TECHNICAL WHITE PAPER

Splunk SmartStore on Pure FlashBlade™

Pure Storage® delivers faster performance, better reliability, and a lower TCO for Splunk SmartStore implementations.

Contents

Introduction

Splunk® Enterprise helps companies solve complex data management and network security challenges, enabling them to more easily scale their business. It enables search, analysis, and visualization of machine data from IT infrastructure or business applications. The traditional Splunk Enterprise indexer cluster deployment model (also known as the distributed scale-out model) provides high data availability and fidelity but presents significant cost challenges as data volumes grow. Splunk SmartStore, the latest evolution of this model, provides efficient compute and storage elasticity along with simpler and more flexible data management infrastructure, which can help lower costs. The new model relies on an object data store located on remote storage. Because data availability and fidelity are trusted to remote storage, the choice of platform for the object store is critical.

Pure Storage is the market leader for all-flash storage arrays, offering simplified operations and improved efficiency. Splunk SmartStore on Pure Storage FlashBlade® accelerates access to critical, real-time data while reducing overhead costs, increasing availability, and improving operational efficiencies.

This document explains the benefits of deploying Splunk SmartStore on Pure FlashBlade and showcases the resulting performance improvements. The addition of Pure Storage FlashBlade provides:

- Maximum reliability 24/7 from resilient hardware
- Lower cost of ownership achieved by optimizing utilization and lowering storage requirements
- Faster insights in an all-flash infrastructure and consistent performance in data operations and searches
- Simple data management at scale with faster data rebalancing
- Increased Splunk availability through SmartStore that enables instant Indexer failure recovery
- Incremental storage space through inline compression
- Inline encryption, securing Splunk data at rest

Technology Overview

Pure FlashBlade

Pure Storage developed FlashBlade architecture to meet the storage needs of data-driven businesses. FlashBlade is an all-flash system, primarily optimized for storing and processing unstructured data. A FlashBlade system can simultaneously host multiple file systems and multi-tenant object store for thousands of clients.

FlashBlade's ability to scale performance and capacity is based on five key innovations:

- **High-performance storage:** FlashBlade maximizes the advantages of an all-flash architecture by storing data in storage units and ditching crippling, high-latency storage media such as traditional spinning disks and conventional solid-state drives. The integration of scalable NVRAM into each storage unit helps scale performance and capacity proportionally when new blades are added to a system.
- **Unified network:** A FlashBlade system consolidates high communication traffic between clients and internal administrative hosts into a single, reliable high-performance network that supports IPv4 and IPv6 client access over Ethernet links up to 100Gb/s.
- **Purity//FB storage operating system:** With the Purity//FB symmetrical operating system running on FlashBlade's fabric modules, Purity//FB minimizes workload balancing problems by distributing all client operation requests evenly among blades.
- **Common media architectural design for files and objects:** FlashBlade's single underlying media architecture supports concurrent access to files via a variety of protocols such as NFSv3, NFS over HTTP, and SMB (with Samba-level functionality) and objects via S3 across the entire FlashBlade configuration.
- **Simple usability:** Purity//FB on FlashBlade alleviates system management headaches as it simplifies storage operations by autonomously performing routine administrative tasks. With a robust operating system, FlashBlade is capable of self-tuning and providing system alerts when components fail.

A full FlashBlade system configuration consists of up to five self-contained rack-mounted chassis interconnected by high-speed links to two external fabric modules (XFM). At the rear of each chassis are two on-board fabric modules for interconnecting the blades, other chassis, and clients using TCP/IP over high-speed Ethernet. Both fabric modules are interconnected and each contains a control processor and Ethernet switch ASIC. For reliability, each chassis is equipped with redundant power supplies and cooling fans. The front of each chassis holds up to 15 blades for processing data operations and storage. Each blade assembly is a self-contained compute module equipped with processors, communication interfaces, and either 17TB or 52TB of flash memory for persistent data storage.

The current FlashBlade system can support more than 1.5 million NFSv3 getattrs per second, or >17 GiB/sec of 512KiB reads or >8 GiB/sec of 512KiB overwrites on a 3:1 compressible dataset in a single 4U chassis with 15 blades and can scale compute and performance up to a 5 x 4U chassis with 75 blades. A new FlashBlade system in 10 x 4U chassis with 150 blades is also in GA now.

Splunk Enterprise

Splunk Enterprise enables you to search, analyze, and visualize the machine data from your IT infrastructure or business applications. Splunk Enterprise takes in data from various sources (e.g. websites, applications, sensors, devices, etc.). Once you define the data source, it indexes the data stream and parses it into a series of individual events for viewing and searching.

Splunk SmartStore

Splunk SmartStore, the latest evolution of Splunk's traditional distributed scale-out deployment, provides a simpler and more flexible data management infrastructure. As a deployment's data volume increases, demand for storage outpaces demand for compute resources. SmartStore allows you to manage your indexer storage and compute resources in a cost-effective manner by scaling those resources separately.

With SmartStore, you can reduce the indexer storage footprint to a minimum and choose I/O optimized compute resources. Most data reside on remote storage, while the indexer maintains a local cache that contains a minimal amount of data: hot buckets, copies of warm buckets participating in active or recent searches, and bucket metadata.

Deploying Splunk SmartStore provides key advantages, including:

- Significant TCO reduction with decoupled compute and storage
- Performance at scale with application and data-aware SmartStore cache
- High-availability and data-resiliency features provided by remote object storage such as FlashBlade
- Efficient independent scaling of compute and storage
- Quick recovery from peer failure and fast data rebalancing, requiring only metadata fixups for warm data
- Full recovery of warm buckets even when the number of peer nodes that goes down is greater than or equal to the replication factor
- Simplified operations with setup/teardown of clusters on demand

SmartStore provides fast access to searchable data by retaining frequently accessed data on local indexers while offloading aged/unused data to remote storage, reducing the cost of storage. The cache brings in data from remote storage closer to compute on-demand with parallelized prefetch, ensuring performance while searching aged-out data.

CentOS Linux

CentOS version 7.5, a Linux distribution derived from Red Hat Enterprise Linux sources, provides free, enterprise-class computing platform functionalities, including support for Linux Containers. Splunk supports Splunk Enterprise on Linux, Windows, and MacOS. It supports Linux 3.x and 4.x kernel versions for Splunk Enterprise and offers RPM or DEB packages or a tar file depending on the version of the Linux on the host. Since a majority of the Splunk implementations have been on the Linux operating system, we opted to use CentOS Linux for our tests.

Challenges for Enterprises

Co-Located Storage and Compute

The Splunk Enterprise indexer cluster deployment model (also known as the distributed scale-out model) provides high data availability and fidelity but also presents significant cost challenges. In this deployment, the Splunk Enterprise indexers are configured to replicate each other's data, thus preventing data loss and facilitating searches. The distributed scale-out model is aligned with earlier big data technologies such as Hadoop, which relied on close server-storage proximity to achieve high performance.

In a distributed scale-out model, every Splunk indexer is configured—predominantly through direct-attached storage (DAS)—to have similar-sized storage for Hot/Warm and Cold tiers. This model worked well in the past to process the necessary volumes of data. However, as data volumes grow, storage and compute requirements do not scale linearly. Adding a node of the same type with compute and storage to address the storage requirement is suboptimal and cost-prohibitive (Figure 1).

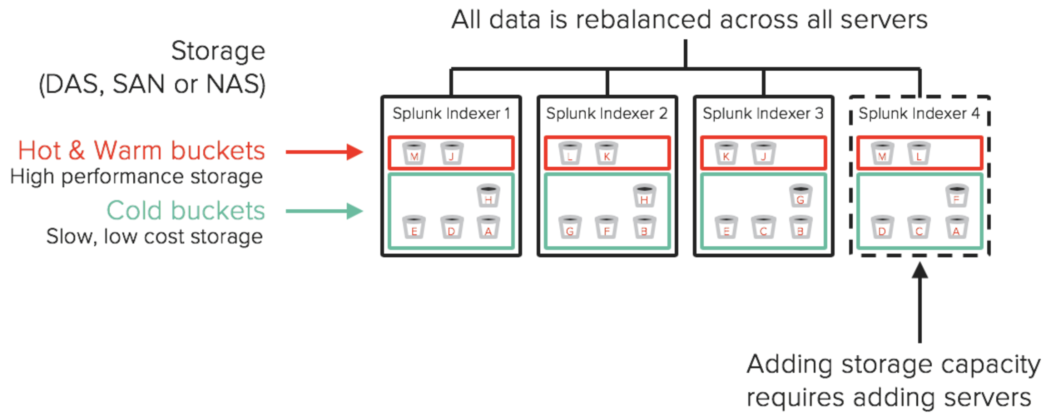


Figure 1: Adding a new node

A key requirement of a distributed scale-out model is the copy or replica of the data on the additional nodes based on the replication factor (Figure 2). An indexer cluster with replication factor (RF) of two and search factor (SF) of two requires the data to be ingested on a primary node and replicated to a secondary node. Hence the storage requirement spikes considerably in an indexer cluster environment based on the RF and SF. Splunk offers the TSIDX reduction feature to reduce the time series index files (tsidx) on data that requires infrequent searching as it ages, which reduces the disk usage. While this delivers substantial savings in disk usage, it comes at the cost of search performance when data is searched.

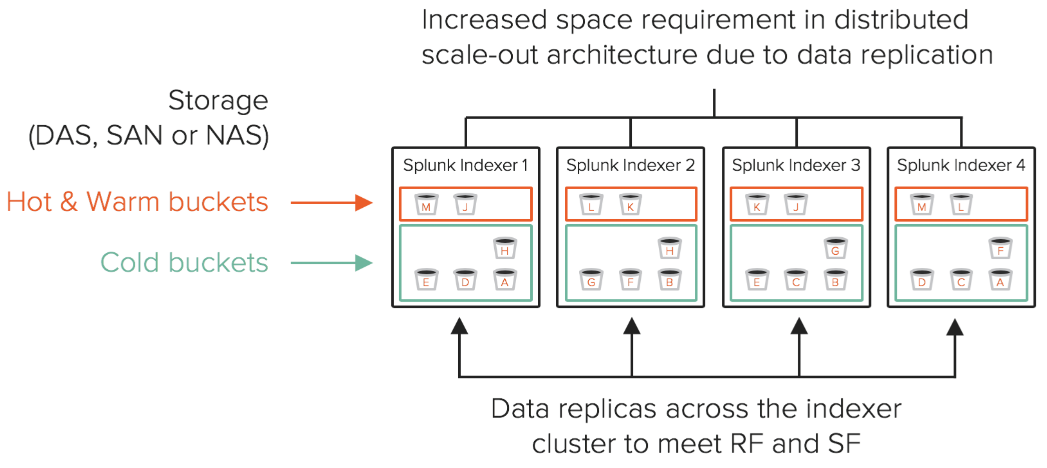

Increased space requirement in distributed
scale-out architecture due to data replication
Storage
(DAS, SAN or NAS)
Hot & Warm buckets
Cold buckets
Splunk Indexer 1
Splunk Indexer 2
Splunk Indexer 3
Splunk Indexer 4
Data replicas across the indexer
cluster to meet RF and SF

Figure 2: Distributed scale-out model

Operational Overhead

The addition of a new indexer into the cluster requires a data rebalance to distribute the Splunk bucket copies across the peer nodes to optimize each peer's search load and disk storage (Figure 3). The rebalance process can take hours per server and days or weeks for an entire cluster.

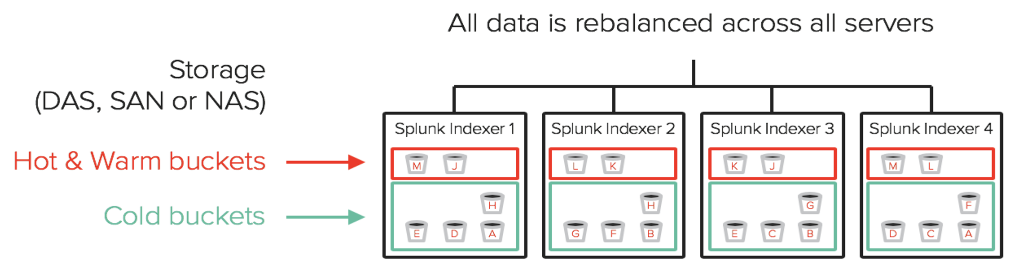


Figure 3: Rebalancing data across servers

Meanwhile, when a peer node goes down intentionally (offline command) or unintentionally (server crash), bucket fixup activities are performed by the cluster master, which instructs the remaining peers to fix the cluster's set of buckets to return the cluster to a complete state. The time that the cluster takes to return to a complete state can be significant because it must first stream buckets from one peer to another and make non-searchable bucket copies searchable. For more information on estimating the cluster recovery time when a peer node goes down, please see the Splunk documentation.

Upgrading or updating software can be lengthy and costly due to disruption in productivity. To minimize downtime and to limit any disruption to indexing and searching, the recommendation is to upgrade one peer node at a time (Figure 4).

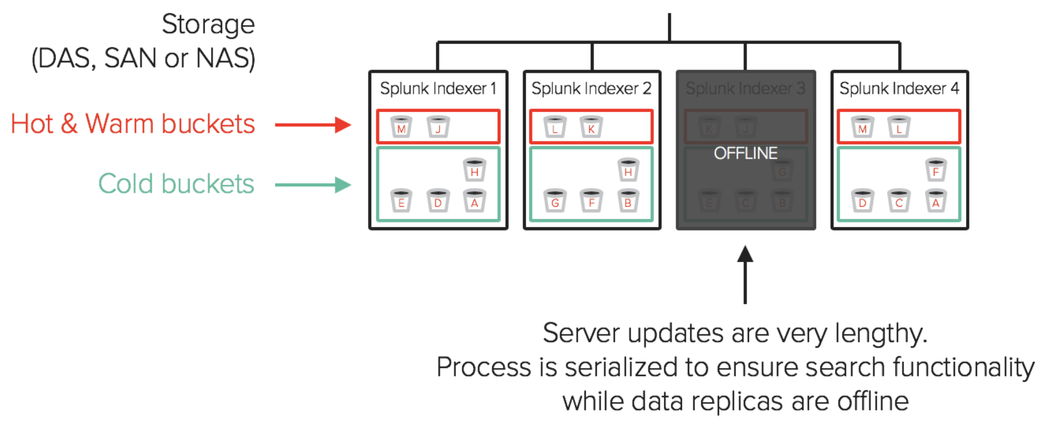

Storage (DAS, SAN or NAS)
Hot & Warm buckets
Cold buckets
Splunk Indexer 1
Splunk Indexer 2
Splunk Indexer 3
OFFLINE
Splunk Indexer 4
Server updates are very lengthy.
Process is serialized to ensure search functionality while data replicas are offline

Figure 4: Upgrading one peer node at a time

The Solution

The solution presented here is comprised of Splunk SmartStore using ultra-fast S3 protocol on FlashBlade for the remote storage tier. FlashBlade is a groundbreaking, scale-out, all-flash file and object storage system from Pure Storage architected to consolidate complete data silos while accelerating real-time insights from machine data using applications such as Splunk.

Disaggregated Storage

Splunk SmartStore disaggregates the storage from the compute to offer efficient utilization of compute and storage while enabling independent scaling of compute and storage to meet business needs. This means storage space can be scaled independently by adding blades or additional chassis into the FlashBlade as needed, rather than adding more indexer nodes as in the classic Splunk model. This results in reduced storage costs and a reduction in the number of indexer nodes under SmartStore and reduces the TCO.

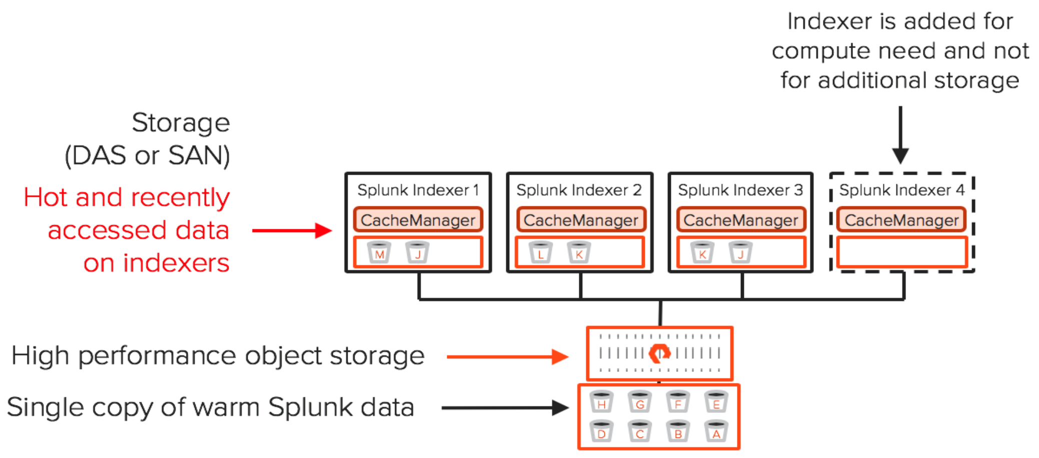

Indexer is added for compute need and not for additional storage
Storage (DAS or SAN)
Hot and recently accessed data on indexers
Splunk Indexer 1
CacheManager
M
J
Splunk Indexer 2
CacheManager
L
K
Splunk Indexer 3
CacheManager
K
J
Splunk Indexer 4
CacheManager
High performance object storage
Single copy of warm Splunk data
H
G
F
E
D
C
B
A

Figure 5: Adding blades

Operational Efficiency

Splunk SmartStore also provides simplified and efficient data management. Operational activities (e.g. indexer node additions, removals, failures, and, data rebalance) require only metadata updates between the nodes in the cluster to meet the replication factor and search factor parameters, instead of physically transferring the data between the nodes. This requires less time than the full data replication, enabling increased operational efficiencies. Figure 6 demonstrates the ability to update servers within minutes, because data balancing involves only metadata movement with SmartStore and not the actual data movement as required in the classic model.

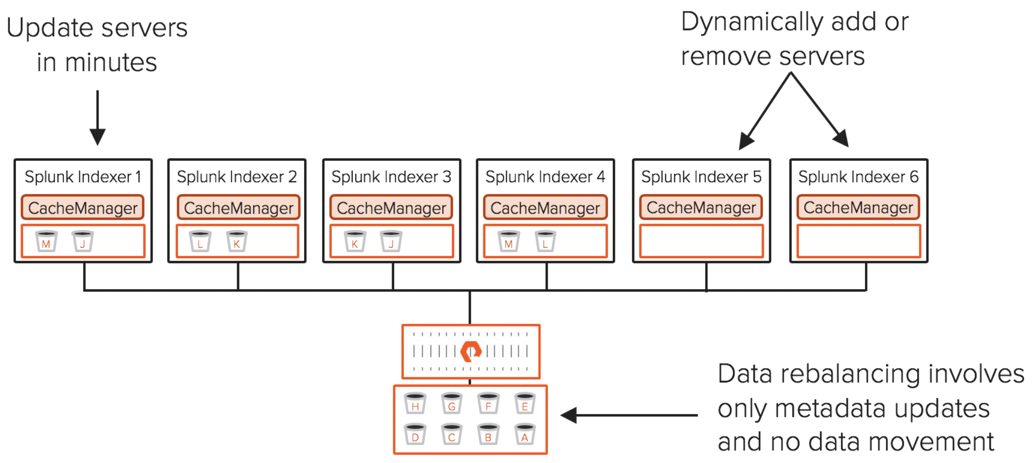


Figure 6: Updating servers quickly

Splunk SmartStore Accelerated by Pure

Splunk SmartStore, with FlashBlade as the high-performance object store, holds the single master copy of the warm/cold (in case of migration from classical to SmartStore) data while a cache manager on the indexer maintains the recently accessed data. Data movement between the indexer and the remote storage tier is managed by the cache manager. The data availability and fidelity functions are off-loaded to the remote object storage, FlashBlade, which offers N+2 redundancy.

Figure 7 shows Splunk SmartStore using FlashBlade for the remote object store.

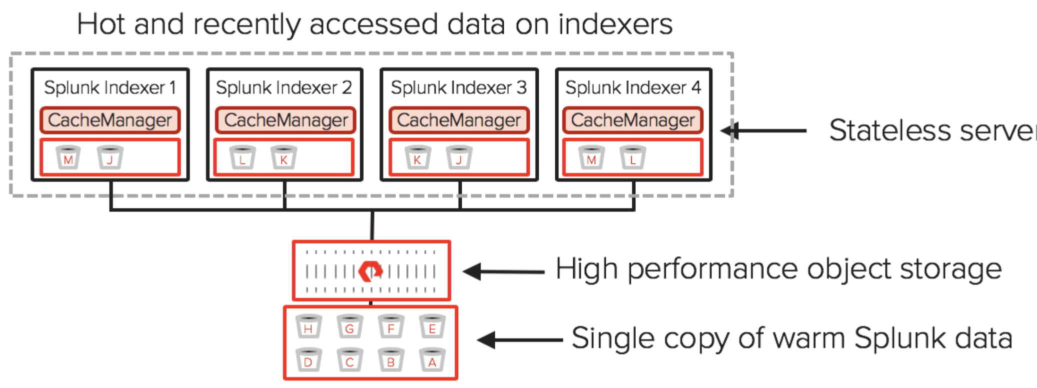


Figure 7: Splunk SmartStore using FlashBlade for the remote object store

Overall, Splunk SmartStore on FlashBlade:

- Enables dynamic cluster scaling by adding compute and storage independently
- Simplifies indexer cluster maintenance without impacting data integrity and availability
- Offers faster data rebalancing and faster peer node recovery
- Provides all-flash performance with no storage tiering
- Further reduces space usage through compression
- Protects Splunk data at rest on the remote storage through encryption
- Reduces indexer servers while supporting similar or higher storage capacity requirements
- Offers elastic architecture to scale seamlessly up to multi-petabyte capacity by simply adding more blades or additional chassis to the FlashBlade
- Improves performance when uploading and downloading data between the cache and FlashBlade
- Enables resiliency, thanks to FlashBlade's in-built N+2 redundancy and self-healing blades

Splunk SmartStore on FlashBlade Design

Design Topology

This section describes the design topology for the Splunk SmartStore on FlashBlade that was tested in our lab.

The solution included three chassis with 20 Intel CPU-based Cisco UCS B-series M4 and M5 blade servers for hosting the Splunk Enterprise configuration. The Splunk configuration was comprised of eight indexers, three search heads, one cluster master and eight universal forwarders. The Pure Storage FlashBlade was used for the remote object storage, to host warm data from Splunk.

We do not have any preferences regarding customers using commercially available Intel x86-based servers instead of Cisco UCS-based servers. Customers are advised to use their preferred choice of servers and storage.

Note: Splunk's recommended configuration is to use direct attached storage (DAS) for the hot and cache tier on the indexers. We recommend the same because any searches that do not find the data on the cache have to be downloaded from the object storage back to the cache. The download can be slowed by the write bandwidth of the hot/cache tier if an enterprise SAN storage is used instead of the direct attached storage.

The following set of tables show the respective component's configuration used in our tests. They do not represent any maximum available configuration of that component.

FlashBlade Configuration

The FlashBlade is the remote object storage that stores Splunk's warm data, compressed and encrypted at the storage level. In case of migration from Classic Splunk to the SmartStore, this will also hold the data from the cold buckets.

Note: The minimum Purity//FB version required to run SmartStore is 2.3.0.

Component	Description
FlashBlade	15 x 17TB blades
Capacity	240TB raw 162.46TB usable (with no data reduction)
Connectivity	4 x 40GB/s Ethernet (data) 2 x 1GB/s redundant Ethernet (Management port)
Physical	4U
Software	Purity//FB 2.3.0

Table 1: FlashBlade configuration

FlashBlade can support network connectivity up to 320GB/s per chassis (8 x 40GB/s or 32 x 10GB/s). To scale beyond single chassis, 2 FlashBlade external FabricModules (XFM) are required.

Operating System and Software Configuration

OS and Software	Description
Linux	CentOS 7.5 (64 bit)
Splunk	Splunk 7.2.6
Cisco UCS Manager	3.2 (1d)

Table 2: Operating system and software configuration

Physical Topology

The solution, Splunk SmartStore on FlashBlade, consists of a combined stack of hardware (compute, storage, network) and software (Splunk Enterprise, CentOS Linux, Cisco UCS Manager).

Component	Description
Indexer	8 Cisco UCS B200-M5 server blades, each with: 2 x Intel Xeon Gold 6138 @ 2GHz (20 cores) 128GB of memory, 40Gbps connectivity
Search Head	3 Cisco UCS B200-M4 server blades each with: 2 x Intel Xeon processor E5-2670 v3 CPUs (12 cores) 128GB of memory
Cluster Master	1 Cisco UCS B200-M4 server blade with: 2 x Intel Xeon processor E5-2609 v4 CPUs (8 cores) 64GB of memory
Forwarders	8 Cisco UCS B200-M4 server blades, each with: 2 x Intel Xeon processor E5-2680 v4 CPUs (14 cores) 64GB of memory
Networking	2 Cisco UCS 6332 UP 16-Port fabric interconnects 2 Cisco MDS 9148S fabric 16G FC switches 2 Cisco Nexus 9372PX Ethernet switches
Virtual Interface Card	40Gbps unified I/O ports on Cisco UCS VIC 1340
Chassis	3 x Cisco UCS 5108 Chassis (6RU each)

Table 3: Physical topology

Logical Topology

Figure 8 shows the logical architecture of the solution with the components listed below:

- Eight indexers in an indexer cluster setup
- Three search heads in a search head clustering
- One cluster master
- Eight universal forwarders

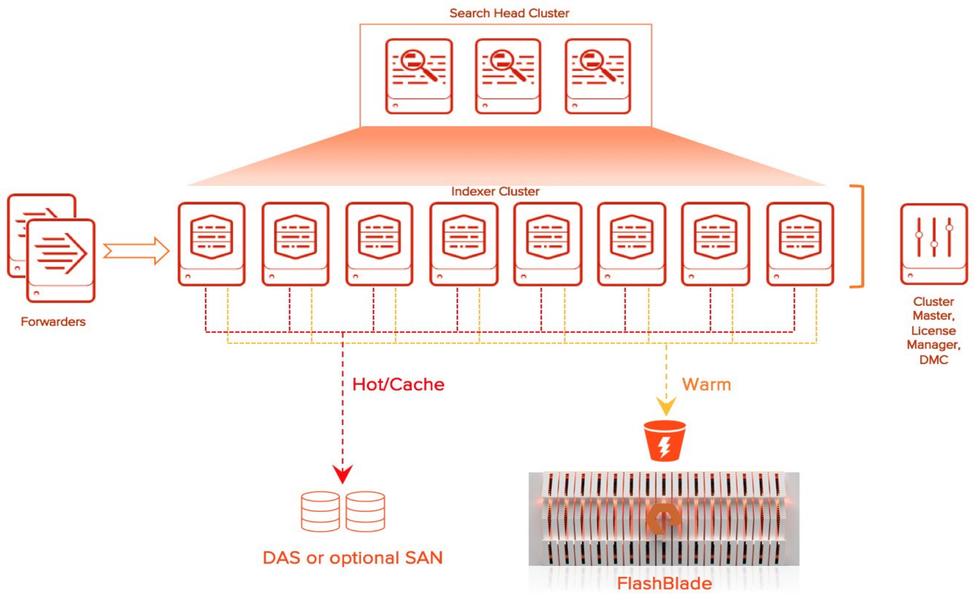


Figure 8: logical architecture of the solution

Design Considerations

Splunk Enterprise deployments are generally based on the following:

- **Daily ingest rate or indexing volume:** Higher index rates require more indexers.
- **Number and type of searches:** Numerous concurrent searches or resource-intensive searches (i.e. dense search) can tax the search heads and indexers.
- **Number of concurrent users:** Numerous users viewing dashboards or running searches would require more search heads, ideally a search head cluster.
- **Data fidelity:** An indexer cluster is required to ensure against data loss.
- **Data availability:** Deploy both an indexer cluster and a search head cluster to get access to the full set of data.
- **Disaster recovery requirements:** A multisite indexer cluster spread between two data centers enables identical data sets for quick recovery.

Indexing volumes using SmartStore do not differ from those of the non-SmartStore solution.

When data is in the cache, SmartStore searches are expected to be similar to non-SmartStore searches. When data is not in the cache, search times are dependent on the amount of data to be downloaded from the remote object store to the cache. Hence, searches involving rarely accessed data or data covering longer time periods can have longer response times than experienced with non-SmartStore indexes. The FlashBlade accelerates the download time considerably in comparison to any other cheap-and-deep object storage in the market.

Note that data fidelity, availability, and disaster recovery requirements for warm buckets are offloaded to the remote object storage and impacted by your choice of the remote object storage solution. Meanwhile, for hot buckets, Splunk still provides high availability.

Capacity

In a SmartStore deployment, storage sizing should be considered for two entities: cache size and remote object store.

SmartStore Cache Sizing Guidelines

As Splunk still manages the availability of the hot buckets, it still replicates them between the peer nodes based on the RF setting. Splunk's suggested SmartStore Cache sizing guidelines. are:

$$Global\ cache\ size = (daily\ ingest\ rate * RF + (cache\ retention - 1) * daily\ ingest\ rate)$$

Cache Retention: Searches the timespan for the majority of your searches. Splunk's recommendation is 30 days for Splunk Enterprise and 90 days for Splunk Enterprise Security. Configure this as needed.

RF: Replication factor, recommended setting is 2.

To calculate the cache size for each indexer, divide the global cache size by the number of indexers.

Ingest Rate	1TB	1TB	5TB	5TB
Cache Retention (Days)	10	30	10	30
Replication Factor (RF)	2	2	2	2
Global Cache Size (TB)	11TB	31TB	55TB	155TB

Table 4: Splunk SmartStore sizing capacity

Remote Object Store Sizing Guidelines

In classic Splunk, sizing is based on the hot/warm tier and cold tier retentions. For example, a one-year retention requirement might be split between hot/warm tier at one month and cold tier at 11 months but with replicated copies based on RF and SF. In the case of SmartStore, the remote object store should hold single copy of the data for the whole one-year retention period. Hence, there would be no split of the retentions between hot/warm and remote object store.

The guideline for the remote object store is to be sized to hold all the ingested data for the full retention period as a single copy irrespective of the replication factor is:

$$Remote\ Object\ Store\ Sizing\ =\ Daily\ Ingest\ Rate\ *\ Compression\ Ratio\ *\ Retention\ Period$$

Note: The Splunk compression ratio is generally 50% (15% from the compression of raw data and 35% from the metadata or the index files) but this is entirely dependent on the type of data. For higher-cardinality data, this percentage can go down, resulting in lower compressed data overall or increase in the storage sizing requirement.

As FlashBlade offers further data reduction through compression of the index files, conservatively at 1.3:1 data reduction ratio, the storage requirement goes down further.

Remote Object Store Sizing Summary

Ingest Rate	1TB	2TB	5TB	10TB
Retention Period (Days)	365	365	365	365
Compression Ratio	0.5	0.5	0.5	0.5
Object Store Sizing	183TB	365TB	913TB	1.83PB
FlashBlade Sizing	140TB	281TB	702TB	1.4 PB

Table 5: Remote object sizing

Performance

Planning system resources and bandwidth to enable search and index performance in a distributed indexer cluster environment must factor in the total volume of data being indexed and the number of active concurrent searches (scheduled or other) at any time.

Per Splunk's performance recommendation, an indexer that meets the Splunk's reference hardware specifications of 12 CPU cores at 2GHz or greater and 12GB RAM can ingest up to 300GB/day while supporting search load in a Splunk Enterprise deployment and 100GB/day in an Enterprise Security deployment. Splunk has introduced two new hardware specifications for better indexing performance and search concurrency over a distributed Splunk Enterprise deployment. The mid-range specification recommends 24 CPU cores at 2GHz or greater with 64GB RAM while the high-performance specification recommends 48 CPU cores at 2GHz or greater and 128GB RAM.

Since Splunk search heads and indexers are CPU-intensive, it is recommended to split the search and indexing functions with distributed searching enabled. This enables search heads to parallelize searches.

For further performance recommendations, please refer to the Splunk Enterprise Capacity Planning Manual.

Solution Validation and Testing

The architecture goal of the SmartStore feature is to decouple storage from compute to enable independent scaling of resources while maintaining the fast indexing and search capabilities characteristic of Splunk Enterprise deployments as well as optimized time-efficient indexer cluster management.

The Splunk solution on FlashBlade is validated by testing the above three key functions of Splunk Enterprise, namely data ingestion, search behavior, and cluster operational efficiency.

Data Ingestion

During data ingestion, hot buckets and recently ingested/searched warm buckets are placed in local storage. Master copies of other warm buckets reside on the object store. As in the case with non-SmartStore indexes, data ingested into the hot buckets hosted on the indexer's local storage cache is replicated to the hot buckets in local storage across the replication factor number of peer nodes.

When hot buckets are rolled into warm, a copy of the bucket is uploaded to the remote storage, which takes over the responsibility of maintaining the high availability of that bucket. Once the upload is acknowledged, the replicated copies on the target peer nodes are evicted. The source peer node maintains the local copy on the cache for the time determined by the cache eviction policy.

From this point, the replication factor still controls the metadata that the cluster maintains for the bucket. Unlike non-SmartStore indexes, the cluster does not attempt to maintain multiple copies of the warm buckets with SmartStore indexes.

Ingest Test Overview

Ingest 64TB of data into the indexer cluster with eight peer nodes configured with replication factor and search factor at 2 and all indexes configured for SmartStore. Measure the time taken to load 64TB of data as well as capture system utilization, upload throughput to remote object storage, and other pertinent metrics.

Ingest Test Setup

To scale test the SmartStore ingestion, four batches of 16TB of data was ingested, amounting to 64TB of overall data. The test data that was generated was the Apache-type log of sourcetype *access_combined* with additional keywords on every event that can be used for sparse and rare searches.

The dataset was generated through a custom script developed with the Go programming language. The data generation script was run on eight forwarders, which generated 2TB per server, totaling 16TB for each batch.

To speed up the data through forwarders, the throughput entry maxKBps in $SPLUNK_HOME/etc/system/local/limits.conf was set to 0, which means there was no regulation of the data throughput from the forwarder to the indexers. The default throughput of universal forwarders is 256KBps. For testing purposes, larger throughput (e.g. 10240 or 0) might be acceptable but for standard operating procedures, set this to a value that reflects the environment, so as not to overwhelm the network infrastructure.

The 16TB of data was ingested into four SmartStore indexes named apache-pure1, apache-pure2, apache-pure3 and apache-pure4 sequentially from the eight universal forwarders using the oneshot method.

```
$SPLUNK_HOME/bin/splunk add oneshot <source log file> -index apache-pure1 -sourcetype access_combined -auth
admin:splunk123
```

Ingest Test Results

The data ingestion of 64TB from eight universal forwarders into the indexer cluster with eight indexers took 79 hours 55 minutes at an average rate of 233MBps and peak around 275MBps.

Ingested Data	Elapsed Time	Indexing Rate	Indexing Rate per Indexer
65TB	79 hours, 55 min	233MB/s	29.13 MB/s

Table 6: Ingest test results

As the dataset was loaded manually across each index, there were dips in the indexing rate as seen in Figure 9. These certainly impacted the average indexing rate.

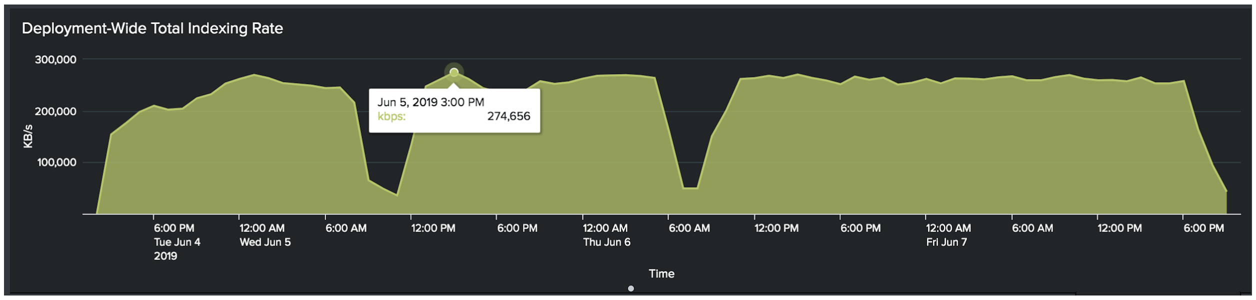


Figure 9: Deployment-wide total indexing rate

During the ingest, the overall CPU utilization of the eight indexer nodes was under 25%. The fill ratio of all the key data processing queues were never saturated and remained under 35% throughout the ingest process.

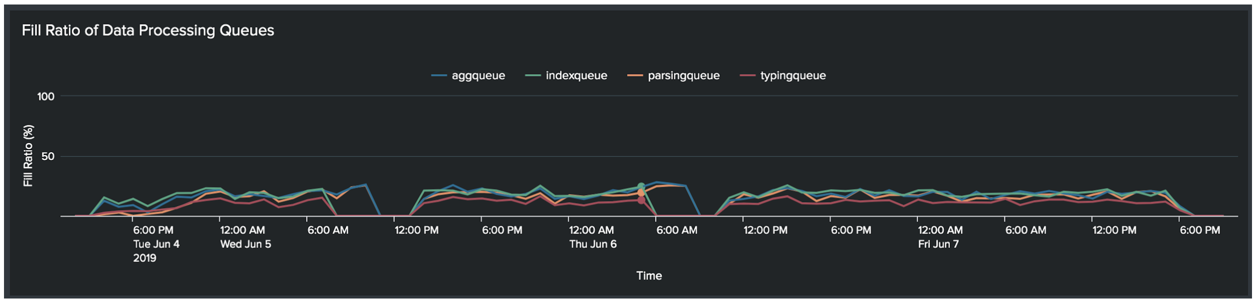


Figure 10: Fill ratio of data processing queues

As the hot buckets are rolled into warm, Splunk uploaded the warm buckets onto the remote object storage. Figure 11 is a screenshot of the FlashBlade GUI showing the write IOPS and bandwidth on the FlashBlade.

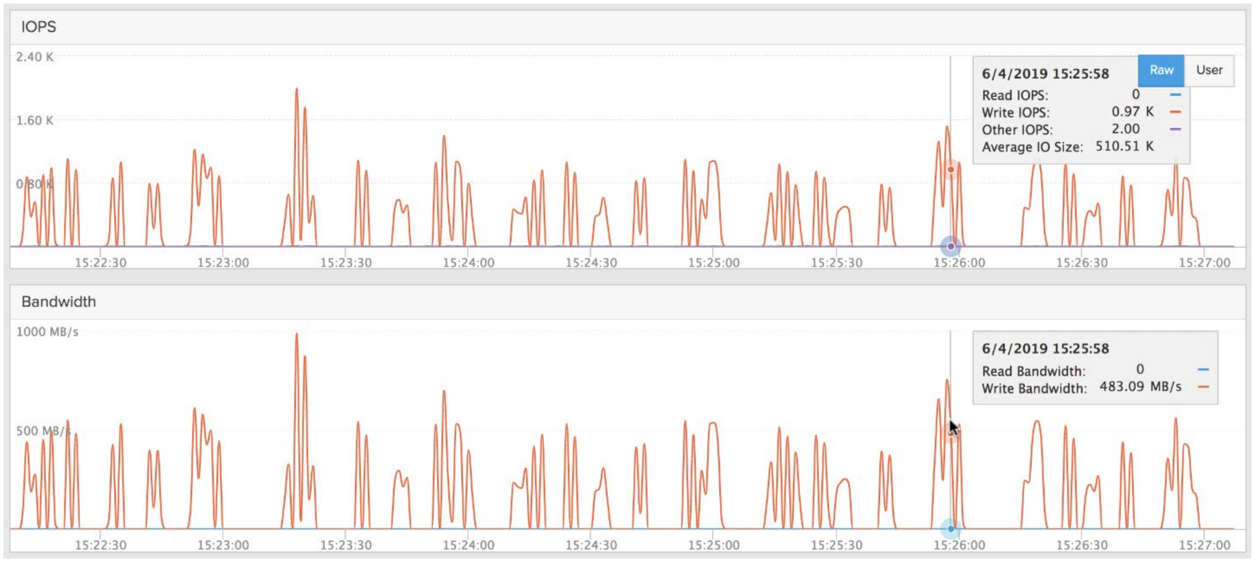


Figure 11: Write IOPS and bandwidth on the FlashBlade in Flashblade

Space Usage

Splunk consumed only 29.78TB of storage space for the 64TB of ingested data. This is a 2.15:1 data reduction at the Splunk level which is slightly higher than Splunk's recommendation on disk usage.

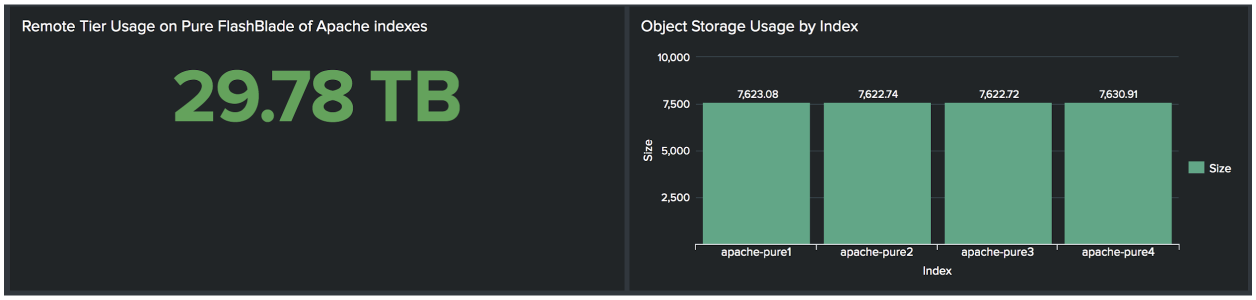


Figure 12: Splunk usage on FlashBlade

In addition to the data reduction by Splunk, the remote object storage, FlashBlade also compresses the Splunk data, resulting in a data reduction which varies between 1.2:1 to 1.5:1 depending on the type of data and cardinality.

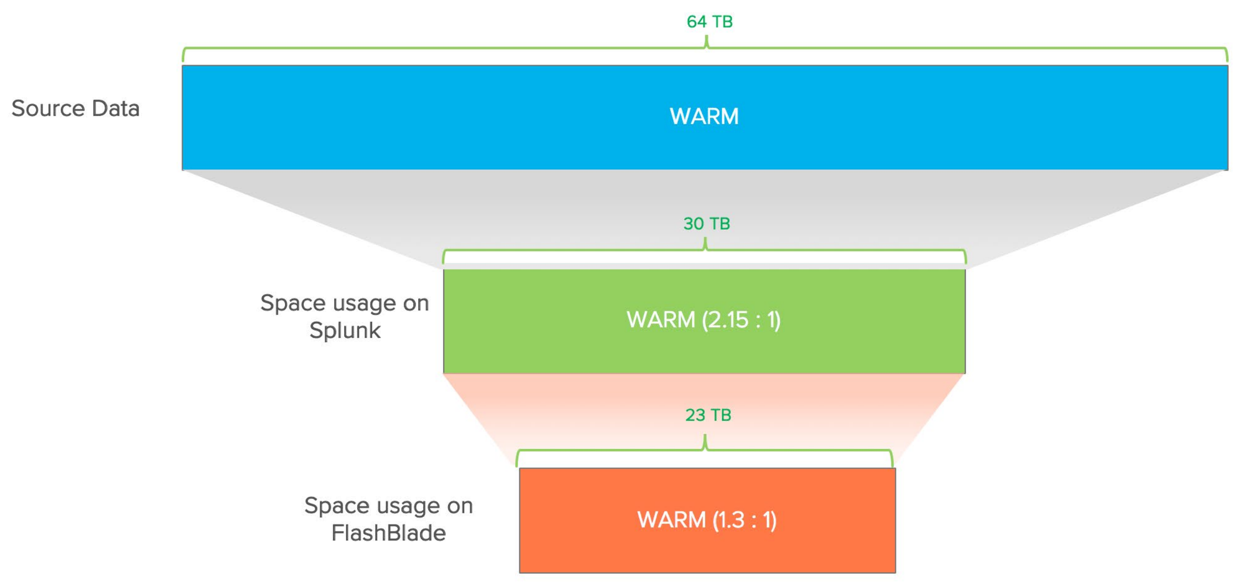


Figure 13: Flashblade data reduction compared to source data and Splunk

Splunk's recommended[v] daily ingestion rate per indexer based on reference hardware with 12 cores at 2GHz is 300GB per day for Splunk Enterprise, while supporting a search load. This equates to *2.4TB per day* of ingest, per our eight-indexer node configuration. For enterprise-scale, Splunk recommends high-performance reference hardware with 48 cores which can yield higher than 300GB per day ingest rate while supporting search load.

In our test, with concurrent search load, the theoretical daily ingest that was achieved with eight indexer nodes under 25% of CPU utilization and with healthy indexing queues of **19.2TB per day**. This is because of the selection of Cisco UCS M5 blades for indexer nodes, which are made up of 2x20 CPU cores (40 cores in total). This aligns with Splunk's high-performance reference hardware's performance capability and clearly demonstrates the possibility of improving performance on a superior infrastructure.

Search Behavior

SmartStore introduces a Cache Manager layer on the indexers, to provide optimal search performance at scale, for datasets that may not reside locally on the indexers.

Research on the usage patterns of Splunk customers indicates that a majority of searches involve near-term data and have spatial and temporal locality. When a search finds an event at a specific time or in a specific log, it is likely that other searches will look for events within a closely similar time range or in that log. Hence, the cache manager favors recently created buckets and recently accessed buckets to ensure that most of the data that are likely to be searched is available on the cache.

If the required buckets are not in the cache, the cache manager downloads them from the FlashBlade to the local cache. When the search data is found in the cache, the search times will be almost equivalent to those of traditional Splunk (non-SmartStore). When the data is not in the cache, search times will depend on the amount of data to be downloaded from the remote object storage to the cache.

Hence, it is very critical to right-size the cache settings on the Splunk SmartStore indexers and have a high-performance remote object storage, which accelerates data transfer to cache with low latency.

Apart from the cache size, the type of searches can also affect the Splunk Enterprise performance as different types of search like dense, sparse, and rare have different performance characteristics:

- Dense search returns a large percentage of matching results for a given dataset in a given period. Dense search is usually CPU intensive because of the overhead required to uncompress the raw data stored on the index.
- Sparse search returns a smaller amount of results for a given dataset in a given period and can be both CPU- and I/O-bound.
- Super-sparse search returns a small number of results from each index bucket that matches the search. This is I/O intensive because the indexer must look through all of the buckets of an index to find the results. Super-sparse search can take a long time to finish based on the amount of data available on the indexers.
- Rare search returns a smaller number of results but can receive assistance from bloom filters that help eliminate index buckets that don't match the search request. They are also I/O bound but can return results faster than super-sparse search.

Search Test Overview

We performed the following search tests and measured critical metrics (e.g. search time, download size):

- Cache usage test
- Concurrent searches test

The first search test is to showcase the impact of cache usage and the second test to measure the overhead of running different types of concurrent searches on three search heads and eight indexers. For the concurrent searches test, we opted to perform sparse and rare searches.

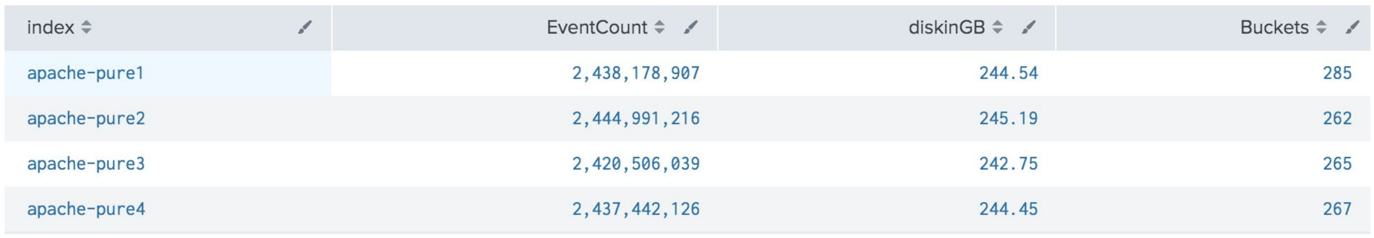
Search Test Setup

The total dataset of 64TB spanned 16 days with each day holding one TB per apache index or four TB for all four indexes (apache-pure1, apache-pure2, apache-pure3, and apache-pure4). Each index on an average consumes around 500 buckets per day. The cache size for each indexer used in this test was set to 375GB.

Cache Usage Test

In this test, a single search against each apache index was performed multiple times by evicting the cache and loading a specific amount of data upfront every time to mimic the varying cache usage. The test was repeated to achieve cache-hits of 0%, 25%, 50%, 75% and 100%.

The cache usage search test would go after 12 hours of dataset, which is the equivalent of 270 buckets and approximately 244GB of data per index, or 975GB across all four indexes. The following table gives the summary details of the buckets per index for the 12-hour search.



index	EventCount	diskinGB	Buckets
apache-pure1	2,438,178,907	244.54	285
apache-pure2	2,444,991,216	245.19	262
apache-pure3	2,420,506,039	242.75	265
apache-pure4	2,437,442,126	244.45	267

Figure 14: Buckets per index over the 12-hour search

Table 7 illustrates the characteristics of the search in terms of data size and events.

Data Set	12 Hours
Data Size	975GB (1079 buckets)
Total Events	9.7 billion
Result Set	94,368 events

Table 7: Search characteristics

The following searches would be run across four indexes at the same time.

```
index=apache-pure1 every100k MadlyOdd starttime="05/20/2019:08:00:00" endtime="05/20/2019:20:00:00"
index=apache-pure2 every100k raymondoesit starttime="05/20/2019:08:00:00" endtime="05/20/2019:20:00:00"
index=apache-pure3 every100k Diamondsda starttime="05/20/2019:08:00:00" endtime="05/20/2019:20:00:00"
index=apache-pure4 every100k Mathioluiz starttime="05/20/2019:08:00:00" endtime="05/20/2019:20:00:00"
```

Here is the test sequence to measure the search run time and the cache overhead.

1. Evict the local cache by running the following REST API across all indexers
   `$SPLUNK_HOME/splunk _internal call /services/admin/cacheman/_evict -post:mb 10000000 -post:path <homePath> -method POST -auth admin:splunk123`
2. Run the search to go after 12 hours of dataset, which should bring up all data from the remote object store to the cache. This is equivalent to 0% cache-hit.
3. Evict the cache again on all indexers.
4. Run the search to go after three hours of dataset, which should bring up 25% of the data into the cache.
5. Run the search to go after all 12 hours of dataset, which should download the remaining 75% of data with 25% of cache-hit.
6. Evict the cache again on all indexers.
7. Run the search to go after six hours of dataset to download 50% of the data into the cache.

9. Run the search to go after all 12 hours of dataset, which should download the remaining 50% of data with 50% of cache-hit.
10. Evict the cache again on all indexers.
11. Run the search to go after nine hours of dataset to download 75% of the data into the cache.
12. Run the search to go after all 12 hours of dataset, which should download the remaining 25% of data with 75% of cache-hit.
13. Run the search again to go after all 12 hours of dataset, which should not download any data as all the data should be in the cache which is equivalent of 100% cache-hit.

The above searches were configured as reports and were scheduled to run at specific times.

Concurrent Searches Test

The concurrent searches test was performed against two dimensions: search type (sparse and rare) and cache-hits (100% and 90%). In these tests, we ran 120 concurrent searches with 40 searches against each of the three search heads. Table 8 illustrates the search type (sparse and rare), per search dataset in terms of size, events per search, and the expected events.

Search Distribution	Search Type	Events per Search	Total Events per Search	Dataset per Search
100%	Sparse	9820	98.2 million	15GB
90%	Sparse	9820	98.2 million	15GB
100%	Rare	2	32.7 million	10GB
90%	Rare	2	32.7 million	10GB

Table 8: Search type per search dataset

In the case of 100% concurrent searches, all 120 searches go after the data in the cache. As each sparse search goes after approximately 15GB of Splunk data, running all 120 searches is expected to cover 1.8TB of Splunk dataset. The rare search goes after 1.2TB of Splunk data in the cache.

Table 9 illustrates the number of searches on cache tier and non-cache tier based on the search distribution and the corresponding dataset.

Search Distribution	Search Type	Searches on Cache Tier	Cache Tier Dataset	Searches on Non-cache Tier	Non-cache Dataset
100%	Sparse	120	1.8TB	0	-
90%	Sparse	108	1.6TB	12	180GB
100%	Rare	120	1.2TB	0	-
90%	Rare	108	1.08TB	12	120GB

Table 9: Search type per search tier

In the case of concurrent searches at 90% cache-hit, 90% of the searches (108) should go after the data in the cache and 10% of the searches (12) should go after the data that is not in the cache. The expectation in this search test is that 10% of searches would take up more time as they have to download the buckets from the remote object storage into the cache.

The searches were submitted through curl using the REST endpoints for the searches. The search results were extracted through the same mechanism.

Search Test Results

Cache Usage Test Results: Based on the bucket distribution, the total cache usage for the search that goes across four indexes is 976GB.

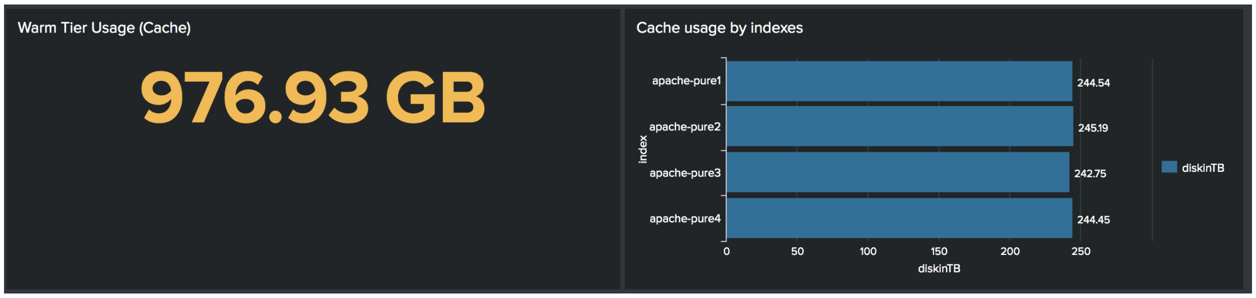


Figure 15: Total cache usage

Figure 16 shows the download details of the search when 100% of the data was downloaded from the remote object Storage, FlashBlade, for the first time after evicting all the data from the cache.

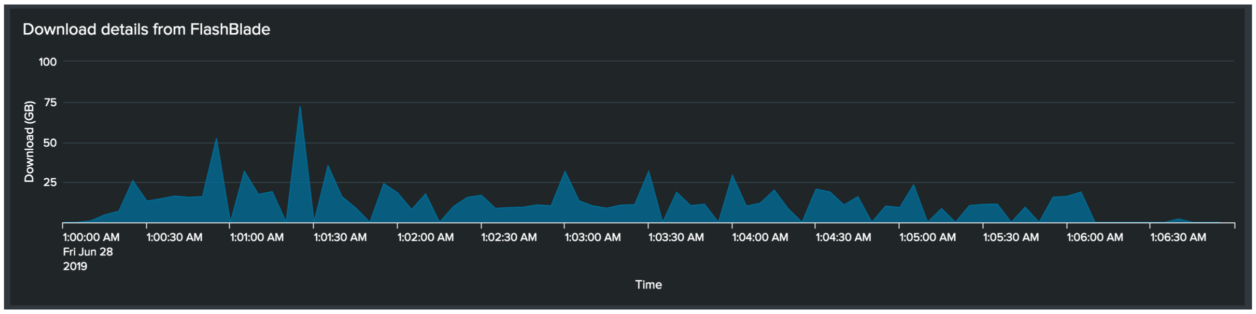


Figure 16: Download details from Flashblade.

Figure 17 shows the run time of the searches with varying degrees of cache-hits.

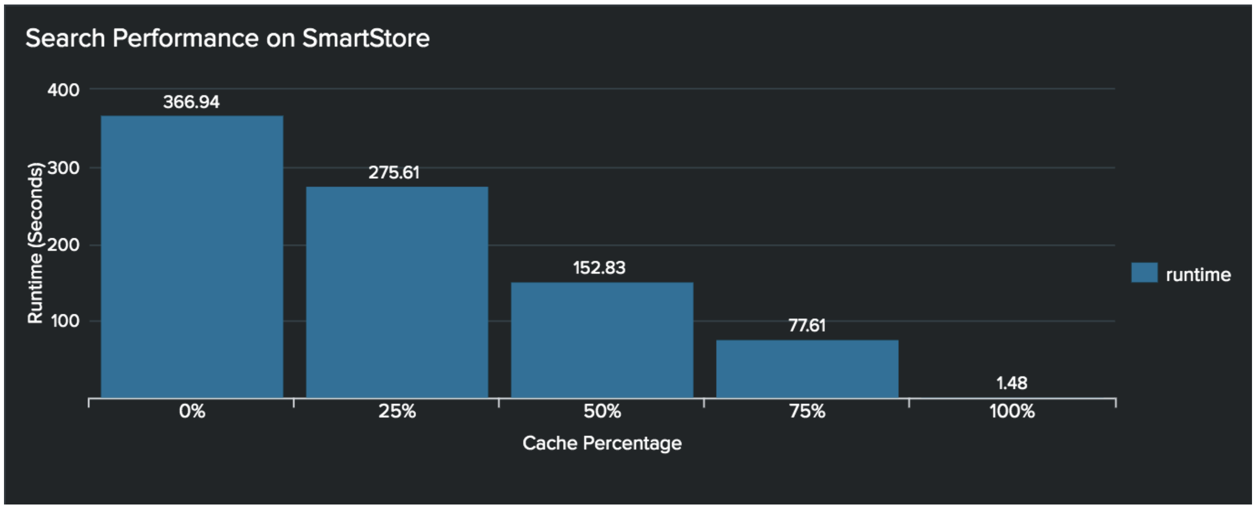


Figure 17: Search performance on FlashBlade

At 0% cache-hit, the search took 366.94 seconds to download the 976GB of data from the FlashBlade to the local cache. At 100% cache-hit, the search took 1.48 seconds to search 1079 buckets to return the 94,368 events out of the 9.7 billion events. As you can see, the search performs linearly in proportion to the availability of the data in the local cache and reiterates the importance of cache sizing with SmartStore.

Concurrent Searches Test Results: Figure 19 illustrates the search results of the 120 concurrent searches under 100% cache-hits. The 100% cache-hits series corresponds to all the 120 searches going after the data in the cache.

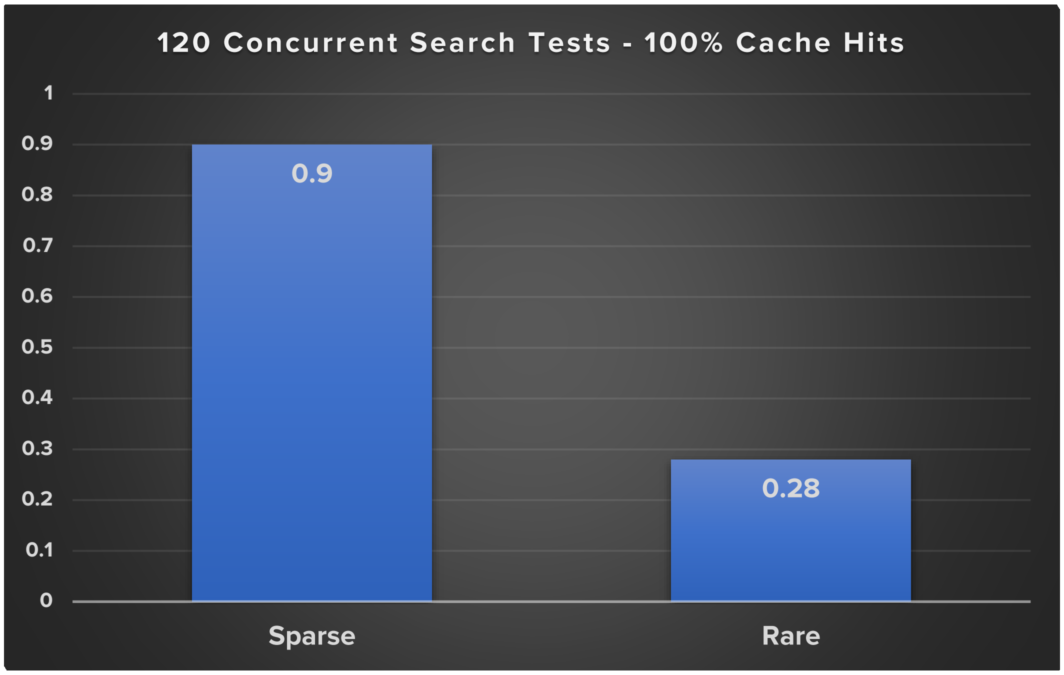


Figure 18: 100% cache hits in sparse vs rare searches

As seen in Figure 18, each sparse search took 0.9 seconds to complete while each rare search took 0.28 seconds.

Figure 19 shows the run time of the sparse and rare searches with 90% cache-hits. This means that out of the 120 searches, 90% of the searches, or 108, went after the data in the cache, and the remaining 10%, or 12 searches, went after the data not in the cache.

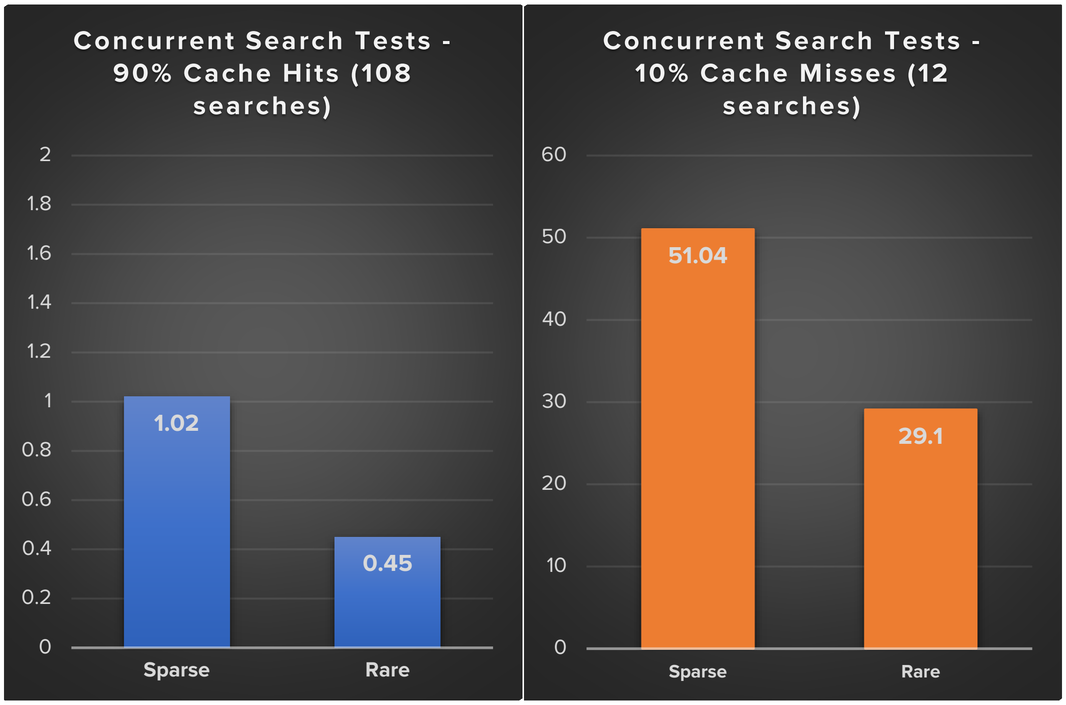


Figure 19: Run time of sparse and rare searches with 90% cache-hits

The 10% searches had to download the data from the FlashBlade onto the cache to complete the search. The run timings are directly proportional to the amount of data to be downloaded from the remote object storage back to the cache.

In the above test for the dense search, approximately 180GB of data had to be downloaded from the FlashBlade onto the cache for the 10% of the searches. The dashboard in Figure 20 shows the overall downloaded data and the details. The data was extracted from the Metrics component of the Splunk "internal" index.

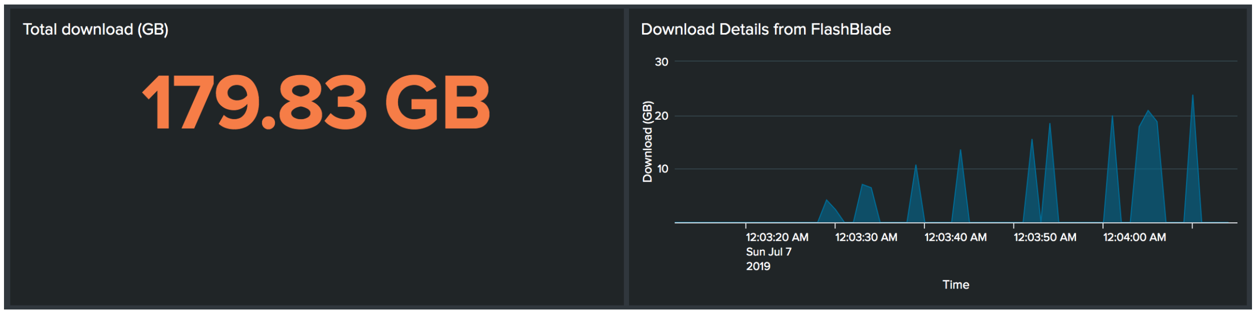


Figure 20: Download details from FlashBlade

Overall, each dense search took 51.04 seconds to complete. This included the I/O time to download the 15GB of data per search (180GB of data for all 12 searches). Similarly, each sparse search that went after the data not in the cache took 29.1 seconds to complete. This included the download from FlashBlade to the cache of 10GB of Splunk data per search (120GB of data for all 12 searches). As you can see from Figures 20 and 21, 120 sparse searches on 100% cache-hit took an average 0.9 seconds to complete while the 108 searches on 90% cache-hit took 1.02 seconds to complete. Similarly, 120 rare searches on

100% cache-hit took on an average 0.28 seconds to complete while the 108 searches on 90% cache-hit took 0.45 seconds to complete. This is because searching data in the object store added the overhead from the cache manager.

Figure 21 shows host-wide CPU usage while the 120 concurrent searches were run with 100% of the data in the cache. Interestingly, the maximum CPU utilization that we noticed across indexers and search heads was under 4%.

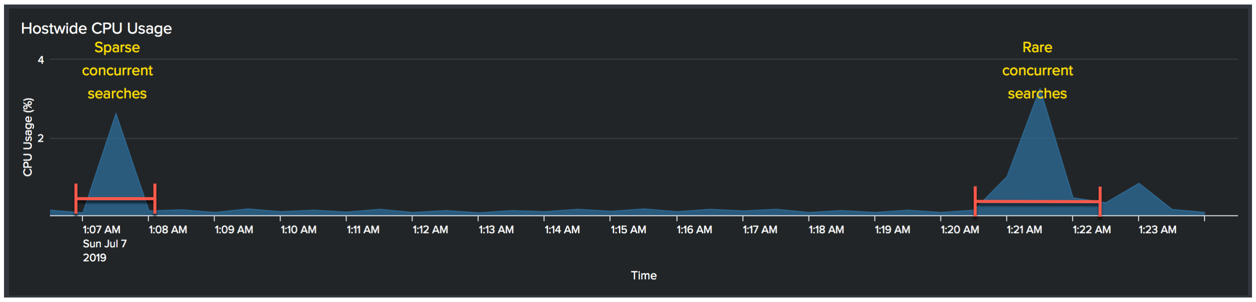


Figure 21: Host-wide CPU usage

Overall, the search performance on Splunk SmartStore is directly proportional to the availability of data on the local cache and the performance of the remote object store. This is where FlashBlade can accelerate downloads by an order of magnitude when compared to the legacy cheap and deep object storage solutions.

Operational Efficiency

As the storage is disaggregated from the compute in the Splunk SmartStore architecture, the responsibility of data availability and disaster recovery of the SmartStore warm buckets shifts from the Splunk indexer cluster to the remote storage service, FlashBlade in this instance.

The SmartStore cluster uses the replication factor to determine the number of copies of SmartStore warm bucket metadata it maintains. In the case of warm bucket fixup, the cluster needs to replicate only the bucket metadata and not the complete contents of the warm bucket.

In contrast to non-SmartStore indexes, a cluster can recover most of the data in SmartStore indexes if it loses peer nodes that equal or exceed the replication factor in number. In such cases, the cluster can recover all of the SmartStore warm buckets because those buckets are stored remotely and are unaffected by peer node failure. The cluster may lose some of its SmartStore hot buckets because they are stored locally rather than on the remote object store.

Over time, the indexer cluster can become imbalanced in terms of bucket distribution across all its peer nodes. This may be caused by forwarders sending unequal amounts of data to peer nodes. The imbalance increases the likelihood that one or more peers will run out of disk space and transition into a detention state in which it no longer indexes new data. Such an imbalance also increases the chances of not all peer nodes participating in the distributed searches. With data unequally spread across nodes, some peer nodes may be overutilized.

To overcome the imbalance and rearrange the buckets equally across all the peer nodes, data rebalancing can be performed on the indexer cluster. In the case of non-SmartStore, data rebalancing will literally move bucket copies from peers with more copies to peers with fewer copies This can be time-consuming depending on the number of buckets to be redistributed. In SmartStore, as the bucket copies on the peer nodes contain only the metadata, and not the actual data, the rebalance activity is faster.

To illustrate the operational improvements with SmartStore, the following two tests were performed on non-SmartStore and SmartStore environments that had exactly similar data:

- Indexer node addition and data rebalance
- Indexer node failure

Indexer Node Addition and Data Rebalance Test Setup

In this test, we added a new indexer node to the existing indexer cluster comprised of eight indexers, and performed a data rebalance activity to rearrange the buckets equally across all the nine indexers. During this test, both concurrent searches and data ingest were performed.

> **Note:** As per Splunk's guidelines, do not use searchable data rebalance with SmartStore indexes. The Searchable mode is not optimized for SmartStore and can cause the data rebalance to proceed slowly.

The fully populated indexer cluster in our environment had approximately 78K buckets across eight indexers.

A new peer node (splunk-ix09) was set up with Splunk 7.2.6 installed and configured with similar hot tier/cache configuration as other indexers. Once Splunk was started and pointed to the license manager, the cluster config on the peer node was updated with the mode "slave" and pointed to the cluster master.

```
[root@splunk-ix09 ~]# $SPLUNK_HOME/bin/splunk edit cluster-config -mode slave -master_uri https://splunk-
cm01:8089 -replication_port 8080 -secret PureSplunk -auth admin:splunk123
```

Data rebalance can be invoked through Splunk UI or via the command line. We configured the data rebalance threshold to 95%, which means the rebalancing continued until the number of copies on each peer was within a range of .95 to 1.05 of the average. In this test, we configured the data rebalance threshold and initiated data rebalance through the command line with the following commands.

```
$SPLUNK_HOME/bin/splunk edit cluster-config -mode master -rebalance_threshold 0.95 -auth admin:splunk123
$SPLUNK_HOME/bin/splunk rebalance cluster-data -action start -auth admin:splunk123
```

The bucket usage across the indexer cluster was captured before and after the indexer node addition and data rebalance, along with the time it took for the data rebalance to complete.

Indexer Node Failure Test Setup

To simulate the node failure, we brought the peer node offline with the enforce-counts option to ensure all data was replicated before the peer was fully shutdown. This is the process typically followed when retiring an indexer. During this test, concurrent searches and data ingest were performed.

Again, the indexer cluster in our environment had approximately 78K buckets spread across nine indexers.

The bucket usage across the indexer cluster was captured before and after the indexer node was taken offline along with the time it takes for the data redistribution to complete.

The following command will be issued from the indexer node (splunk-ix09) to initiate the index offline process:

```
$SPLUNK_HOME/bin/splunk offline --enforce-counts
```

Operational Efficiency Test Results

Indexer node addition and data rebalance test: A new indexer was added, and the data rebalance command as given below was performed on both classic Splunk and SmartStore environment which had a similar dataset (32TB) and bucket distribution (78K buckets). During this test, data were ingested and concurrent searches were performed.

```
[root@splunk-cm01 opstest]# date; splunk rebalance cluster-data -action status -auth admin:splunk123
Mon Jun 24 18:30:09 PDT 2019
Data rebalance is running
Status (-1=not running, 0-100 for percentage): 100.00
Last completed rebalance in minutes (-1=never ran): -1
[root@splunk-cm01 opstest]# date; splunk rebalance cluster-data -action status -auth admin:splunk123
Mon Jun 24 18:30:59 PDT 2019
Data rebalance is running
Status (-1=not running, 0-100 for percentage): 100.00
Last completed rebalance in minutes (-1=never ran): -1
[root@splunk-cm01 opstest]# date; splunk rebalance cluster-data -action status -auth admin:splunk123
Mon Jun 24 18:32:19 PDT 2019
Data rebalance is not running
Status (-1=not running, 0-100 for percentage): -1
Last completed rebalance in minutes (-1=never ran): 0
```

The data rebalance command took 11 hours and 45 minutes to complete under the Classical Splunk whereas it just took **2 minutes and 10 seconds** to complete on the SmartStore environment.

During the data rebalance, there was no hiccup or disruption to either the concurrent searches or the data ingestion.

> Note: The non-SmartStore environment used similar all-flash hot/warm tier and all-flash cold tier over NFS. The data rebalance times can take significantly longer if they are on a traditional disk-based storage tier.

Figures 22 and 23 show the bucket fixup details between the SmartStore and classic mode.

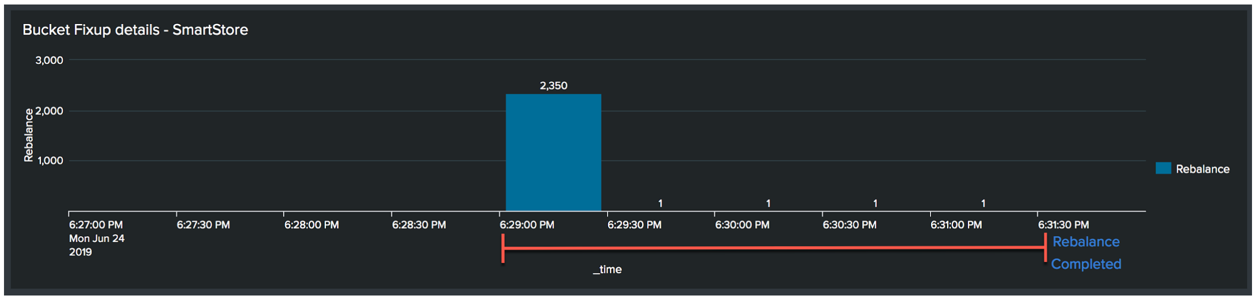


Figure 22: Bucket flip details: SmartStore

Figure 23 shows the rebalance activities on a similar dataset on non-SmartStore indexes. During this process, Splunk replicated the data across the nodes as part of redistribution, made the buckets searchable, and truncated the size of the buckets that were moved.

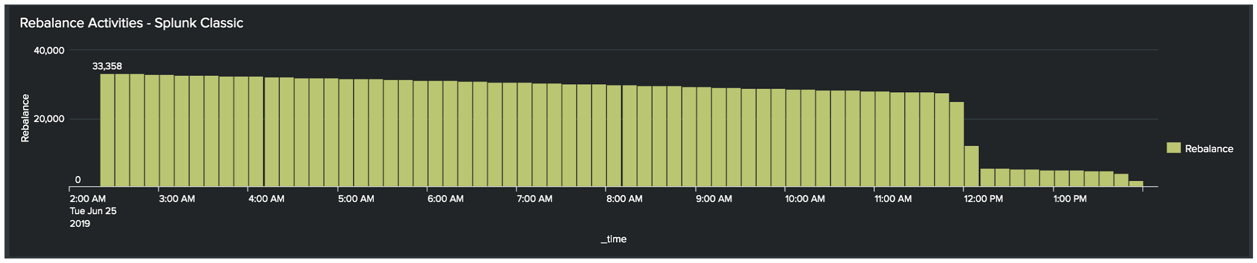


Figure 23: Rebalace activities on SmartStore

Figure 24 summarizes the comparison of the data rebalance activity between non-SmartStore and SmartStore environment with the same dataset.

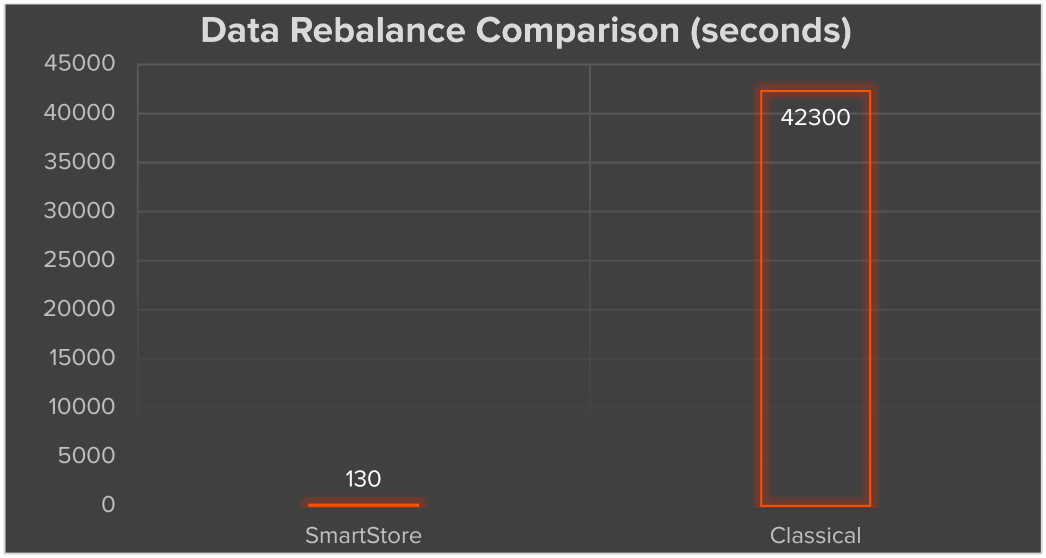
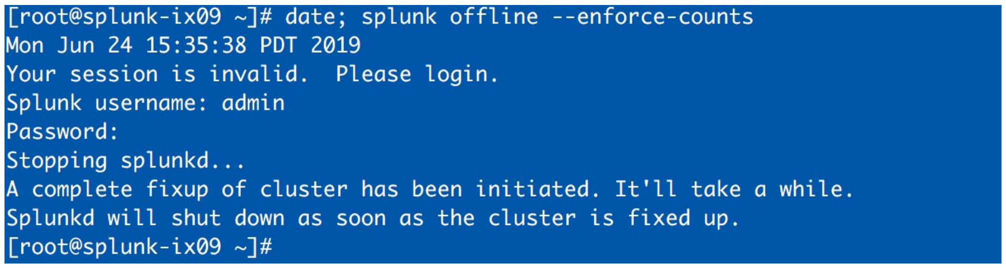


Figure 24: Data rebalance comparison

The test reaffirms the SmartStore optimization where the data were rebalanced at a faster rate because only the buckets' pointers, and not the actual data, were redistributed.

Indexer node failure test: In this test, an indexer was offlined on both classic Splunk and SmartStore environment which had a similar dataset (32TB) and bucket distribution (78K buckets).

The offline command was performed on the indexer node with the enforce-counts option. During this time, the concurrent searches and data ingestion were performed and ran without any issues.



```
[root@splunk-ix09 ~]# date; splunk offline --enforce-counts
Mon Jun 24 15:35:38 PDT 2019
Your session is invalid.  Please login.
Splunk username: admin
Password:
Stopping splunkd...
A complete fixup of cluster has been initiated. It'll take a while.
Splunkd will shut down as soon as the cluster is fixed up.
[root@splunk-ix09 ~]#
```

Figure 25: Offline command

Once it is kicked off, the offline command starts the decommissioning process. The cluster master starts the remedial bucket-fixing processes to return the cluster to a complete and valid state. The enforce-counts ensures the indexer's replicated data is available on other peer nodes in the cluster. Only after all the bucket-fixing activities needed to return the cluster to a complete state have completed and ongoing searches are done does the peer node go offline. Under the Indexer clustering page, the off-line peer node will show up with status "Graceful shutdown."

Overall it took 2 hours and 35 minutes to complete the node decommission process on the non-SmartStore environment as majority of the time was spent on meeting the search factor and replication factor by copying the data between the surviving nodes. Meanwhile, the same process just took **9 minutes** to complete under the Splunk SmartStore.

Figures 26 and 27 show Bucket Fixups activities between Classical and SmartStore mode.

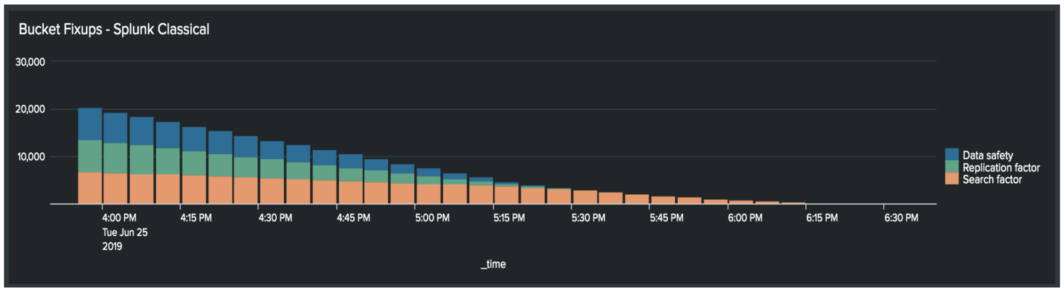


Figure 26: Bucket fixups with Splunk Classical mode

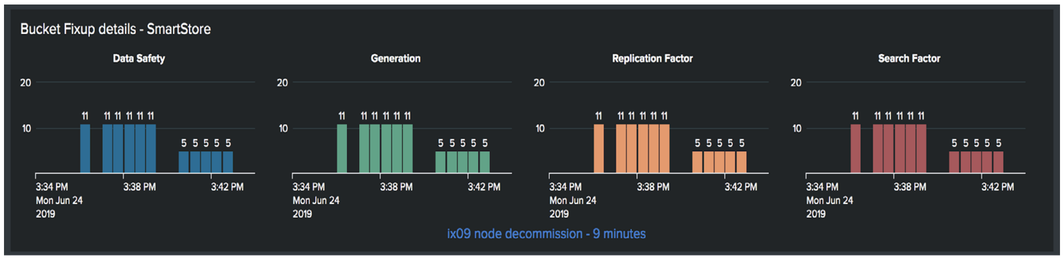


Figure 27: Bucket fixups with SmartStore

Figure 28 shows the comparison of node offline process between SmartStore and non-SmartStore environment with the same dataset.

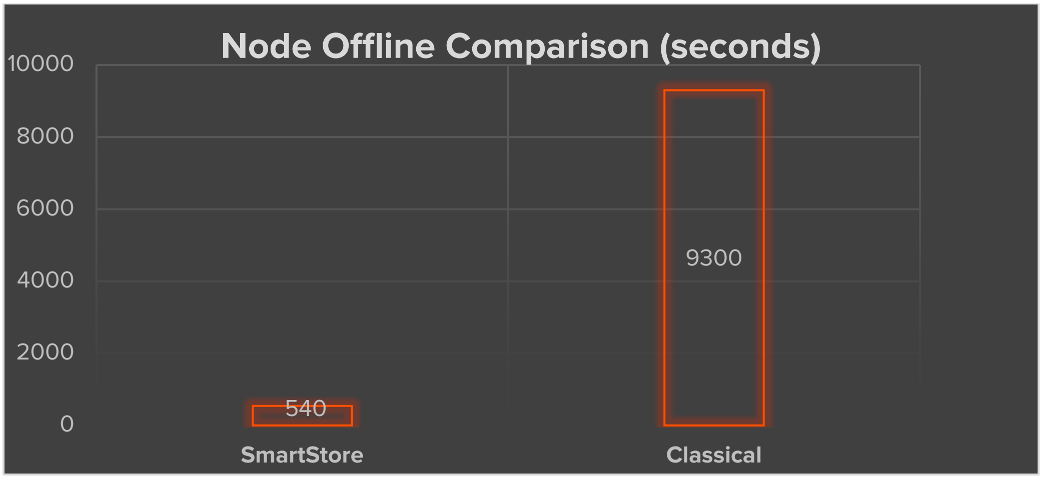


Figure 28: Node offline comparison: SmartStore vs. Classical

In summary, by disaggregating the storage from compute and offloading data to the remote object store, SmartStore provides a very efficient indexer cluster management. The node addition with data rebalance and node removal tests have clearly exhibited advantages in using SmartStore, which can significantly lower the management costs and at the same time enable faster recovery.

Best Practices

Minimum Purity//FB Version to Run Splunk SmartStore: The minimum Purity version for FlashBlade to run Splunk SmartStore is 2.3.0. This includes all the object-related functionalities that are required to host Splunk index data on FlashBlade using S3 protocol.

Remote Volume

The volume definition for the remote storage in indexes.conf points to the remote object store where Splunk SmartStore stores the warm data. The remote volume definition looks like the following.

```
[volume:remote_store]
storageType = remote
path = s3://<bucket name>
remote.s3.access_key = <access key of the account that holds the bucket>
remote.s3.secret_key = <secret key of the account that holds the bucket>
remote.s3.endpoint = http://<FlashBlade-data-vip>
```

Here are some of the key limitations to be aware of when using the remote volumes:

- Each remote volume definition can have only one path meaning a single S3 bucket name.
- The remote volume which refers to the S3 bucket on a FlashBlade should be limited to an indexer cluster or a standalone indexer. The same S3 bucket cannot be shared across two clusters or standalone indexers.
- An indexer cluster or a standalone indexer can have one or more remote volumes.
- A SmartStore index is limited to a single remote volume and cannot be spread across multiple remote volumes.
- All peer nodes of an indexer cluster should use the same SmartStore configurations.

Bucket Size: Splunk has predefined sizes for the bucket that can be configured under the maxDataSize parameter in `indexes.conf as maxDataSize = <positive integer>|auto|auto_high_volume`

The default is "auto" at 750MB whereas auto_high_volume is 10GB on 64-bit systems and 1GB on 32-bit systems.

The general recommendation by Splunk for high volume environment is to set the bucket size to auto_high_volume but for Splunk SmartStore indexes, the specific recommendation is to use "auto" (750MB) or lower. This is to avoid timeouts when downloading big sized buckets from the remote objectstore back to the cache.

Recommended setting:
`maxDataSize = auto`

TSIDX Reduction: SmartStore doesn't support TSIDX reduction. Do not set the parameter enableTsidxReduction to "true" for SmartStore indexes.

The recommended setting is:
`enableTsidxReduction: false`

Bloom Filters: Bloom filters play a key role with SmartStore in reducing the download of tsidx data from the remote object store to the cache. Do not set the parameter createBloomfilter to "false."

The recommended setting is:
`createBloomfilter: true`

Cache Sizing: The cache sizing in the SmartStore environment has a global scope and cannot be specified at an index level. Use the following factors to determine cache sizing.

- **Daily ingest rate:** Daily ingested source data size that counts against the license.
- **Cache Retention:** Search timespan for majority of your searches. Splunk's recommendation is 30 days for Splunk Enterprise and 90 days for Splunk Enterprise Security. Configure this as needed.
- **Replication Factor (RF):** Number of raw data copies to maintain on the indexer cluster, recommended setting is 2.

$$Global\ Cache\ Sizing\ = Daily\ Ingest\ Rate\ *\ RF\ +\ (Cache\ Retention — 1)\ *\ Daily\ Ingest\ Rate)$$

$$Cache\ Sizing\ per\ Indexer\ = Global\ Cache\ Sizing\ /\ Indexers$$

The cache size is set at each indexer level and to set the cache size, update the parameter `max_cache_size` under cachemanager stanza in the `$SPLUNK_HOME/etc/master-apps/_cluster/local/server.conf` file.

```
[cachemanager]
max_cache_size = <positive integer - value in MB>
```

homePath: Requires a path value as this is the location on local storage where hot and cached warm buckets reside.

coldPath: Requires a path value, even though the setting is ignored except in the case of migrated indexes. After migration, the path can be deleted and reconfigured to be the same as the homePath. This would enable the removal of cold tier/storage.

Data availability and fidelity: In SmartStore, the data availability and data fidelity functions are offloaded to the remote object store. This means the ingested data has to be rolled over from hot buckets to warm buckets as soon as possible. The rolling of hot to warm is controlled by size or age. The parameter maxDataSize determines the size of the bucket. When the bucket is full, it rolls from hot to warm. Limiting bucket size to "auto" or 750MB should be sufficient for customers with high volume of data ingestion. Forcing the hot bucket rolling based on other factors can result in smaller buckets which can have negative impacts on the search and performance.

Versioning: FlashBlade supports versioning, which is required by SmartStore to protect against any accidental deletion. Splunk data is generally deleted when it surpasses the configured data retention period. Setting this parameter to **false** on a S3 storage like FlashBlade that supports versioning allows Splunk to put a delete marker on the objects rather than physically deleting them which makes it possible to protect against the accidental deletion. If this parameter is set to true, which is the default setting, all versions of the data is deleted permanently by Splunk SmartStore when it ages out and cannot be recovered.

Recommended setting: `remote.s3.supports_versioning = false #(default is true)`

As the data is not physically removed when data ages out while this parameter is set to false, it is recommended to set a lifecycle policy at the FlashBlade S3 bucket level to physically remove the deleted data and reclaim the space. See Appendix-D for setting the lifecycle policy on a FlashBlade S3 bucket.

List Object Version

FlashBlade supports object listing version V2 which is much more performant than version V1. To improve performance when Splunk is listing objects, V2 is highly recommended.

The recommended setting is: `remote.s3.list_objects_version = v2`

Multi-part upload/download: FlashBlade supports multipart upload and download and the default setting of 128MB should be good enough and it is not recommended to modify this setting unless the new value has been proven to improve throughput.

SmartStore indexes.conf configuration

```
[default]
maxDataSize: auto
# 750 MB to roll from Hot to Warm
remotePath = volume:remote_store/$_index_name
#Enables SmartStore across all indexes when remotePath is defined in the default section

[volume:remote_store]
storageType = remote
path = s3://<bucket name>
# The following S3 settings are required only if you're using the access and secret keys.

remote.s3.access_key = <access key of the account that holds the bucket>
remote.s3.secret_key = <secret key of the account that holds the bucket>
remote.s3.endpoint = http://<FlashBlade-data-vip>
remote.s3.supports_versioning = false
remote.s3.list_objects_version = v2

[index-name]
remotePath = volume:remote_store/$_index_name
homePath = volume:hot/index-name/db
coldPath = volume:cold/index-name/colddb
thawedPath = $SPLUNK_DB/index-name/thaweddb
repFactor = auto
```

See Splunk's documentation for features not supported by SmartStore and current restrictions on SmartStore use.

Conclusion

Splunk is the market leader in Security Information and Event Management (SIEM) and ranked #2 in IT Operations Management. The usage of Splunk has been growing considerably across various organizations that want to add more data sources while retaining the data for a longer period, thus stressing the Splunk infrastructure. Splunk SmartStore on FlashBlade is designed to provide a scalable architecture for organizations to achieve these goals.

The tests clearly demonstrate that by deploying Splunk SmartStore on FlashBlade, organizations can:

- Scale servers and storage independently for improved asset utilization and reduced TCO.
- Improve ingest performance with best-of-breed infrastructure components.
- Achieve significant improvement in the operational efficiencies of indexer cluster management activities without impacting data integrity.
- Accelerate data transfer between the indexer cache and the object tier.
- Increase availability while scaling massively.
- Enhance data reduction through compression while achieving additional security through encryption.

Appendix

Appendix A: Splunk Components

A **bucket** is a file system directory containing a portion of Splunk Enterprise index.

An **index** is a repository for data ingested into Splunk. Splunk manages all its index data in the flat file format on indexer and doesn't use any sophisticated database management systems.

An **indexer** is a Splunk Enterprise instance that indexes the incoming machine data, transforming them into events and storing the results into an index. The other function of indexer is to search the indexed data in response to search requests.

An **indexer cluster** is a group of indexers configured to replicate each other's data, so that the system keeps multiple copies of all data to prevent data loss while enabling data availability for searching in case of an indexer node failure. As the Indexer cluster features automatic failover, in case of an indexer failure Splunk automatically fails over that indexer to the next indexer which enables the incoming data to be indexed, making it available for searching.

Cache manager is part of an indexer that manages the SmartStore cache. The cache manager's goal is to optimize the use of the local storage and also handle the transfer of bucket copies between local and remote storage. It also evicts bucket copies from the local storage as needed to make space for incoming data from the remote storage.

SmartStore is a Splunk Enterprise indexer feature to store indexed data into remote object stores such as FlashBlade or Amazon S3. By disaggregating the storage from the compute, SmartStore allows independent scaling of compute and storage resources separately, thus improving the efficiency of resource usage.

Search head is a Splunk Enterprise instance that handles incoming search requests. In a distributed search environment, the search head sends requests to a group of indexers, aka "search peers," which perform the actual searches on their indexes and send the results back. The search head merges the results and presents them to the user. Dedicated search heads don't have any indexes dedicated to performing search management functions such as consolidate and display.

A **search head cluster** is a group of interchangeable and highly available search heads, aka **cluster members** that share configurations, job scheduling, and search artifacts. By increasing concurrent user capacity and eliminating single points of failure, search head clusters enable highly available and scalable search services.

Forwarder is a small-footprint version of a Splunk instance that forwards data to remote indexers for data processing and storage.

Cluster master or **master node** is another Splunk Enterprise instance that regulates the functioning of an indexer cluster.

Deployer is a Splunk Enterprise instance that distributes apps and other configurations to the search head cluster members. The deployer cannot be run on the same instance as a cluster member.

The monitoring console is the Splunk Enterprise monitoring tool that allows you to view detailed performance information about your Splunk Enterprise deployment through pre-defined dashboards. Some of the available dashboards are indexing performance, index and volume usage, license usage, search performance, search head, and indexer clustering, etc.

The license master controls one or more license slaves and is generally used when you have more than one indexer and want to manage indexer access to purchased license capacity from a central location.

The deployment server is a Splunk Enterprise instance that acts as a centralized configuration manager. It is the tool for distributing configurations, apps, and content updates to groups of Splunk Enterprise instances. This is generally used to update forwarders, non-clustered indexers, and search heads. This is not a required component to manage forwarders and other instances. Based on your preference and/or standard operating procedures, tools like Chef, Puppet, Salt, etc., can be used.

Appendix B: FlashBlade Configuration for SmartStore

To use FlashBlade to hold the Splunk SmartStore data, a bucket within the FlashBlade object store has to be created. This section details the process to create the S3 bucket on the FlashBlade that will hold the Splunk SmartStore index data.

FlashBlade object structure involves the following hierarchy (Figure 29):

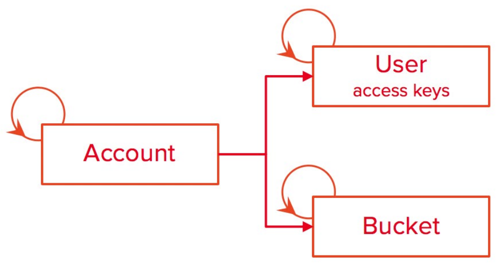
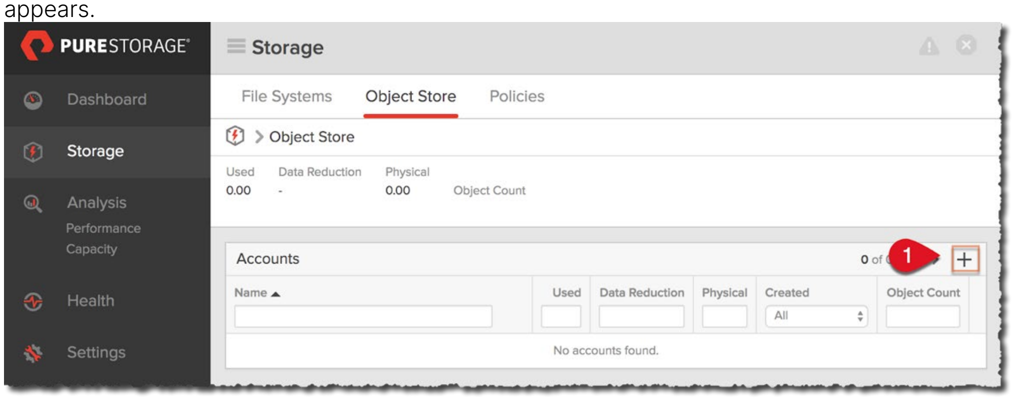


Figure 29: FlashBlade object structure

An object will eventually be stored under a bucket and a bucket is owned by an account. The objects can be accessed from the bucket with a user access key and secret key within that account. FlashBlade object store can have multiple accounts to segregate the objects by the line of business or by the organization. Each account can have one or more users with each user can have one or more access keys.

1. In the Accounts section of the Storage > Object Store page, click the add (+) button. The Create Account pop-up window appears.



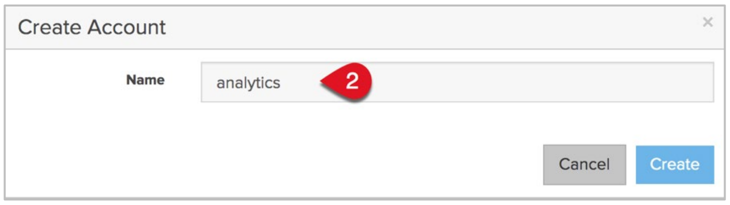
0.00 - 0.00

0 of 0

No accounts found.

Figure 30: Adding accounts

2. Enter the new account's name in the Name field and click Create.



Name: analytics

Figure 31: Adding account name

3. From the Storage > Object Store page, click the account that was created in prior step. In the Users section, click the add (+) button to create a new user.

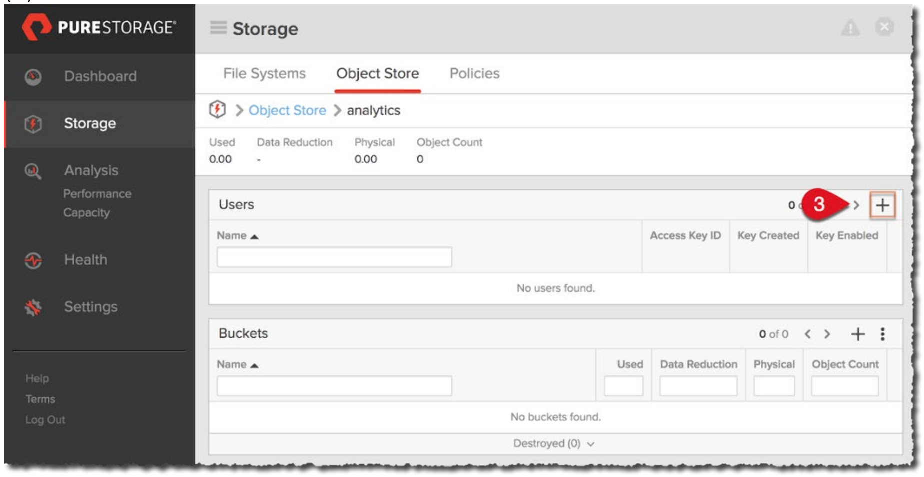
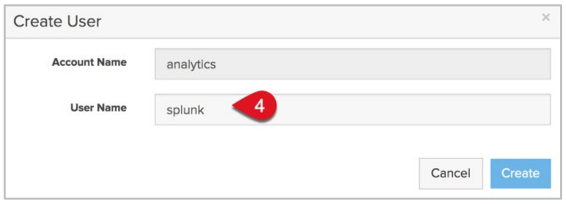


Figure 32: Adding users

4. Enter the new user's name in the User Name field and click Create.


Create User

Account Name: analytics

User Name: splunk

Figure 33: Creating the user name

5. Under the Users section, click the More Options > Create access key button on the same row as the user under whom to create an access key. Make note of the access key and secret key information displayed in the confirmation popup. Optionally use the CSV or JSON button to download the key information.

   Note: You must save or download your key information before closing the popup. The secret key can be accessed only during the key creation confirmation and can't be viewed or accessed after the confirmation popup is closed.

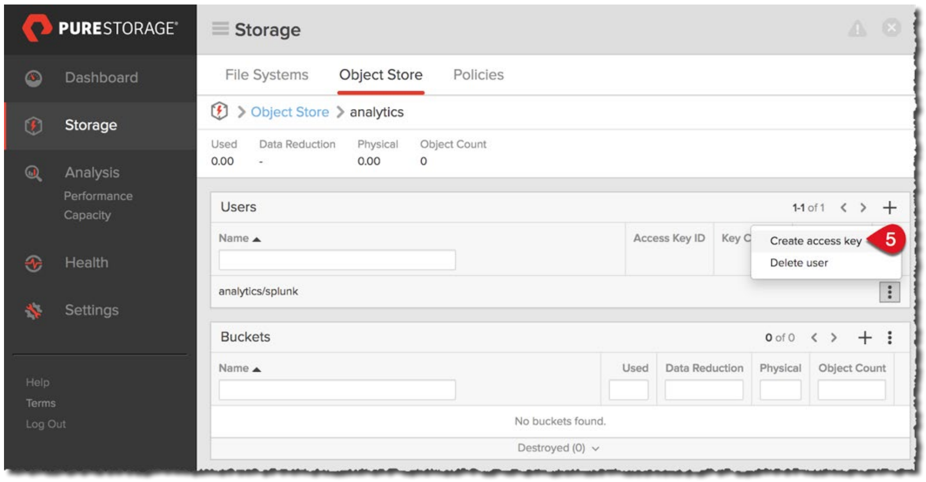
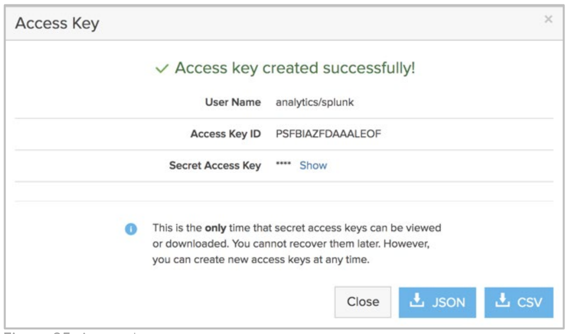


Figure 34: Creating the access key


Access Key

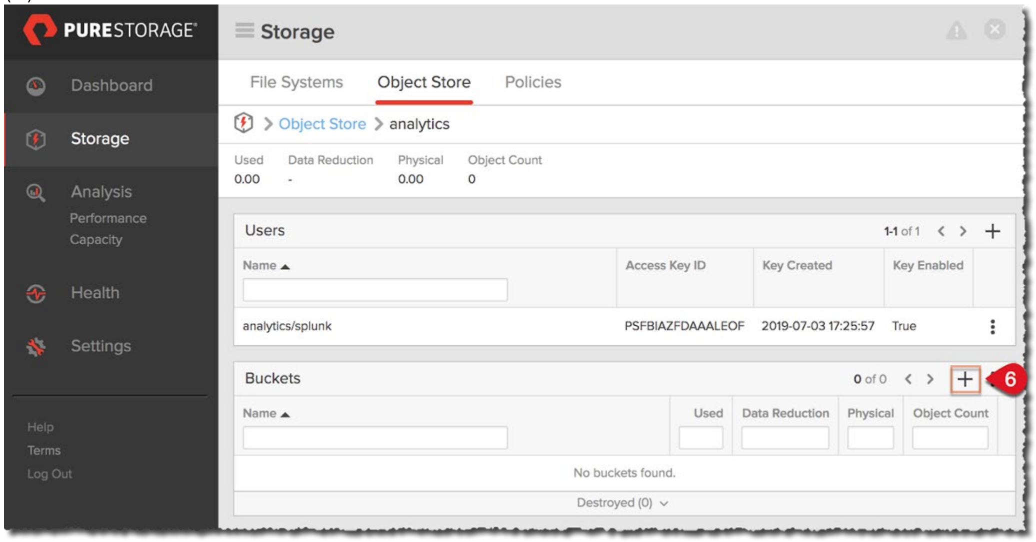
✓ Access key created successfully!

User Name	analytics/splunk
Access Key ID	PSFBIAZFDAAALEOF
Secret Access Key	**** Show

This is the **only** time that secret access keys can be viewed or downloaded. You cannot recover them later. However, you can create new access keys at any time.

Figure 35: Access key

6. From the Storage > Object Store page, click the account that was created earlier. From the Buckets section, click the add (+) button to create a new bucket.



Used	Data Reduction	Physical	Object Count
0.00	-	0.00	0

Users 1-1 of 1

Name ▲	Access Key ID	Key Created	Key Enabled
analytics/splunk	PSFBIAZFDAAALEOF	2019-07-03 17:25:57	True

Buckets 0 of 0

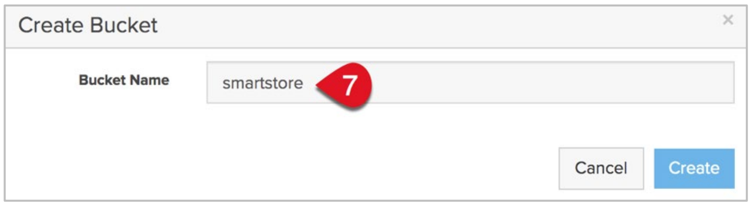
Name ▲	Used	Data Reduction	Physical	Object Count

No buckets found.

Destroyed (0)

Figure 36: Creating a bucket

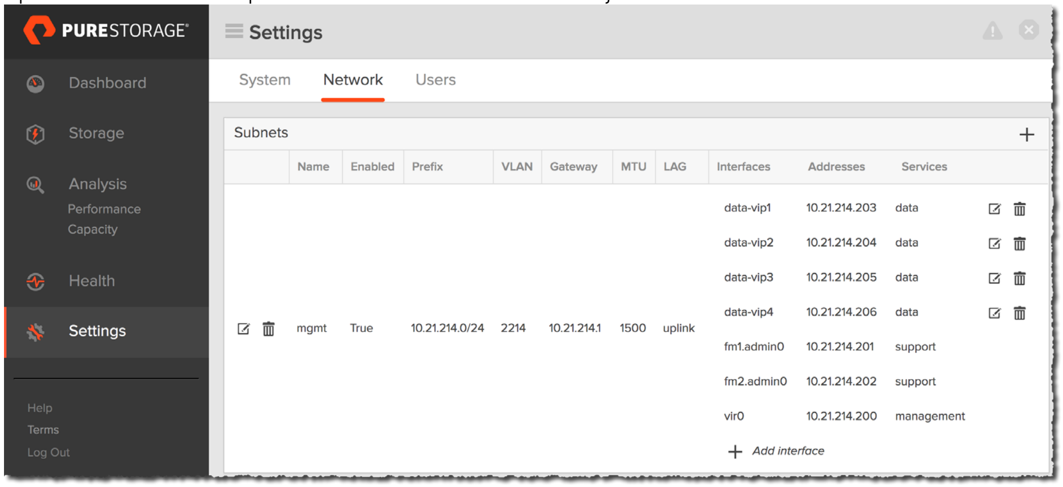
7. Enter a name for the bucket and click the Create button.


Create Bucket

Bucket Name: smartstore

Figure 37: Creating a bucket

8. From the Settings > Network page, select the appropriate network interface from the data services interfaces for the Splunk SmartStore endpoint to connect to the remote object store.



Name	Enabled	Prefix	VLAN	Gateway	MTU	LAG	Interfaces	Addresses	Services
mgmt	True	10.21.214.0/24	2214	10.21.214.1	1500	uplink	data-vip1	10.21.214.203	data
							data-vip2	10.21.214.204	data
							data-vip3	10.21.214.205	data
							data-vip4	10.21.214.206	data
							fm1.admin0	10.21.214.201	support
							fm2.admin0	10.21.214.202	support
							vir0	10.21.214.200	management

Add interface

Figure 38: Selecting a network interface

9. You can now use the bucket created above along with the access key, secret key, and the endpoint to configure the remote store on the indexes.conf file.

```
path = s3://smartstore
remote.s3.access_key = <access key>
remote.s3.secret_key = <secret key>
remote.s3.endpoint = http://10.21.214.203
remote.s3.supports_versioning = false
```

Appendix C: Useful Splunk searches

Hot Tier Space Usage

```
| dbinspect index=<index-name> cached=false | search state=hot | stats sum(sizeOnDiskMB) AS diskTotalinMB
|eval diskinGB = diskTotalinMB/1024 |fields diskinGB
```

Warm Tier Space Usage

```
| dbinspect index=<index-name> cached=false | search state=warm | stats sum(sizeOnDiskMB) AS diskTotalinMB
|eval diskinGB = diskTotalinMB/1024
```

Hot/Warm Tier Space Usage

```
| dbinspect index=* cached=false | stats sum(sizeOnDiskMB) AS diskTotalinMB by state
```

Bucket Details of an Index

```
|dbinspect index=<index-name>|eval start_time=strftime(startEpoch,"%m/%d/%y %H:%M:%S"),
end_time=strftime(endEpoch,"%m/%d/%y %H:%M:%S") |fields start_time, end_time, state, bucketId, eventCount
```

SmartStore Space Usage

```
| rest /services/admin/cacheman | dedup title | search title="*|apache-pure*" |eval Index=substr(title,5,12)
|stats sum(cm:bucket.estimated_size) as InCache by Index|eval Size=round(InCache/1024/1024/1024,2) |fields
Index, Size
```

Summarized Bucket Details by Index That Are Participating in the Search by the Date Range

Note: In our search, we used the date range from 05/20/19 08:00:00 to 05/20/19 20:00:00. Please modify the index names and date range to meet your needs.

```
|dbinspect index=apache-pure* |dedup bucketId| eval start_time=strftime(startEpoch,"%m/%d/%y %H:%M:%S"),
end_time=strftime(endEpoch,"%m/%d/%y %H:%M:%S")
| where ("05/20/19 08:00:00" >= start_time AND "05/20/19 08:00:00" <= end_time) OR ("05/20/19 20:00:00" >=
start_time AND "05/20/19 20:00:00" <= end_time) OR ( start_time >= "05/20/19 08:00:00" AND end_time <=
"05/20/19 20:00:00")|stats sum(eventCount) as ec sum(sizeOnDiskMB) as mb count as Buckets by index |eval
diskinGB=round(mb/1024,2), EventCount=tostring(ec, "commas") |fields index, EventCount, diskinGB, Buckets
```

Appendix D: Bucket Lifecycle Policy

Space Reclamation

As the parameter `remote.s3.supports_versioning` is set to false, the data is not physically removed when data ages out. Hence it is recommended to set a lifecycle policy at the FlashBlade S3 bucket level to physically remove the deleted data and reclaim the space.

Note: As of Purity//FB 3.0, the lifecycle policy can only be set through python code and not through the GUI.

The option to enable lifecycle policy through GUI is planned for a future Purity//FB release. Use the sample python code to set the lifecycle policy of a given bucket in a FlashBlade. This code will remove all noncurrent versions of the objects (deleted or overwritten objects), say after 7 days. You'll need to update the **NoncurrentDays** value per your requirements.

```
import boto3
s3 = boto3.resource(service_name='s3', use_ssl=False,
      aws_access_key_id='<access_key>',
      aws_secret_access_key='secret_key',
      endpoint_url='http://<FB data-vip>')

s3.meta.client.put_bucket_lifecycle_configuration (
  Bucket='<S3 bucket-name>',
  LifecycleConfiguration={
    'Rules': [
      { 'ID' : 'rule1',
        'Filter' : { 'Prefix' : '/', },
        'Status' : 'Enabled',
        'NoncurrentVersionExpiration': { 'NoncurrentDays': 7 },
      }
     ]
    }
   )
```