one or more devices with accelerator capabilities simultaneously among multiple Linux* containers or Dockers* as well as the host. The following sections describe the steps necessary to install the Intel® QuickAssist Technology driver for both SR-IOV enabled or disabled use cases.

2.1 Installing Intel® QAT Software on Host with SR-IOV/IOMMU disabled

If you are not using SR-IOV and trying to enable a Physical Function (PF) for acceleration services for the Linux* host, Linux* container or Docker*, it is very straightforward to install the Intel® QAT Software package on the host. This section describes how to install the driver software on the host with only the PF enabled.

2.1.1 Updating the BIOS Setting

Before installing Intel® QAT software, update the BIOS to the latest stable version for your platform. Perform the steps in this section to ensure a smooth installation and validation.

1. Reset all BIOS settings to their default.
2. Disable all power saving options such as: Power performance tuning, CPU P-State, CPU C3 Report and CPU C6 Report.
3. Disable all virtualization options like VT-D and SR-IOV.

Note: Some example BIOS virtualization options are listed below. Yours may vary according to your vendor.

Advanced > System Agent (SA) Configuration > SRIOV

Advanced > System Agent (SA) Configuration > VT-D

4. Set the PCIe links to the highest possible speed, e.g., PCIe* Gen3 instead of PCIe Gen2 or Gen1.
5. Ensure that the PCIe links have trained to the expected width, e.g., x8 or x16.

2.1.2 Installing Intel® QAT Software

1. Change the current directory to the directory where you want to install the QAT software (for example, `/QAT`), referred to as <QATdir> in this document.
2. Set the following environment variable:
   `export ICP_ROOT=`<QATdir>
3. Unpack the Intel® QAT software and run the following commands to build and install the driver on the host:
   ```
   # tar -zxvf <QAT tarball name>
        (for example, qat1.7.l.4.11.0-00001.tar.gz)
   # ./configure
   # make install
   ```
4. To install the sample code as well on the host:
   ```
   # make samples-install
   ```

Note: To uninstall the driver and sample code completely, run: `# make uninstall`

5. Verify the QAT service by running the following command on the host OS.
   ```
   # service qat_service status
   ```

   As an example, with one Intel® Communications Chipset 8925 to 8955 Series device in the system, the output would appear as below:

   ```
   Checking status of all devices.
   There is 1 QAT acceleration device(s) in the system: qat_dev0 - type:
   dh895xcc,  inst_id: 0,  bsf: 83:00.0,  #accel: 6 #engines: 12 state:
   up
   ```

   As another example, with one Intel® C62X Series Chipset device in the system, the output would appear as below:

   ```
   Checking status of all devices.
   There is 3 QAT acceleration device(s) in the system:
    qat_dev0 - type: c6xx,  inst_id: 0,  node_id: 0,  bsf: 0000:1a:00.0,
   #accel: 5 #engines: 10 state: up
    qat_dev1 - type: c6xx,  inst_id: 1,  node_id: 0,  bsf: 0000:1c:00.0,
   #accel: 5 #engines: 10 state: up
    qat_dev2 - type: c6xx,  inst_id: 2,  node_id: 0,  bsf: 0000:1e:00.0,
   #accel: 5 #engines: 10 state: up
   ```

6. Verify the QAT service by running the RSA test code on the host OS.
   ```
   # cpa_sample_code runTests=2
   ```

   As an example, with one Intel® C62X Series Chipset device in the system, part of the performance output would appear as below:
   ```
   ---------------------------------------
   RSA CRT DECRYPT
   Modulus Size                2048 Number of Threads     18
   Total Submissions      1800000
   Total Responses        1800000
   Total Retries          100659543
   Clock Cycles Start     0
   ```

```
Clock Cycles End       0
Total Cycles           0
CPU Frequency(kHz)     2294915
Operations per second    101474
---------------------------------------
```

Note: The rest of the steps in this section are required only if you are using OpenSSL* or Nginx*. Otherwise, you can skip them.

7. Download the OpenSSL* and QAT engine software, following the instructions in `README.md` on https://github.com/intel/QAT_Engine.

   Ensure that the `[SSL]` section in the QAT driver configuration file (eg. `/etc/c6xx_dev0.conf`) has been replaced with the `[SHIM]` section information below:

```
   ##############################################
   # User Process Instance Section
   ##############################################
   [SHIM]
   NumberCyInstances = 1
   NumberDcInstances = 0
   NumProcesses = 32  //this might vary with CPU numbers on your
platform
   LimitDevAccess = 0
   # Crypto - User space
   Cy0Name = "UserCY0"
   Cy0IsPolled = 1
   Cy0CoreAffinity = 1
```

   You can configure the QAT engine via the OpenSSL* configuration file (default is <path to> `openssl/install/ssl/openssl.cnf`):

```
openssl_conf = openssl_def
[openssl_def]
engines = engine_section
[engine_section]
qat = qat_section
[qat_section]
engine_id = qatengine
dynamic_path = <path to>openssl/install/lib/engines-1.1/qatengine.so
default_algorithms = RSA, EC, DH
```

8. Run the following commands to check if the Intel® QAT OpenSSL* Engine is loaded correctly in the host system:

```
# cd <path to>openssl/bin
# ./openssl engine -t -c -vvvv qatengine
```

   The following output should appear with QAT engine information.

```
   (qat) Reference implementation of QAT crypto engine
   [RSA, DSA, DH, AES-128-CBC-HMAC-SHA1, AES-256-CBC-HMAC-SHA1,
   AES-128-CBC-HMAC-SHA256, AES-256-CBC-HMAC-SHA256, TLS1-PRF]
   [ available ]
   ENABLE_EXTERNAL_POLLING: Enables the external polling
interface to the engine.
        (input flags): NO_INPUT
   POLL: Polls the engine for any completed requests
        (input flags): NO_INPUT
   SET_INSTANCE_FOR_THREAD: Set instance to be used by this thread
        (input flags): NUMERIC
   GET_NUM_OP_RETRIES: Get number of retries
        (input flags): NO_INPUT
   SET_MAX_RETRY_COUNT: Set maximum retry count
        (input flags): NUMERIC
   SET_INTERNAL_POLL_INTERVAL: Set internal polling interval
        (input flags): NUMERIC
   GET_EXTERNAL_POLLING_FD: Returns non blocking fd for crypto
engine
        (input flags): NO_INPUT
   ENABLE_EVENT_DRIVEN_POLLING_MODE: Set event driven polling mode
        (input flags): NO_INPUT
   GET_NUM_CRYPTO_INSTANCES: Get the number of crypto instances
        (input flags): NO_INPUT
   DISABLE_EVENT_DRIVEN_POLLING_MODE: Unset event driven polling
mode
        (input flags): NO_INPUT
   SET_EPOLL_TIMEOUT: Set epoll_wait timeout
        (input flags): NUMERIC
```

The following speed command can be used to measure the performance of rsa2048 with the Intel® QAT OpenSSL* Engine. You can change the multi parameter `$number` based on the QAT configuration file setting. For more information on the process calculation, refer to Section 4.3.2.1 of *Intel® QuickAssist Technology Software for Linux* - Getting Started Guide.*

```
# ./openssl speed -engine qatengine -elapsed -multi $number -
async_jobs 72 rsa2048
```

Note: If the environment variables have not been set correctly, error messages such as `error while loading shared libraries: libssl.so.1.1: cannot open shared object file: No such file or directory` will appear. If this occurs, export the environment variable LD_LIBRARY_PATH via the command:
`# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:`<path to>`openssl/lib`

As an example, with one Intel® Communications Chipset 8925 to 8955 Series device in the system, performance output would appear as below:

```
engine " qatengine " set.
engine " qatengine " set.
+DTP:2048:private:rsa:10
+DTP:2048:private:rsa:10
+R1:195765:2048:10.01
+R1:214750:2048:10.01
+DTP:2048:public:rsa:10
+DTP:2048:public:rsa:10
+R2:2111634:2048:10.00
+R2:2096699:2048:10.00
Got: +F2:2:2048:0.000051:0.000005 from 0
Got: +F2:2:2048:0.000047:0.000005 from 1
OpenSSL 1.1.0e  16 Feb 2017
built on: reproducible build, date unspecified
options:bn(64,64) rc4(16x,int) des(int) aes(partial) idea(int)
blowfish(ptr)
compiler: gcc -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -
DOPENSSL_THREADS -DOPENSSL_NO_STATIC_ENGINE -DOPENSSL_PIC -
DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -
```

```
DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DRC4_ASM -
DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -
DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM -
DOPENSSLDIR="\"/home/nginx_test/openssl.bin/ssl\"" -
DENGINESDIR="\"/home/nginx_test/openssl.bin/lib/engines-1.1\"" -Wa,--
noexecstack
              sign     verify     sign/s verify/s
rsa 2048 bits 0.000024s 0.000003s  40884.4 400000.0
```

As another example, with one Intel® C62X Series Chipset device in the system, performance output would appear as below:

```
engine " qatengine " set.
engine " qatengine " set.
+DTP:2048:private:rsa:10
+DTP:2048:private:rsa:10
+R1:567141:2048:10.00
+R1:448483:2048:10.00
+DTP:2048:public:rsa:10
+DTP:2048:public:rsa:10
+R2:2403088:2048:10.00
+R2:2738731:2048:10.00
Got: +F2:2:2048:0.000022:0.000004 from 0
Got: +F2:2:2048:0.000018:0.000004 from 1
OpenSSL 1.1.0f  25 May 2017
built on: reproducible build, date unspecified
options:bn(64,64) rc4(16x,int) des(int) aes(partial) idea(int)
blowfish(ptr)
compiler: gcc -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -
DOPENSSL_THREADS -DOPENSSL_NO_STATIC_ENGINE -DOPENSSL_PIC -
DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -
DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -
DRC4_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM
-DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM -
DOPENSSLDIR="\"/root/kpt/openssl/openssl.bin/ssl\"" -
DENGINESDIR="\"/root/kpt/openssl/openssl.bin/lib/engines-1.1\""
-Wa,--noexecstack
                  sign     verify     sign/s verify/s
rsa 2048 bits 0.000010s 0.000002s 101010.1 500000.0
```

9. Download the Nginx* 1.18.0 and Nginx* patch for Intel® QuickAssist Technology OpenSSL* Engine, under the Nginx* patch and follow the README within the patch for installation.

Note: As of this writing, the latest Nginx* Patch is available on the link below: https://github.com/intel/asynch_mode_nginx

You can change some Nginx* configuration variables such as `worker_processes` and `worker_cpu_affinity` to fully utilize the asynchronous capability of the Intel® QAT OpenSSL* Engine and achieve maximum performance. The following example shows how to edit the Nginx* configuration file <path to >`nginx/install/conf/nginx.conf` :

Or you can copy <path to >`nginx/install/conf/nginx.QAT-sample.conf` to <path to >`nginx/install/conf/nginx.conf` and edit as following example:

```
  ************************************************
  worker_processes 4;
  worker_cpu_affinity 0001 0010 0100 1000;
```

```
worker_rlimit_nofile 200000;

load_module modules/ngx_ssl_engine_qat_module.so;

events {
    use epoll;
    worker_connections 102400;
    accept_mutex off;
}

# Enable QAT engine in heuristic mode.
ssl_engine {
    use_engine qatengine;
    default_algorithms RSA,EC,DH,DSA;
    qat_engine {
        qat_offload_mode async;
        qat_notify_mode poll;
        qat_poll_mode heuristic;
        qat_sw_fallback on;
    }
}

http {
    gzip off;
    gzip_min_length     128;
    gzip_comp_level     1;
    gzip_types  text/css text/javascript text/xml text/plain text/x-
component application/javascript application/json application/xml
application/rss+xml font/truetype font/opentype application/vnd.ms-
fontobject image/svg+xml;
    gzip_vary            on;
    gzip_disable        "msie6";
    gzip_http_version   1.0;

    # HTTP server with QATZip enabled.
    server {
        listen       80;
        server_name  localhost;
        location / {
            root   html;
            index  index.html index.htm;
        }
    }

    # HTTPS server with async mode.
    server {
        #If QAT Engine enabled,  `asynch` need to add to `listen`
directive or just add `ssl_asynch  on;` to the context.
        listen       443 ssl asynch;
        server_name  localhost;

        ssl_protocols       TLSv1.2;
        ssl_certificate      cert.pem;
        ssl_certificate_key  cert.key;

        location / {
            root   html;
            index  index.html index.htm;
        }
        }
    }
    *************************************************
```

Run Nginx* with the Intel® QAT OpenSSL* Engine as below:

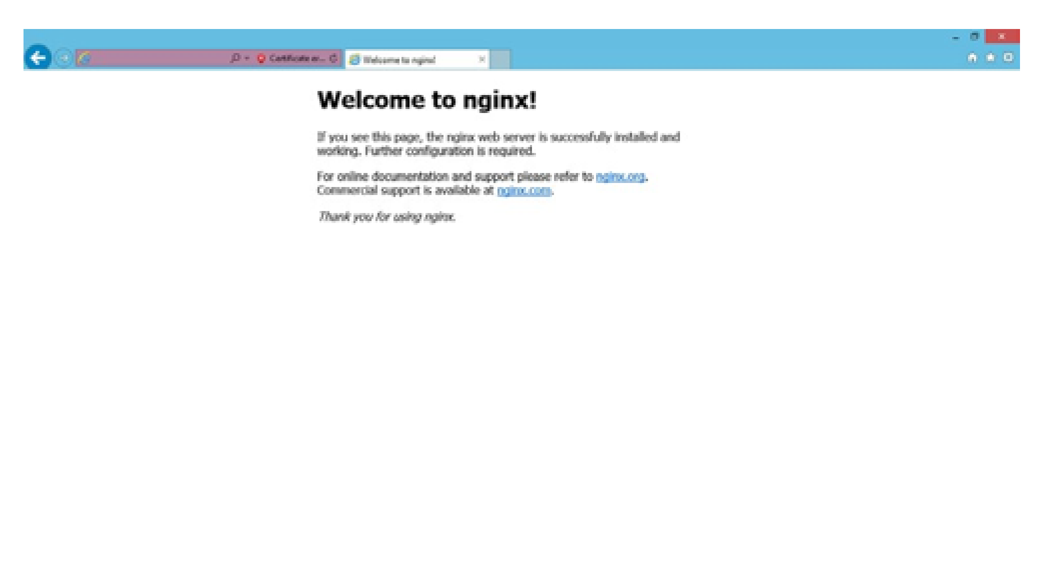
```
# <path to>Nginx/sbin/nginx -c <path to>Nginx/conf/nginx.conf
```

Note: Check the status of Nginx* to ensure that it has been launched successfully. As an example, the output of Nginx* with 16 workers is similar to the following:

```
# ps -ef | grep nginx
root   285      1  0 Mar30 ?   00:00:00 nginx: master process
./nginx -c /home/nginx_test/nginx/conf/nginx.conf
root   287    285  3 Mar30 ?   00:30:07 nginx: worker process
root   288    285  3 Mar30 ?   00:33:10 nginx: worker process
root   289    285  3 Mar30 ?   00:32:32 nginx: worker process
root   290    285  3 Mar30 ?   00:34:25 nginx: worker process
root   291    285  3 Mar30 ?   00:32:29 nginx: worker process
root   292    285  3 Mar30 ?   00:32:21 nginx: worker process
root   293    285  3 Mar30 ?   00:37:52 nginx: worker process
root   294    285  3 Mar30 ?   00:32:29 nginx: worker process
root   295    285  3 Mar30 ?   00:37:36 nginx: worker process
root   296    285  2 Mar30 ?   00:25:16 nginx: worker process
root   297    285  5 Mar30 ?   00:53:27 nginx: worker process
root   298    285  3 Mar30 ?   00:33:41 nginx: worker process
root   299    285  3 Mar30 ?   00:30:48 nginx: worker process
root   300    285  3 Mar30 ?   00:37:24 nginx: worker process
root   301    285  3 Mar30 ?   00:33:21 nginx: worker process
root   302    285  3 Mar30 ?   00:32:37 nginx: worker process
root   398    388  0 10:16 ?   00:00:00 grep --color=auto nginx
```

The client will display a Nginx* login screen.

Figure 1. Nginx* login screenshot



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

2.2 Installing Intel® QAT Software on Host with SR-IOV/IOMMU enabled

Section 2.2 describes how to install the driver software on the host with Virtual Function (VF) enabled.

2.2.1 Updating the BIOS Setting

Before installing Intel® QAT Software, update the BIOS to the latest stable version for your platform. Then follow the steps in this section to ensure stable operations.

1. Reset all BIOS settings to their default.
2. Disable all power saving options such as: Power performance tuning, CPU P-State, CPU C3 Report and CPU C6 Report.
3. Enable all virtualization options like VT-D and SR-IOV.

Note: Some example BIOS virtualization options are listed below. Yours may vary according to your vendor.

Advanced > System Agent (SA) Configuration > SRIOV

Advanced > System Agent (SA) Configuration > VT-D

4. Set the PCIe links to the highest possible speed, e.g., PCIe* Gen3 instead of PCIe Gen2 or Gen1.
5. Ensure that the PCIe links have trained to the expected width, e.g., x8 or x16.

2.2.2 Configuring the host operating system with SR-IOV/IOMMU

1. Update the kernel boot parameter with `intel_iommu=on`. For more information, refer to *Intel® QuickAssist Technology Software for Linux* - Getting Started Guide,* section 2.4, "Updating `grub` Configuration File".

   The following is a short summary of how to update the grub2 in CentOS and reboot the OS to activate SR-IOV/IOMMU functionality.

   ```
   # vim /etc/default/grub
   # change GRUB_CMDLINE_LINUX_DEFAULT=" … quiet" to
       GRUB_CMDLINE_LINUX_DEFAULT=" … quiet intel_iommu=on"
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   Reboot the system after the `grub` file has been updated.

   For more information on updating `grub2` and rebooting, refer to https://wiki.centos.org/HowTos/Grub2 or *Intel® QuickAssist Technology Software for Linux* - Getting Started Guide.*

2. Verify SR-IOV hardware capabilities using the command:

   ```
   # lspci -vnc 8086:<Device ID>
   ```

   The output should display one of the capabilities as:

   ```
   Capabilities: [140] Single Root I/O Virtualization (SR-IOV)
   ```

For more detail about SRIOV configuration, refer to *Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note* Section 2.2, "Installing and Configuring the Host Operating System".

2.2.3 Installing Intel® QuickAssist Technology Software

1. Change the current directory to the directory where you want to install the QAT software (for example, `/QAT`), referred to as <QATdir> in this document.
2. Set the following environment variable:

   ```
   export ICP_ROOT=<QATdir>
   ```

3. Unpack the Intel® QAT software and run the following commands to build and install the driver on the host:

   ```
   # tar -zxvf <QAT tarball name> (for example
   qat1.7.l.4.11.0-00001.tar.gz)
   # ./configure --enable-icp-sriov=host
   # make install
   ```

4. To install the sample code as well on the host:

   ```
   # make samples-install
   ```

Note: To uninstall the driver and sample code completely, run: `# make uninstall`

5. Verify the QAT service by running the following command on the host OS.

   ```
   # service qat_service_vfs status
   ```

   As an example, with two Intel® Communications Chipset 8925 to 8955 Series devices in the system, the output would appear as below:

   ```
   Checking status of all devices.
   There is 66 QAT acceleration device(s) in the system:
   qat_dev0 - type: dh895xcc,  inst_id: 0,  bsf: 06:00.0,  #accel: 6
   #engines: 12 state: up
   qat_dev1 - type: dh895xcc,  inst_id: 1,  bsf: 0d:00.0,  #accel: 6
   #engines: 12 state: up
   ...
   qat_dev64 - type: dh895xccvf,  inst_id: 62,  bsf: 0d:04.6,
   #accel: 1 #engines: 1 state: up
   qat_dev65 - type: dh895xccvf,  inst_id: 63,  bsf: 0d:04.7,
   #accel: 1 #engines: 1 state: up
   ```

   As another example, with one Intel® C62X Series Chipset device in the system, performance output would appear as below:

   ```
   # service qat_service_vfs status

   qat_dev3 - type: c6xxvf,  inst_id: 0,  node_id: 0,  bsf:
   0000:1a:01.0,  #accel: 1 #engines: 1 state: up

    qat_dev4 - type: c6xxvf,  inst_id: 1,  node_id: 0,  bsf:
   0000:1a:01.1,  #accel: 1 #engines: 1 state: up

   …

    qat_dev49 - type: c6xxvf,  inst_id: 46,  node_id: 0,  bsf:
   0000:1e:02.6,  #accel: 1 #engines: 1 state: up
     qat_dev50 - type: c6xxvf,  inst_id: 47,  node_id: 0,  bsf:
   0000:1e:02.7,  #accel: 1 #engines: 1 state: up
   ```

6. If you are using OpenSSL* or Nginx*, refer to Section 2.1.2 and follow the procedure starting at Step 5.

Note: To enable VFs for OpenSSL*/Nginx* usage in Linux* container or Docker*, set the `LimitDevAccess` value to `1`. The following configuration is for crypto operation only:

```
#############################################
# User Process Instance Section
##############################################
[SHIM]
NumberCyInstances = 1
NumberDcInstances = 0
NumProcesses = 1
LimitDevAccess = 1
# Crypto - User space
Cy0Name = "UserCY0"
Cy0IsPolled = 1
```

2.2.4 Enabling devices with more than 32 physical functions and virtual functions

By default, the Intel® QAT driver is limited to supporting no more than 32 physical functions and virtual functions. To eliminate this restriction, comment out the following lines in `quickassist/lookaside/access_layer/src/qat_direct/include/icp_adf_init.h`:

```
#ifdef ADF_MAX_DEVICES
#undef ADF_MAX_DEVICES
#endif
#define ADF_MAX_DEVICES 32
```

§

3 Using Intel® QAT Software in Linux* Containers

This chapter describes the steps necessary to enable Intel® QuickAssist Technology functionality in Linux* containers (LXC). These procedures can be used whether or not SR-IOV/IOMMU is enabled.

3.1 Installing LXC Virtualization in Linux*

This section describes how to install, deploy and run LXC containers on a CentOS*/RHEL* distribution.

Note: For more details of LXC installation, refer to the link:

https://www.tecmint.com/install-create-run-lxc-linux-containers-on-centos/

1. Open a terminal.
2. LXC virtualization is provided through Epel repositories. Install Epel repositories in your system using the command:
   ```
   # yum install epel-release
   ```
3. The Perl language interpreter and debootstrap packages are required. Install them using the command:
   ```
   # yum install debootstrap perl libvirt
   ```

Note: Choose QEMU/KVM for Hypervisor.

4. Install the LXC virtualization solution with the command:
   ```
   # yum install lxc*
   ```
5. After installing LXC service, verify that LXC and the `libvirt` daemon are running.
   ```
   # systemctl status lxc.service
   # systemctl start lxc.service
   # systemctl start libvirtd
   # systemctl status lxc.service
   ```
   Check LXC kernel virtualization status using the command:
   ```
   # lxc-checkconfig
   ```

3.2 Setting up Privileged LXC container with QAT in Linux*

This section describes how to create the LXC container and set it up with QAT acceleration service. The process of creating a LXC container is very simple.

1. To create a new container, enter the command:
   ```
   # lxc-create -n <container_name> -t <container_template>
   ```

For example, to create a new container named `qat` based on a CentOS template which is provided in the LXC repositories, enter the command:

```
# lxc-create -n qat -t centos
```

2. The root password is set up as expired and must be changed at first login, which you should do as soon as possible. If you lose the root password or wish to change it without starting the container, you can change it from the host by running the following command (which will also reset the expired flag):
   ```
   # chroot /var/lib/lxc/qat1/rootfs passwd
   ```
3. To start a created container with a specified name as a daemon, enter the command:
   ```
   # lxc-start -n qat -d
   ```
4. Ensure the QAT driver has been installed successfully in the host system. Then add the related QAT devices to the running container based on the matching devices on the host, using the commands:
   ```
   # lxc-device -n qat add /dev/usdm_drv
   # lxc-device -n qat add /dev/qat_dev_processes
   # lxc-device -n qat add /dev/qat_adf_ctl
   # modprobe uio
   # for dev in `ls /dev/uio*`;do lxc-device -n qat add $dev;done
   ```

3.3 Running acceleration driver sample code in privileged LXC container

This section describes how to run the QAT driver sample code in a LXC container.

1. Install the sample code on the host.
   ```
   # make samples-install
   ```
2. Copy the working directory of QAT driver installed in the Host to LXC container.
   ```
   # cp -r $ICP_ROOT /var/lib/lxc/qat/rootfs/$ICP_ROOT
   ```

Note: If the QAT working directory is not `$ICP_ROOT`, modify the above command accordingly.

3. Create a new shell running inside an existing container using the command:
   ```
   # lxc-attach -n qat
   ```
4. Run sample code inside the Linux* container using the command:
   ```
   #  cd $ICP_ROOT/build
   # ./cpa_sample_code
   ```

Note: Error messages such as error while loading shared libraries: `libqatengine s.so: cannot open shared object file: No such file or directory` will appear if environment variables have not been set. You can specify them by exporting the environment variable `LD_LIBRARY_PATH` via the command:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ICP_ROOT/build
```

3.4 Setting up UnPrivileged LXC container with QAT in Linux*

1. Create a new user for lxc

```
adduser mylxcusr
```

2. Find out allocated subuids and subgids for the lxc user

```
grep mylxcusr /etc/sub{gid,uid}

/etc/subgid:mylxcusr:231072:65536

/etc/subuid:mylxcusr:231072:65536
```

3. Create a default container configuration file for lxc user.

   Make sure the user “mylxcusr” is allowed up to 10 veth type devices to be created and added to the bridge called lxcbr0. Networking will only work if the following lines are added.

```
vi /etc/lxc/lxc-usernet

mylxcusr veth lxcbr0 10
```

4. Switch to lxcuser

```
su  mylxcusr
```

5. Once logged in as mylxcusr create below directories structures and files for mylxcusr

```
mkdir -p ~/.config/lxc

cp /etc/lxc/default.conf ~/.config/lxc/default.conf
```

6. Append the configuration as follows (use mapped user and group id ranges 100000:65536 from step 2 to ~/.config/lxc/default.conf

```
lxc.idmap = u 0 231072 65536

lxc.idmap = g 0 231072 65536
```

7. Provide required permissions and create a new container from mylxcusr account

```
chmod 777 /run/user/0

    chmod 777 /run/user/0/lxc/lock/home/

    #example below is based on centOS template

    lxc-create -t download -n container_name -- -d centos -r 8 -a amd64
```

8. Start the new container and login to the container

```
lxc-start -n container_name
```

```
lxc-attach -n container_name
```

9. Create below folders inside the unprivileged container "container_name"

```
mkdir  /QAT

mkdir /usr/lib/firmware

lxc-stop  -n container_name
```

10. Exit and stop the container

```
exit

lxc-stop  -n container_name
```

11. As root user install the QAT driver in the host system and add permissions for below

```
chmod 777 /dev/usdm_drv

chmod 777 /dev/qat_dev_processes

chmod 777 /dev/qat_adf_ctl

chmod 777 /dev/uio*
```

12. Login as mylxcusr and mount QAT related devices to the container inside container_name config file

Add below mount commands into /home/mylxcusr/.local/share/lxc/container_name/config file

```
lxc.mount.entry = /QAT QAT none bind 0 0

lxc.mount.entry = /usr/lib/firmware usr/lib/firmware none bind 0 0

lxc.mount.entry = /dev/usdm_drv dev/usdm_drv none bind,optional,create=file

lxc.mount.entry = /dev/qat_dev_processes dev/qat_dev_processes none bind,optional,create=file

lxc.mount.entry = /dev/qat_adf_ctl dev/qat_adf_ctl none bind,optional,create=file

for dev in `ls /dev/uio*`; do u="$(cut -d'/' -f3 <<<"$dev")"; echo "lxc.mount.entry = /dev/$u dev/$u none bind,optional,create=file" >> /home/mylxcusr/.local/share/lxc/container_name/config ;done
```

13. Restart lxc services (optional incase if any network error pop up)

```
systemctl stop lxc-net.service

systemctl start lxc-net.service

systemctl stop lxc.service

systemctl start lxc.service
```

3.5 Running acceleration driver sample code in Unprivileged LXC container

This section describes how to run the QAT driver sample code in an unprivilaged LXC container.

1.Install the sample code on the host as root user.

```
make samples-install
```

2. Login as mylxcusr, start and create a new shell running inside an existing container using the command:

```
lxc-start -n container_name

lxc-attach -n container_name
```

3.Run sample code inside the unprivilaged container "container_name" using the command:

```
export ICP_ROOT=/QAT

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH$ICP_ROOT/build

cd $ICP_ROOT/build

./cpa_sample_code
```

Note: when running QAT services from within an unprivileged LXC, there are additional memory requirements. Your system's max locked memory size must exceed 64 KB (you can check this with the ulimit –a command). If it is not large enough, edit /etc/security/limits.conf to set memlock to mylxcusr - memlock 4096.

3.6 Running OpenSSL* and Nginx* with Acceleration Services in LXC container (Optional)

This section describes how to run the optional Nginx* and OpenSSL* applications with QAT in an LXC container.

1. **Make sure Nginx* and OpenSSL* have been configured properly in the host.** Then copy the relevant Nginx*, OpenSSL* binary and QAT driver installed in the host to the LXC container.

Note: If Nginx*, the OpenSSL* binary and QAT are not installed in the default directories `/root/$NGINX_INSTALL_DIR`, `/root/$OPENSSL_INSTALL_DIR`, and `$ICP_ROOT`, modify these commands accordingly.

```
# cp -r /root/$NGINX_INSTALL_DIR /var/lib/lxc/qat/rootfs/root
# cp -r /root/$OPENSSL_INSTALL_DIR /var/lib/lxc/qat/rootfs/root
# cp -r $ICP_ROOT /var/lib/lxc/qat/rootfs/$ICP_ROOT
```

2. Create a new shell running inside an existing container using the command:

```
# lxc-attach -n qat
```

3. Run the following commands to verify that the Intel® QAT OpenSSL* Engine is loaded correctly in LXC container:

```
# cd <path to>openssl/bin
# ./openssl engine -t -c -vvvv qatengine
# ./openssl speed -engine qatengine -elapsed -multi 2 -async_jobs 72
rsa2048
```

Note: Error messages such as error while loading shared libraries: `libssl.so.1.1: cannot open shared object file: No such file or directory` will appear if environment variables have not been set. You can specify them by exporting the environment variable `LD_LIBRARY_PATH` via the command:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path to>/openssl/lib
```

4. Run Nginx with QAT within the LXC container.

```
# <path to>Nginx/sbin/nginx -c <path to>Nginx/conf/nginx.conf
```

5. Check the status of Nginx* to ensure that it has been launched successfully.

```
# ps -ef | grep nginx
```

Note: As an example, the output of Nginx* with 16 workers appears below:

```
root   285      1  0 Mar30 ?   00:00:00 nginx: master process
./nginx -c /home/nginx_test/nginx/conf/nginx.conf
root   287    285  3 Mar30 ?   00:30:07 nginx: worker process
root   288    285  3 Mar30 ?   00:33:10 nginx: worker process
root   289    285  3 Mar30 ?   00:32:32 nginx: worker process
root   290    285  3 Mar30 ?   00:34:25 nginx: worker process
root   291    285  3 Mar30 ?   00:32:29 nginx: worker process
root   292    285  3 Mar30 ?   00:32:21 nginx: worker process
root   293    285  3 Mar30 ?   00:37:52 nginx: worker process
root   294    285  3 Mar30 ?   00:32:29 nginx: worker process

root   295    285  3 Mar30 ?   00:37:36 nginx: worker process
root   296    285  2 Mar30 ?   00:25:16 nginx: worker process
root   297    285  5 Mar30 ?   00:53:27 nginx: worker process
root   298    285  3 Mar30 ?   00:33:41 nginx: worker process
root   299    285  3 Mar30 ?   00:30:48 nginx: worker process
root   300    285  3 Mar30 ?   00:37:24 nginx: worker process
root   301    285  3 Mar30 ?   00:33:21 nginx: worker process
root   302    285  3 Mar30 ?   00:32:37 nginx: worker process
root   398    388  0 10:16 ?   00:00:00 grep --color=auto nginx
```

§

4 Using Intel® QAT Software in Docker*

This chapter describes how to enable Intel® QuickAssist Technology functionality in Docker*. These procedures apply whether or not SR-IOV/IOMMU is enabled.

4.1 Installing Docker* in Linux*

This section describes how to install, deploy and run Docker* on a CentOS/RHEL distribution.

For more details of Docker* installation, refer to the links: http://blog.csdn.net/xixiworld/article/details/71438794

https://docs.docker.com/engine/installation/linux/docker-ce/centos/#install-usingthe-repository https://docs.docker.com/engine/admin/systemd/#httphttps-proxy

1. Uninstall the old versions of Docker*.

Note: Older versions of Docker* were called `docker` or `docker-engine`. If these are installed, uninstall them, along with associated dependencies:

```
# yum remove docker docker-common container-selinux docker-
selinux docker-engine
```

2. If you are installing Docker* for the first time on a new host machine, set up the Docker* repository.

   ```
   # yum install -y yum-utils (Optional)
   # yum-config-manager --add-repo \
   https://download.docker.com/linux/centos/docker-ce.repo
   ```

3. Install the Docker* CE.

   ```
   # yum install docker-ce
   ```

4. Start Docker* and check the running status of Docker*.

   ```
   # systemctl start docker
   # systemctl status docker
   ```

5. Set up HTTP/HTTPS proxy for Docker* (optional)

   If you are behind an HTTP or HTTPS proxy server, for example in corporate settings, you will need to add this configuration in the Docker* `systemd` service file.

   a. Create a `systemd` drop-in directory for the Docker* service.

      ```
      # mkdir -p /etc/systemd/system/docker.service.d
      ```

   b. Create a file called `/etc/systemd/system/docker.service.d/httpproxy.conf` that adds the HTTP_PROXY or HTTPS_PROXY environment variables:

      ```
      [Service]
      Environment="HTTP_PROXY=http://proxy.example.com:80/"
      Environment="HTTPS_PROXY=https://proxy.example.com:443/"
      ```

c. If you have internal Docker* registries that you need to contact without proxying you can specify them via the `NO_PROXY` environment variable:

```
Environment="HTTP_PROXY=http://proxy.example.com:80/"
NO_PROXY="localhost,127.0.0.1,dockerregistry.somecorporation.com"
```

d. Flush changes and restart Docker*:

```
 # systemctl daemon-reload
 # systemctl restart docker
```

6. Verify that Docker* is installed correctly by running the `hello-world` image. This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

```
# docker run hello-world
```

4.2 Setting up Docker* with QAT in Linux*

This section describes how to set up Docker* with the QAT acceleration service.

1. Pull a CentOS image and check the local image in the server.

```
# docker pull centos
# docker images
```

Note: If the QAT working directory is not `$ICP_ROOT`, modify the following commands, accordingly, using the same path as the host.

2. If you are running QAT services from within a privileged Docker*, enter the following commands:

```
# docker run -it -v $ICP_ROOT:$ICP_ROOT --privileged=true centos
/bin/bash
```

Note: If you are running QAT services from within an unprivileged Docker*, there are additional memory requirements. Your system's max `locked memory size` must exceed 64 KB (you can check this with the `ulimit -a` command). If it is not large enough, edit `/etc/security/limits.conf` to set `memlock` to unlimited.

Note: If you encounter performance issues, you may also want to edit <path to>`system/system/docker.service` to add the lines:

```
LimitMEMLOCK=infinity
LimitNOFILE=infinity
```

Note: If you are not running Docker* as root, you may need to use the `chmod` command to grant permission to the QAT devices listed in the following procedure.

3. To run QAT services within an unprivileged Docker* instance, enter the following commands:

```
# unset devpara

# modprobe uio

# for dev in `ls /dev/uio*`; do devpara=$devpara" --
device="$dev":"$dev; done

# export devpara=$devpara" --
device=/dev/qat_adf_ctl:/dev/qat_adf_ctl"
```

```
# export devpara=$devpara" --
device=/dev/qat_dev_processes:/dev/qat_dev_processes"

# export devpara=$devpara" --device=/dev/usdm_drv:/dev/usdm_drv"

# vim <path to>systemd/system/docker.service
  (Add the memlock setting --- LimitMEMLOCK=infinity)

# systemctl daemon-reload
# systemctl restart docker.service

# docker run -it -v $ICP_ROOT:$ICP_ROOT $devpara centos /bin/bash
```

4.3 Running Acceleration Driver sample code in Docker*

Once Docker* has been set up, enter the following commands to execute QAT driver software package sample code:

```
# cd $ICP_ROOT/build
# ./cpa_sample_code
```

Note: Error messages such as error while loading shared libraries: libqatengine_s.so: cannot open shared object file: No such file or `directory` will appear if environment variables have not been set. You can specify them by exporting the environment variable `LD_LIBRARY_PATH` via the command:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ICP_ROOT/build
```

4.4 Running OpenSSL* and Nginx* with Acceleration Services in Docker* (Optional)

The following sections detail the steps to run the Nginx* and OpenSSL* applications with QAT acceleration service in Docker*.

1. Make sure Nginx* and OpenSSL* have been configured properly in the host.

Note: If Nginx*, the OpenSSL* binary and QAT are not installed in the default directories `/root/$NGINX_INSTALL_DIR`, `/root/$OPENSSL_INSTALL_DIR`, and `$ICP_ROOT`, modify the following commands accordingly.

2. Run Docker* with the following commands which map the working directories of Nginx*, OpenSSL* binary and QAT driver:
```
# docker run -it -v /root/$NGINX_INSTALL_DIR:/root/$NGINX_INSTALL_DIR
-v /root/$OPENSSL_INSTALL_DIR:/root/$OPENSSL_INSTALL_DIR -v
$ICP_ROOT:$ICP_ROOT $devpara centos /bin/bash
```
3. Run the following commands to verify the Intel® QAT OpenSSL* Engine has been loaded correctly in Docker*:
```
# cd <path to>openssl/bin
# ./openssl engine -t -c -vvvv qatengine
# ./openssl speed -engine qatengine -elapsed -multi 2 -async_jobs 72
rsa2048
```

Note: Error messages such as `error while loading shared libraries: libssl.so.1.1: cannot open shared object file: No such file or directory` will appear if environment variables have not been set. You can specify them by exporting the environment variable `LD_LIBRARY_PATH` via the command:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path to>/openssl/lib
```

4. Run Nginx* with QAT within Docker*:
   ```
   # <path to>Nginx/sbin/nginx -c <path to>Nginx/conf/nginx.conf
   ```
5. Check the status of Nginx* to ensure that it has been launched successfully.
   ```
   # ps -ef | grep nginx
   ```

As an example, the output of Nginx* with 16 workers appears below:

```
root   285      1  0 Mar30 ?   00:00:00 nginx: master process

./nginx -c /home/nginx_test/nginx/conf/nginx.conf

root   287    285  3 Mar30 ?   00:30:07 nginx: worker process
root   288    285  3 Mar30 ?   00:33:10 nginx: worker process
root   289    285  3 Mar30 ?   00:32:32 nginx: worker process
root   290    285  3 Mar30 ?   00:34:25 nginx: worker process
root   291    285  3 Mar30 ?   00:32:29 nginx: worker process
root   292    285  3 Mar30 ?   00:32:21 nginx: worker process
root   293    285  3 Mar30 ?   00:37:52 nginx: worker process
root   294    285  3 Mar30 ?   00:32:29 nginx: worker process
root   295    285  3 Mar30 ?   00:37:36 nginx: worker process
root   296    285  2 Mar30 ?   00:25:16 nginx: worker process
root   297    285  5 Mar30 ?   00:53:27 nginx: worker process
root   298    285  3 Mar30 ?   00:33:41 nginx: worker process
root   299    285  3 Mar30 ?   00:30:48 nginx: worker process
root   300    285  3 Mar30 ?   00:37:24 nginx: worker process
root   301    285  3 Mar30 ?   00:33:21 nginx: worker process
root   302    285  3 Mar30 ?   00:32:37 nginx: worker process
root   398    388  0 10:16 ?   00:00:00 grep --color=auto nginx
```

§

5 Limitations of Running Intel® QAT Software in Multiple Linux* Containers or Dockers*

As of this writing, there are some limitations to run applications like OpenSSL* or Nginx* with acceleration services simultaneously in multiple Linux* containers or Dockers*.

- It is not possible to assign a specified amount of QAT instance/VF resources to one container or Docker* and isolate the access from others. All Linux* containers, Dockers* and the host share instances and VFs on a first-come, first-served basis.
- All QAT-related UIO devices must be added to each container or Docker*. Errors will occur if only some of the UIO devices are added to a container or Docker*.
- Stopping or restarting QAT devices in a container or Docker* will impact all other containers and Dockers* immediately.
- The total number of QAT instances should not exceed the maximum number of instances specified in the QAT configuration file. Depending on which hardware QAT device is installed, the configuration file name is:
  - `etc/dh895xcc_dev0.conf`
  - `etc/c6xx_dev0.conf`
  - `etc/c3xx_dev0.conf`
- With SR-IOV enabled, QAT acceleration running on the host does not use the PF. It uses one or more VFs instead.

§

Appendix A FAQ

The error message `mmap on memory allocated through ioctl failed` appeared when QAT ran in Docker*. How do I fix it?

Double-confirm the value of `max memory size` by running the `ulimit -a` command in both the host and a container. If the default value of `max memory size` is too small or the `LimitMEMLOCK` setting does not take effect, you will see the following error message when running QAT in Docker*:

```
ioctl_alloc_slab:893 mmap on memory allocated through ioctl
failed
ADF_UIO_PROXY err: adf_init_ring: unable to get
ringbuf(v:(nil),p:(nil)) for rings in bank(0)
ADF_UIO_PROXY err: icp_adf_transCreateHandle: adf_init_ring
failed
```

You can fix it by manually specifying the `ulimit memlock` setting when running Docker*:

```
# docker run -it --ulimit memlock=-1:-1 -v /ICP_ROOT:/ICP_ROOT $devpara
centos /bin/bash
```

§