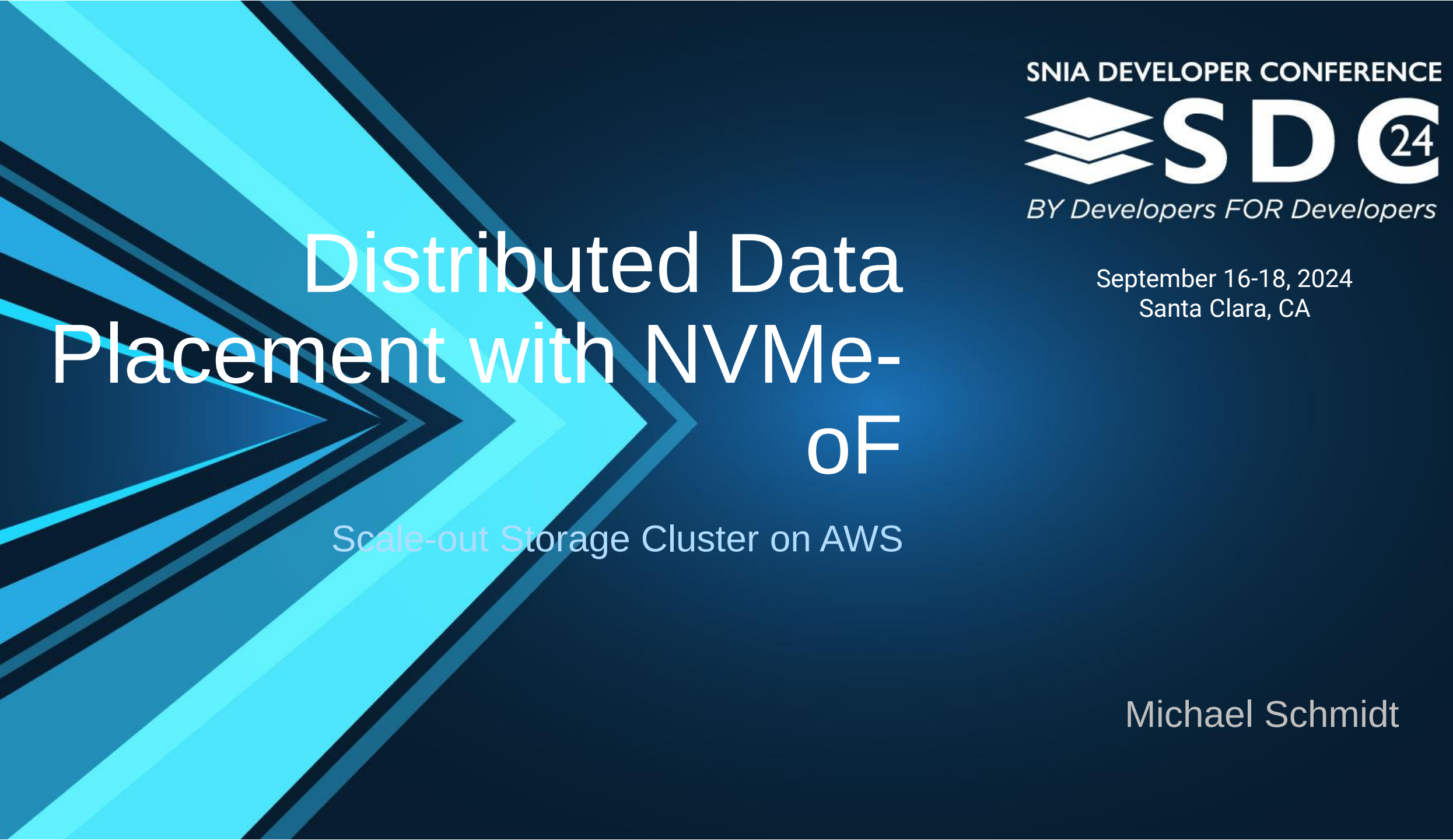


Distributed Data Placement with NVMe-oF

Scale-out Storage Cluster on AWS

September 16-18, 2024
Santa Clara, CA

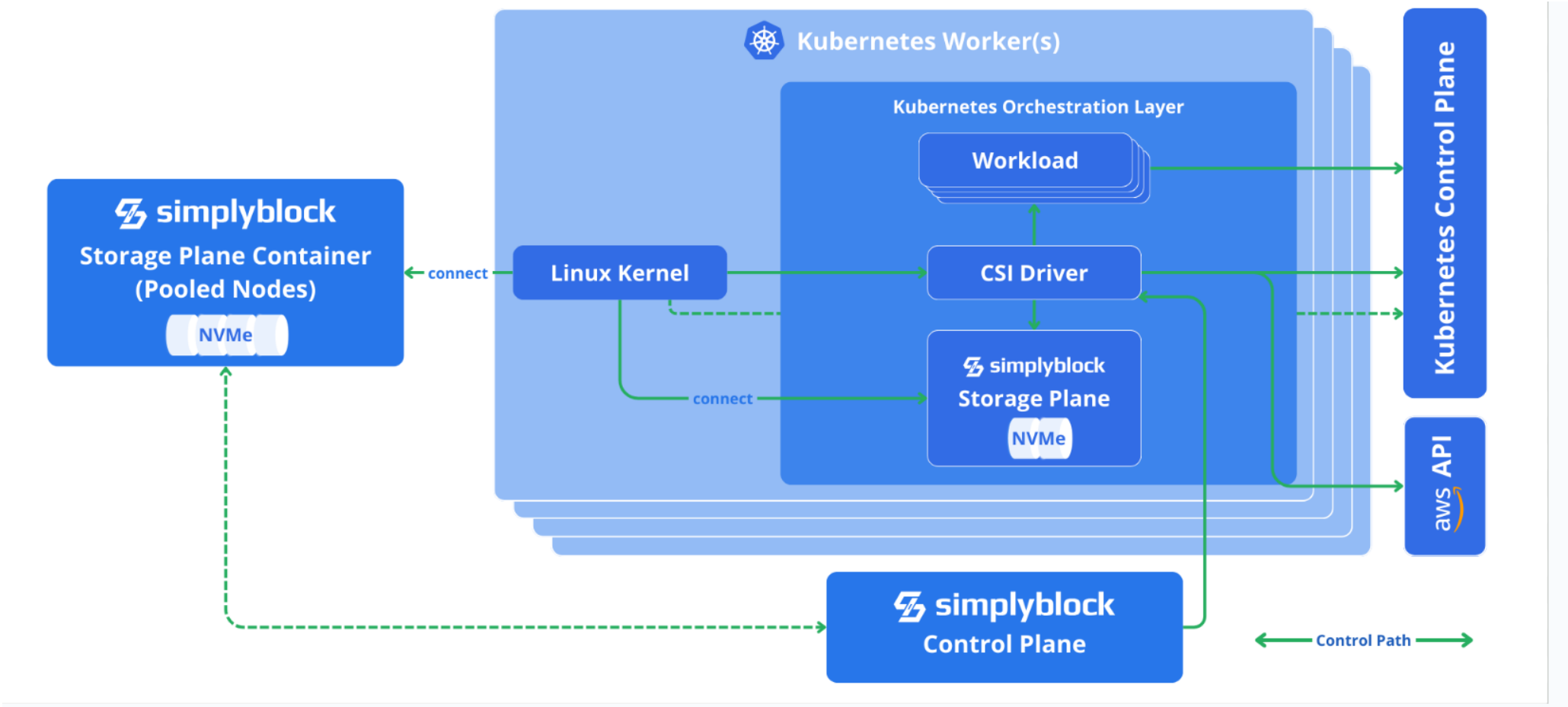
Michael Schmidt

Introduction

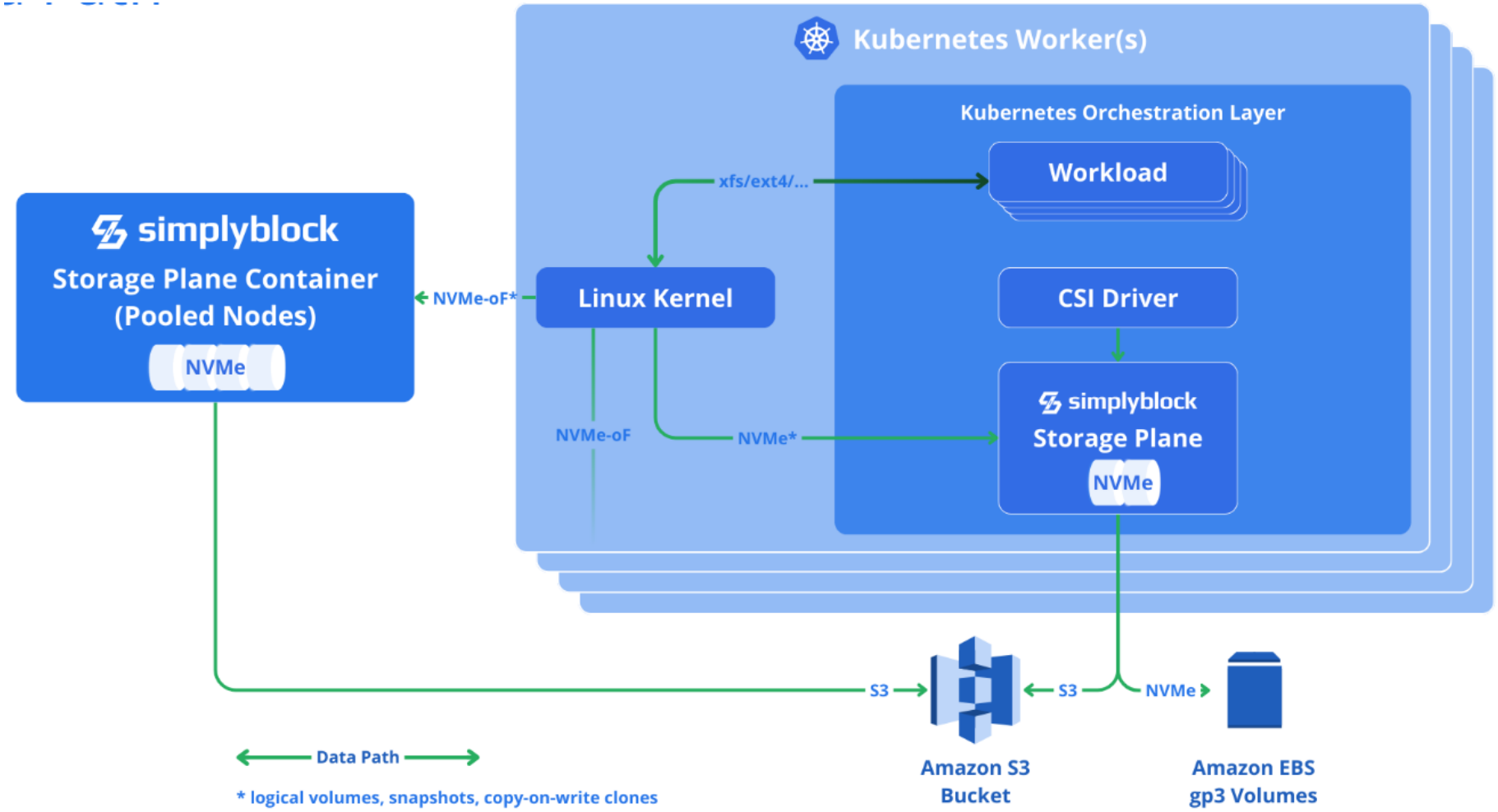
Product

- A *scalable* storage cluster technology and optimization layer for clouds
- Virt. of multiple cloud storage services behind **NVMe-oF/tcp** and a **CSI driver**
- Storage for any host or container with NVMe-oF/tcp initiators
- Tight integration into Kubernetes ecosystem
- Fully containerized solution
- Storage plane is entirely built on top of spdk
- Control plane provides CLI, API, Deployment, Monitoring, Alerting, Log Management, Repair and IO Statistics Services
- Designed to be cloud-agnostic, but the current releases focus on AWS

Control Path

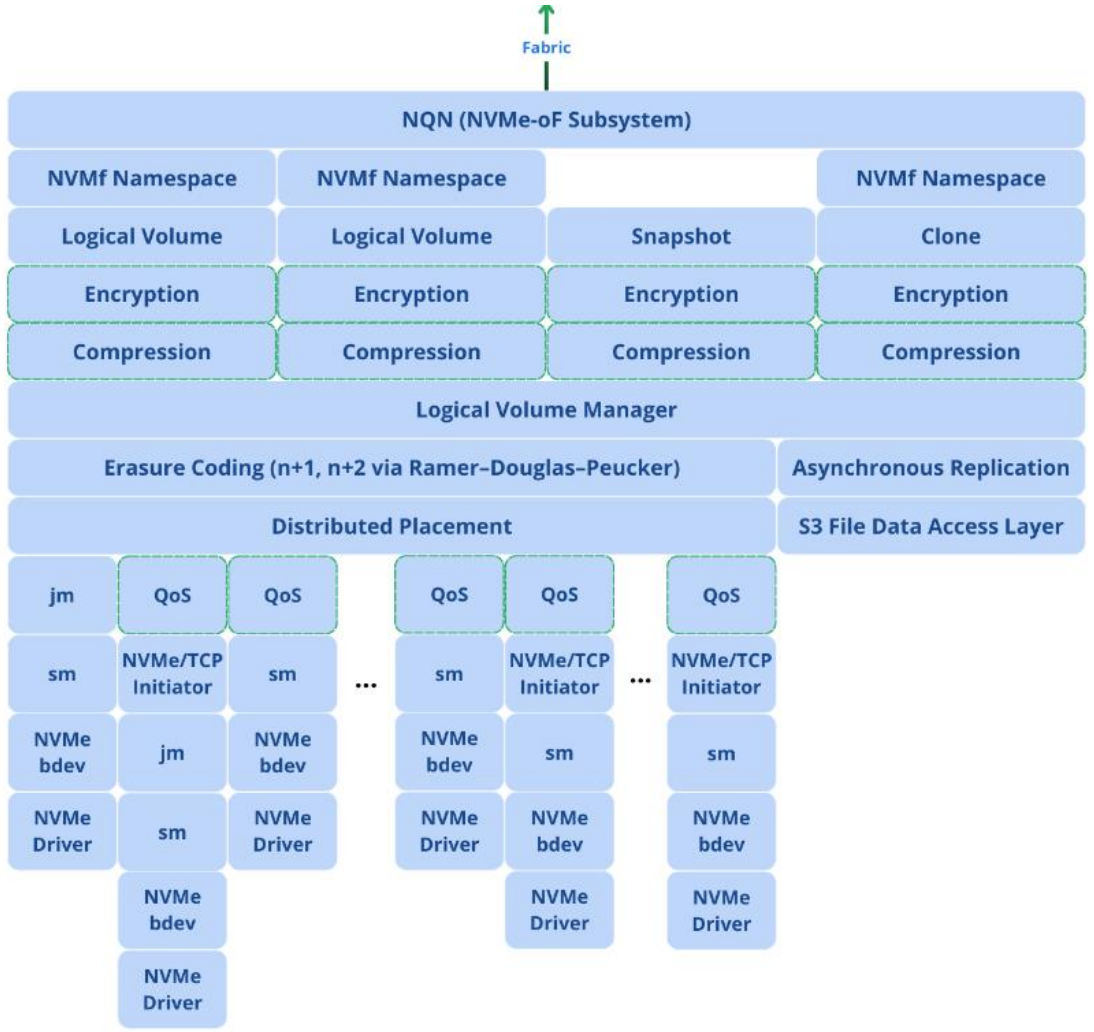


Data Path

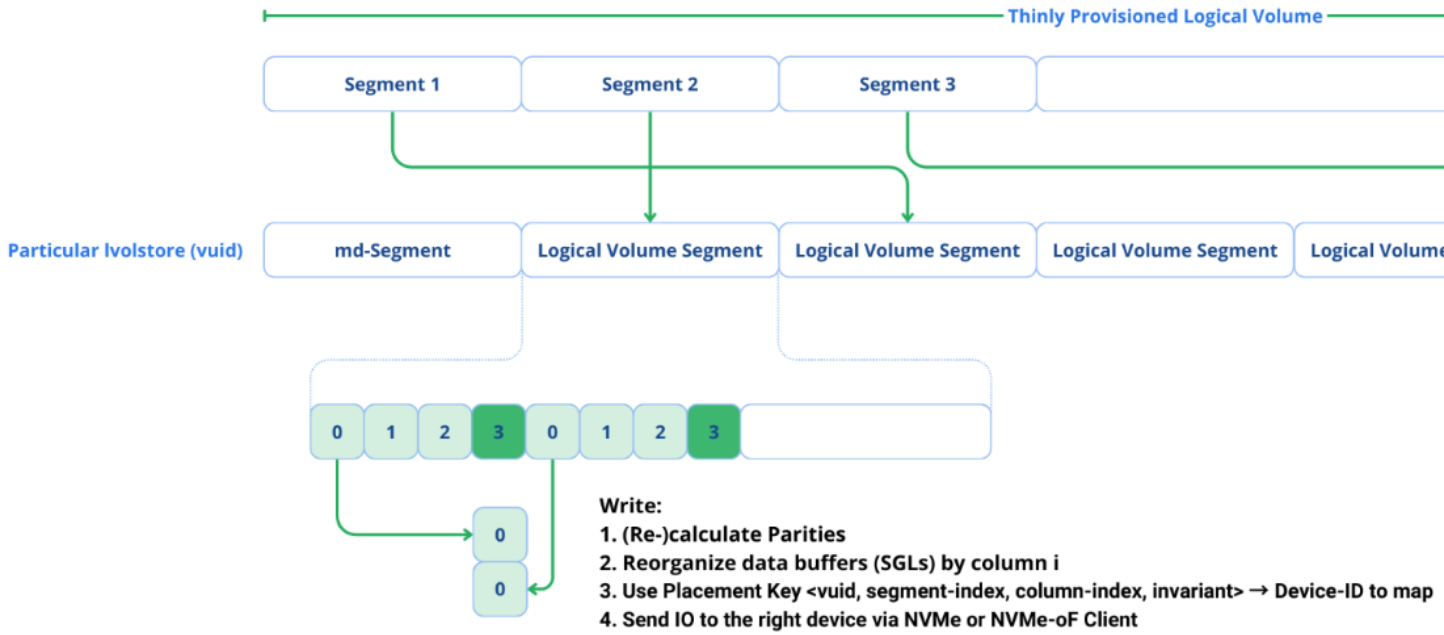


Concepts

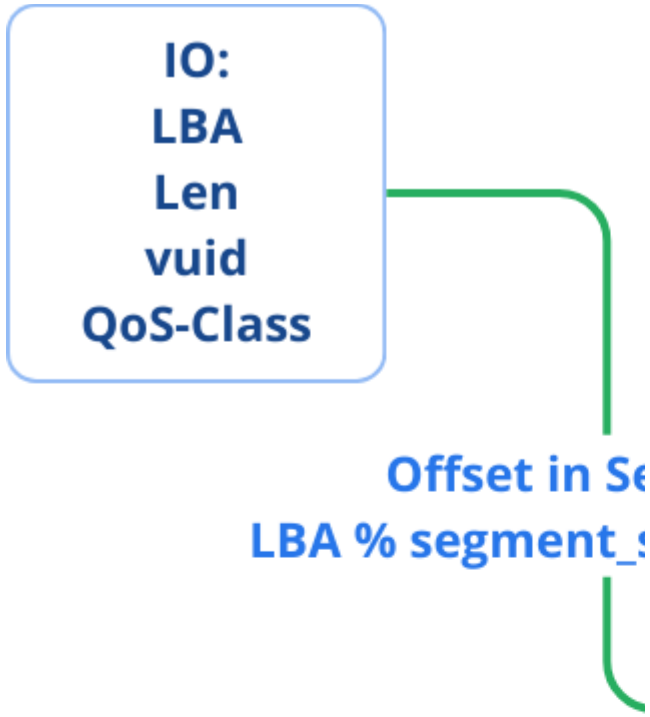
data services container: data path



data services: Distributed placement



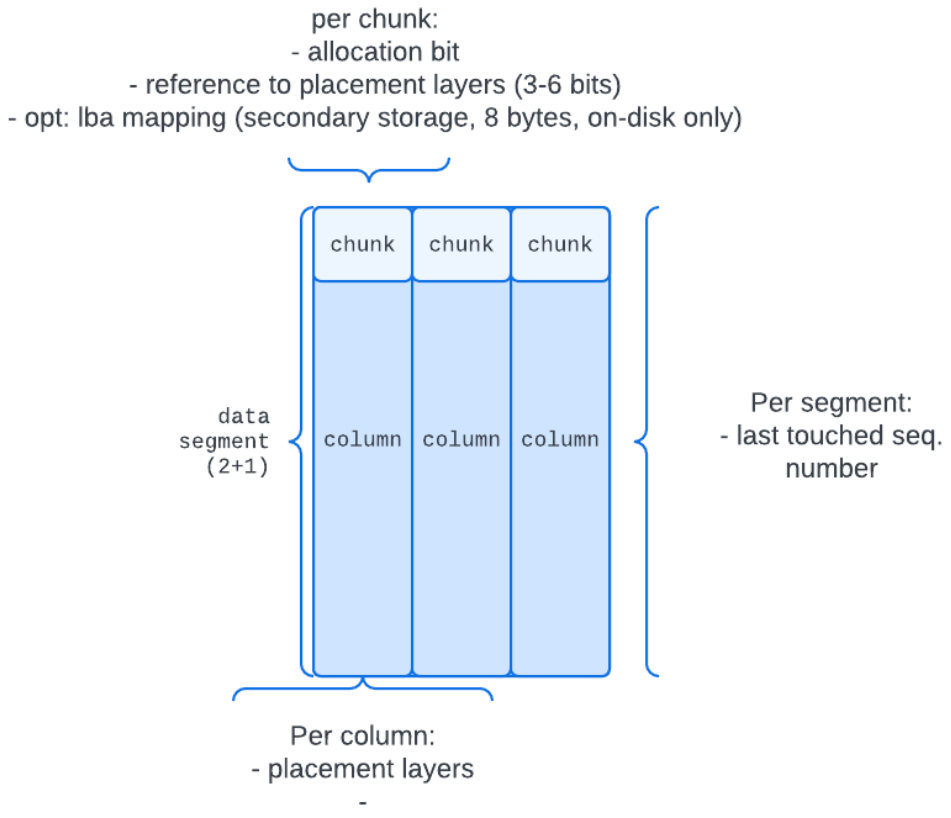
data services container: device-level placement



- Virtual block device, which manages a particular local NVMe namespace
- It organizes the linear address space into segments and manages pools of segments assigned to particular logical devices (vuids)
- It works single threaded, but actual IO can be served by multiple threads used to serve multiple queue pairs in parallel

meta-data journal

- Stores information per segment (compacted - in mem and on disk) and journals each IO request (on disk)
- Regular compaction to update per-segment data from journal
- block-level allocation map
- Placement map
- Mapping of data on secondary storage
- Per segment “last-touched” sequencing



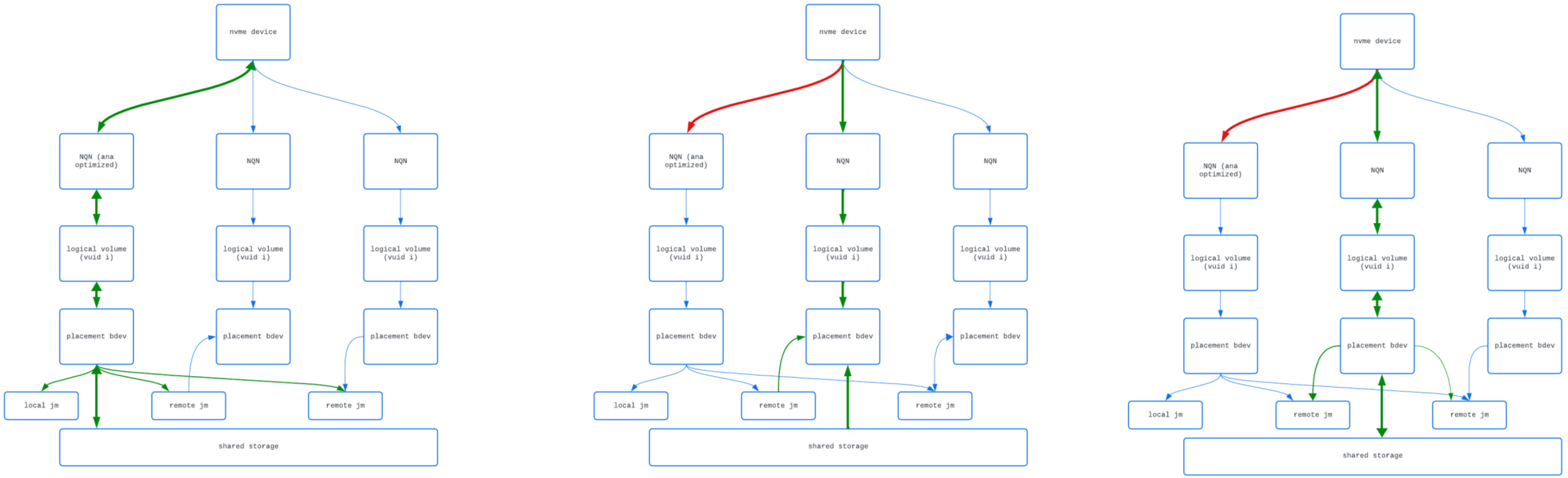
meta data journal: further design decisions

- Replicated across 3 different nodes (6 copies)
- Replicas kept in synch or synchronized after unavailability
- Compaction of meta data: io request-level journal is processed and compacted into on-disk meta-data segments as a background task
- Current write amplification to data is about 25% only

meta data journal: further design decisions

- Replicated across 3 different nodes (6 copies)
- Replicas kept in synch or synchronized after unavailability
- Compaction of meta data: io request-level journal is processed and compacted into on-disk meta-data segments as a background task
- Current write amplification to data is about 25% only

HA volumes



- A logical volume is instantiated on 3 nodes and exposed to the fabric via an identical NVMe Qualified Name (NQN)
- ANA Optimized Namespace on primary
- NVMe/TCP initiator multipathing function chooses the first path per default.

- In case the first path is down, IO is sent to a secondary node. The placement bdev initiates the fail-over (leadership acquisition).
- For this purpose, it needs to update the logical volume in-memory data structures from the underlying storage and journal replica
- Takeover takes a few milliseconds, after which the new leader will serve the incoming IO.

Asynchronous replication to S3

- Amazon S3 is about 5 times cheaper than aws NVMe block storage (EBS)
- It is replicated across Availability Zones (AZs)
- It can be write- and deletion-protected for a defined retention period.
- By using parallelism, it is possible to write and read sequentially at relatively high throughput rates, but access latency is very high and IOPS are very expensive.
- Simplyblock replicates storage writes asynchronously to S3 in the form of a WAL. It is written entirely sequentially in batches of fixed-size S3 objects (WAL segments).
- A compaction mechanism is used to regularly remove stale blocks and re-organize compacted data into new objects with the help of the md journal.
- This allows implementation of Disaster Recovery and recovery from ransomware attacks with near-zero RPO. It also serves as a basis for our storage tiering.

QoS

- When the total IOPS and throughput load on a cluster increases, so does the queue depth and the average completion time of IO on each of the devices.
- The load is equally distributed across all devices and the observed increase in access latency is similar.
- To support logical volumes with different performance requirements on the same cluster, a QoS component was introduced.
- Simplyblock supports 4 priority classes. A class can be chosen on the logical volume level. In addition, internally, all md writes take place at the highest priority class.
- A virtual QoS device on top of each physical NVMe device or NVMe partition takes care that the queue depth to the physical device is limited. While highest priority IO is always allowed, IO at a lower priority is held back (queued separately) until the amount of IO in flight (qd) decreases sufficiently.

Cluster (Re-)Balancing

- Auto-Rebalancing triggered by 3 events: device (node) restarted after temp. outage, device (node) persistent failure, device (node) added to cluster
- All re-distributions of data are serialized on a per-device and per logical volume basis, but are fully parallelized between logical volumes.
- Each logical volume in the cluster is responsible for “updating” the data placement (re-distribution) in case of such an event.
- All affected <segment, column> pairs (or partial data within) are first identified
- Then data is re-distributed according to the most current version of the cluster map and node / device statuses.
- The redistribution takes place within the IO functions of the placement bdevs
- Each column, including both data and parity, is handled individually and parity re-calculation is generally not required (exception: re-distribution of data in case of failed device requires rebuild of some of the data).

Performance & Scaleability

Baseline Performance - Measurements

- *Software performance* based on 3 indicators: **net IO access latency per qd** (without disk and network), **total vCPU consumption per K-IOPS effective**, **Read/Write-Amplification**
- *System performance* based on defined storage instance types with *dedicated* nw bandwidth: **total io access latency per IO load**, **total IOPS**, **total throughput**
- AWS: On storage node instances with dedicated network bw, there are no noisy-neighbour effects and performance can be measured in a repeatable and reliable manner to assign SLAs for particular IO patterns
- SLAs are harder to achieve for arbitrary, variable patterns, because nitro disk performance is very sensitive to IO patterns.

Basic Findings

- Software
  - read amplification: 1
  - write amplification (worst-case, 4KiB random writes): writes: k + 0.25, reads: 0 (n=1), 1 (n=2), 2 (n>2)
  - approx. cpu consumption / K-IOPS (depending on io pattern): 0.05 cpu cores (2.1 Ghz arm)
  - io access latency (spdk+simplyblock path): 40 us

- System
  - disaggregated storage cluster with 3 nodes of type i3en.2xlarge, single AZ
  - approx. 1.200.000 IOPS on read (4 KiB random)
  - approx. 600.000 IOPS on write (4 KiB random)
  - max r/w bandwidth 8 GB/s

Spdk: Scaling on AWS (single node)

- AWS offers a broad range of Intel and Graviton instances with local instance storage up to bare-metal instances with 96 vCPUs and 60TB of local instance storage.
- Spdk is based on several principles, which make it highly scalable: user space polling mode driver, zero-copy architecture, lockless, asynchronous messaging btw. threads
- Tests show that the overall system performance of the NVMe/TCP target on AWS does *not* scale entirely linearly with the amount of vCPUs and disks added, while shared resources (L3, RAM, PCI) are not fully loaded
- preferable to work with more and smaller nodes

Simplyblock: Scaling on AWS (single node)

- Simplyblock threads (in addition to spdk) : (1) EC + Distributed Data Placement - one to many (2) single device dynamic data placement (main) - one per physical disk (3) single device dynamic data placement (io workers) - up to three per physical disk (3) journal manager - usually one per node (4) tiering manager - one per node (5) replication - one per node
- All threads principally avoid cross-thread dependencies and each thread can work nearly entirely independently (theoretically)
- The highest load is on (1). While there is no pressure on shared resources and no dependencies btw. threads in this group, we still observe an optimal amount of threads of that type per node - after that threshold, the cpu consumption / K-IOPS starts to rise significantly
- The reason for this behaviour is not fully understood and requires further research
- Therefore smaller nodes are preferable

Simplyblock: Scaling on AWS (cluster / multiple nodes)

- Each node in the cluster delivers about 75% of the performance of a single node deployment
- This decrease is caused by networking / coordination btw. remote nvme-oF and local IO requests to the same disks
- Vertical scalbility of cluster by adding node is extremely good - nearly linear
- Maximum number of nvme-oF initiators per node - about 1.000

spdk on aws: findings

qp saturation

- Spdk is built on the assumption that a physical NVMe device can be saturated (IOPS, throughput) with a single qp.
- When using the spdk NVMe-oF target, a single qp is used by the kernel initiator per client thread. This qp maps to an spdk target qp, and then, down in the block device stack, to a single qp on the target NVMe device.
- But AWS local instance storage (Nitro NVMe) requires 2-3 queue pairs to reach full saturation.
- For this reason, it is not simply possible to saturate a NVMe device with a single-threaded client IO using spdk (on AWS).

of qps

- NVMe base specification defines that devices may support up to 65K queue pairs.
- As a general rule, spdk requires one qp per core in the core mask (however, this depends on the block device configuration and cpu masks used).
- While AWS provides sufficient queue pairs for local instance storage (depending on the type and size of disk - typically around 64), this is not the case for EBS volumes. For example, a single gp3 volume only has 2 queue pairs.
- This can cause issues with spdk terminating due to insufficient queue pairs could be allocated.

Write Atomicity

- The NVMe identify controller / namespace reports that AWS Nitro NVMe devices support only 512 bytes blocks and single block write atomicity.
- However, the write and read performance of those NVMe devices is optimized / throttled to around 4K block size (max. IOPS on 4K, same as on 512 bytes).
- Also, the spdk logical volume layer (logical volume virtualization, snapshots, copy-on-write clones), require 4K write atomicity.
- Good news: AWS supports a “torn write protection” feature for both instance storage, as well as block storage. It provides 4K (in some cases up to 16K) write atomicity.

Multipathing

- NVMe base specification defines multipathing options with support for both active/passive (via asynchronous Namespace Access aka ANA) and active/active (round-robin).
- NVMe-oF multipathing is available in Linux kernel since version 5. Not to confuse with the generic mpio device!
- On Amazon Linux 2 (kernel: 5.10), the NVMe-oF multipathing feature is not enabled and the pending issue is not yet fixed. It works on Amazon Linux 2023 though.

CPU Architecture

- Spdk runs on both Graviton and x86 instances.
- However, a vCPU on AWS is usually a thread on x86 and a core on graviton.
- E.g. on a system with 16 vCPUs, the first 8 and the second 8 correspond to thread pairs per core, e.g. 0x1010 defines the first core.
- This can cause troubles, if an spdk process or container runs in parallel with other work loads.
- It is important to set the spdk core mask accordingly to avoid noisy neighbours.
- For example, to allocate 4 vCPUs on an instance with 16 vCPUs, use (binary)
- 01100000 01100000 (always leave vCPU 0 for system).

Thank you

michael@simplyblock.io

