# KVCEPH USER GUIDE

# v0.8

MSL/ADS

# Table of Contents

# About KVCEPH

KVCEPH is a fork of CEPH object storage system, optimized to support Samsung Key-Value SSDs. It provides a new object store called KvsStore with the following two helper components: an onode prefetcher and an event –driven scheduler that can eliminate synchronous IOs in the data path to the devices. This document describes the high-level design of KVCEPH and introduces how to set up and run KVCEPH.

## I. Revision history

| Date | Description | Author |
|---|---|---|
| 2019-02-01 | First draft | Yangwook Kang |
| | | |
| | | |
| | | |
| | | |

## II. References

| Name | URL |
|---|---|
| KVCEPH SOURCE CODE REPOSITORY | https://github.com/OpenMPDK/KVSSD |
| CEPH SOURCE CODE REPOSITORY | https://github.com/ceph/ceph/ |
| CEPH Documentation | http://docs.ceph.com |
| | |
| | |

## III. Definitions, acronyms and abbreviations

| Term | Definition |
|---|---|
| KVSSD | Samsung Key-Value SSD |
| KVCEPH | A modified CEPH storage systems with the support for KVSSDs |
| OP Workers | A set of threads that process incoming requests |
| Object | A key-value pair |
| Onode | Object metadata |
| Object Store | An internal IO interface in CEPH that accepts object requests |
| KvsStore | A CEPH object store for KVSSDs |
| PG | Placement Group: a group of objects used for object placement |
| RocksDB | A default host-side key-value store used in CEPH |
| BlueStore | A default object store in CEPH |

# Introduction to KVCEPH

Based on CEPH Luminous release, KVCEPH provides the additional features, including KvsStore, Onode prefetcher and Event-driven request handler, to support KVSSDs efficiently.

Figure 1 shows the design of the event-driven scheduler, which manages a queue per placement group (PG) and notifies the events to workers using signals and *epoll* system call. This eliminates the static association between a group of request queues and OP workers. In the case of the default Sharded scheduler in CEPH, this association causes under-utilization as all *op* workers might not be used in some cases. Onode prefetcher is designed to better utilize SSD bandwidth by converting blocking attribute read operations inside OP workers to non-blocking reads. Prefetching takes place when OP worker deques an item from the per-PG queue, and an object in the queue is dynamically selected based on the current processing time. Finally, KvsStore processes transactions, read, and all other object requests such as attribute requests and ls, communicating with KVSSDs.

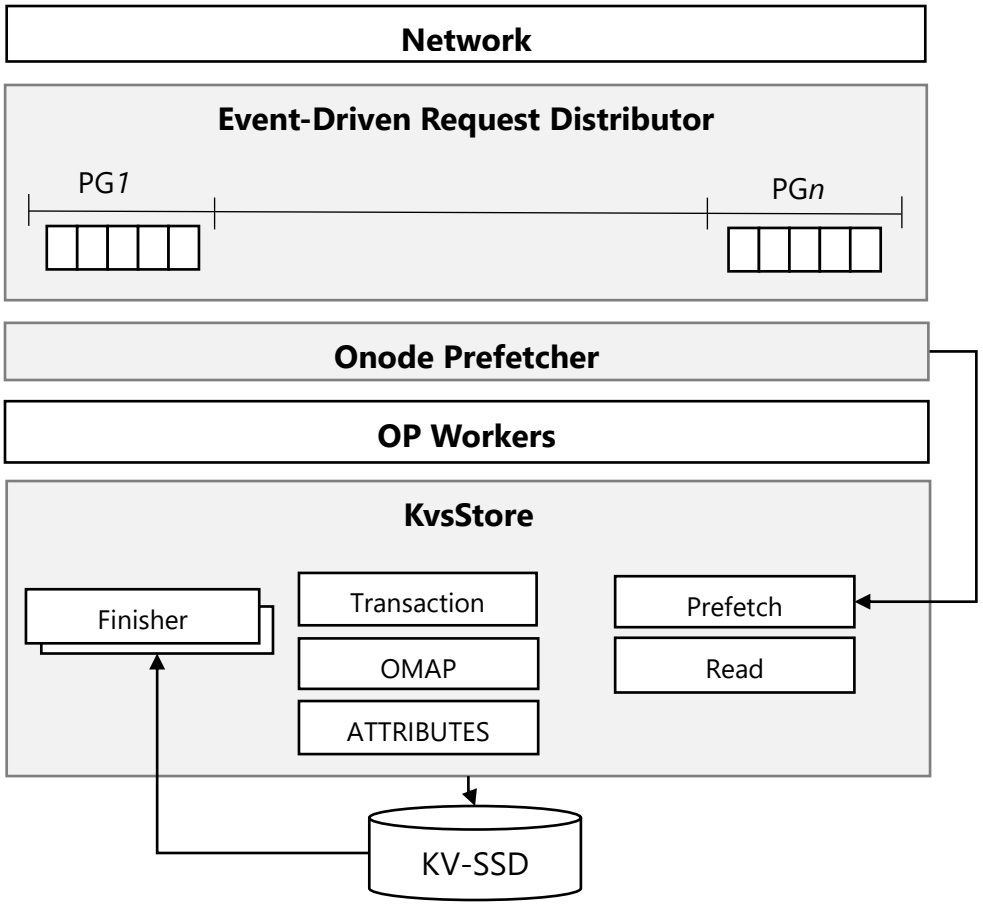


**Figure 1. Structure of KVCEPH OSD**

KvsStore does not require an additional IO stack for key-value processing as the key-value interface is now directly supported by the device. For example, compared to BlueStore (refer to Figure 2), KvsStore does not require the use of resource-intensive RockDB and a file system for data management. KvsStore simplifies metadata processing and recovery, while significantly saving compute and IO amplification. This reduced over-all system resource usage, and thus allowing better in-node scalability due to availability of more CPUs for device IOs.

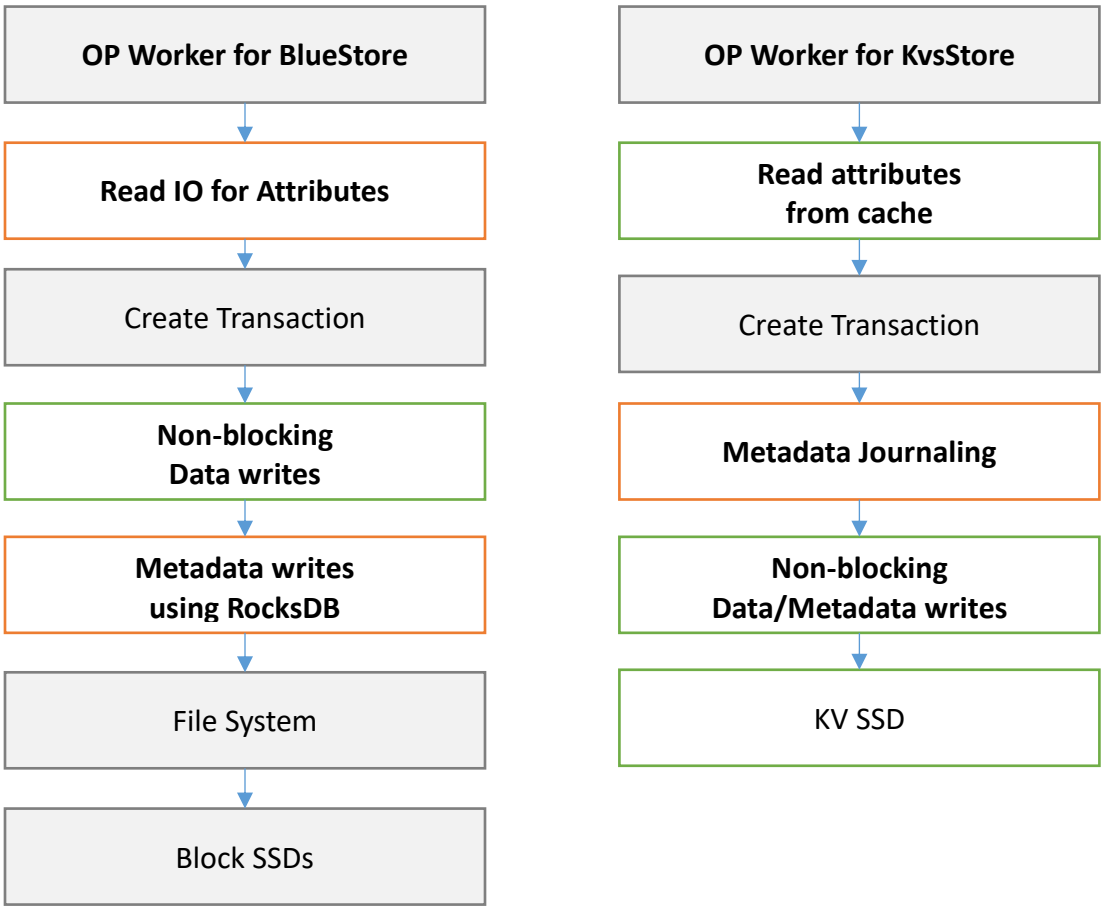


**Figure 2. A typical data path for write operations**

Note that metadata journaling in KvsStore is still being used to support transaction (all or nothing consistency). It can be removed in the future once the device firmware supports batch operations.

# Install and Configure

KVCEPH runs on any Linux environment CEPH is supported. The procedure of building KVCEPH is the same as that of CEPH but to access KVSSDs directly, a KV kernel driver needs to be installed and loaded prior to run KVCEPH. This section describes the steps to install KVCEPH and the kernel driver.

## I. Checking out the source

You can clone from github with

```
git clone git@github.com/OpenMPDK/KVCeph.git
```

or, if you are not a github user,

```
git clone https://github.com/OpenMPDK/KVCeph.git
```

Note that the source code needs to be stored in the root file system, not KVSSD.

## II. Install KV Kernel Driver

You can download the KV kernel driver from the following github repository.

git clone git@github.com/OpenMPDK/KVSSDCore.git

The source code of a KVSSD kernel driver is under ./PDK/driver/PCIe/kernel_driver. Find the directory that matches your kernel version and run the following commands to load the driver

```
make
./re_insmod.sh
```

## III. Building KVCEPH

*cmake* is used to set compile time parameters and output directories. The dependent libraries will be installed by install-deps.sh. KVCEPH does not compile-time parameters.

### III-1. Install dependent packages

Install-deps.sh is available under the KVCEPH directory.

```
./install-deps.sh
```

### III-2. Compile

The original Ceph build instruction can be used. For example:

```
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=./ceph-runtime ..
make
make install
(This will install the binaries under the ceph-runtime directory)
```

Note that this target directory specified by CMAKE_INSTALL_PREFIX is the directory that needs to be copied to other OSD nodes.

For a list of Ceph cmake options: https://github.com/ceph/ceph/

## IV. Configuring KVCEPH

The default CEPH configuration file locations include $CEPH_CONF, /etc/ceph/ceph/conf, ~/.ceph/conf, ./ceph.conf as explained in the CEPH documentation (http://docs.ceph.com/docs/jewel/rados/configuration/ceph-conf/).

Prior to running KVCEPH, the following two parameters need to be configured.

- osd_objectstore = kvsstore
    - sets the type of object store to use
- kvsstore_dev_path = /dev/device_path
    - sets the path to a Samsung KV SSD per OSD

Note that these two parameters are mandatory. *osd objectstore* needs to be defined in the [*global*] section in ceph.conf, and *kvsstore_dev_path* should be set in each OSD section. Please also see a sample configuration file, build/sample.conf under the KVCEPH root folder, which demonstrates the use of these parameters.

Optionally, the following parameters can be set to tune the performance and memory usage.

- kvsstore_readcache_bytes = <size in bytes>
    - the maximum size of read cache in bytes
    - default value: 1073741824 (1GB)
- kvsstore_max_cached_onodes= <number of onodes>
    - the maximum number of cached onodes
    - default value: 100000
- enable_onode_prefetch
    - Enable onode prefetching
    - Possible values:
        - "deq": dequeuer-time prefetching
        - "enq": enqueue-time prefetching
        - "disabled" : disable prefetching [default]
- op_scheduler
    - Set the type of scheduler. Possible values:
        - "rr": Round-robin scheduler
        - "epoll": Epoll scheduler
        - "sharded": Sharded scheduler – CEPH's default scheduler [default]
    - Note that epoll scheduler might open many files (>= 1024) internally so the number of maximum open files in the operating system can be reached. Please change the limit to a large number (e.g. 999999) in your operating system.

**Please note:**

1. The default value of osd_heartbeat_grace is changed from 20 to 90, because deep scrub operations can take longer than 20 seconds.
2. The default value of osd_max_object_size is changed from 128*1024L*1024L to 2*1024L*1024L, as KV-SSD supports objects upto 2 MB (2097152 bytes).

# Quickstarts

There are various ways to set up and deploy CEPH as explained at http://docs.ceph.com/docs/jewel/rados/deployment/. Since KVCEPH does not limit any functionality in CEPH, the same procedure can be used to deploy KVCEPH. This document describes one of the possible deployment with the following simple cluster configuration as an example.

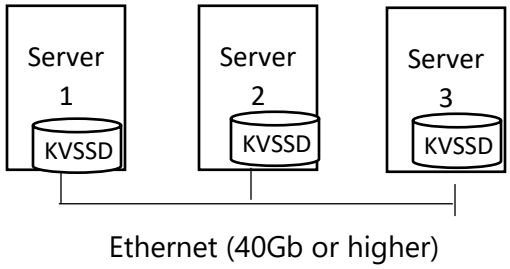


**Figure 3. A cluster with three nodes**

CEPH requires at least three server daemons to be started: a monitor server, a master server, and an OSD server. In this example, we assume that one monitor and one master servers run on the server 1, and each server contains one OSD server and one KVSSD.

## I. Running KVCEPH Locally

Before deploying KVCEPH to multiple storage nodes, a basic functionality and configuration can be checked by running KVCEPH locally in a debug mode.

For convenience, the CEPH utility vstart.sh, which set up a local test cluster, is modified to support KVCEPH. This script internally generates a configuration file under the current directory so configuration file does not need to be created manually. To run a local cluster with vstart.sh:

### I-1. Set the device path in vstart.sh

By default, kvsstore_dev_path is set to /dev/nvme0n1 in vstart.sh.
Please modify the line that defines kvsstore_dev_path in vstart.sh if needed.

### I-2. Run a test cluster with KvsStore

```
cd build
make vstart        # builds just enough to run vstart
sudo CEPH_DEV=1 MON=1 MDS=0 OSD=1 ../src/vstart.sh -n -x --kvsstore
./bin/ceph -s
```

Please also check out/osd.0.log file to see if there were any issue in setting up KVCEPH.

## I-3. Generate workloads

Once KVCEPH is loaded successfully, it can be tested with *rados* utility, which is under the /build/bin directory. It contains various object IO commands including put, get, delete, and a simple read/write benchmark.

For example:

```
# create a pool that does not generate a replication
build$ sudo ./bin/ceph osd pool create pool1 100
build$ sudo ./bin/ceph osd pool application enable pool1 cephfs
build$ sudo ./bin/ceph osd pool set pool1 size 1
build$ sudo ./bin/ceph osd pool set pool1 min_size 1
# write 1000 4KB objects
build$ sudo ./bin/rados bench -p pool1-b 4096 --max-objects 1000 --run-name 1 -t 64 40 write --no-cleanup
```

**Disclaimer:** The block-size for rados bench needs to be set using –b <objectsize> option, as by default the object size is assumed to be 4MB and KV-SSD supports objects upto 2 MB (i.e. –b 2097152).

## II. Running KVCEPH with Multiple Nodes

We introduce the manual deployment method in this document, which does not utilize CEPH deploy utilities. Other methods can be found at Original Ceph Deploy Manual Page .

To deploy CEPH, CEPH binary files and configuration files need to be copied to all OSD nodes. In this example, we assume that CEPH binaries are under /build/ceph-runtime (can be specified with cmake) and configuration files will be generated in Server 1 and copied to Server 2 and Server 3 using the same 'cephuser' account in all three nodes.

### II-1. Prepare user accounts

- Create a 'cephuser' account in all three nodes and enable passwordless login from server 1.

### II-2. Set up Monitors, Masters, and OSDs

In this step, we need to create a unique ID of a cluster and shared key rings, and add that information to a configuration file. We provide a script under /build/kvceph-scripts that can automate this process and generate a configuration file.

Please open each script to adjust the device paths in each server node and node IPs defined in the top of the file before running it.

The script runs the following tasks internally:

- Copy /build/ceph-runtime to the ~cephuser/ceph-runtime directory in other server nodes.
- Create keyrings for OSD, clients, and monitors and update ceph.conf accordingly
- Create a ceph-deploy directory in cephuser's home directory in each server and store keyrings and configuration files
- Run Monitors, Masters, OSDs

The configuration file generated by the script would be something like this and the full version of this sample configuration file is also available at /build/sample.conf:

```
[global]
        ...
        osd_objectstore = kvsstore
        ...
    [osd.0]
        # settings affect osd.0 only
        kvsstore_dev_path = (nvme device path in Server 1)
        cluster addr = (cluster network IP of Server 1)
        public addr =  (public network IP of Server 1)
        host = Server 1
        ...
    [osd.1]
        ...
        kvsstore_dev_path = (nvme device path in Server 2)
        cluster addr = (cluster network IP of Server 2)
        public addr =  (public network IP of Server 2)
        host = Server 2

    [osd.2]
        ...
        kvsstore_dev_path = (nvme device path in Server 3)
        ...
```

The script can create one monitor, one master, and many OSDs, but it can be easily changed to support multiple monitors and masters.

Another way to set up the CEPH servers is to manually create and deploy them using a set of commands introduced in this CEPH document [http://docs.ceph.com/docs/mimic/install/manual-deployment/], or use their ceph-deploy utility.

Once the cluster is correctly set and running, the 'ceph -s' command would show that the current status of each server.

# Howto-guides

## I. Run unit tests

Unit tests can be a quick way to check if the KVSSD and KvsStore are working as intended. To build and run unit tests:

```
cd build
sudo make ceph_test_kvsstore
(optional) sudo rm -rf kvsstore.test_temp_dir
sudo ./bin/ceph_test_kvsstore
```

kvsstore.test_temp_dir is created during unit testing and deleted at the end. If the unit tests were interrupted, this file can be remained undeleted and needs to be manually deleted.

## II. Format KVSSD

**Quick format:**

```
sudo nvme format (device path) -s0 -n1
```

**Secure erase:**

```
sudo nvme format (device path) -s1 -n1
```

## III. Log file locations

In a local test cluster, the log files will be created under /build/out/osd.0.log

If the cluster is deployed with our deploy scripts, they will be stored under ~cephuser/ceph-deploy/out/osd.x.log

# IV. KVSSD firmware update

```
sudo nvme fw-download (device path) –f=/path/to/kvfirmware
sudo nvme fw-activate (device path) --slot=0 --action=1

<powercycle>
```