NUTANIX

# Software-Defined Storage Explained

## Nutanix Strategic Technology Paper

# Software-defined Storage Explained

## Control Plane, Data Plane, Decoupled Planes

In the world of networking, SDN initially meant the ability to decouple the control plane (QoS, diagnostics, discovery) from the data plane. Nicira quickly realized that such a transition could take a long time because southbound APIs and dumber Taiwan hardware will take years to standardize. They were smart in moving over to VXLAN — or network virtualization — as their new business mantra. VMware is confident it can sell this idea of a software-defined overlay a whole lot better in the coming three years than try to boil the ocean by decoupling control from data. Storage companies took this control-plane-data-plane jargon and coined the idea of Software-defined Storage (SDS). No one, however, has spent time to explain what problem it will solve for customers. Decoupling the control plane from the data plane is no more a means, but an end in itself.

## SDS ≠ Federated Storage

Federated storage as a concept has existed for at least two decades, if not more. Veritas virtualized storage with a file system and a volume manager a generation ago. VMware VMFS did the same thing a decade ago, and was hammered in the coming years for creating the IO Blender problem. SANs were the new enemy of virtualization because accountability began at the federated storage software, but ended at the underlying storage system. Finger pointing ran amuck in federated storage for a good reason. Data Domain diskless gateways died a quiet death because the gateway vendor and the array vendors played the game of hot potatoes while a customer system was down. Acopia (now F5), OnStor (now defunct), and NetApp V-series were fringe products because the control planes of individual systems were never stitched together. For example, you could never take a simple snapshot that spanned two arrays — there were no APIs to momentarily freeze multiple arrays for a consistent snapshot. Federated storage continues to be a problem today. Without a distributed data fabric (i.e., a clustered file system, or an industry-standard API for snapshots, clones, data protection, etc.) large companies are dangerously playing with fire and customers' emotions.

## SDS: True North

SDS is more than marketing jargon. It is a profound concept, and we are only vaguely beginning to understand what the true north for this term really is. Unlike the networking industry, where systems from different vendors have communicated to form working fabrics, storage arrays have been dispersed into silos for disparate workloads. Without truly understanding what control plane commonly means across all array vendors, it is almost inane to apply the SDN concepts to SDS verbatim. And honestly, we are only scratching the surface of SDS — with time and customer experience, our grasp of this paradigm will grow.

As we see it now, here are the seven principles of SDS:

**1. Software-defined Controller:** A storage controller must be provision-able via software orchestration. A new instance of a storage controller can be instantiated on a hypervisor just like one instantiates a virtual machine — on-demand, using APIs, or within a few mouse clicks. This is only possible if the controller does not run directly on bare metal, but rather on top of the hypervisor, which is now the de facto OS of the next-generation datacenter.

**2. Zero Hardware Crutch:** A software-defined controller must not use any proprietary hardware. That means no dependence on special-purpose FPGA, ASIC, NVRAM, battery-backup, UPS, modem etc. Use dynamic HTTP-based tunnels instead of modems. Use inexpensive flash instead of ASIC or NVRAM. Use industry-standard PCIe pass through if you must bypass the hypervisor.

**3. x86-based Convergence:** Storage as a datacenter service must run on the same hardware as the rest of the datacenter services. It can then share CPU, memory, and network with firewalls, WAN accelerators, load balancers, deep packet inspectors, and all other business applications.

**4. Virtual Hardware:** With aforementioned x86-based hardware sharing, storage controllers can be provisioned virtual hardware resources — vCPU, vRAM, virtual ports, vSwitch QoS, etc. — at will. A storage vendor need not go back to "taping out" a new array with larger memory, faster CPUs, or faster networks. If a performance problem requires bumping up "hardware" for storage controllers, one can simply use the hypervisor knobs to configure faster storage. If offline compression needs to kick in at night when load on other services is low, compute resources can be dynamically passed on to the storage tier. And that is powerful!

**5. Factory-defined Nothing:** No data management feature is factory-stitched. For example, it should be SDS heresy to ship dual controller arrays such that every workload gets HA. Perhaps non-persistent virtual desktops or test-and-dev VMs don't require HA. Similarly, it's heresy to ship an array with RAID-6 such that every workload gets erasure encoding. While read-mostly workloads embrace RAID-6, write-intensive workloads abhor them.

**6. Mechanism, Policy, and Late Binding*:** A corollary of factory-defined nothingness is VM-aware Everything. Every data management service — snapshot, cloning, backup, DR, compression, dedup, performance QoS (and debugging), HA, RAID, etc. is defined at a VM-level. Factory ships with mechanism. Deployment worries about policies. Factory never hardcodes policies. And only that brings out the beauty of undifferentiated hardware, and software-based differentiation. On the same hardware, one could have some VMs with RAID-10, others with RAID-6, some with HA, some without HA, some with three copies of data, others with one, some with 15-min RPO, others with 1-hour RPO, and so on. This is the true essence of Cloud Computing — policies are late-bound at deployment, not early-bound in the factory. Policies are late-bound to software (virtual) constructs like VMs, not early-bound to hardware in the factory or to coarse-grained storage entities such

as LUNs or volumes. This one virtue is the awesomeness of SDS; it's preposterous to talk of software-defined anything without decoupling mechanism from policy, and without applying policies to virtual (software-based) constructs. Of course, to invoke mechanisms and to configure policies require next-generation RESTful APIs. The Virtual Hardware aspect of SDS is yet another example of late binding of resources to datacenter services.

*— * The separation of mechanism and policy was prevalent in operating systems and programming languages research. For example, the Mach operating system, a precursor to Windows NT, argued for microkernels by keeping mechanisms within the OS and policies above the OS in user-space where other services run (also refer this).*

**7. Active Systems (Liveness):** Storage up until now has been a passive bit keeper of data, a glorified byte shuttler between the network and the disk. In the past decade, vendors that did anything intelligent in the background, e.g., auto-tiering, were handsomely rewarded by customers. Hardware-based storage is passive. SDS is live and active. It is constantly reflecting on IO and data access patterns to create "system tasks" that move hot data closer to compute and cold data away from compute, that keep sequential workloads away from flash and random workloads journaled on flash, that compress or erasure-encode (RAID-6'ed) cold data offline, that pre-fetch and uncompress hot data in DRAM, that archive older snapshots into a WAN-based cloud, etc. How data lives and on what storage medium is yet another late-binding example. Policies are applied much later in the data lifecycle, not when I/O is passing the bits to the storage controller. SDS "wakes up" a sleepy storage tier, brings software to a world of hardware, and brings life to data. Active systems make data resemble a "plasma", a fabric, a constantly shifting liquid of red, yellow, green, and blue.

## SDS: True North

People confuse companies that shift hardware such as Nutanix as companies that cannot be SDS. We adhere to every SDS principle I mention above, but integrate commodity hardware in order to enable an iPhone-like datacenter experience. An all-software SDS inevitably turns into a support nightmare when emotions run high during customer escalations. Attempting to please everybody all the time by promising to run software on any and every hardware is a fallacy. The converged infrastructure solutions success shows customer preference for a single throat to choke versus the lack of accountability in a multi-vendor setup. The largest enterprise software companies of our times — Oracle, VMware, and SAP — have constantly had the SAN to blame whenever there was system slowness or data corruption. SDS integrated with HCL hardware in the field is laissez faire route to storage, especially at a time when hypervisor device drivers are not an open API-based plug-n-play ecosystem, and when flash reliability is hugely variable. Imagine a SATA drive that is not hot-swappable, or a failed drive that does not glow the red LED, or a SATA controller that silently loses data on power loss.

We ship hardware because integrating it with software extremely late in the "factory" makes common sense. We ship hardware because we need to own the problem first. We ship hardware because we have to.

## In Conclusion

SDS is a mandatory component of a software-defined datacenter (SDDC). Wikipedia defines software-defined storage as “a marketing theme for promoting storage technologies”. While this may be a natural conclusion based upon the large storage companies’ SDS positioning, it does injustice to a promising shift in datacenter architectures. SDS results from separating mechanism from policy and from late binding. These are the same concepts that encapsulate the true meaning of software-defined anything, including the datacenter itself.