ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

Persistent Storage Options for Stateful Containerized Applications

Bilal Homsi[1], Mohamed Sierafi[2]

[1,2]Dhahran, Saudi Arabia

Abstract: **There has been a major shift towards building stateless applications using containers due to added benefits in terms of portability, agility and resilience containers provide to the application runtime. With the added growth of container adoption, a breed of new applications stateful in nature are being built which would necessitate the need for persistent storage for containers. This paper examines the architecture requirements for stateful applications in a containerized environment and discusses the different persistent storage options available and needed for stateful applications using containers.**

Keywords: **Containers, File Storage, Block Storage, MySQL, Postgres, Windows, Linux, HA Clustering, CI/CD, HPC, ML, SLA, CNS, NAS, SAN, HCIS, SDS.**

I. INTRODUCTION

Application container technology provides the basic standards based packaging and runtime management of the underlying application modules. Containers are quick to deploy optimizing use of system resources. With containers, developers benefit from application portability, programmable image management. IT operations leverage a standard runtime environment for deployment and management. With all these benefits, a common misperception still exists that containers are good only for stateless applications and it is not doable to containerize stateful applications.

II. STATEFUL CONTAINERIZED APPLICATIONS TYPES

Persisting application state is about preserving the state across application restarts and outages. This type of state is usually stored in a redundant database tier with regular backup performed on it. Stateful applications suitable for containerization can be categorized into the following classes, each of which has different clustering characteristics:

A. SQL Databases

Open-source database servers (MySQL, PostgreSQL) are popular with developers of cloud-native applications. Commercially supported versions of these databases (Microsoft SQL) are available for deployment in container environment in both Windows and Linux environment. The Linux container images are suitable for production use, while the Windows container images are provided only for development and testing use cases. When High Availability (HA) Clustering is deployed in Containers, it is possible to use the container orchestrator to automatically restart a failed database server instance — as long as the container is reconnected with its original storage after failover. Several vendors provide SQL databases that are designed to scale out performance in clusters and that can be deployed in containers. When these database servers are containerized, the container orchestrator can be used to schedule multiple parallel instances of their server applications.

B. NoSQL Databases

NoSQL databases are architected for expected performance on large big data lakes. This capability requires support for scaling out multiple instances in a cluster. Their components can be deployed in containers that can be managed with an orchestrator. Some of them use data replication to make sure that copies of data remain available if an instance of a component fails.

C. Analytics

Analytics applications use a diverse list of methods for scaling in clusters while load balancing and maintaining availability. Their core components are usually suitable for scale-out deployment in containers, but their functions for accessing data may require some re-engineering. Typically, for instance, Elasticsearch can perform its own data replication. Hadoop and Spark depend on the Hadoop Distributed File System (HDFS) to synchronize access to data. HDFS requires some effort to implement in containers.

D. Content Management

Content management applications maintain their state in databases. Therefore, the persistence of these containers can be managed as discussed in SQL Databases.

E. Continuous Integration/Continuous Delivery (CI/CD)

Containers can be used to implement clusters for CI/CD tools. Such tools use a master-slave model for distributed builds, and it relies on a single file directory to maintain its state. Therefore, it is necessary to use a shared file system to share that directory between containers running on different hosts.

F. Data Processing

Lots of data processing applications, such as graphics-rendering and video-transcoding tools, can use clusters of application instances to scale out processing for large batch jobs. They usually rely on a shared file system to coordinate the state of distributed tasks and to transfer data.

G. High Performance Computing (HPC)

HPC applications are usually optimized for scaling out. When these applications are deployed in containers, a container orchestrator can be used to schedule multiple instances of them to process tasks in parallel across a cluster of hosts. As with data processing applications, HPC applications usually rely on a parallel shared file system to transfer data between application instances.

H. Machine Learning (ML)

Containers are commonly used to deploy ML applications. It uses distributed computing to work on portions of graphs in different processes, each of which can run in a separate container.

III. APPLICATION CONTAINERIZATION APPROACHES

In order to choose the right storage method for stateful containerized applications, there is need to identify the use case for containers and fully understand the benefits that containerization is expected to provide for that particular application. Containers can help in the efforts to streamline and accelerate application delivery pipelines, increase workload portability for cloud migration, and break up and refactor monolithic applications. Containerization use cases fall into three broad categories:

A. Rehosting Existing Applications

Some legacy applications can be adapted for deployment in containers without significant changes to their architecture or functionality. This approach requires minimal efforts. Despite a change to its deployment model, the application remains monolithic. However, rehosting applications can still benefit from leveraging orchestration for the purposes of scalability, availability and redundancy.

B. Revising Existing Applications

Revising an existing application involves taking an existing monolithic application and altering its architecture to decompose functionality into smaller components based on microservices. A common approach with stateful applications is to break apart and containerize only the stateless components or services, leaving the data-persisting functionality as is.

Containerized stateful applications deployed in these scenarios will have a variety of storage requirements depending on their scale and performance needs (throughput, latency).

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

C. Building New Cloud-Native Applications

Building new applications remains the most common containerization use case. Cloud-native applications are often designed as highly dynamic distributed microservices. Storage persistence is needed to deal with the behaviors exhibited by cloud-native applications, such as high churn rate. In traditional centralized storage cases, applications tend to be tied to certain hosts, and storage systems tend to have relationships with the hosts rather than with the applications directly. With new cloud-native applications that are distributed and containerized, Service Level Agreements (SLAs), including those related to storage, must be enforced at the application level. Since orchestrated containers can move between available hosts in the cluster to accommodate the specified desired state and application SLAs, one of the two following accommodations must be enforced:

1. Force host affinity for persistent containers to prevent the orchestration system from scheduling them to nodes that do not have the required dataset available locally. This option is good for small clusters and less dynamic workloads.

2. Deploy additional software to enable containerized workloads to access their associated datasets over the network, regardless of which hosts provide the compute resources. This option is good for any cloud native application planned for containerization.

IV. APPLICATION CONTAINERIZATION REQUIREMENTS

For implementing persistence in a stateful application, the requirements that will drive the choice of a particular approach need to be defined. Below are the fundamental factors to consider when containerizing a stateful application:

A. Application Clustering

When containerized applications are clustered, two separate logical layers of clustering have to be implemented

a. The cluster of container host systems on which the containers are running. The host systems may be physical servers, Virtual Machines (VMs) or cloud Infrastructure as a Service (IaaS) instances. Storage is attached to each node of the host cluster.

b. The logical cluster of application instances running in containers. Each application instance may share its state with other instances.

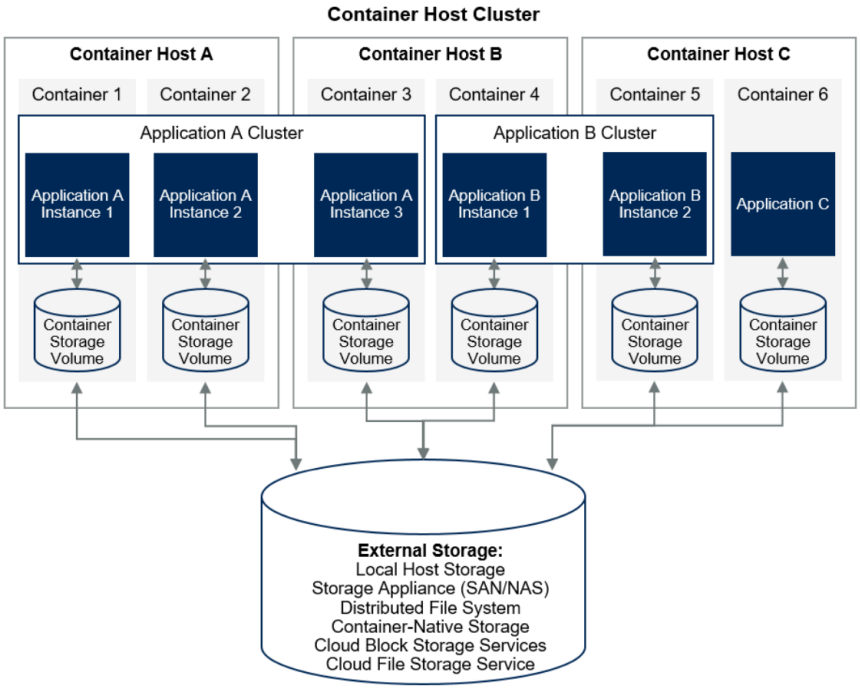

Fig 1: Clustering with Containers (Courtesy of Gartner)

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

Since storage remains attached to a particular underlying host, a key consideration to account for is whether the containerized application will run on a stand-alone basis or in a cluster. If the containerized applications will be clustered, it will be necessary to determine if the clustered application instances need to share their state through a storage system. If so, it will be necessary to take advantage of Container-Native Storage (CNS) solutions. CNS solutions work with the container orchestrator to optimize the scheduling of application instances in a cluster, based on storage access.

Application clusters may be implemented for purposes of High Availability (HA), scalability or both. There are two approaches for application clustering:

a. High Availability Clustering

A standby application instance can take over when a primary instance has crashed or has stopped processing normally, allowing processing to continue.

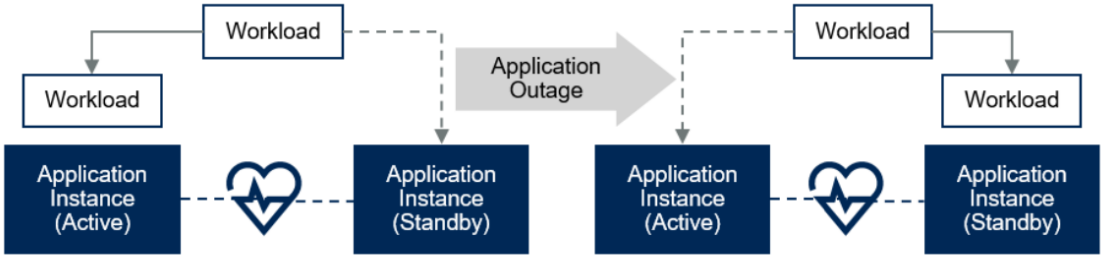

Fig 2: High Availability Clustering with Containers (Courtesy of Gartner)

b. Performance Clustering

With scale-out performance clustering, multiple instances of an application work in concert, thus providing the capacity needed to efficiently process large individual batch workloads or to support high volumes of user traffic.

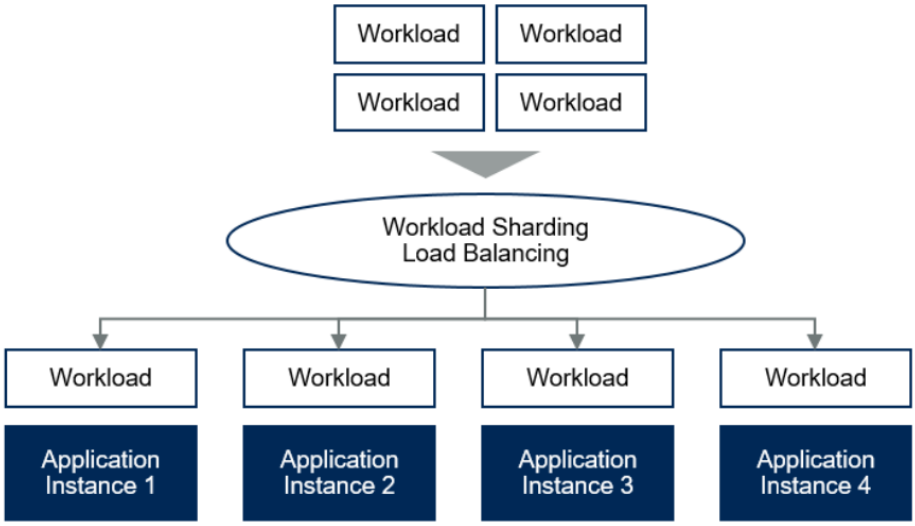

Fig 3: Performance Clustering with Containers (Courtesy of Gartner)

In either approach, Container orchestrator will launch another instance of its container. If an entire host fails, the orchestrator will automatically reschedule all of the containers from the failed host to the surviving hosts. The actual implementation of each clustering method will vary by application type.

B. Data Synchronization

When applications are clustered, the method through which multiple application instances persist and share data is a critical operational characteristic of the cluster. There are different approaches that containerized applications can synchronize data when running in various configurations:

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

a. Standalone Configuration

In this approach, an application instance connects to a local storage resource that is bound to its container with a storage volume driver. The instance could also maintain its data on a Storage Area Network (SAN) appliance, a Network Attached Storage (NAS) appliance or a distributed file system to gain access to advanced data services.

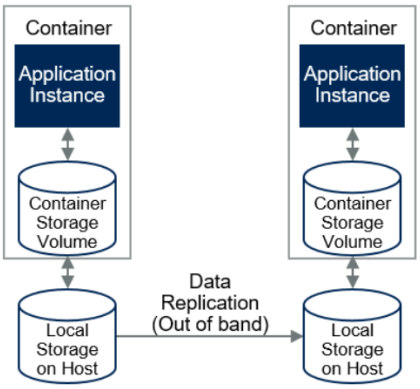

Fig 4: Standalone Configuration Where Application is Not Clustered (Courtesy of Gartner)

For a stateful container to gain mobility within a cluster, or between clusters, data centers, cloud services, one of the following approaches must be used:

1. Attach the volume to a storage resource residing on centralized or distributed storage. In this case, a volume in a container on another host can be reattached to the storage resource, as long as the resource can be identified in a namespace that is consistent across the mobility domain.

2. Use a snapshot mechanism in storage management software to create an out-of-band copy of the local storage resource that is replicated to the destination host. In this case, the container or container orchestration platform has to implement a method for connecting a volume in the new container (running on the destination host) to the replicated storage resource.

b. Shared Storage With Distributed File System

In this approach, multiple containers use container storage volumes to connect with shared storage based on a distributed file system, such as NFS or some other file service implementation.

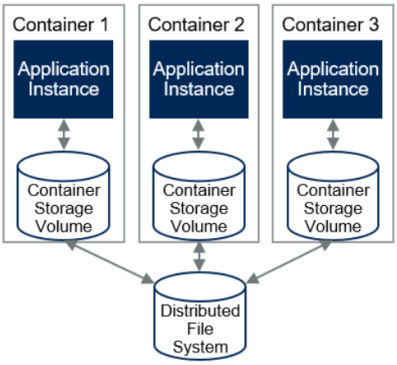

Fig 5: Shared Storage With Distributed File System Configuration (Courtesy of Gartner)

When applications write data to the shared storage volume, it becomes accessible to applications in all other containers attached to that storage.

c. Master-Slave Data Replication

In this approach, clustered applications do not rely on the storage system to share data. Instead, a master instance of the application maintains its data on local storage.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

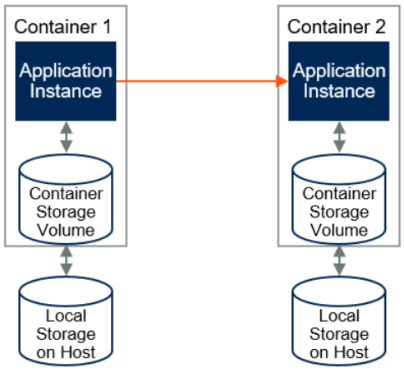

Fig 6: Master-Slave Data Replication Configuration (Courtesy of Gartner)

However, when state has to be persisted across the cluster, the application explicitly replicates its transactions to one or more instances of the application running in standby slave mode. These instances are attached to their own local storage in another container.

d. Multi-Master Data Replication

In this approach, applications do not rely on the storage system to share data either. Instead, clustered application instances maintain their data on local storage. The application instances communicate directly with each other to replicate data as needed, to scale performance and to maintain data availability.

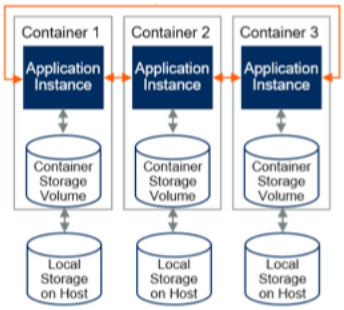

Fig 7: Multi-Master Data Replication Configuration (Courtesy of Gartner)

V. PERSISTENT STORAGE APPROACHES

Development communities for several container orchestration platforms, collaborated on developing a new specification for container storage called Container Storage Interface (CSI). CSI defines both block- and file-storage abstractions for container storage volumes.

The CSI specification standardizes the integration of external storage systems with many container orchestration systems. The application only has to write its data to a container storage volume representing a file system or block device, and the orchestrator will handle the rest transparently. Container orchestration platforms can now seamlessly integrate any third-party drivers that implement CSI for storage products or services.

Different methods for provisioning container storage volumes, using drivers based on CSI or older specifications, are listed below

A. Local Host Storage

When backed by local storage, a container's storage volume is linked to the host on which the container is running. The container orchestrator provisions the local storage resources requested by the containerized application, and allows them to be used on the same host as long as they are needed. Once they are no longer needed, those resources are released.

This approach is a good choice for nodes in shared-nothing clusters of databases, and for use cases that involve high degrees of churn with container creation and removal. It may also be used to implement single-system HA for a database, whereby the container reattaches to the local storage when restarted after a failure.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

B. Storage Appliance

Storage appliances, such as Network-Attached-Storage (NAS) platforms, Storage-Array-Network (SAN) arrays and Hyperconverged Infrastructure Systems (HCIS) platforms, are designed to manage storage with a centralized approach.

Storage appliances are an option only for on-premise container deployments. While these systems provide the highest level of enterprise data management capabilities, they will incur some performance overhead due to limited support for shared access by multiple containers.

C. Distributed File System

Distributed file system storage uses a parallel file system to cluster multiple storage nodes together. It presents a single namespace and storage pool to provide high-bandwidth persistence for multiple hosts in parallel. Distributed file systems can be deployed on-premises or in the public cloud. A key feature of using distributed file systems is that they can usually be accessed by multiple containers simultaneously, either on the same host or on different hosts. This capability allows distributed file systems to support containerized stateless applications requiring shared storage. With hybrid and multi-cloud deployments, the use of distributed file systems provides operational consistency when stateful containerized applications have to access storage resources on different platforms.

D. Container Native Storage

By default, most container orchestration systems treat all hosts in the cluster as equally suitable for running a container with the possibility to impose host affinity and other restrictions in cases where the application prefers or avoids a particular host.

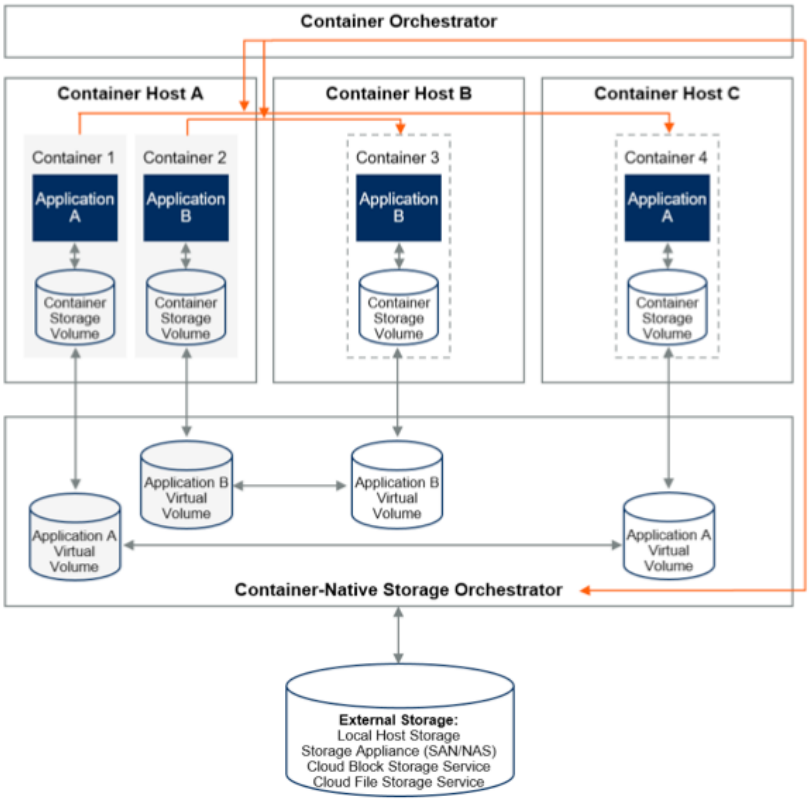

Fig 8: Cloud Native Storage Orchestration (Courtesy of Gartner)

When making a decision on where to schedule a stateful container, the orchestrator should prioritize nodes that have direct access to the data the container requires. Operations can define virtual storage volumes in these pools of locally attached storage. Also, they can define policies for dynamically replicating the virtual storage volumes across multiple hosts in a container cluster or even to other data centers, or public cloud services.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 2, pp: (34-42), Month: April - June 2021, Available at: www.researchpublish.com

VI. FUTURE TRENDS

With the momentum build-up for container adoption in stateful applications, two areas are still maturing to achieve these objectives:

A. Adoption of Container Standard Interface (CSI)

Although CSI is considered production-ready, standard to be fully supported in storage products and in production releases of container orchestration platforms is still ongoing. Also, the current release of the CSI specification is fairly limited in scope. It allows applications in containers the ability to control underlying storage resources. To have a more complete specification, the CSI specification needs to extend its capabilities to support management of data across multiple hosts and multiple sites.

B. Container Storage Hyperconvergence

In traditional storage systems, control over storage resources resides within the storage systems. While this integration enables the highest levels of scalability and reliability, this is in favor of flexibility, making it difficult to adjust the behavior of storage resources to the needs of specific applications. Hyperconvergence architecture based on commodity hardware with local attached storage can be leveraged using the below approaches:

a. **Software-Defined Storage (SDS)** is a storage deployment model that improves the flexibility of storage systems by introducing a logical separation between the actual storage resources (data plane) and the software layer that controls those resources (control plane). This concept can be enhanced where containers are used to host elements of the control plane software. With this approach, the containers become building blocks for assembling SDS control planes. The SDS control planes can be customized based on the storage resources that are available to container hosts and based on the needs of applications which are hosted in containers.

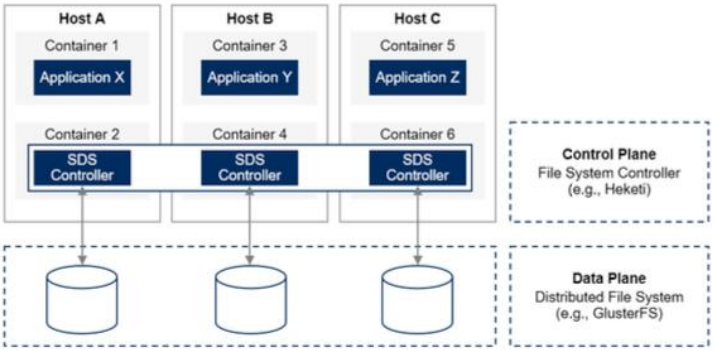

Fig 9: Containers Storage Persistence Based on Hyperconverged Servers (Courtesy of Gartner)

b. Container Native Storage Orchestration

A storage orchestrator that integrates with the container orchestrator to ensure that container storage volumes are available for containers, regardless of the host they are scheduled on.

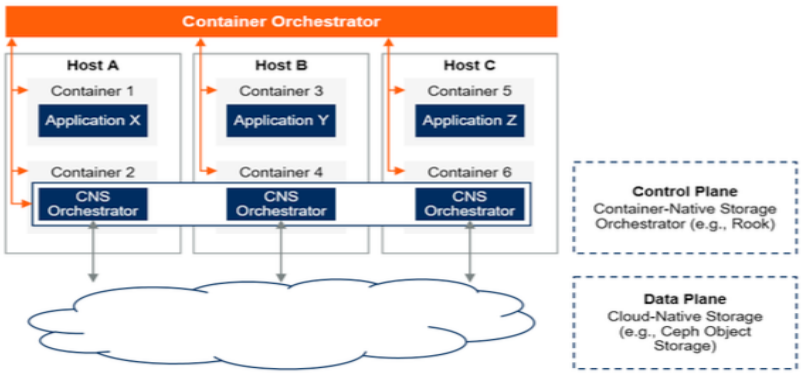

Fig 10: Containers Native Storage Orchestration (Courtesy of Gartner)

If a container moves from one host to another, the storage orchestrator will unmount its storage volume and remount it in the container on the other host. This has the added benefit of simplifying the deployment of SDS with containers.

VII. CONCLUSION

There is a strong drive fueling interest in running stateful applications in containers. This cannot be achieved without ensuring operational consistency, solution availability, containerization of long-running and data-rich applications, fulfilling data-sharing requirements, and delivery of database services. Persisting data for a single application instance in a container that runs on only one host is simple. When multiple instances of stateful applications run in orchestrated containers that must share state, their relationship with underlying storage resources requires careful management using robust processes that can ensure their binding. As the containerization of stateful applications becomes more common, users must consider a variety of possible approaches for providing storage persistence to containerized applications. These approaches all have different deployment and operational characteristics, both on-premises and in the cloud.

REFERENCES

[1] Tony Lams, "Decision Point for Selecting Stateful Container Storage", Gartner Paper G00383151, April, 2019.

[2] Julia Palmer, "Hype Cycle for Storage and Data Protection Technologies, 2020", Gartner Paper G00441602, July 2020.

[3] Technology Org Website, Boris Kurktchiev, "3 Reasons to Bring Stateful Applications to Kubernetes", thenewstack.io [https://thenewstack.io/3-reasons-to-bring-stateful-applications-to-kubernetes/]. (Accessed on November 1, 2020)

[4] Technology Org Website, Jim Bugwadia August 2016, "Containerizing stateful applications", infoworld.com [https://www.infoworld.com/article/3106416/containerizing-stateful-applications.html]. (Accessed on November 1, 2020)

[5] Julia Palmer, Arun Chandrasekaran, "An I&O Leader's Guide to Storage for Containerized Workloads", Gartner Paper G00380359, January 2019.

[6] Technology Org Website, Pete Brey, November 2019, "Persistent Data Storage Integral for Containers", containerjournal.com [https://containerjournal.com/topics/container-networking/persistent-data-storage-integral-for-containers] (Accessed on November 1, 2020)

[7] Technology Org Website, Anthony Adshead, March 2020, "Container storage 101: What is CSI and how does it work?", computerweekly.com [https://www.computerweekly.com/feature/Container-storage-101-What-is-CSI-and-how-does-it-work] (Accessed on November 1, 2020)