NVIDIA DOCA RegEx

Programming Guide

MLNX-15-060489 _v2.0.2 | May 2023

Table of Contents

Chapter 1. Introduction 1
Chapter 2. Prerequisites 2
Chapter 3. Architecture 3
3.1. Rule Compilation 3
3.2. RegEx Implementations 3
3.3. Huge Job Emulation 3
Chapter 4. API 5
4.1. Enumerated Types 5
4.1.1. doca_regex_job_types 5
4.1.2. doca_regex_search_job_flags 5
4.1.3. doca_regex_status_flag 5
4.2. Structures 6
4.2.1. doca_regex_job_search 6
4.2.2. doca_regex_search_result 6
4.2.3. doca-regex 7
4.3. Instance Construction/Destruction 7
4.3.1. doca_regex_create 7
4.3.2. doca_regex_destroy 7
4.3.3. doca_regex_as_ctx 7
4.4. Device Query API 8
4.4.1. doca_regex_is_supported 8
4.4.2. doca_regex_get_hardware_supported 8
4.4.3. doca_regex_get_maximum_job_size 8
4.4.4. doca_regex_get_maximum_non_huge_job_size 9
4.4.5. doca_regex_job_get_supported 9
4.4.6. doca_regex_search_job_flag_get_highest_priority_match_supported 9
4.4.7. doca_regex_search_job_flag_get_stop_on_any_match_supported 10
4.5. Programming RegEx 10
4.5.1. Reprogramming Guidelines 10
4.5.2. doca_regex_set_hardware_compiled_rules 11
4.5.3. doca_regex_get_hardware_compiled_rules 11
4.5.4. doca_regex_set_hardware_uncompiled_rules 12
4.5.5. doca_regex_get_hardware_uncompiled_rules 12
4.6. DOCA RegEx Setup 13
4.6.1. doca_regex_set_workq_matches_memory_pool_size 13

4.6.2. doca_regex_get_workq_matches_memory_pool_size 14
4.7. Configuration Options 14
4.7.1. doca_regex_set_huge_job_emulation_overlap_size 14
4.7.2. doca_regex_get_huge_job_emulation_overlap_size 15
4.7.3. doca_regex_set_in_order_responses_enabled 15
Chapter 5. DOCA RegEx Samples 16
5.1. Sample Prerequisites 16
5.2. Running the Sample 16
5.3. Samples 17
5.3.1. RegEx Scan 17

Chapter 1. Introduction

DOCA RegEx is a library that provides RegEx pattern matching to DOCA applications. It provides access to the regular expression processor (RXP) , a high-performance, hardware-accelerated RegEx engine available on the NVIDIA® BlueField® DPUs, and can utilize software-based engines when required.

Using DOCA RegEx, developers can easily execute complex regular expression operations in an optimized, hardware-accelerated way.

This document is intended for software developers wishing to accelerate their regular expressions operations.

Chapter 2. Prerequisites

DOCA RegEx-based applications can run either on the host machine or on the DPU target.

The RegEx engine is enabled by default on the DPU. However, to enable RegEx offloading on the host, run:

```
host> sudo /etc/init.d/openibd stop
host> sudo echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

Then enable host access to the RegEx engine on the DPU:

```
dpu> echo 1 > /sys/bus/pci/devices/0000\:03\:00.0/regex/pf/regex_en
```

Chapter 3. Architecture

DOCA RegEx provides a flexible API for programming regular expression databases, enqueuing jobs and dequeuing results. The API operates asynchronously allowing many pattern matching operations to be executed in parallel.

3.1. Rule Compilation

Regular expressions can be provided as:

- A "compiled" rules file where the external RXPC (RXP compiler) is used to generate a compiled data file; or
- An "uncompiled" where the DOCA Regex library compiles the supplied regular expressions when initialized

The external compiler is termed "RXPC" (RXP compiler) and generates RXP object format (ROF) binary files that represent the compiled regular expressions. For more information on RXPC, please refer to chapter "RXP Compiler Utility" in the NVIDIA RXP Compiler Tool Guide.

When uncompiled rules are provided, the library utilizes a set of default options during compilation. For complete control and optimization, it is recommend you use compile rules with custom compiler options.

3.2. RegEx Implementations

The library itself is designed to support multiple RegEx engine implementations. Currently, only hardware devices are supported. Software devices will be introduced in the future.

3.3. Huge Job Emulation

The library includes a facility to accept job lengths that are greater than the maximum size supported by an engine. The library fragments incoming jobs into smaller fragments and processes them sequentially looking for potential matches. The "huge job emulation" mechanism takes data from the end of the previous fragment and appends it to the

start of the next fragment (the "size" of the overlap) to find additional matches. See the `doca_regex_property_huge_job_emulation_overlap_set` API call for more information.

Chapter 4. API

This section details the specific enumerated types, structures, and API operations related to the DOCA RegEx library.

Note: The pkg-config (`*.pc` file) for the RegEx library is included in DOCA's regular definitions (i.e., `doca`).

4.1. Enumerated Types

4.1.1. doca_regex_job_types

This enumerated type provides the available job types for RegEx operations.

```
enum doca_regex_job_types {
        /** Default RegEx search mode */
        DOCA_REGEX_JOB_SEARCH = DOCA_ACTION_REGEX_FIRST + 1,
};
```

4.1.2. doca_regex_search_job_flags

This enumerated type provides the flags which are applicable to RegEx jobs.

```
enum doca_regex_search_job_flags {
        DOCA_REGEX_SEARCH_JOB_FLAG_HIGHEST_PRIORITY_MATCH = 1 << 1,
        DOCA_REGEX_SEARCH_JOB_FLAG_STOP_ON_ANY_MATCH = 1 << 2,
};
```

DOCA_REGEX_SEARCH_JOB_FLAG_HIGHEST_PRIORITY_MATCH
When a RegEx job is submitted for searching, a number of regular expressions can be tested for in parallel. This flag results in only the match with the lowest rule ID being returned.

DOCA_REGEX_SEARCH_JOB_FLAG_STOP_ON_ANY_MATCH
BlueField-3 only. If this option is set on a RegEx job, the engine stops and returns the first RegEx match detected in the input data.

4.1.3. doca_regex_status_flag

This enumerated type provides flags that indicate the status of a job response.

```
enum doca_regex_status_flag {
 DOCA_REGEX_STATUS_SEARCH_FAILED = 1,
};
```

DOCA_REGEX_STATUS_SEARCH_FAILED
This is a general failure indication for any RegEx job.

4.2. Structures

4.2.1. doca_regex_job_search

This structure contains information required when sending a job to the RegEx engine.

```
struct doca_regex_job_search {
 struct doca_job base;
 uint16_t rule_group_ids[4];
 struct doca_buf const *buffer;
 struct doca_regex_search_result *result;
 uint8_t allow_batching;
};
```

base
Common DOCA job data.

rule_group_ids
An array of IDs which can be used to select which groups of rules are used to process this job. Set each value to a non-zero value to enable group selection, or to 0 to ignore it.

buffer
A `doca_buf` representing the data to be scanned for RegEx matches.

result
Pointer to where the job response is stored. The caller must ensure this pointer is valid when submitting a job and it must remain valid until a response for the job has been retrieved from the engine.

allow_batching
Setting this field to 1 allows the RegEx device to aggregate jobs into batches if this is the optimal method for the supplied data. Batching can improve throughput at the cost of latency. Set this field to 0 to force this job to begin executing immediately. This also forces any previously enqueued jobs that have been batched and not yet dispatched to begin processing.

4.2.2. doca_regex_search_result

This structure contains result information from a previous RegEx search.

```
struct doca_regex_search_result {
 uint64_t status_flags;
 uint32_t detected_matches;
 uint32_t num_matches;
 struct doca_regex_match *matches;
 struct doca_regex_mempool *matches_mempool;
};
```

status_flags
This field indicates any status flags that have been set as a result of the RegEx operation. See doca_regex_status_flag enumerated type for more information.

detected_matches
The total matches that have been detected by the RegEx operation.

num_matches
The actual number of matches returned.
matches
A linked list of doca_regex_match elements. The linked list is num_matches long.
matches_mempool
The memory pool that owns the matches.

4.2.3. doca-regex

This is an opaque structure used to represent a RegEx instance and is used with API calls.

```
struct doca_regex ;
```

4.3. Instance Construction/Destruction

This section details API calls related to the creation and destruction of DOCA RegEx instances.

4.3.1. doca_regex_create

Creates a DOCA RegEx instance.

```
doca_error_t doca_regex_create(struct doca_regex **regex);
```

regex [out]
A pointer to be populated with the address of the newly created RegEx context.
Returns

- doca_error_t return code with DOCA_SUCCESS if successful
- DOCA_ERROR_INVALID_VALUE – indicates an invalid input to the API call
- DOCA_ERROR_NO_MEMORY – indicates a failure to allocate memory for the instance

4.3.2. doca_regex_destroy

Destroys a previously created DOCA RegEx instance.

```
doca_error_t doca_regex_destroy(struct doca_regex *regex);
```

regex [out]
A pointer to be populated with the address of the newly created RegEx context.
Returns

- doca_error_t return code with DOCA_SUCCESS if successful
- DOCA_ERROR_INVALID_VALUE – indicates an invalid input to the API call

4.3.3. doca_regex_as_ctx

Converts a RegEx instance into a generic doca_ctx. See the NVIDIA DOCA Core Programming Guide for more information on DOCA contexts.

```
struct doca_ctx *doca_regex_as_ctx(struct doca_regex *regex);
```

regex [in]
The RegEx instance to convert.

> Note: Must remain valid until after the context is no longer required.

Returns
`doca_ctx` object on success; otherwise `NULL`.

4.4. Device Query API

This section details API calls that can be used to query a DOCA device regarding its RegEx functionality.

4.4.1. doca_regex_is_supported

Validates whether a DOCA device supports RegEx.

```
doca_error_t doca_regex_is_supported(struct doca_devinfo const *devinfo);
```

devinfo [in]
The device to check.

Returns

- `DOCA_SUCCESS` – device can be used with `doca_regex`
- `DOCA_ERROR_INVALID_VALUE` – received invalid input; the `devinfo` is not correct
- `DOCA_ERROR_NOT_SUPPORTED` – device cannot be used with `doca_regex`

4.4.2. doca_regex_get_hardware_supported

Validates whether a DOCA device supports hardware accelerated RegEx operations.

```
doca_error_t doca_regex_get_hardware_supported(struct doca_devinfo const *devinfo);
```

devinfo [in]
The device to check.

Returns

- `DOCA_SUCCESS` – hardware accelerated RegEx offloading is supported
- `DOCA_ERROR_INVALID_VALUE` – received invalid input; the `devinfo` is not correct
- `DOCA_ERROR_NOT_SUPPORTED` – device cannot hardware accelerate RegEx

4.4.3. doca_regex_get_maximum_job_size

Returns the maximum accepted job size for the selected device.

```
doca_error_t doca_regex_get_maximum_job_size(struct doca_devinfo const *devinfo,
 uint64_t *max_job_len);
```

devinfo [in]
The device to check.

max_job_len [out]
The maximum job size in bytes.

Returns

- `DOCA_SUCCESS` – `max_job_len` is populated correctly
- `DOCA_ERROR_INVALID_VALUE` – received invalid input; the `devinfo` is not correct
- `DOCA_ERROR_NOT_SUPPORTED` – device does not support RegEx

4.4.4. doca_regex_get_maximum_non_huge_job_size

Determines the maximum job size supported by this device without requiring the huge job emulation feature.

```
doca_error_t doca_regex_get_maximum_non_huge_job_size(struct doca_devinfo const
 *devinfo, uint64_t *max_job_len);
```

devinfo [in]
The device to check.

max_job_len [out]
The maximum job size in bytes.

Returns

- `DOCA_SUCCESS` – `max_job_len` is populated correctly
- `DOCA_ERROR_INVALID_VALUE` – received invalid input; the `devinfo` is not correct
- `DOCA_ERROR_NOT_SUPPORTED` – device does not support RegEx

4.4.5. doca_regex_job_get_supported

Determines if a given job type is supported for a given device.

```
doca_error_t doca_regex_job_get_supported(struct doca_devinfo const *devinfo, enum
 doca_regex_job_types job_type);
```

devinfo [in]
The device to check.

job_type [in]
Job type to validate.

Returns

- `DOCA_SUCCESS` – job type is supported by device
- `DOCA_ERROR_INVALID_VALUE` – received invalid input; the `devinfo` is not correct
- `DOCA_ERROR_NOT_SUPPORTED` – job type is not supported by device

4.4.6. doca_regex_search_job_flag_get_highest_priority_r

Determines if highest priority match is supported for a given device when submitting `doca_regex_job_search` jobs.

```
doca_error_t doca_regex_search_job_flag_get_highest_priority_match_supported(struct
 doca_devinfo const *devinfo);
```

devinfo [in]
The device to check.

Returns

- `DOCA_SUCCESS` – job type is supported by device

- DOCA_ERROR_INVALID_VALUE – received invalid input; the devinfo is not correct
- DOCA_ERROR_NOT_SUPPORTED – job type is not supported by device

4.4.7. doca_regex_search_job_flag_get_stop_on_any_mat

Determines if "stop on any" match is supported for a given device when submitting doca_regex_job_search jobs.

```
doca_error_t doca_regex_search_job_flag_get_stop_on_any_match_supported(struct
 doca_devinfo const *devinfo);
```

devinfo [in]
The device to check.

Returns

- DOCA_SUCCESS – job type is supported by device
- DOCA_ERROR_INVALID_VALUE – received invalid input; the devinfo is not correct
- DOCA_ERROR_NOT_SUPPORTED – job type is not supported by device

4.5. Programming RegEx

The RegEx engine must be programmed prior to submitting doca_regex_job_search jobs.

While the RXP engines are programmed using a compiled rules database, doca_regex supports programming using either compiled or uncompiled rules. For uncompiled rules doca_regex compiles them internally (using a default configuration) to generate a compiled database rules file. For a finer control over the compilation process (e.g., to optimize or alter the default compiler configuration and behavior), use the external RXP Compiler (rxpc) and provide the compiled database to the relevant API calls.

Note: Calling doca_regex_set_hardware_xxxx_rules can occur before or after starting doca_regex but the device must be programmed before enqueuing jobs.

4.5.1. Reprogramming Guidelines

While doca_regex supports on-the-fly reprogramming of the RXP hardware devices, care should be given to ensure the reprogramming action does not interrupt or produce undesired results.

The BlueField DPU series provides two instances of the RXP hardware engine and (re)programming is commenced serially to provide uninterrupted regular expression processing (i.e., zero downtime). This process produces some caveats that must be understood to prevent undesired behavior.

As most reprogramming events typically last only a few milliseconds, it is recommended that, prior to calling the (re)programming APIs, all results from doca_regex are ignored until the API's call return. However, if you want uninterrupted processing of regular expressions, either:

- Ensure that the new rules being programmed contain all the old rules, and you are simply adding more rules (with new rule IDs and subsets); or
- Understand that any rules that are existed in the old database (i.e., using the same rule ID) but are now different in the new database, will return incorrect matches until the API returns, and that any new rules not existing in the old database (i.e. they are using a new rule ID in the new database), will be valid immediately

As previously stated, when the (re)programming APIs return, the regular expression matches will wholly reflect the rules present in the new reprogrammed rules database.

4.5.2. doca_regex_set_hardware_compiled_rules

This function specifies the compiled rules data to be used by the RegEx engine.

```
doca_error_t doca_regex_set_hardware_compiled_rules(struct doca_regex
 *regex, void const *rules_data, size_t rules_data_size);
```

regex [in]
The DOCA RegEx instance.

rules_data [in]
A pointer to a buffer of pre-compiled binary rules data.

rules_data_size [in]
The size of the binary rules data in bytes.

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields are invalid
- `DOCA_ERROR_NO_LOCK` – unable to gain exclusive control of RegEx instance
- `DOCA_ERROR_IN_USE` – RegEx instance is currently started and in-use
- `DOCA_ERROR_NO_MEMORY` – unable to allocate memory to store a copy of the rules

Note: The caller retains ownership of the data pointed to by `rules_data` and is responsible for freeing it when they no longer require it. The engine will make a copy of this data for its own purposes.

Note: This API call is mutually exclusive with the uncompiled rules API call (doca_regex_set_hardware_uncompiled_rules).

4.5.3. doca_regex_get_hardware_compiled_rules

This function gets the compiled rules data that is currently in use by the RegEx engine.

```
doca_error_t doca_regex_get_hardware_compiled_rules(struct doca_regex
 *regex, void const *rules_data, size_t rules_data_size);
```

regex [in]
The DOCA RegEx instance.

rules_data [out]
Values to populate with a pointer to an array of bytes containing the compiled rules in used by the RegEx engine.

rules_data_size [out]
The size, in bytes, of the memory pointed to by the `rules_data` field (assuming data != `NULL`).

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields are invalid
- `DOCA_ERROR_NO_MEMORY` – unable to allocate memory to store a copy of the rules

Note: The caller is responsible for the memory pointed to by `rules_data` field and therefore must free it when they no longer require it.

4.5.4. doca_regex_set_hardware_uncompiled_rules

This function specifies the compiled rules data to be used by the RegEx engine.

```
doca_error_t doca_regex_set_hardware_uncompiled_rules(struct doca_regex
 *regex, void const *rules_data, size_t rules_data_size);
```

regex [in]
The DOCA RegEx instance.

rules_data [out]
Values to populate with a pointer to an array of bytes containing the compiled rules in used by the RegEx engine.

rules_data_size [out]
The size, in bytes, of the memory pointed to by the `rules_data` field (assuming data != `NULL`).

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields are invalid
- `DOCA_ERROR_NO_MEMORY` – unable to allocate memory to store a copy of the rules

Note: The caller is responsible for the memory pointed to by `rules_data` field and therefore must free it when they no longer require it.

Note: This API call is mutually exclusive with the compiled rules API call (doca_regex_set_hardware_compiled_rules).

4.5.5. doca_regex_get_hardware_uncompiled_rules

This function gets the uncompiled rules data that is currently in use by the RegEx engine.

```
doca_error_t doca_regex_get_hardware_uncompiled_rules(struct doca_regex
 *regex, void const *rules_data, size_t rules_data_size);
```

regex [in]
The DOCA RegEx instance.

rules_data [out]
Values to populate with a pointer to an array of bytes containing the compiled rules in used by the RegEx engine.

rules_data_size [out]
The size, in bytes, of the memory pointed to by the `rules_data` field (assuming data != `NULL`).

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields are invalid
- `DOCA_ERROR_NO_MEMORY` – unable to allocate memory to store a copy of the rules

Note: The caller is responsible for the memory pointed to by `rules_data` field and therefore must free it when they no longer require it.

4.6. DOCA RegEx Setup

This section details the API calls required to setup DOCA RegEx with memory to store received matches, adjust the number of queue pairs, etc.

4.6.1. doca_regex_set_workq_matches_memory_pool_size

Each work queue attached to the RegEx instance gets a pool allocator for matches. Set this value to set the maximum number of matches that can be stored for a given work queue.

```
doca_error_t doca_regex_set_workq_matches_memory_pool_size(struct doca_regex *regex,
 uint32_t pool_size);
```

regex [in]
The DOCA RegEx instance.

pool_size [in]
The number of items to have available to each work queue.

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields are invalid
- `DOCA_ERROR_NO_MEMORY` – unable to allocate memory to store a copy of the rules
- `DOCA_ERROR_IN_USE` – RegEx instance is currently started and in-use

Note: The range of valid values for this property depend upon the device in use.
This means that acceptance of a value through this API does not ensure the value is acceptable. This is validated as part of starting the context.

4.6.2. doca_regex_get_workq_matches_memory_pool_siz

This function gets the uncompiled rules data that is currently in use by the RegEx engine.

```
doca_error_t doca_regex_set_workq_matches_memory_pool_size(struct doca_regex *regex,
 uint32_t pool_size);
```

regex [in]
The DOCA RegEx instance.

pool_size [out]
The number of items to have available in each work queue.

Returns

- DOCA_SUCCESS – the RegEx instance accepted the supplied rules data
- DOCA_ERROR_INVALID_VALUE – one or more input fields are invalid

4.7. Configuration Options

DOCA RegEx has options that alter its mode of operation and control certain features. This section details those API calls and their related impact.

4.7.1. doca_regex_set_huge_job_emulation_overlap_size

This API call enables the Huge Job Emulation functionality of the DOCA RegEx instance, allowing it to find matches in data that exceeds the maximum job length of a particular RegEx device. For example, the BlueField RXP hardware device has a maximum job size of 16KB.

This function is provided with a size parameter that indicates the size of overlap to use in the Huge Job Emulation algorithm. This algorithm breaks up the incoming job data into fragments. Therefore, the overlap size causes data from the previous fragment to be prepended to the start of the next fragment.

As this overlap impacts performance (job data may get searched multiple times) the overlap size should be kept to a minimum value that still guarantees that matches are found.

```
doca_error_t doca_regex_set_huge_job_emulation_overlap_size(struct doca_regex
 *regex, uint16_t nb_overlap_bytes);
```

regex [in]
The DOCA RegEx instance.

nb_overlap_bytes [in]
The number of items to have available to each work queue.

Returns

- DOCA_SUCCESS – the RegEx instance accepted the supplied rules data
- DOCA_ERROR_INVALID_VALUE – one or more input fields were invalid
- DOCA_ERROR_NO_LOCK – unable to gain exclusive control of RegEx instance
- DOCA_ERROR_IN_USE – RegEx instance is currently started and in-use

4.7.2. doca_regex_get_huge_job_emulation_overlap_size

Gets the size of overlap to use when a job exceeds a devices maximum search size.

```
doca_error_t doca_regex_get_huge_job_emulation_overlap_size(struct doca_regex const
 *regex, uint16_t *nb_overlap_bytes);
```

regex [in]
The DOCA RegEx instance.

nb_overlap_bytes [out]
The number of bytes to overlap.

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields were invalid

4.7.3. doca_regex_set_in_order_responses_enabled

Configure `doca_regex` to ensure the ordering of responses or matches is kept in the same order that jobs are sent. This option must be set prior to starting the DOCA RegEx instance.

As the RXP hardware processes RegEx jobs in parallel, results can be returned out of order. If your application expects to see results flow back in the same order they are submitted, use this API call to enable in-order responses.

```
doca_error_t doca_regex_set_in_order_responses_enabled(struct doca_regex const
 *regex, bool enabled) ;
```

regex [in]
The DOCA RegEx instance.

enabled [in]
Boolean value indicating if results should be returned in-order.

Returns

- `DOCA_SUCCESS` – the RegEx instance accepted the supplied rules data
- `DOCA_ERROR_INVALID_VALUE` – one or more input fields were invalid
- `DOCA_ERROR_NO_LOCK` – unable to gain exclusive lock of the Regex instance
- `DOCA_ERROR_IN_USE` – RegEx instance was in-use

Chapter 5. DOCA RegEx Samples

This document describes RegEx samples based on the DOCA RegEx library. These samples illustrate how to use the DOCA RegEx API to configure, send, and receive data buffers to and from the BlueField RegEx engine.

Processing the data to detect matches requires compilation of regular expressions rules file. The compiled file, `.ROF2`, is loaded to the RegEx engine using DOCA RegEx APIs.

5.1. Sample Prerequisites

Developing an application that leverages the RegEx engine requires pre-run setup:

1. Allocate hugepages:
   ```
   echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```
2. Make sure the RegEx engine is active:
   ```
   systemctl status mlx-regex
   ```
   If the status is inactive (`Active: failed`), run:
   ```
   systemctl start mlx-regex
   ```

To run the application, the RegEx compiled rule file (`.rof2.binary`) must be supplied with it. To compile the sample rules file, run:

```
cd /opt/mellanox/doca/samples/doca_regex/<sample_name>/
rxpc -V bf2 -f <rules_file_name>.txt -p 0.01 -o /tmp/sample_regex_rules
```

Note: For more information, refer to NVIDIA RXP Compiler Tool Guide.

5.2. Running the Sample

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA samples.
2. To build a given sample:
   ```
   cd /opt/mellanox/doca/samples/doca_regex/<sample_name>
   meson build
   ```

```
ninja -C build
```

Note: The `binary doca_<sample_name>` will be created under `./build/`.

3. Sample (e.g., `regex_scan`) usage:

```
Usage: doca_regex_scan [DOCA Flags] [Program Flags]
DOCA Flags:
  -h, --help                        Print a help synopsis
  -v, --version                     Print program version information
  -l, --log-level                   Set the log level for the program
 <CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60>

Program Flags:
  -p, --pci-addr <PCI-ADDRESS>      RegEx device PCI address
  -r, --rules <path>                Path to compiled rules file (rof2.binary)
  -d, --data <path>                 Path to data file
```

For additional information per sample, use the `-h` option:

```
./build/doca_<sample_name> -h
```

5.3. Samples

5.3.1. RegEx Scan

This sample illustrates how to scan data to find matches according to regular expression patterns.

The sample logic includes:

1. Accepting RegEx rules file path and data to scan.
2. Configuring DOCA RegEx device (loading compiled rules, mempool allocation, etc.) to enable RegEx engine to receive jobs.
3. Splitting the user data to 6 chunks.
4. Sending the chunks to RegEx engine, each chunk is a RegEx job.
5. HW scanning data and returning a list of matches.
6. Reporting the results.

References:

- `/opt/mellanox/doca/samples/doca_regex/regex_scan/regex_scan_sample.c`
- `/opt/mellanox/doca/samples/doca_regex/regex_scan/regex_scan_main.c`
- `/opt/mellanox/doca/samples/doca_regex/regex_scan/meson.build`
- `/opt/mellanox/doca/samples/doca_regex/regex_scan/regex_rules.txt`
- `/opt/mellanox/doca/samples/doca_regex/regex_scan/data_to_scan.txt`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: “NVIDIA”) make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com