



*BY Developers FOR Developers*

**Storage Developer Conference**  
**September 22-23, 2020**

# **pynvme: an open, fast and extensible NVMe SSD test tool**

**Crane Chu, Engineer, Founder**  
**GENG YUN Technology Pte Ltd**





# Requirement

# Changes in SSD

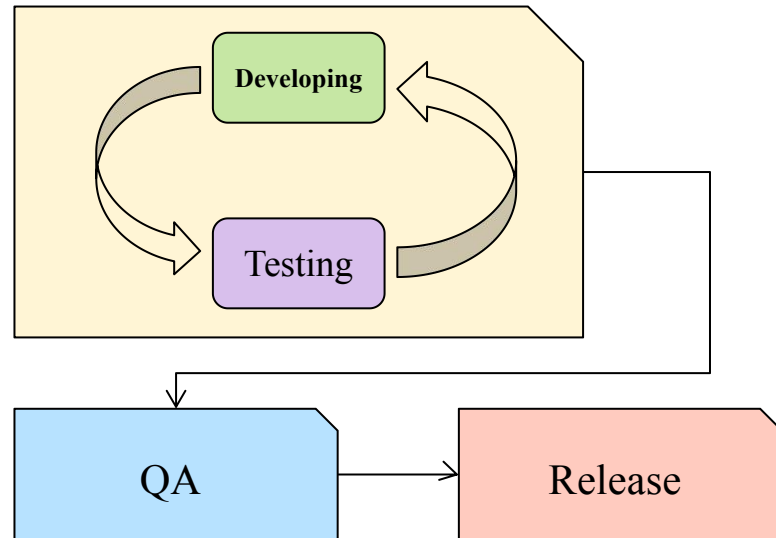
- SSD has been changing for the decade:
  - media:
    - SLC, MLC, TLC, QLC
    - 2D => 3D
    - PCM, 3D-Xpoint...
  - host:
    - PATA, SATA, PCIe/NVMe
    - open-channel
    - up coming: ZNS, KV, ... ?
  - DRAM
  - form factor
- Agile: good for the constant change and uncertainty



# Agile Testing

- Developing and Testing are done interactively and iteratively.
- QA verifies the product of Dev/Test for customers.
- Testing and QA are different. Testing tools and QA tools are also different.
- Most available tools in the market are QA tools.

test	QA
for developer	for customer
before checkin	before release
automatic	manual
white-box	black-box
short	long
changing	stable



# Experience with dnvme

- extended dnvme with a python wrapper in user space, thus developers can write test scripts in Python.
- Some essential problems:
  - performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - memory: user can only allocate virtual memory
- Then, SPDK comes ...
  - high performance
  - user space driver
  - DMA memory

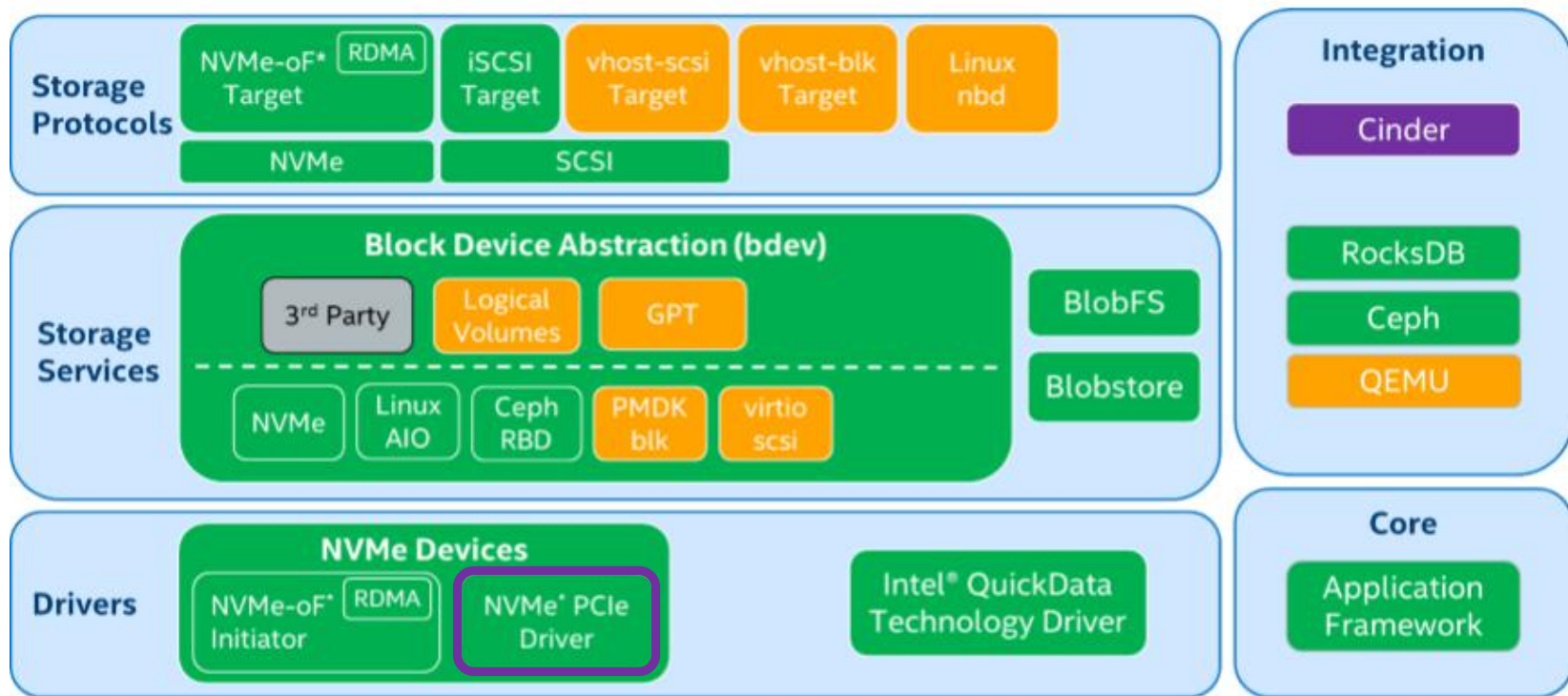


- python + nvme: pynvme <https://github.com/pynvme/pynvme>



**Design**

# SPDK



# Architecture

library (ongoing):

ZNS

TCG

psd

API:

controller

namespace

qpair

pcie

buffer

SPDK: nvme driver

cmdlog

checksum

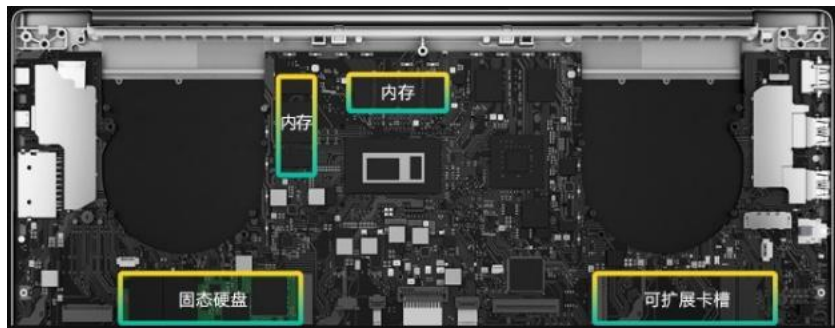
ioworker



# Open Ecosystem



# Hardware



# High Performance

fio (unit: K IOPS)

Q count	1	2	4
test 1	200	332	353
test 2	211	319	340
test 3	211	248	354

pynvme (unit: K IOPS)

Q count	1	2	4
test 1	358	359	359
test 2	358	358	359
test 3	358	356	359



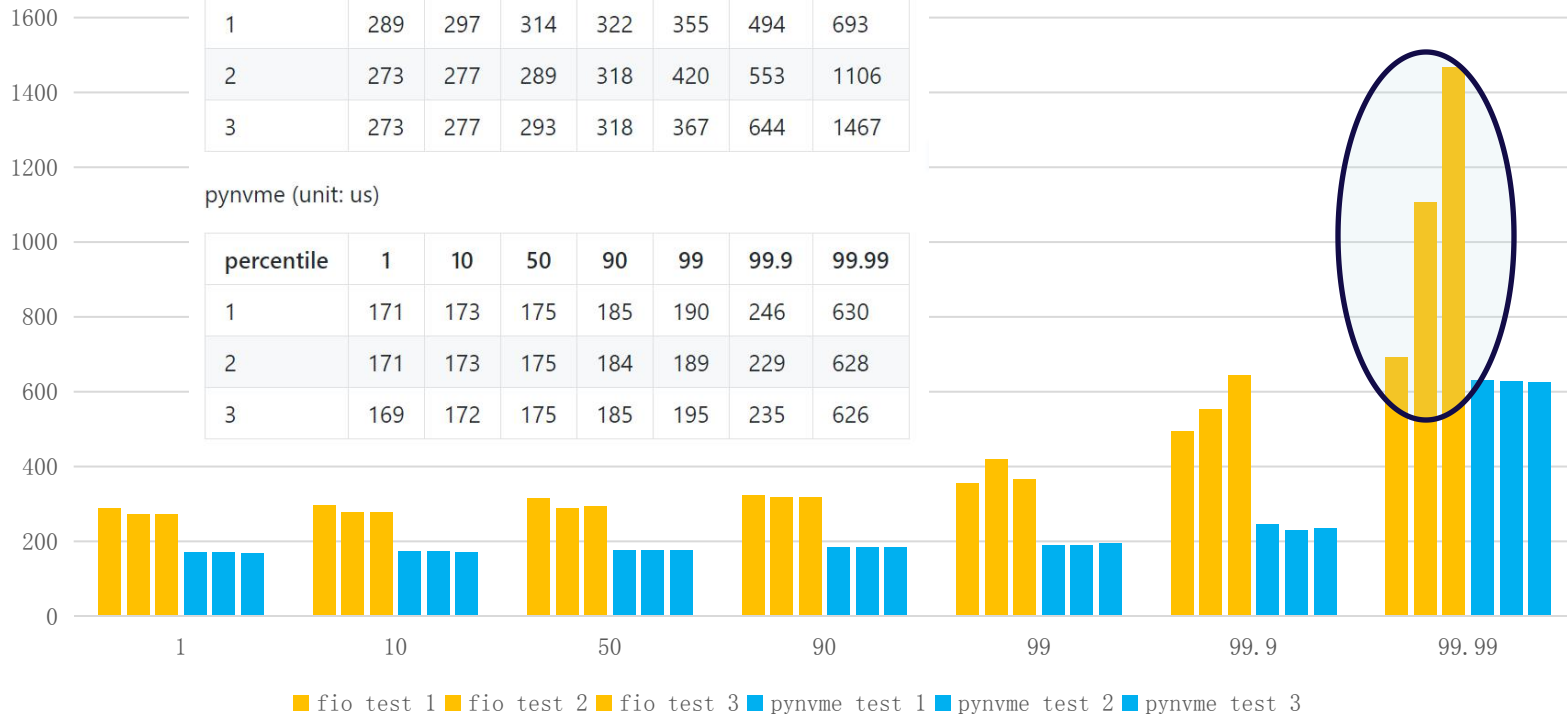
# Low Latency

fio (unit: us)

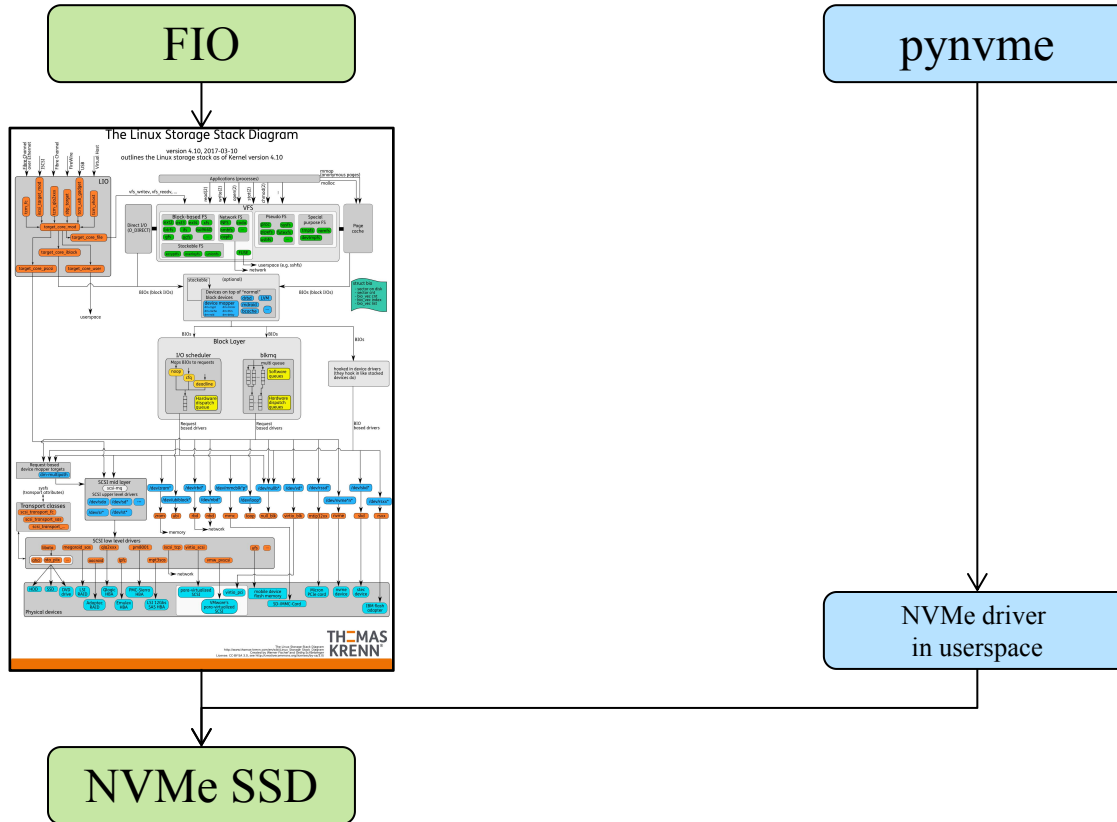
percentile	1	10	50	90	99	99.9	99.99
1	289	297	314	322	355	494	693
2	273	277	289	318	420	553	1106
3	273	277	293	318	367	644	1467

pynvme (unit: us)

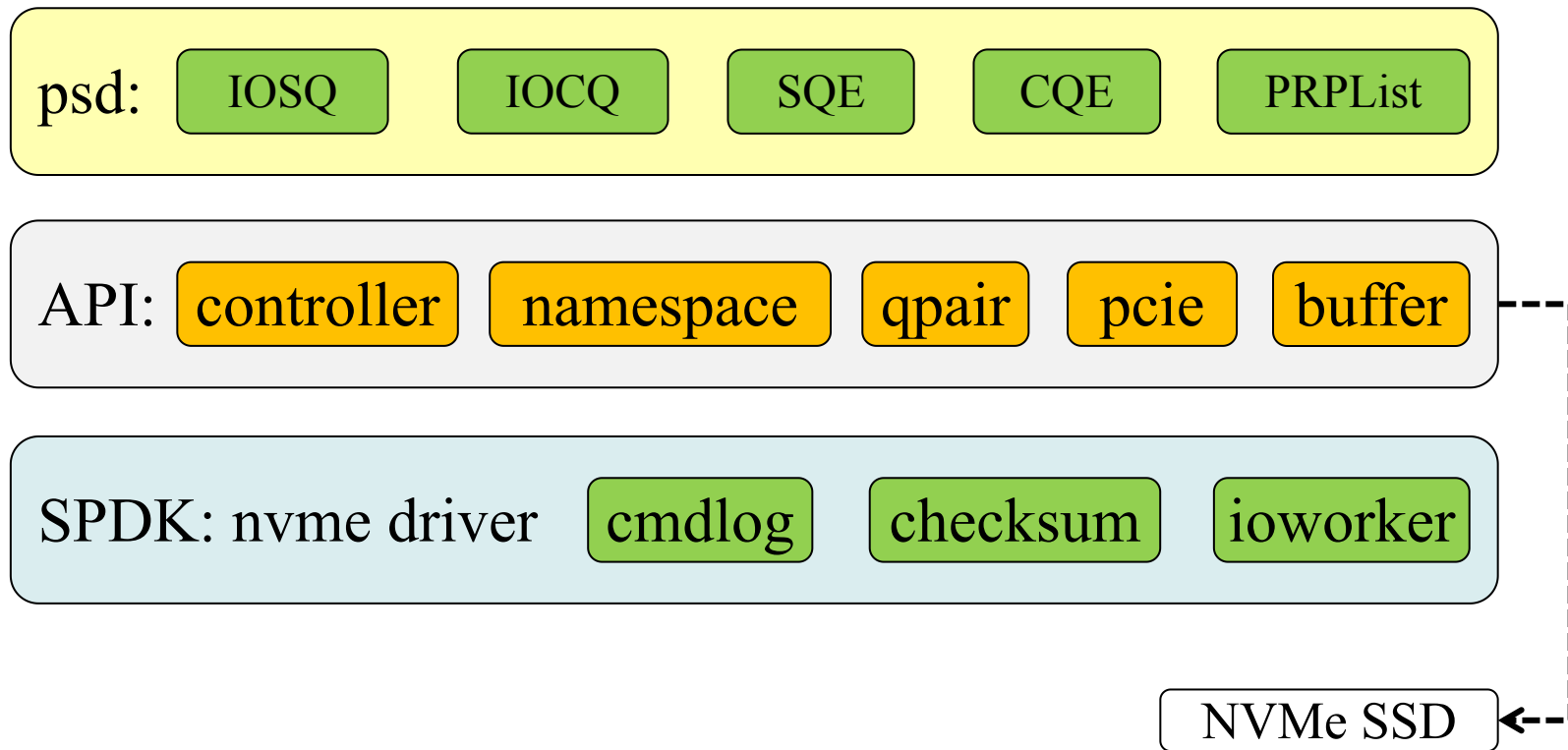
percentile	1	10	50	90	99	99.9	99.99
1	171	173	175	185	190	246	630
2	171	173	175	184	189	229	628
3	169	172	175	185	195	235	626



# Design



# psd: Python Space Driver



# Features

- access PCI configuration space
- access NVMe registers in BAR space
- send any NVMe admin/IO commands
- support callback functions for NVMe commands
- support MSI/MSIx interrupts
- transparent checksum verification on every LBA
- generates IO workload of high performance and low latency
- support multiple namespaces
- support multiple tests on different controllers
- integrate with the test framework pytest
- integrate with VSCode to display cmdlog in GUI
- support NVMe over TCP targets
  
- doc: <https://pynvme.readthedocs.io/>



# Examples



# Example: hello world

```
1 import time
2 import pytest
3 import logging
4
5 import nvme as d
6
7
8 # intuitive, spec, qpair, vscode, debug, cmdlog, assert
9 def test_hello_world(nvme0, nvme0n1, qpair):
10     # prepare data buffer and IO queue
11     read_buf = d.Buffer(512)
12     write_buf = d.Buffer(512)
13     write_buf[10:21] = b'hello world'
14
15     # send write and read command
16     def write_cb(cdw0, status1): # command callback function
17         nvme0n1.read(qpair, read_buf, 0, 1)
18     nvme0n1.write(qpair, write_buf, 0, 1, cb=write_cb)
19
20     # wait commands complete and verify data
21     assert read_buf[10:21] != b'hello world'
22     qpair.waitdone(2)
23     assert read_buf[10:21] == b'hello world'
```

- pytest fixtures
  - callback functions for commands
  - sync point: waitdone()
- 
- run test:
    - > make setup
    - > make test TESTS=scripts/test\_examples.py::test\_hello\_world
    - > make test TESTS=scripts/test\_examples.py::test\_hello\_world pciaddr=0000:3d:00.0
    - > make test TESTS=scripts/test\_examples.py::test\_hello\_world pciaddr=172.168.5.44

# Example: psd

```
698 def test_write_before_power_cycle(nvme0, subsystem):
699     cq = IOCQ(nvme0, 1, 128, PRP(2*1024))
700     sq = IOSQ(nvme0, 1, 128, PRP(8*1024), cqid=1)
701
702     #burst write
703     for i in range(127):
704         cmd = SQE(1, 1)
705         buf = PRP(512, ptype=32, pvalue=i)
706         cmd.prp1 = buf
707         cmd[10] = i
708         sq[i] = cmd
709
710     # write 127 512byte at one shot
711     sq.tail = 127
```

- create IOCQ/IOSQ respectively with specified sqid and cqid
- fill commands in SQ entries
- manually trigger doorbell

# Example: ioworker

```
754 def test_ioworker_single(nvme0n1):
755     nvme0n1.ioworker(io_size=2, time=2).start().close()
756
757
758 def test_ioworker_multiple(nvme0n1, qpair):
759     with nvme0n1.ioworker(io_size=8, lba_random=False,
760                           read_percentage=100, time=2), \
761         nvme0n1.ioworker(io_size=64, lba_random=True,
762                           read_percentage=50, time=2):
763         pass
764
```

- fio-like IO generator
  - Python API
  - better performance
- sending IO in separated processes
- define IO pattern in parameters
- support multiple ioworkers
- support workload from simple to complex

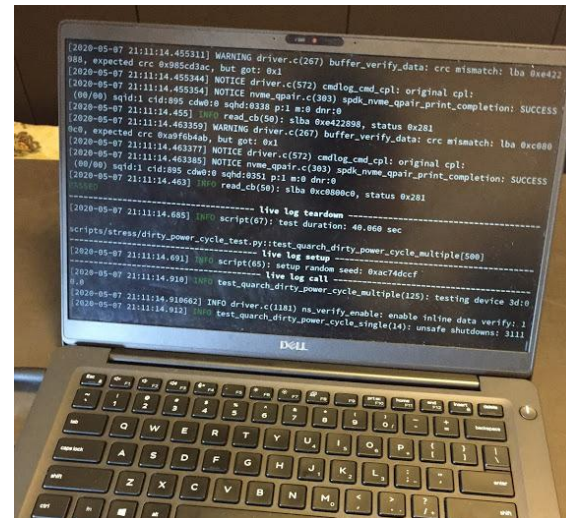
# Example: dirty power cycle

```

33 # 128K random write
34 cmdlog_list = [None]*1000
35 with nvme0n1.ioworker(io_size=256,
36                        lba_random=True,
37                        read_percentage=30,
38                        region_end=region_end,
39                        time=30,
40                        qdepth=qdepth,
41                        output_cmdlog_list=cmdlog_list):
42     # sudden power loss before the ioworker end
43     time.sleep(10)
44     subsystem.poweroff()
45
46 # power on and reset controller
47 time.sleep(5)
48 subsystem.poweron()
49 nvme0.reset()

```

- ioworker sending IO in a separated process
- cut power in main process when ioworker is working



# Example: customized nvme init

```
755 def test_weighted_round_robin_arbitration(pcie):
756     def nvme_init_wrr(nvme0):
757         # disable cc.en
758         nvme0[0x14] = 0
759         while not (nvme0[0x1c]&0x1) == 0: pass
760         nvme0.init_adminq()
761
762         # enable WRR and cc.en
763         nvme0[0x14] = 0x00460801
764         while not (nvme0[0x1c]&0x1) == 1: pass
765
766         nvme0.identify(d.Buffer(4096)).waitdone()
767         nvme0.init_ns()
768         nvme0.setfeatures(0x7, cdw11=0xffffefffe).waitdone()
769         cdw0 = nvme0.getfeatures(0x7).waitdone()
770         nvme0.init_queues(cdw0)
771
772     nvme0 = d.Controller(pcie, nvme_init_func=nvme_init_wrr)
773     if (nvme0.cap>>17) & 0x1 == 0:
774         pytest.skip("WRR is not supported")
775
776     # set arbitration weight
777     assert nvme0[0x14] == 0x00460801
778     nvme0.setfeatures(1, cdw11=0x07030103).waitdone()
779     cdw0 = nvme0.getfeatures(1).waitdone()
780     assert cdw0 == 0x07030103
781
```

- pynvme has a default nvme initializaiton process
- users can define different nvme initialization process in test scripts

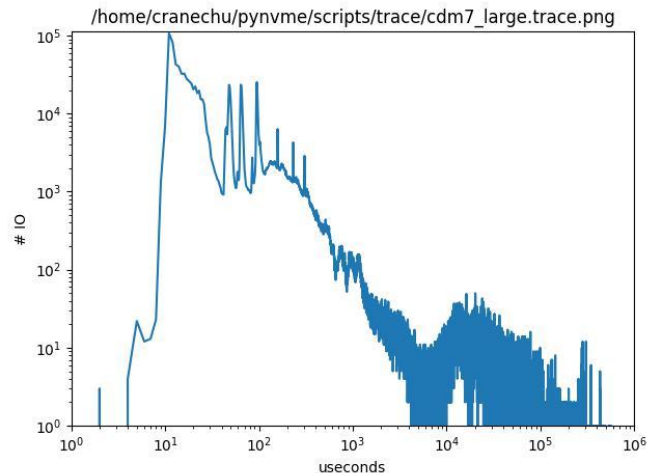
# Example: latency of JEDEC workload

```

794 def test_ioworker_jedec_enterprise_workload_512(nvme0n1):
795     distribution = [1000]*5 + [200]*15 + [25]*80
796     iosz_distribution = {1: 4,
797                          2: 1,
798                          3: 1,
799                          4: 1,
800                          5: 1,
801                          6: 1,
802                          7: 1,
803                          8: 67,
804                          16: 10,
805                          32: 7,
806                          64: 3,
807                          128: 3}
808
809     output_percentile_latency = dict.fromkeys([99, 99.99, 99.9999])
810     nvme0n1.ioworker(io_size=iosz_distribution,
811                      lba_random=True,
812                      qdepth=128,
813                      distribution = distribution,
814                      read_percentage=0,
815                      ptype=0xbeef, pvalue=100,
816                      time=30,
817                      output_percentile_latency=\
818                          output_percentile_latency).start().close()
819     logging.info(output_percentile_latency)
820

```

- ioworker generates IO according to JEDEC enterprise workload
- scripts get latency at different percentile (99%, 99.99%, 99.9999%)





# Example: check on error code

```
42 def test_read_large_lba(nvme0, nvme0n1, buf, qpair):
43     ncap = nvme0n1.id_data(15, 8)
44
45     nvme0n1.read(qpair, buf, ncap-1).waitdone()
46     with pytest.warns(UserWarning, match="ERROR status: 00/80"):
47         nvme0n1.read(qpair, buf, ncap).waitdone()
48     with pytest.warns(UserWarning, match="ERROR status: 00/80"):
49         nvme0n1.read(qpair, buf, ncap+1).waitdone()
50     with pytest.warns(UserWarning, match="ERROR status: 00/80"):
51         nvme0n1.read(qpair, buf, ncap-1, 2).waitdone()
52     with pytest.warns(UserWarning, match="ERROR status: 00/80"):
53         nvme0n1.read(qpair, buf, 0xffffffff00000000).waitdone()
54
```

- pynvme through a warning when command completes with an error
- scripts can use pytest to capture and check the warning

# Example: sanitize an aer

```
93 def test_write_in_sanitize_operations(nvme0, nvme0n1, buf, qpair):
94     if nvme0.id_data(331, 328) == 0: #L9
95         pytest.skip("sanitize operation is not supported") #L10
96
97     logging.info("supported sanitize operation: %d" % nvme0.id_data(331, 328))
98     nvme0.sanitize().waitdone() #L13
99
100     with pytest.warns(UserWarning, match="ERROR status: 00/1d"):
101         nvme0n1.write(qpair, buf, 0).waitdone()
102
103     # check sanitize status in log page
104     with pytest.warns(UserWarning, match="AER notification is triggered"):
105         nvme0.getlogpage(0x81, buf, 20).waitdone() #L17
106         while buf.data(3, 2) & 0x7 != 1: #L18
107             time.sleep(1)
108             nvme0.getlogpage(0x81, buf, 20).waitdone() #L20
109             progress = buf.data(1, 0)*100//0xffff
110             logging.info("%d%%" % progress)
111
112     # check sanitize status
113     nvme0.getlogpage(0x81, buf, 20).waitdone()
114     assert buf.data(3, 2) & 0x7 == 1
115
```

- check if sanitize is supported
- start sanitize operation
- send a write command, and check if it is aborted due to the sanitize operation in progress
- monitor the sanitize progress till it is completed
- check the final sanitize status in logpage



# Example: multiple processes

```
774 def test_ioworker_with_temperature(nvme0, nvme0n1, buf):
775     with nvme0n1.ioworker(io_size=256,
776                           time=30,
777                           op_percentage={0:10, # flush
778                                          2:60, # read
779                                          9:30}), \
780         nvme0n1.ioworker(io_size=8,
781                           time=30,
782                           op_percentage={0:10, # flush
783                                          9:10, # trim
784                                          1:80}):# write
785         for i in range(40):
786             time.sleep(1)
787             nvme0.getlogpage(0x02, buf, 512).waitdone()
788             ktemp = buf.data(2, 1)
789             from pytemperature import k2c
790             logging.info("temperature: %.2f degreeC" %
791                          k2c(ktemp))
792
```

- process 1: ioworker with flush, read, trim commands
- process 2: ioworker with flush, write, trim commands
- main process: print temperature value in SMART data for every second

# Live Demo in VSCode

The screenshot displays the VS Code interface during a live demo. The left sidebar shows the file explorer with a project named 'TEST'. The code editor in the center shows a file named 'test\_examples.py' with a line of code: 'CAZ-82256-Q11 NVMe LITEON 256GB'. The terminal on the right shows the output of the test, including timestamps and temperatures. At the bottom, a performance gauge shows two metrics: 'pynvme' (219.26 MB/s) and 'IOPS' (003.25 K/s).

**File Explorer:**

- TEST
- ▼ PYNVME QPAIRS
- 0000:3d:00.0 Q00:
- 0000:3d:00.0 Q01: >>>>
- 0000:3d:00.0 Q02: >>>>
- ▼ PYTHON
- test\_create\_qpairs
- test\_namespace\_multi...
- test\_dsm\_trim
- test\_ioworker\_perform...
- test\_ioworker\_jedec\_e...
- test\_power\_on\_off
- test\_init\_nvme\_custo...
- test\_ioworker\_op\_dict...
- test\_ioworker\_io\_sequ...
- test\_aer\_with\_multiple...
- test\_verify\_partial\_na...
- test\_jsonrpc\_list\_qpairs
- test\_powercycle\_with...
- test\_reset\_time
- test\_ioworker\_with\_te...
- test\_different\_io\_size...
- test\_getlogpage\_nsid
- test\_ioworker\_fixed\_iops
- test\_ioworker\_iops\_m...
- test\_power\_state\_tran...
- test\_utilities.py

**Code Editor:**

```
1 2020-08-25 17:11:15.660273 [cmd001: Dataset Mar
2 0x00100009, 0x00000001, 0x00000000, 0x00000000
3 0000, 0x3ebff000, 0x00000000, 0x00000000
4 0x00000000, 0x00000000, 0x00000000, 0x00000004
5 0x00000000, 0x00000000, 0x00000000, 0x00000000
6 not completed
7 ...
8
9 2020-08-25 17:11:15.660268 [cmd002: Read]
10 0x00030002, 0x00000001, 0x00000000, 0x00000000
11 0x00000000, 0x00000000, 0x3ef3f000, 0x00000000
12 0x3edc3038, 0x00000000, 0x0a9a793c, 0x00000000
13 0x000000ff, 0x00000000, 0x00000000, 0x00000000
14 not completed
15 ...
16
17 2020-08-25 17:11:15.660261 [cmd003: Dataset Mar
18 0x00160009, 0x00000001, 0x00000000, 0x00000000
19 0x00000000, 0x00000000, 0x3ea7f000, 0x00000000
20 0x00000000, 0x00000000, 0x00000000, 0x00000004
21 0x00000000, 0x00000000, 0x00000000, 0x00000000
22 not completed
23 ...
```

**Terminal:**

```
[2020-08-25 17:11:09.073] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:09.715] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:10.754] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:11.796] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:12.835] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:13.883] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:14.920] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:15.964] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:17.084] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:18.123] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:19.361] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:20.400] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:22.136] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:23.174] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:24.206] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:25.243] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:26.298] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:27.348] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
```

**Performance Gauge:**

- pynvme: 219.26 MB/s
- IOPS: 003.25 K/s

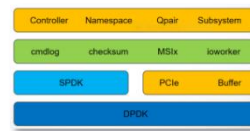
# pynvme builds your own tests.



Ecosystem



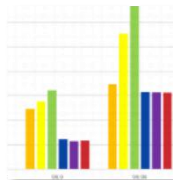
Scripts



Extendability



Hardware

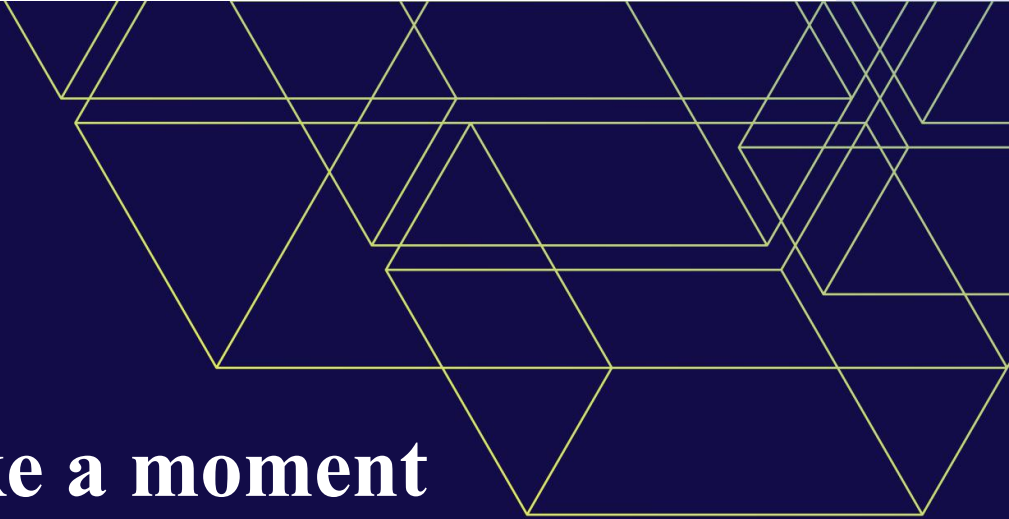


Performance



Service





**Please take a moment  
to rate this session.**

**Your feedback matters to us.**

**Thank you!!!**