PURESTORAGE®
Uncomplicate Data Storage, Forever

TECHNICAL WHITE PAPER

Splunk SmartStore on Kubernetes with Portworx and Pure Storage FlashBlade

Improve server utilization and enhance Splunk performance with containerized architecture from Pure Storage.

Contents

Executive Summary

Data is driving business growth, compelling organizations across industries to adopt data-driven practices at an increasing rate. Over the past decade, businesses have harnessed the power of "big data" to unlock new possibilities and enhance their analytical capabilities. Today, those businesses must accelerate their capabilities by moving beyond experimentation with analytics toward mature investments and capabilities—or risk losing a competitive edge.

Splunk Enterprise enables organizations to make meaningful decisions quickly, by helping them unify their data from various sources, analyze it to drive business resilience, and then visualize it to unlock new and valuable insights. While Splunk can turn data into action, the significant data growth still poses numerous challenges like the management of data, underutilization of the infrastructure components and sub-par search performance. The increased data usage in the traditional Splunk architecture with data replicated across multiple indexer nodes not only increases the storage usage but also makes it arduous to manage. Additionally, with the distributed scale-out architecture model where compute and storage are bundled together, the higher data usage requires additional servers, which in turn leads to underutilized servers. Increased data also severely affects the search performance when the data is tiered into different storage media, generally done to curb the storage cost.

With the advent of Splunk SmartStore, an application feature within Splunk Enterprise, the storage is disaggregated from the compute and offloaded to a S3 compatible storage like Pure Storage® FlashBlade®. This enables independent compute and storage growth while lowering the storage costs and significantly improving operational efficiencies.

The Splunk Operator for Kubernetes, along with the Splunk SmartStore, offers new architectural options for containers, along with a data services platform for persistent storage and S3 compatible object storage. These new cloud-native technologies, running on the latest bare-metal servers along with Pure Storage FlashBlade as the object storage and Portworx®, a leading Kubernetes data services platform, can help improve application performance, optimize server and storage utilization and simplify the management of the cloud native environments.

In our tests running Splunk SmartStore on Kubernetes with Pure Storage FlashBlade and Portworx, a four-node cluster performed three times better than bare-metal servers when ingesting data and twice as better in running dense and sparse searches while maximizing the utilization by a factor of 1.6 times.

This reference architecture will help you further understand the suggested architecture, test cases, test results, and solution benefits.

Audience

The target audience for this reference architecture includes Splunk administrators, system and storage administrators, IT managers, system architects, sales engineers, field consultants, and professional service engineers. A working knowledge of Splunk and Linux and an understanding of server, storage, containers, and networking is assumed, but is not a prerequisite to read this document.

Objective

The purpose of this reference architecture is to show how to design a solution for Splunk SmartStore on Kubernetes with FlashBlade and Portworx as well as demonstrate the benefits of using the solution by highlighting the test results.

Business Challenge

According to the latest estimates based on the findings, 328.77 million terabytes or 0.33 zettabytes of data are created every day. The burgeoning data growth within the Organizations are putting a lot of stress on the systems that manage the data. Various analytical applications including Splunk's deployment model was based on a distributed scale-out model which provides high data availability through multiple copies of data and fidelity through security and hashing but presents various challenges as the data volumes grow.

The Increasing cost of Infrastructure

In the distributed scale-out model, the data is collocated within the indexer, generally on a locally attached storage. For high availability of data, it is replicated with one or more indexers within the cluster based on the replication factor (RF). The indexer cluster can sustain RF-1 indexer failures. For example, with a replication factor of 2, the cluster can tolerate just one node failure, while with a replication factor of 3, the cluster can tolerate two concurrent node failures. Unfortunately, the downside of this is the need for additional storage to hold these replica copies. The distributed scale-out model uses similarly-configured servers with pre-defined storage capacity; adding more storage capacity means adding additional servers, even if there is no need for the additional compute resources. Similarly, adding more servers for compute resources comes along with the additional storage capacity albeit there was no need for additional storage.

This dependent scaling of compute and storage becomes cost-prohibitive very quickly when data usage increases. This also leaves the servers under-utilized at either the compute or at storage layers.

Operational Overhead

With multiple copies of data (based on the replication factor in the distributed scale-out model), any indexer cluster expansion, hardware refresh, or imbalanced disk usage across indexers requires a data rebalance. Data rebalancing balances the storage distribution across all the indexer nodes. During data rebalancing, Splunk physically moves the data between the indexer nodes and, based on the amount of data, this can be a slow and lengthy process. To reduce the impact of rebalancing data, the storage capacity on the server is reduced, which limits the amount of data to be distributed. Unfortunately, this increases the number of servers needed to host the overall required storage that the index cluster needs.

In addition to the above, an indexer failure will force a reconstruct of the data on the surviving indexer nodes to meet the replication factor requirement, which is again a physical data movement that is not only resource-intensive but also takes away some of the CPU cycles from Splunk.

Underutilized Compute Resources

The bare-metal servers that are added to the indexer cluster to meet the storage requirements result in most of the compute resources being underutilized. This is an expensive proposition and adds a significant overhead in maintaining the servers as well as patching and upgrading the software components. Also, not all analytical applications, including Splunk, take advantage of the available cores/threads for functions like ingest when running a single instance of the application. This is reaffirmed by the sizing recommendations given by Splunk where they suggest an indexer that meets the minimum reference hardware requirements can ingest up to 300GB/day while supporting other functions like searches.

Lack of Faster Insights

Increased data volumes not only cause data management challenges, but also create performance degradation as the data ages. As it ages, data is moved to cheaper and lower-performance storage tiers as it is very expensive to hold them in faster, high-performing storage along with the replica copies. This tiered approach quickly becomes impractical when responding to historical searches related to regulatory or compliance requirements, cybersecurity, and legal discovery, all of which demand information beyond the most recent data be readily accessible.

The Solution

The fundamental issue with the distributed scale-out model is the collocation of the storage with the server, hence the primary solution is to disaggregate the storage from the server. The disaggregated storage has to be in a centralized location accessible to all the Splunk indexers. It should also provide high data availability so there is no need to maintain multiple copies of data, thus reducing the storage requirements.

To maximize the utilization of the compute resources, multiple instances of the applications should be deployed on the same bare-metal server either through virtualization or containerization. Running numerous application instances through containers on the same bare-metal servers can take better advantage of all the available CPU resources than virtualization. Containers have moved into the mainstream after years of experimentation and Enterprises are adapting them at a faster pace. Adopting a container orchestration like Kubernetes would be more effective in managing multi-container workloads with automation of deployment and scalability.

Disaggregated Storage

Splunk helps address data management and enterprise security challenges. It enables the search, analysis, and visualization of the machine data from IT infrastructure and business applications, as well as delivery of insights and business value to customers.

Splunk Enterprise introduced SmartStore, which uses the remote storage tier and a cache manager. These allow data to reside either locally on indexers or on the remote storage tier. The data movement between the indexer and the remote storage tier is managed by the cache manager, which resides on the indexer. The supported remote storage services include cloud storage like AWS S3, GCP, and Azure Blob and any S3-compatible object storages like Pure Storage FlashBlade on premises. With the introduction of SmartStore, whenever hot buckets are rolled into warm, they are uploaded into the remote storage, thus eliminating the cold tier.

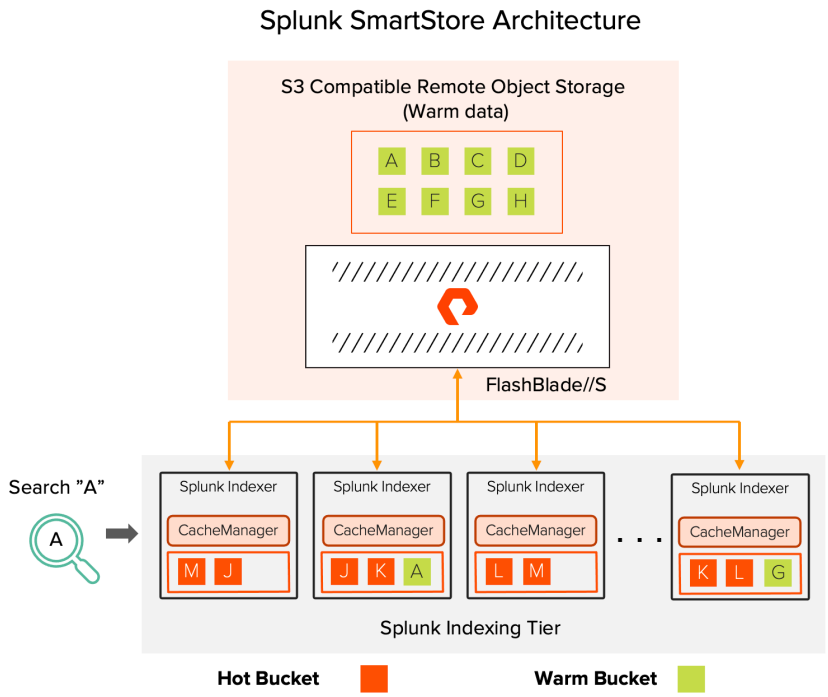

Figure 1: Splunk SmartStore Architecture

SmartStore also introduced an application-aware cache that automatically evaluates the data age, priority, and user's data access patterns to determine which data needs to be in the cache to fulfill any real-time analytics. SmartStore reduces the storage required by keeping only one copy on the remote storage, effectively enabling the customers to grow volumes of data to be retained for a longer period while keeping the data searchable. Splunk SmartStore with FlashBlade as the high-performance remote object store accelerates historical search performance while reducing the space usage by in-line compression and protecting the data at rest through encryption.

By disaggregating the storage from compute, any operational activities like indexer node addition, removal, failures that require a data rebalance are now limited to metadata updates between the nodes eliminating the expensive and time-consuming physical movement of data between the nodes. Importantly, the disaggregation allows independent scaling of compute and storage based purely on the respective resource's demand.

Application Containerization

Containerization is the next evolution of server virtualization. Containers encapsulate an application with all its dependencies, including system libraries, binaries, and configuration files. They differ from virtual machines by not having their own operating systems and share the existing OS provided by the host system. Containers are extremely lightweight as they don't bundle an extra OS along with the application and hence can be launched very fast and scaled instantaneously.

Containerization of applications brings many benefits, including the following:

- **Scalability:** Through containerization, container instances like Splunk indexers and search heads can be added quickly and efficiently to handle increased application load.
- **Portability:** The container image can be deployed on any platform that provides access to the same base OS, giving the flexibility throughout the lifecycle of the application. Once developed, the containers can be run anywhere: on-prem or on various platforms and clouds.
- **Resource efficiency:** Containers use far fewer resources than virtual machines and deliver higher utilization of compute resources.
- **Lightweight:** Containers hold only the dependencies needed for the application and don't require a separate OS, so they can start up much more quickly than virtual machines and in any host environment.
- **Agility:** Containers accelerate application development and deployment by streamlining DevOps CI/CD.
- **Simplified management:** Easier management since install, upgrade, and rollback processes are built into the container orchestrators like Kubernetes's platform.

Kubernetes with Portworx

Kubernetes is a portable, extensible, open-source container orchestration platform for managing containerized applications such as Splunk. Containers enable applications to be broken into smaller, independent pieces and can be deployed and managed dynamically—rather than as a monolithic stack—with predictable application performance and resource utilization. Kubernetes provides a framework to run these containers on distributed systems resiliently and enables horizontal scaling.

Kubernetes offers portability and flexibility and works with virtually any type of container runtime. It is the market leader in the container orchestrator arena and the adoption has continuously grown in recent years. According to industry analyst firm Gartner, "by 2027, more than 90% of global organizations will be running containerized applications in production, which is a significant increase from fewer than 40% in 2021."[1]

The Splunk Operator for Kubernetes simplifies the deployment of Splunk Enterprise in a cloud-native environment that uses containers. The Operator simplifies the scaling and management of Splunk Enterprise by automating administrative workflows using Kubernetes best practices.

One of the limitations of containers is the lack of built-in persistent storage. Any stateful services that run on containers require storage to persist the state across container restarts. The majority of the analytics applications maintain their state and hence applications like Splunk require persistent storage to host their configuration and runtime state.

Portworx by Pure Storage is the leading Kubernetes data services platform and provides a fully-integrated solution for persistent storage, data protection, disaster recovery, data security, cross-cloud and data migrations, and automated capacity management for applications like Splunk running on Kubernetes. Designed specifically for cloud-native applications, Portworx delivers the performance, reliability, and security you require from traditional enterprise storage but is built from the ground-up for Kubernetes. Portworx is the most complete Kubernetes data services platform. GigaOm called Portworx "the gold standard for cloud-native Kubernetes storage for the enterprise." With Portworx, you can run modern analytics like Splunk in production at scale with all of the enterprise capabilities mentioned above.

[1] Gartner, CTO's Guide to Containers and Kubernetes - Answering the Top 10 FAQs,, Arun Chandrasekaran, 31 May 2022, ID G00763328

Solution Architecture

Our solution was designed to take advantage of the Splunk SmartStore feature implemented on the containers to maximize the utilization of the compute resources along with Portworx for maintaining statefulness while accelerating the ingest and search performance by hosting the data on the S3-compatible Pure Storage FlashBlade.

Here are the key software and hardware components and technologies included in this reference architecture:

- Splunk Enterprise with Splunk SmartStore
- Splunk Operator for Kubernetes
- Pure Storage FlashBlade
- Portworx by Pure Storage
- Six Bare-metal servers (four servers were dedicated for indexers)

The figure below shows the overall technology stack for running Splunk SmartStore on Kubernetes with Pure Storage products. Pure Storage developed FlashBlade architecture to meet the storage needs of data-driven businesses. FlashBlade is an all-flash scale-out system, primarily optimized for storing and processing unstructured data. The integration of scalable NVRAM into each blade helps scale performance and capacity proportionally when new blades are added to a system.

The solution is complemented by Portworx by Pure Storage to run stateful applications like Splunk Enterprise without worrying about the operational overhead of Splunk server failure or managing the storage space to accommodate the massive data growth. Portworx supports various backing stores, from directly-attached NVMe or SSD drives to on-prem SANs such as Pure Storage FlashArray™ to cloud-based block storage such as Amazon EBS or Azure Disk. Portworx provides the same abstraction and benefits for Splunk on Kubernetes architecture irrespective of where the solution is deployed, either on the cloud or on-prem.

The solution included six PowerEdge R740xd servers with local NVMe drives to run the Splunk Enterprise with four servers dedicated to host the indexer functionality. The bare-metal configuration used for comparison comprises one Cluster Manager, one Search Head and four Indexers on those six servers. The Pure Storage FlashBlade was used as the remote object storage, to host the warm data from Splunk SmartStore.

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

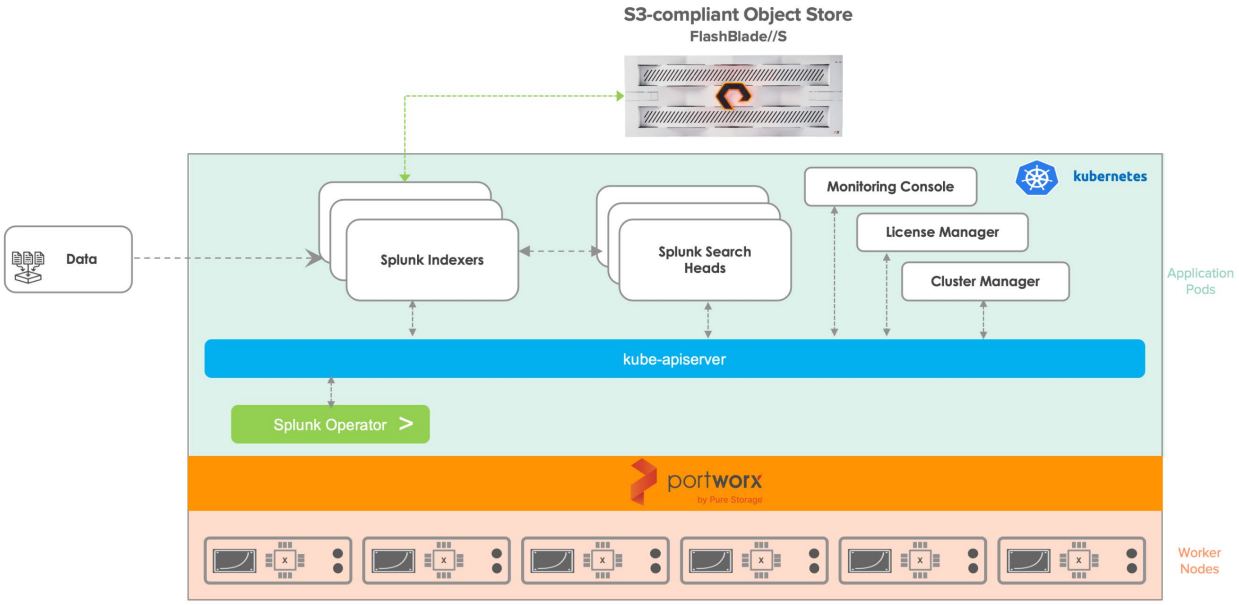

Figure 2: Solution architecture for Splunk SmartStore on Kubernetes with Portworx and Pure Storage FlashBlade

Overall, this reference architecture uses high performance hardware and cloud computing software to help increase performance, improve hardware utilization, and operational efficiency.

Reference Architecture Environment Overview

The reference architecture environment consisted of six bare-metal servers running Ubuntu 20.04.6. The servers were equipped with two Intel Xeon Platinum 8260 @2.4 GHz with 24 cores or 48 cores in total, 256GB of RAM, two 100GbE network interfaces and four 1.5 TB local NVMe drives.

For the bare-metal server tests, four servers were dedicated to host four indexers, with one indexer per server while cluster manager and a search head were hosted on the remaining two servers. The same servers were used as the worker nodes in the Kubernetes environment with four servers were configured to host the indexers alone through anti-affinity rules.

Kubernetes can be deployed through multiple ways like bootstrapping clusters with kubeadm or through deployment tools like kubespray, kops etc. We used kubespray to deploy the Kubernetes cluster but feel free to follow your organization's preferred choice for deploying Kubernetes. Note, this document will not show how to deploy the Kubernetes cluster but have included relevant links in the References section.

The following sections show the configuration of each component used in our tests.

> **Note**: They do not represent any maximum available configuration of that component.

System Configuration Details

Server Configuration

Component	Physical Servers
CPU	2 x 24 Cores (Intel Xeon 8260 @ 2.4GHz)
Memory	256GB
Network	2 x 100GbE
Portworx Backing Disk(s)	4 x 1.5 TB Local NVMe drives, FlashArray attached volumes

Software Configuration

Component	Physical Servers
Operating System	Ubuntu 20.04.6
Kubernetes	1.24.6
Portworx	2.13.5
Splunk Enterprise	9.0.3
Splunk Operator for Kubernetes	2.2.1
Kubespray	2.20

Pure Storage FlashBlade Configuration

The S3-compliant FlashBlade is the remote object storage that stores Splunk's warm data, compressed and encrypted at the storage level.

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

Component	Physical Servers
FlashBlade	7 blades with 2 DFM each @24TB
Capacity	329TB raw
Connectivity	4 x 100Gb/s Ethernet (data) 2 x 1Gb/s redundant Ethernet (management port)
Physical	5U
Software	Purity//FB 4.0.0

Portworx Enterprise Configuration

Portworx Enterprise 2.13.5 was deployed on-prem. The Portworx cluster used the spec generator tool located at https://central.portworx.com to generate the installation resources.

The Splunk instances require two volumes: *etc* to host the Splunk Enterprise and application configs and *var* to host the event and runtime data that includes the indexes. In case of the indexers, the var volume hosts the cache data. Portworx was used to provision the etc and var volumes for all the Splunk instances using the storage class resources.

The etc volume for all the Splunk instances was hosted on a FlashArray system which was setup as the backing store within Portworx. The FlashArray also hosted the var volume of all the Splunk instances except the indexers. For the var volume of the indexers, the local NVMe drives were used as the backing store within Portworx.

Note: The etc volumes can be provisioned on any backend store on the Portworx and FlashArray is not a requirement.

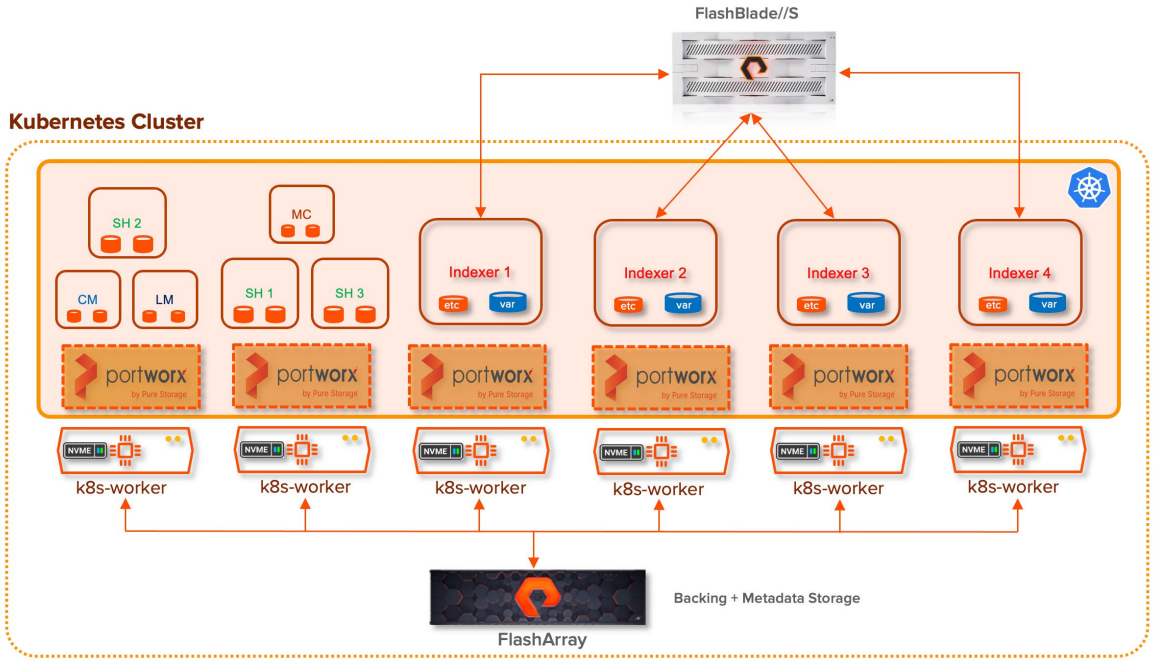

Figure 3: Portworx configuration with FlashBlade

The Portworx Spec Generator tool was configured to use the Portworx Operator to deploy Portworx with the built-in etcd cluster to maintain the metadata. This reference architecture took advantage of the new functionality within Portworx PX-Fast. PX-Fast enables better utilization of high-performance storage such as NVMe devices for persistent storage with Kubernetes. PX-Fast uses the new foundational storage layer in the Portworx Enterprise StorageCluster named PX-StoreV2. For PX-StoreV2, a pre-provisioned metadata path is required on all the Portworx storage nodes with space greater than or equal to 64GB. FlashArray volumes were created and attached to the worker nodes to host the metadata. The following figure illustrates the configuration used in our environment.

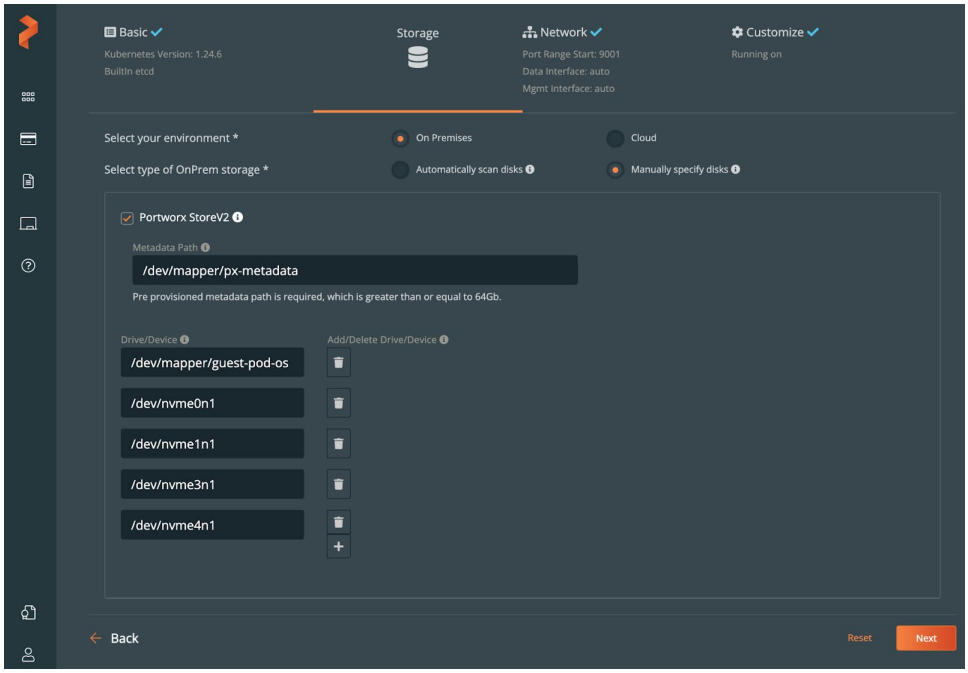

Figure 4: Portworx Enterprise spec generator

Splunk Configuration on Kubernetes

Splunk Operator and Splunk Enterprise on Kubernetes can be installed through yaml files or helm charts. Please see the Splunk documentation for more installation information using the helm charts.

For this reference architecture, the Splunk Operator was deployed by running the following command:

```
kubectl apply -f https://github.com/splunk/splunk-operator/releases/download/2.2.1/splunk-operator-cluster.yaml
```

The indexer cluster used in this reference architecture was deployed through a Helm chart with the configuration values included in a file as below:

```
helm install -f values.yaml splunk-enterprise-k8s splunk/splunk-enterprise -n splunk-operator
```

See Appendix A for the configuration values (values.yaml) used in this reference architecture.

Prior to deploying the Splunk Operator and the Splunk Enterprise, numerous prerequisites that are listed in the following section were performed.

StorageClass

A StorageClass allows Kubernetes operators to define the type of storage offerings available within a Kubernetes cluster. For Splunk on Portworx, two different storage class resources were created using Portworx as the provider. The first StorageClass provides high availability with two replicas. The etc volume for all the instances and the var volume for all instances except the indexers were provisioned through this StorageClass.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-splunk-vol
provisioner: kubernetes.io/portworx-volume
allowVolumeExpansion: true
parameters:
  repl: "2"
allowedTopologies:
- matchLabelExpressions:
  - key: medium
    values:
    - STORAGE_MEDIUM_SSD
```

The above StorageClass was made as the default that would make it easier to provision the persistent storage for the Splunk operator deployment.

The second StorageClass uses the fastpath parameter to take advantage of the PX-Fast with a single replica. The var volume for all the indexers were provisioned through this StorageClass.

> Note: At this time, the PX-Fast is available only for single-replica volumes within Portworx. Multi-replica support will be available in a future release soon.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-fast-nvme
provisioner: kubernetes.io/portworx-volume
allowVolumeExpansion: true
parameters:
  repl: "1"
```

```
    fastpath: "true"
  allowedTopologies:
  - matchLabelExpressions:
    - key: medium
      values:
      - STORAGE_MEDIUM_NVME
```

ConfigMaps

A ConfigMap is a Kubernetes feature that allows users to inject configuration data into application pods. This enables application portability by decoupling environment-specific configuration from the application container images.

The recommended option to provide the Splunk license file to the License Manager is through a ConfigMap which will be referred to in the LicenseManager pod creation through volumes and licenseUrl configuration parameters.

The below command creates a ConfigMap named splunk-licenses that includes a license file named enterprise.lic.

```
kubectl create configmap splunk-licenses --from-file=enterprise.lic -n splunk-operator
```

Secrets

A secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Secret enables the use of confidential data in the application without including them in a pod specification or in a container image.

For this reference architecture, the Splunk admin password, the indexer cluster, and search head cluster secrets were included in a generic secret resource created manually. This makes it easier to login to the Web UI without having to decode the default secret object created by Splunk.

```
kubectl create secret generic splunk-splunk-operator-secret --from-literal='hec_token=<token>' --
from-literal='idxc_secret=<secret> --from-literal='password=<password>' --from-
literal='pass4SymmKey=<secret>' --from-literal='shc_secret=<secret>' -n splunk-operator
```

Similarly, the access key and secret key of the S3-compliant object store for SmartStore were stored in a Secret resource. This is referred to in the ClusterManager pod creation through the secretRef configuration parameter under the remote_store volume definition under the smartstore section.

```
AK=xxxxxxxx
SK=xxxxxxxxx
kubectl create secret generic s3-secret --from-literal=s3_access_key=$AK --from-
literal='s3_secret_key=$SK -n splunk-operator
```

Labels

The nodes were labeled to limit the Splunk instances from running across all the available worker nodes. As the goal is to compare the performance of the indexers in bare-metal servers against the same running on containers within the same set of servers, the servers were labeled as follows:

```
kubectl label nodes k8s-worker01 k8s-worker02 servertype=nonidx
kubectl label nodes k8s-worker03 k8s-worker04 k8s-worker05 k8s-worker06 servertype=indexers
```

The two servers which would host the `ClusterManager`, `LicenseManager`, `MonitoringConsole`, `SearchHeads` were labeled as `servertype=nonidx` while all the indexer nodes were labeled as `servertype=indexers`. The label was used in the `nodeSelectorTerms` to limit the pods to nodes with specific labels.

Test Dataset

To simulate real-world scenarios for testing the ingest and searches, apache logs were generated through a custom-built utility with varying details including keywords to search events at every 1, 10, 100, 1K, 10K, 1M, 10M levels. The keyword placed at the numerical sequence helps to perform rare and very rare searches. The data generation tool can be found at this link.

Solution Validation and Test Results

The following tests were performed as part of the Solution validation to ensure it indeed addresses the challenges that were discussed in the earlier section.

To show the improvement, these tests were initially performed on a Splunk Enterprise cluster running on bare-metal environment and later Splunk Enterprise on the Kubernetes environment with PX. In both cases, the same FlashBlade was used as the remote object storage.

- Ingest tests
- Search tests
- Indexer failure test

The above tests were initially performed on the bare-metal deployment with four indexers. In the case of containers, the tests were repeated by scaling the indexers to 4, 8, 16, and 32 and limiting those indexers to the same four bare-metal servers.

Tests Overview

The following sections describe the test details and the metrics captured.

Ingest Test Overview

For the ingest test, 18TB of data at 1 TB/day was pre-generated using the custom-built data generator utility. The 1 TB/day was further divided into a number of sub-sets equaling the number of indexers with each datafile at a 16GB denomination. For example, in the case of bare-metal servers where there were four indexers, the 1 TB dataset was split into 4 sets of 256 GB with each set containing 16 x 16GB files which were ingested into Splunk. Similarly in the case of Kubernetes with 32 indexers, 1 TB dataset was split into 32 sets of 32 GB with each set containing 2 x 16GB files.

The per day data of 1TB was ingested into one of the SmartStore indexes named *pure-apache*, *pure-apache1*, *pure-apache2*, and *pure-apache3* using the oneshot method. The ingest time was captured across all the tests along with the CPU utilization of the indexers to compare them between the bare-metal vs containers deployment.

Search Test Overview

The search tests were performed by running concurrent searches that went across the four SmartStore indexes, pure-apache, pure-apache1, pure-apache2, and pure-apache4 at varying intervals. Half of the searches were dense search, which are generally CPU intensive, with each one of them accessing over multiple millions of events. The other half of the searches were sparse search, which are generally IO intensive, with a specific keyword like every1m or every10m that is searching for 1 event in every 1 million or 1 event in every 10 million.

The cache size was adjusted to keep only four days of data in the cache whenever the indexer count was changed. For example, with four indexers, the cache size was set to 512GB which can host four 512GB or 2TB of cached data across the deployment equaling four days' worth of data. In the case of 32 indexers, the cache size was set to 64GB, which again can host 32 64GB or 2TB of cached data across the deployment. The searches were not limited to for days but six days, meaning one third of the searches won't find the data in the cache and must download the data from the remote object store.

The elapsed time of the searches and the CPU utilization of the searches were captured across all the tests. The search tests were performed standalone as well as with the ingest which would require higher read and write bandwidth out of the remote object store and would consume more CPU utilization.

Indexer node Failure tests overview

An indexer node is failed in both bare-metal and container deployment. The indexer node failure causes bucket fixup tasks like replication factor, search factor, generation etc., to get the indexer cluster back to complete and valid state. The bucket fixup and the time was measured to show the impact of an indexer failure. In the case of containers, the *kubectl delete pod* command was used to force an indexer node failure.

Test Results

Ingest Tests

The initial test was performed on the bare-metal servers with four indexers. To ingest 1TB, it took 5454 seconds at an average throughput of 191.21 MBps. Next, we deployed Splunk Enterprise on Kubernetes using the Splunk Operator and started with 4 indexer pods and gradually scaled them to 8, 16, and 32 indexer pods on the same four bare-metal servers. As the objective was to measure the ingest performance there were no other activities like searches were performed during the ingest test.

The following graph illustrates the Splunk ingest behavior on the Kubernetes at 4, 8, 16 and 32 indexer pods. Except for the four indexer pods on Kubernetes, the performance of Splunk ingest was better when the indexer instances were increased. The reason for the lower performance on four indexers on Kubernetes is the use of compute resources settings at the Splunk Operator. As part of the Splunk Operator deployment, we sized the CPU resources for an indexer pod at 12 and memory at 24GB. Hence the four indexer pods on Kubernetes were using way lower resources than that of the bare-metal setup.

The ingest throughput with 32 indexers on the four bare-metal servers reached the peak of 625MBps.

> Note: The throughput numbers listed in the graph is the average ingest throughput for the systems and not the maximum possible throughput. Hence the mileage can vary based on the underlying server configuration and the CPU and memory resources configuration at the Splunk level.

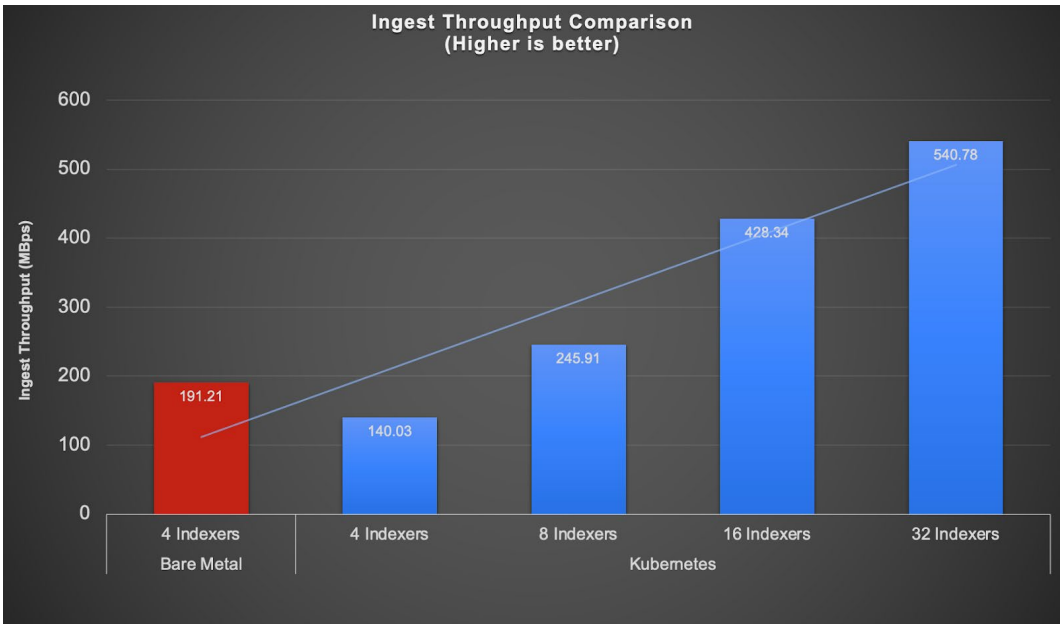

Figure 5 : Splunk Ingest throughput comparison between indexers on bare-metal and indexers on Kubernetes

The following graph shows the CPU utilization of the servers hosting the indexers. The CPU utilization went up almost four times when the number of indexer instances went up eight times from four indexers on the bare-metal servers to 32 indexer pods on the same bare-metal servers. With the increased CPU utilization, the ingest performance also went up almost three times from 191.21MBps to 540.78MBps.

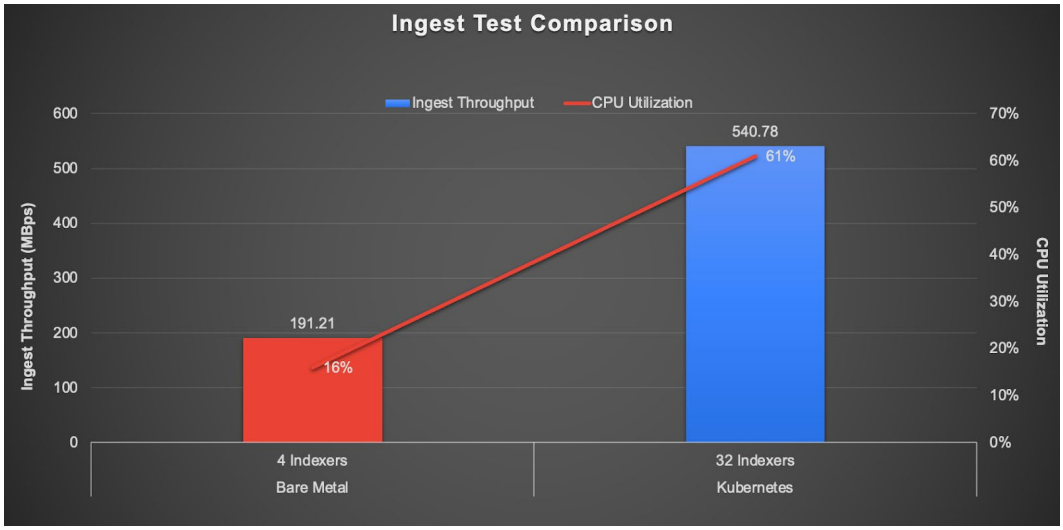

Figure 6: Ingest throughput comparison between indexers on four bare-metal servers and 32 indexer pods on Kubernetes on the same four bare-metal servers

Search Tests

Two types of search tests were performed. One with the ingest to mimic the real-world scenario where both ingest and searches happen all the time and the other with just the search tests alone. In each scenario, the searches were run for one hour while the ingest was in progress. The following graph shows the comparison of the search tests between the bare-metal vs kubernetes deployment when it was performed along with the ingest.

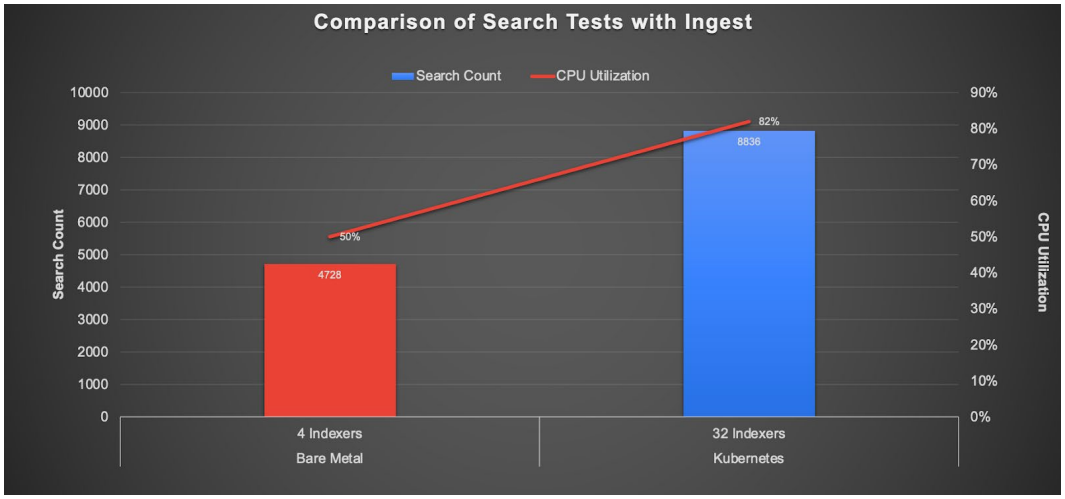

Figure 7: Comparison of searches along with ingest when indexers on bare-metal vs Kubernetes

In the case of bare-metal servers hosting the indexers, there were a total of 4728 successful searches (2364 dense and 2364 sparse searches) that ran for one hour or 79 searches/minute. In comparison, with 32 indexer pods on Kubernetes, the number of successful searches went up to 8836 searches (4418 dense and 4418 sparse) in one hour or 147 searches/minute. During this test, the CPU utilization of the servers went all the way up to 82% on Kubernetes in comparison to 50% when Splunk was run on bare-metal servers.

With more instances of the Splunk indexer pods on Kubernetes, the successful searches per minute almost doubled while the CPU utilization was maximized.

In the second scenario, the search tests were performed standalone for one hour. On bare-metal servers, there were a total of 7344 successful searches that ran in an hour or 122 searches/minute. With 32 indexer pods on Kubernetes, it went up to 13890 searches in one hour or 231 searches/minute, almost two times improvement with increased number of Splunk instances. Interestingly, the CPU utilization went down to 25% when run on 32 indexers on Kubernetes as opposed to 30% when run on 4 indexers on bare-metal servers.

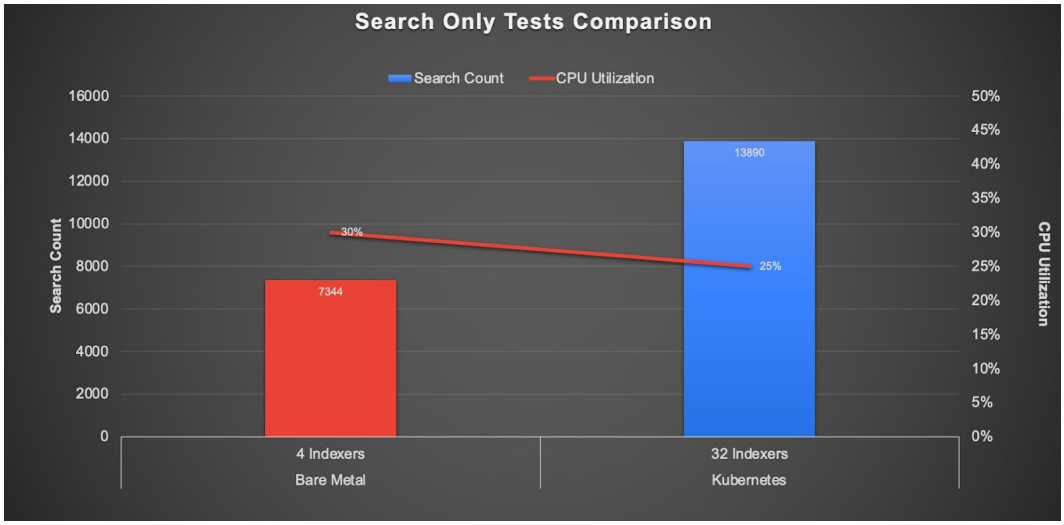

Figure 8: Comparison of Search only test when indexers on bare-metal vs Kubernetes

In both scenarios with the search tests, there was a clear indication of improved searches and maximized CPU utilization when the indexer instances were scaled up on Kubernetes, thus addressing the challenges of underutilized resources and improved performance.

Indexer Failure Test

For the indexer failure test, the indexer cluster had a dataset of 18TB (9TB on remote object store) on 45K buckets for both the bare-metal and Kubernetes.

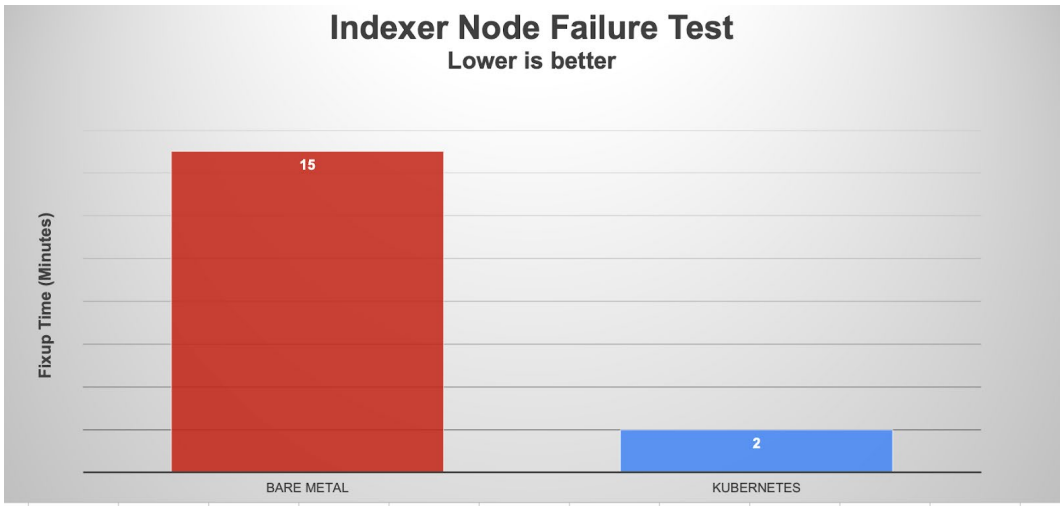

Figure 9: Indexer node failure tests when indexers on bare-metal vs Kubernetes

As the above graph shows, the indexer failure on bare-metal servers with 45K buckets (almost 11.4K fix up activities) took almost 15 minutes to complete to get the indexer cluster to a complete and valid state. Almost all time was spent on fixing the replication factor and search factor requirements. While the cluster returns to a valid state after 15 minutes, the failed indexer results in an indexer cluster with reduced resources to support the ongoing ingest and search.

In the case of Kubernetes, an indexer pod failure only took 2 minutes of bucket fixups before the cluster returned to a complete and valid state. This is due to the fact that Kubernetes will automatically restart a failed pod and it took two minutes for the pod to come up with the same persistent volumes.

Best Practices to Configure Splunk SmartStore on Kubernetes with Portworx

We recommend the following best practices for configuring Splunk SmartStore on Kubernetes with Portworx and FlashBlade systems.

1. Use the Apps Framework to configure SmartStore indexes, cache settings at the cluster scope level.
2. Setup a separate bucket on the FlashBlade to host the Apps Framework.
3. As the indexer nodes in SmartStore are stateless, there is no need to enable HA for the persistent storage volume of the var volume on Portworx that hosts the cache data. Use the replication option to 1.
4. To scale up or down the indexers use the following command:

```
kubectl patch indexercluster <ix name> --type=json -p '[{"op": "replace","path":
"/spec/replicas","value": <idx count>}]' -n splunk-operator
```

See the Best Practices for Splunk on Pure Storage article on Pure Storage support website for additional information.

Conclusion

Based on the tests, the suggested solution of running Splunk SmartStore on Portworx with Pure Storage FlashBlade offers the following benefits:

- Improved performance across the board by scaling the indexers on Kubernetes.
  - Ingest performance improved by a factor of three.
  - Searches per minute improved by a factor of two.
- Maximized CPU utilization by running more instances of Splunk indexers on Kubernetes.
  - CPU Utilization improved by a factor of 1.6.
- Improved operational efficiencies during indexer failure.
- Reduced cost of infrastructure by maximizing the Splunk instances to run on Kubernetes on the same bare-metal server.

Splunk Operator on Kubernetes with Pure Storage FlashBlade and Portworx offers a clear architectural benefit to run critical modern applications like Splunk Enterprise.

Additional Resources

- Splunk Operator for Kubernetes: https://splunk.github.io/splunk-operator/
- App Framework Resource Guide: https://splunk.github.io/splunk-operator/AppFramework.html
- Splunk SmartStore: https://docs.splunk.com/Documentation/Splunk/9.0.5/Indexer/AboutSmartStore
- Portworx documentation: https://docs.portworx.com
- PX-StoreV2 on-prem installation requirements: https://docs.portworx.com/install-portworx/kubernetes/bare-metal/install-px-store-v2/

Appendix: Configuration Values for the Helm installation of the Splunk Enterprise

Here is an excerpt of the configuration values (values.yaml) used with the helm installation of the Splunk Enterprise. Environment specific values like storage classes, resource requests for CPU, memory should be updated.

> Note: This is a suggested configuration. Your mileage can vary based on various factors like servers, network setup, software versions etc.

```
splunk-operator:
  enabled: false

licenseManager:
  enabled: true
  name: lm

  volumes:
  - name: licenses
    configMap:
      name: splunk-licenses

  licenseUrl: /mnt/licenses/enterprise.lic
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: servertype
            operator: In
            values:
            - nonidx
  etcVolumeStorageConfig:
    ephemeralStorage: false
    storageCapacity: 10Gi
    storageClassName: px-splunk-vol
  varVolumeStorageConfig:
    ephemeralStorage: false
    storageCapacity: 100Gi
    storageClassName: px-splunk-vol
  extraEnv:
  - name: TZ
    value: America/Los_Angeles

monitoringConsole:
```

```
    enabled: true
    name: mc

    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: servertype
              operator: In
              values:
              - nonidx
    etcVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 10Gi
      storageClassName: px-splunk-vol
    varVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 100Gi
      storageClassName: px-splunk-vol
    extraEnv:
    - name: TZ
      value: America/Los_Angeles

  clusterManager:
    enabled: true
    name: cm

    appRepo:
      ## Example: Deploy apps on cluster manager instance for local use
      appsRepoPollIntervalSeconds: 300
      defaults:
        volumeName: volume_app_repo_us
        scope: cluster
      appSources:
        - name: idxApps
          location: AppsLoc/
      volumes:
        - name: volume_app_repo_us
          storageType: s3
          provider: minio
```

```
      region: us-west-2
      path: k8s-splunk-apps
      endpoint: https://<FlashBlade_data_vip>
      secretRef: s3-secret

  smartstore:
    volumes:
      - name: remote_store
        path: k8s-splunk-data
        endpoint: http://<FlashBlade_data_vip>
        secretRef: s3-secret
    indexes:
      - name: pure-apache
        remotePath: $_index_name
        volumeName: remote_store
      - name: pure-apache1
        remotePath: $_index_name
        volumeName: remote_store
      - name: pure-apache2
        remotePath: $_index_name
        volumeName: remote_store
      - name: pure-apache3
        remotePath: $_index_name
        volumeName: remote_store

  defaults:
    splunk:
      idxc:
        replication_factor: 2
        search_factor: 2

  etcVolumeStorageConfig:
    ephemeralStorage: false
    storageCapacity: 10Gi
    storageClassName: px-splunk-vol
  varVolumeStorageConfig:
    ephemeralStorage: false
    storageCapacity: 100Gi
    storageClassName: px-splunk-vol

  resources:
```

```
      requests:
        memory: "8Gi"
        cpu: "4"
      limits:
        memory: "16Gi"
        cpu: "8"

    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: servertype
              operator: In
              values:
              - nonidx
    extraEnv:
    - name: TZ
      value: America/Los_Angeles

  indexerCluster:
    enabled: true
    name: ixcl
    replicaCount: 4

    etcVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 10Gi
      storageClassName: px-splunk-vol
    varVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 550Gi
      storageClassName: px-fast-nvme

    resources:
      requests:
        memory: "24Gi"
        cpu: "12"
      limits:
        memory: "32Gi"
        cpu: "16"
```

```
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: servertype
              operator: In
              values:
              - indexers
    extraEnv:
    - name: TZ
      value: America/Los_Angeles

  searchHeadCluster:
    enabled: true
    name: shc

    etcVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 10Gi
      storageClassName: px-splunk-vol
    varVolumeStorageConfig:
      ephemeralStorage: false
      storageCapacity: 100Gi
      storageClassName: px-splunk-vol

    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: servertype
              operator: In
              values:
              - nonidx
    extraEnv:
    - name: TZ
      value: America/Los_Angeles
```

About the Author

Somu Rajarathinam is a Technical Director in the Portfolio Solutions team at Pure Storage. His responsibilities include defining application solutions for Pure Storage products, performing benchmarks, and developing reference architectures for applications running on Pure Storage products. Somu has over 25 years of experience in the database and analytics area. He was a member of the Systems Performance and Oracle Applications Performance Groups at Oracle Corporation. He also worked with Logitech, Inspirage, and Autodesk, where his assignments include delivering database and performance solutions, managing infrastructure, and providing support to database and analytical applications both on-prem and on the Cloud.

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041

PS2461-01-en 07/23