Using Intel® QuickAssist Technology in Linux* Container and Docker

Application Note

January 2018
Contents

1.0 Introduction .............................................................. 6
  1.1 About this Document ............................................... 6
  1.2 Terminology .......................................................... 6
  1.3 Documentation ......................................................... 6
     1.3.1 Where to Find Current Software and Documentation ....... 6
  1.4 Reference Documents ................................................. 7
  1.5 Documentation Conventions .......................................... 7
  1.6 Software Requirements ............................................... 7

2.0 Installing Intel® QuickAssist Technology Software .................. 8
  2.1 Installing Intel® QAT Software on Host with SR-IOV/IOMMU disabled .... 8
     2.1.1 Updating the BIOS Setting ................................... 8
     2.1.2 Installing Intel® QAT Software ................................ 9
  2.2 Installing Intel® QAT Software on Host with SR-IOV/IOMMU enabled ........ 15
     2.2.1 Updating the BIOS Setting ................................... 15
     2.2.2 Configuring the host operating system with SR-IOV/IOMMU ........... 16
     2.2.3 Installing Intel® QuickAssist Technology Software ................. 16
     2.2.4 Enabling devices with more than 32 physical functions and virtual functions ........................................ 18

3.0 Using Intel® QAT Software in Linux Containers ...................... 19
  3.1 Installing LXC Virtualization in Linux ............................. 19
  3.2 Setting up LXC container with QAT in Linux ....................... 19
  3.3 Running acceleration driver sample code in LXC container .......... 20
  3.4 Running OpenSSL* and Nginx* with Acceleration Services in LXC container (Optional) ........................................ 21

4.0 Using Intel® QAT Software in Docker* .................................. 23
  4.1 Installing Docker in Linux ........................................... 23
  4.2 Setting up Docker with QAT in Linux ............................... 24
  4.3 Running Acceleration Driver sample code in Docker ................. 25
  4.4 Running OpenSSL* and Nginx* with Acceleration Services in Docker (Optional) ........................................ 25

5.0 Limitations of Running Intel® QAT Software in Multiple Linux Containers or Dockers .................. 27

Appendix A FAQ .................................................................. 28
Figures

Figure 1. Nginx login screenshot ................................................................. 15

Tables

Table 1. Terminology ................................................................................. 6
Table 2. Reference Documents ................................................................. 7
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2018</td>
<td>001</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

§
1.0 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

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

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

Table 2 includes a list of related documentation.
1.4 Reference Documents

<table>
<thead>
<tr>
<th>Table 2. Reference Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document</strong></td>
</tr>
<tr>
<td>Intel® Communications Chipset 8925 to 8955 Series Software for Linux* Getting Started Guide</td>
</tr>
<tr>
<td>Intel® Communications Chipset 8925 to 8955 Series Software Programmer’s Guide</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology API Programmer’s Guide</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Cryptographic API Reference Manual</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Data Compression API Reference Manual</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology for Linux* Release Notes</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology for Linux* Getting Started Guide</td>
</tr>
<tr>
<td>Intel® Communications Chipset 89xx Series Datasheet</td>
</tr>
<tr>
<td>Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Software for Linux* Getting Started Guide - HW version 1.7</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Software for Linux* Release Notes - HW version 1.7</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Software for Linux* Programmer’s Guide - HW version 1.7</td>
</tr>
<tr>
<td>Intel® QuickAssist Technology Driver for Linux* - HW version 1.7</td>
</tr>
</tbody>
</table>

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.1.0.3-42) 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* 7.2 64-bit version, Kernel: GNU*/Linux* 3.10.0-327.el7.x86_64
2.0 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.upstream.1.1.0.3-42.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® C620 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, bsf: 3d:00.0, #accel: 5 #engines: 10 state: up
   qat_dev1 - type: c6xx, inst_id: 1, bsf: 3f:00.0, #accel: 5 #engines: 10 state: up
   qat_dev2 - type: c6xx, inst_id: 2, bsf: 41:00.0, #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® C620 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/dh895xcc_qa_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]
quat = qat_section
[qat_section]
gine_id = qat
```
8. Run the following commands to check if the Intel® QAT OpenSSL* Engine is loaded correctly in the host system:

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

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.

```bash
# ./openssl speed -engine qat -elapsed -multi $number -async_jobs
72 rsa2048
```
Installing Intel® QuickAssist Technology Software

**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 "qat" set.
engine "qat" 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_NIST256_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® C620 Series Chipset device in the system, performance output would appear as below:

```
engine "qat" set.
engine "qat" 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
```
9. Download the Nginx 1.10.3 and Nginx patch for Intel® QuickAssist Technology OpenSSL* Engine, untar 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](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`:

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

Use the `openssl` command to verify if the changes have been successful:

```
sign verify sign/s verify/s
rsa 2048 bits 0.000010s 0.000002s 101010.1 500000.0
```
... 
ssl_async       on;
ssl_buffer_size  64k;
...
location / {
  ...
}
...
}
... 
*************************************************
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  290    285  3 Mar30 ?   00:34:25 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 an Nginx* login screen.

![Nginx login screenshot](image)

**Figure 1. Nginx login screenshot**

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.upstream.1.1.0.3-42.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 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_dev2 - type: dh895xccvf, inst_id: 0, bsf: 06:01.0, accel: 1
    #engines: 1 state: up
qat_dev3 - type: dh895xccvf, inst_id: 1, bsf: 06:01.1, accel: 1
    #engines: 1 state: up
qat_dev4 - type: dh895xccvf, inst_id: 2, bsf: 06:01.2, accel: 1
    #engines: 1 state: up
... qat_dev9 - type: dh895xccvf, inst_id: 7, bsf: 06:01.7, accel: 1
    #engines: 1 state: up
qat_dev10 - type: dh895xccvf, inst_id: 8, bsf: 06:02.0,
    accel: 1 #engines: 1 state: up
qat_dev11 - type: dh895xccvf, inst_id: 9, bsf: 06:02.1,
    accel: 1 #engines: 1 state: up
... qat_dev17 - type: dh895xccvf, inst_id: 15, bsf: 06:02.7,
    accel: 1 #engines: 1 state: up
qat_dev18 - type: dh895xccvf, inst_id: 16, bsf: 06:03.0,
    accel: 1 #engines: 1 state: up
qat_dev19 - type: dh895xccvf, inst_id: 17, bsf: 06:03.1,
    accel: 1 #engines: 1 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

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
```
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:

```c
#ifdef ADF_MAX_DEVICES
#undef ADF_MAX_DEVICES
#endif
#define ADF_MAX_DEVICES 32
```
3.0 **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/](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:
   ```bash
   # yum install epel-release
   ```
3. The Perl language interpreter and debootstrap packages are required. Install them using the command:
   ```bash
   # yum install debootstrap perl libvirt
   ```

*Note:* Choose QEMU/KVM for Hypervisor.

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

Check LXC kernel virtualization status using the command:
```bash
# lxc-checkconfig
```

3.2 **Setting up 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:
   ```bash
   # 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:
   ```bash
   # lxc-create -n qat -t CentOS
   ```
# 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
   # for dev in `ls /dev/uio*`;do lxc-device -n qat add $dev;done

3.3 Running acceleration driver sample code in 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: libqat_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 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 qat
   # ./openssl speed -engine qat -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>/home/nginx_test/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  290 285 3 Mar30 ? 00:34:25 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
     ```
Using Intel® QAT Software in Linux® Containers

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
4.0 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-using-the-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

      # vi /etc/systemd/system/docker.service.d/http_proxy.conf
      # vi /etc/systemd/system/docker.service.d/https_proxy.conf
      # vi /etc/systemd/system/docker.service.d/no_proxy.conf
b. Create a file called `/etc/systemd/system/docker.service.d/http-proxy.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,docker-registry.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=unfinity
   LimitNOFILE=infinty
   ```

   **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:

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

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

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

# export devpara=$devpara "--device=\n/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:

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

**Note:** Error messages such as error while loading shared libraries: libqat_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:

```bash
# export 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.**
Using Intel® QAT Software in Docker*

**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 qat
   # ./openssl speed -engine qat -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>/bin/nginx --c <path to>/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 290 285 3 Mar30 ? 00:34:25 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
```
5.0 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/dh89xxcc._qa_dev0.conf
  - etc/dh895xcc._qa_dev0.conf
  - etc/c6xx._qa_dev0.conf
  - etc/c3xx._qa_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:

```
iocctl_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=16834000:16834000 -v /ICP_ROOT:/ICP_ROOT $devpara centos /bin/bash
```

§