

Reference Storage Platform (RSP)

USER GUIDE

Release 1.0




If you have any questions, please contact us directly at the FTL channel on Slack using the following [link](#).

Table of Contents


Reference Storage Platform (RSP)	1
Hardware Configuration	4
Storage Target Configuration	4
Initiator Configuration	5
BIOS settings	5
Software Configuration.....	5
StarWind VM.....	6
Build your own (StarWind) VM	6
Container	6
Build your own container.....	6
Target configuration	6
Build CSAL container	8
How to run FIO on the initiator.....	9
Appendix A –Target configuration	10
Appendix B – Initiator configuration.....	12

Hardware Configuration

Storage Target Configuration

Item	Description
Storage Target Platform	 SYS-220U-TNR 2U SuperServer Products Supermicro
BIOS Version	1.4b
CPU Model	Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz
NUMA Node(s)	2
OS	Fedora 37 (Server Edition)
Kernel	6.3.8-100.fc37.x86_64
DRAM Installed	756GB (16x16GB DDR4 3200MT/s [3200MT/s])
Huge Pages Size	2048 kB
NIC Summary	Ethernet Controller X710 for 10GBASE-T, Mellanox Technologies MT27800 Family [ConnectX-5]
Storage	Solidigm QLC P5336 61TB, Solidigm's first generation SLC
SPDK	23.05
CSAL	1.0
FIO	3.29

Initiator Configuration

Item	Description
Initiator Platform	 6019U-TN4RT 1U SuperServers Products Super Micro Computer, Inc.
BIOS Version	3.8b
CPU Model	Intel(R) Xeon(R) Gold 6139 CPU @ 2.30GHz
NUMA Node(s)	2
OS	Fedora 37 (Server Edition)
Kernel	6.3.8-100.fc37.x86_64
DRAM Installed	196 GB (12x16GB DDR4 3200MT/s [3200MT/s])
NIC Summary	Ethernet Controller X710 for 10GBASE-T, Mellanox Technologies MT27800 Family [ConnectX-5]
SPDK	23.05
CSAL	1.0
FIO	3.29

BIOS settings

Item	Description
CPU cores	Hyper-threading: Disabled.
CPU Power and Performance Policy	CPU Power and Performance Policy: <ul style="list-style-type: none">• “Performance” for Target• “Performance” for Initiators
	CPU C-state No Limit CPU P-state Enabled Enhanced Intel® SpeedStep® Tech Enabled Turbo Boost Enabled

Software Configuration

From a software standpoint, there are multiple configurations to realize RSP. All the configurations below refer to available target implementations.

StarWind VM

One option to realize RSP is to use StarWind VM that includes (1) StarWind NVMe-oF target package, (2) StarWind UI (3) CSAL. On the target system, please follow StarWind instructions that can be downloaded here: <https://www.solidigm.com/content/solidigm/us/en/support-page/drivers-downloads/ka-01786.html>. To run FIO on a separate system (initiator), please see [Appendix B](#).

Build your own (StarWind) VM

Another option is to build an open source VM, which can be configured with either (1) StarWind's NVMe-oF target or (2) SPDK NVMe-oF target. On the target system, please follow StarWind instructions that can be downloaded here: <https://www.solidigm.com/content/solidigm/us/en/support-page/drivers-downloads/ka-01786.html>. To run FIO on a separate system (initiator), please see [Appendix B](#).

Container

NOTE: This container image is pre-configured for the RSP very specific hardware config outlined in this document. Do not run this container image on any other hardware configuration. If you want to run CSAL in a container on a different platform, follow the instructions for CSAL container creation below.

On the target system, load container image provided at the Solidigm website: <https://www.solidigm.com/content/solidigm/us/en/support-page/drivers-downloads/ka-01786.html>.

Download "csal-image-v1.tar.gz", unzip it and follow the steps below:

```
gunzip csal-image-v1.tar.gz
sudo docker load < csal-image-v1
sudo docker run --privileged -e "SPDK_ARGS=-m 0x2" -e "SPDK_NO_LIMIT=1" -e "EXPOSE_SPDK=true" --
network host -v /dev/hugepages:/dev/hugepages csal-app
```

To run FIO on a separate system (initiator), please see [Appendix B](#).

Build your own container

There's also an option to build a container from source, using SPDK and patch it with RSP patch (contains CSAL).

Target configuration

Before you start:

- 1) All drives cache and base are located on the same socket when using multi-socket server for optimal performance.
- 2) Have 5GB free space for installation files in your working directory.
- 3) Capture model numbers of the drives that you intend to use as a base device and as the cache device. These will be needed for creating FTL.

- a. Execute the following command to check if NVMe cli is installed: > nvme list
 - b. Model number should also be listed on the drive label itself.
- 4) Switch to superuser:
- a. > sudo su
 - b. > enter password

Item	Description
Install Docker Engine on Fedora Install docker-compose from Fedora repository. Verify docker-compose 1.29.2 version. For more information, please visit: https://docs.docker.com/engine/install/fedora/	<pre># dnf install docker-compose # dnf -y install dnf-plugins-core # dnf config-manager --add-repo https://download.docker.com/linux/fedora/docker-ce.repo # dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin # systemctl start docker</pre>
Install nvme-cli build from source. Default nvme-cli 2.2 has a known issue	<pre># git clone https://github.com/mmkayPL/nvme-cli.git # cd nvme-cli # git checkout origin/nvme-format-block-size-fix-1_2 # dnf install -y zlib-devel # meson setup --force-fallback-for=libnvme .build # make -j # make install # cd .. # rm -rf nvme-cli</pre>
Download SPDK source code hosted in a public repository	<pre># git clone https://github.com/spdk/spdk.git</pre>
Download CSAL patch "csal_rsp_v1.0.patch"	<pre># wget https://www.solidigm.com/content/solidigm/us/en/support-page/drivers-downloads/ka-01786.html/csal_rsp_v1.0.patch</pre>
Initialize SPDK submodules	<pre># cd ./spdk/ # git submodule update --init</pre>
Install SPDK dependencies	<pre># ./scripts/pkgdep.sh</pre>
Checkout the v23.05 Release of SPDK	<pre># git checkout tags/v23.05 -b v23.05</pre>
Update the submodules to point at the right commit	<pre># git submodule update</pre>
From the base of the Container SPDK directory, apply the patch:	<pre># patch -p1 < ../csal_rsp_v1.0.patch</pre>
Tar-ball the CSAL directory so that it will be built and used for the SPDK container. From the base of the CSAL directory	<pre># tar cvfz ../spdk.tar.gz .</pre>
Copy the spdk.tar.gz file into the "Host SPDK" tree (v23.05) and into the directory: spdk/docker/build_base	<pre># cp ../spdk.tar.gz ./docker/build_base</pre>

Build CSAL container

Item	Description
To build the CSAL container, navigate to <path-to-spdk>/docker	<pre># cd ./docker # docker-compose build --no-cache build_base csal-target</pre>
<p>In <path-to-spdk>/docker/csdl-app, there is the “csdl-config.ini” file, which contains all the CSAL parameters.</p> <p>You can find more information at: <path-to-spdk>/docker/csdl-app/README</p>	<p>Configure csdl-config.ini</p> <ol style="list-style-type: none"> 1. Verify that model numbers for Base Device and Cache device match the devices in the server you intend to use for CSAL. <ol style="list-style-type: none"> a. <code>sudo nvme list -o json</code> b. Verify the “ModelNumber” output is what you intend to use. 2. Verify that network interface name matches with a high speed NIC, so max performance can be achieved. <ol style="list-style-type: none"> a. Run the following command: <code># ip a show</code> b. Verify the name of the network interface you want to export NVMe-o TCP targets
Run “discover-drives.py” . Do not edit the ftl-env.sh file, it is generated after execution of discover-drives.py program.	<pre># python discover-drives.py</pre> <p>Verify that ftl-env.sh is present in following location <path-to-spdk>/docker/csdl-app</p>
Build an FTL instance	<pre># docker-compose build csal-target</pre>
Bind the NVMe drives to SPDK, for example:	<pre># cd ../../ # HUGEMEM=8192 ./scripts/setup.sh # cd ./docker</pre>
Start CSAL container. This step builds FTL instance(s), configure target, network, and sub-system and listeners. See Appendix A for details of configuration that container performs.	<pre># docker-compose up csal-target</pre>
<p>Stop CSAL target gracefully</p> <p>Note: If you intend to continue to run the workloads or tests, <u>do not</u> execute this command.</p>	<pre># docker-compose stop csal-target</pre>
<p>To return hugepages to the system:</p> <p>If you intend to continue to run the workloads or tests, <u>do not</u> execute this command.</p>	<pre># rm -f /dev/hugepages/ftl_*</pre>

How to run FIO on the initiator

Before you start:

- 1) Switch to superuser:
 - a. > sudo su
 - b. > enter password

Item	Description
Add nvme module from kernel	<pre># modprobe nvme-fabrics</pre>
Install FIO from the official FIO	<pre># git clone https://github.com/axboe/fio.git # cd fio # ./configure # make -j\$(nproc)</pre>
Install SPDK v23.05 from the official github page https://github.com/spdk/spdk	<pre># cd .. # git clone https://github.com/spdk/spdk.git # cd spdk # git submodule update --init # ./configure --with-fio=<path-to-fio> # make -j\$(nproc)</pre>
Run SPDK setup script to assign hugepages on the initiator	<pre># cd <path-to-spdk> # ./scripts/setup.sh</pre>
You can now run your own workload. For reference, here is a sample FIO command to do sequential preconditioning	<pre><path-to-fio>/fio/fio --direct=1 --size=100% -- filename wsr1n1:wsr2n1:wsr3n1:wsr4n1:wsr5n1:wsr6n1 -- ioengine=<path-to-spdk>/build/fio/spdk_bdev -- name=precondition --rw=write --bs=128k --iodepth=128 - -numjobs=1 --thread=1 --spdk_json_conf=<path-to- bdev.conf file>/bdev.conf</pre>
Create FIO job file (fio.ini). For more information, see Appendix B .	
Run workload dependent precondition (don't record performance)	<pre><path-to-fio>/fio/fio <path-to-fio.ini>fio.ini</pre>
Run workload (record performance)	<pre><path-to-fio>/fio/fio < path-to-fio.ini>fio.ini</pre>
To reproduce Solidigm's performance results, run the scripts that can be found in: https://www.solidigm.com/content/solidigm/us/en/support-page/drivers-downloads/ka-01786.html	<ol style="list-style-type: none">1) Update ip address in bdev_tcp_container.conf2) Update paths in the run_fio.sh

Appendix A –Target configuration

RSP container automatically creates the following CSAL config on target, using 3x SLC + 7x QLC SSDs:

- **SLC:** Solidigm first-gen SLC
- **QLC:** Solidigm QLC P5336 61 TB

For reference, we provide the commands that are executed to attach drives, build FTL instances, config SPDK target (create network transport, add subsystems, add namespaces and add listeners).

Configure target	Create 7x FTL instances <code>bdev_ftl_create -b ftl_0 -d ftl_0_qlc0n1 --core_mask 2 --overprovisioning 20 --cache nvc0n1p0 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_1 -d ftl_1_qlc1n1 --core_mask 4 --overprovisioning 20 --cache nvc0n1p1 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_2 -d ftl_2_qlc2n1 --core_mask 8 --overprovisioning 20 --cache nvc1n1p0 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_3 -d ftl_3_qlc3n1 --core_mask 10 --overprovisioning 20 --cache nvc1n1p1 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_4 -d ftl_4_qlc4n1 --core_mask 20 --overprovisioning 20 --cache nvc2n1p0 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_5 -d ftl_5_qlc5n1 --core_mask 40 --overprovisioning 20 --cache nvc2n1p1 --l2p_dram_limit 2048</code> <code>bdev_ftl_create -b ftl_6 -d ftl_6_qlc6n1 --core_mask 80 --overprovisioning 20 --cache nvc2n1p2 --l2p_dram_limit 2048</code> Create RAID-0 on top of the FTL bdevs: <code>bdev_raid_create -z 128 -r concat -n wsr -b "ftl_0 ftl_1 ftl_2 ftl_3 ftl_4 ftl_5 ftl_6"</code> Split the raid-0 bdev into 28 partitions. Each partition is 12.2 TB (11708004 MiB) <code>bdev_split_create -s 11708004 wsr 28</code>
NVMe-oF CSAL Network Configuration	Create TCP transport

	<pre> nvmf_create_transport --trtype TCP --max-queue-depth 256 --max-io-qpairs-per-ctrlr 127 --in-capsule-data-size 8192 --max-io-size 131072 --io-unit-size 131072 --max-aq-depth 128 --num-shared-buffers 2047 --buf-cache-size 32 --dif-insert-or-strip --c2h-success --sock_priority 0 --abort-timeout-sec 1 </pre>
<p>Configure subsystems and add listeners</p>	<p>Create NVMe-oF subsystems and add NVMe bdevs as namespaces</p> <pre> for i in \$(seq 1 28); do nvmf_create_subsystem nqn.2018-09.io.spdk:cnode\${i} \ -s SPDK00\${i} -d wsr\${(i-1)} -a -m 1; nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode\${i}\ wsrp\${(i-1)}; nvmf_subsystem_add_listener \ nqn.2018-09.io.spdk:cnode\${i} -t TCP -f ipv4 -s 4420 \ -a \${IP}; done </pre>

Appendix B – Initiator configuration

This is the configuration that describes the necessary steps to run FIO on the initiator system, and includes:

1. A sample SPDK configuration file which is used as an input in the FIO job file and
2. A sample FIO job file.

Create an SPDK configuration file (bdev.conf) in a preferred location. This file serves as input to FIO test execution. Please note that the IP address in the "traddr" field needs to be updated to the IP of the high-speed NIC of the target system.

This is an example file that can be used to run FIO against a 6-FTL configuration:

```
{
  "subsystems": [
    {
      "subsystem": "bdev",
      "config": [
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "trtype": "tcp",
            "adrfam": "IPv4",
            "name": "wsr1",
            "subnqn": "nqn.2016-06.io.spdk:cnode1",
            "traddr": "192.168.1.1",
            "trsvcid": "4420"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "trtype": "tcp",
            "adrfam": "IPv4",
            "name": "wsr2",
            "subnqn": "nqn.2016-06.io.spdk:cnode2",
            "traddr": "192.168.1.1",
            "trsvcid": "4420"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "trtype": "tcp",
            "adrfam": "IPv4",
            "name": "wsr3",
            "subnqn": "nqn.2016-06.io.spdk:cnode3",
            "traddr": "192.168.1.1",
            "trsvcid": "4420"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "trtype": "tcp",
            "adrfam": "IPv4",

```

```

        "name": "wsr4",
        "subnqn": "nqn.2016-06.io.spdk:cnode4",
        "traddr": "192.168.1.1",
        "trsvcid": "4420"
    },
    {
        "method": "bdev_nvme_attach_controller",
        "params": {
            "trtype": "tcp",
            "adrfam": "Ipv4",
            "name": "wsr5",
            "subnqn": "nqn.2016-06.io.spdk:cnode5",
            "traddr": "192.168.1.1",
            "trsvcid": "4420"
        }
    },
    {
        "method": "bdev_nvme_attach_controller",
        "params": {
            "trtype": "tcp",
            "adrfam": "Ipv4",
            "name": "wsr6",
            "subnqn": "nqn.2016-06.io.spdk:cnode6",
            "traddr": "192.168.1.1",
            "trsvcid": "4420"
        }
    }
}
]
}
]
}

```

Create an FIO job file and use the "bdev.conf" file that was created in the previous step. This is an example configuration of fio.ini:

```

[global]
name=csal_fio_tcp
ioengine=<path-to-spdk>/build/fio/spdk_bdev
thread=1
group_reporting=1
direct=1
norandommap=1
randrepeat = 0
scramble_buffers=1
verify=0
cpus_allowed_policy=split # one cpu per job
bs=4k
rw=randrw
rwmixread=70
numjobs=1
iodepth=128
runtime=3600s
time_based=1
percentile_list=10:30:50:70:90:99.0:99.9:99.99:99.999:99.9999:100
spdk_json_conf=<path-to-bdev.conf file>/bdev.conf

```

[filename0]
filename=wsr1n1
[filename2]
filename=wsr2n1
[filename3]
filename=wsr3n1
[filename4]
filename=wsr4n1
[filename5]
filename=wsr5n1
[filename6]
filename=wsr6n1