Kubernetes Integration with Virtuozzo Storage

A Technical Whitepaper

OCTOBER, 2017
Application Container Storage

Application containers appear to be the perfect tool for supporting cloud native transformation. Moving applications to a micro-services architecture brings significant advantages, including:

- Faster time to market (hours vs months)
- Rolling deployments and versioning
- Better cluster resource utilization

At the same time, challenges often occur when managing thousands of individual containers, increasing the complexity of cloud transformation. To address this challenge, special orchestration solutions like Kubernetes (K8s), Docker Swarm and Mesos were introduced on the market. Kubernetes in particular is currently drawing the most attention in the market with a growing community behind it.

However, according to a recent survey by the Cloud Native Computing Foundation, early adopters highlight several key challenges in deploying production-ready Kubernetes clusters. The biggest challenge is determining the right persistent storage solution for containers. Kubernetes has an API to integrate with storage systems, but does not provide an actual storage solution itself. While 3rd party storage solutions exist on the market, many do not natively support Kubernetes, which can further complicate deployments.

K8s Storage Data Model and Scenarios

Kubernetes is a complex framework that provides a certain level of abstraction for all types of resources, including storage resources. With that said, the model for storage might be a little confusing for users who have just started getting acquainted with the solution.
Initially, if not stated otherwise, containers do not have any persistent storage; they typically utilize ephemeral disks that are actually on-disk files that are cleared when a container is taken down or crashes. This short, ephemeral disk lifecycle is dependent on the lifecycle of the container which is not suitable for persistent workloads. In order to solve this issue, “volumes” were introduced.

A volume is a directory that can contain data that is accessible by every container in a pod. From the application point of view, applications inside containers see the volume at specific mountpoints for each container.

The volume abstraction lifecycle is linked to the pod, which means that when individual containers in the pod are restarted, the data is preserved. Volumes are also available for all the containers inside the pod. When the pod is deleted, the volume is also deleted.
The volume concept in K8s does not provide any particular storage implementation. Any storage solution that provides storage resources should meet accessibility and lifecycle requirements, and in this case distributed, protected storage is usually the best solution.

While volume resources are still widely used in K8s, as time has passed and more production clusters were deployed, some new requirements for storage management have emerged, including:

• Decoupling storage resource provisioning from consumption
• Extending lifecycle options for storage
• Introducing concepts for traditional storage profiles
• Preserving the resource model where the backend is independent from storage implementations

As the requirements above emerged, the following new API resources were introduced:

**Persistent Volume Claim (PVC)**—Representation of a request for a specific volume created by the user, but it is also the consumed resource itself (which may be a little confusing). This object is namespaced, and it is consumed by pods that are deployed to the same namespace.

**Persistent Volume (PV)**—Represents the storage resource itself. It is binded to the Persistent Volume Claim in case the Persistent Volume satisfies the requirements. PV is created by the cluster administrator (or dynamically), and it has no namespace.

**Storage Class**—Represents requirements for a volume, which contains parameters specific for the storage backend.
Dynamic and Static Provisioning

When the user specifies pods to be deployed, the PVC is specified. After that there are two options to handle storage resource allocation: dynamic and static.

The dynamic resource allocation process is displayed below:

When a PVC is created, it has a reference to the Storage Class, which is created for a specific storage driver. The storage driver creates the PV that satisfies the request from the PVC (size, access mode) and Storage Class (specific parameters of the storage backend). As a result, the PVs are created dynamically.
The static provisioning method is displayed below:

With the static method, the PVs are manually provisioned by a system administrator and the PVCs created by users are bound to those PVs automatically based on size and access mode.

When the pod that utilizes the PVC is deleted, the PVC itself and PV are not and can be reused, which makes the PVC/PV storage management approach the most flexible. When the PVC is deleted the PV can be left for manual retainment, deleted or cleaned up—it depends on the capabilities of the backend.

The concepts described in this paper are basic—details on additional parameters and the differences between implementations for different backends are disclosed in the official documentation: Persistent Volumes.
Virtuozzo Storage

Virtuozzo Storage provides highly available, highly flexible software-defined, distributed storage with built-in redundancy. It runs on top of commodity hardware, using locally attached hard drives to create storage pools that are available to all servers. This approach makes data highly available, eliminates disk fragmentation, and optimizes overall cluster I/O performance.

Core Storage Engine

Figure 1 above illustrates the logical structure and basic components of Virtuozzo Storage. The key components are:

- **Chunk Servers (CSs)**—All data in a Virtuozzo Storage cluster is stored in the form of fixed-size chunks on chunk servers, which provide containers with access to the data as needed. The cluster automatically replicates the data chunks and distributes them across the available chunk servers to provide high availability of data. To achieve this, a Virtuozzo Storage cluster must have multiple chunk servers.
• **Metadata Servers (MDSs)**—To keep track of data chunks and their replicas, the cluster stores metadata about them (i.e. file names) on metadata servers. In addition to managing metadata, the MDSs control how files are split into chunks and where the chunks are stored. They also track versions of chunks, ensure that the cluster has enough replicas, and keep a global log of important events that happen in the cluster. As is the case with CSs, multiple MDSs are needed to provide high availability.

• **Clients**—Clients can manipulate data stored in the cluster by sending different types of file requests, including modifying an existing file or creating a new one. Clients access a storage cluster by communicating with the MDSs and CSs. You can export storage via multiple interfaces: persistent volumes for Kubernetes and Docker, S3 object storage, NFS and iSCSI.

### Replication and Erasure Coding

Virtuozzo Storage provides two options to enable the protection of data: replication or erasure coding.

With replication, Virtuozzo Storage breaks the incoming data stream into 256Mb chunks. Each chunk is replicated, and replicas are stored on different storage nodes. As a result, each node has only one replica of a given chunk. This mode is advised for users who are running IO intensive and performance intensive workloads as replication scales with the number of nodes in the storage cluster.
With erasure coding, Virtuozzo Storage breaks the incoming data stream into fragments of certain sizes, then splits each fragment into a certain number (M) of 1-megabyte pieces, and creates a certain number (N) of parity pieces for redundancy. All pieces are distributed among M+N storage nodes, that is, one piece per node. On storage nodes, pieces are stored in regular chunks of 256Mb, but these chunks are not replicated as redundancy is already enabled. The cluster can survive the failure of any N storage node without data loss. The erasure coding approach is much more efficient in terms of physical storage utilization than replication, but some types of workloads can demonstrate performance loss.

Storage Tiers

Storage tiers represent a way to organize storage space. You can use them to keep different categories of data on different chunk servers. For example, you can use high-speed solid-state drives to store performance-critical data instead of caching cluster operations.
Storage Interfaces

Virtuozzo Storage is designed to run side-by-side with the compute cluster. It only needs one CPU core and several Gb of RAM to operate. As a result, almost all CPU and RAM resources are available for compute technology running on top. Storage capacity can be exposed to the clients with the most popular interfaces, including:

- Persistent Volumes for Kubernetes and Docker containers
- S3 Object Storage
- NFS as File Storage
- iSCSI and loopback interfaces for classic virtualization platforms like OpenStack, ESXi or Hyper-V

In addition, Virtuozzo Storage supports data encryption at rest, compression, S3 geo-replication and more than 30+ features essential for any kind of production storage usage.
Volume Management

In order to make storage resources available for the consumption of the K8s Cluster and to simplify volume management, Virtuozzo Storage for Kubernetes provides the following components:

Storage—Installs and configures storage components as part of the system installation and cluster setup.

FlexVolume Driver—A family of vendor-specific volume drivers for K8s that provide simple APIs to mount and unmount volumes. Virtuozzo Storage provides a ploop flexvolume driver that mounts the ploop block device into K8s-managed workloads. This driver is installed on each Virtuozzo Storage node included in the K8s cluster after the Virtuozzo Storage for K8s solution is installed.
**Provisioner**—Provides a separate piece of software that determines how the PV is provisioned and how the Storage Class parameters are handled. There are internal and external provisioners in Kubernetes: internal ones are shipped together with K8s and can be used out-of-the-box, and external provisioners are shipped by storage vendors. In the case of Virtuozzo Storage, the external provisioner is shipped and installed together with storage for the K8s solution installation package. It is deployed as an application inside the K8s cluster for high availability to be provided by default. The Virtuozzo Storage provisioner handles the connection to the storage cluster and the dynamic creation of volumes. The volume parameters applicable are disclosed in the documentation, [Volume Parameters](#), but the most crucial for each volume are replication/encoding and tiering.

**Snapshot Controller**—Manages taking/restoring/deleting of snapshots and managing of snapshot policies. This controller is implemented as an addition under the K8s aggregation API, which allows the introduction of new API resources. The snapshots and controller are described in detail in the next section.

**Management UI**—Enables a separate application that runs inside the K8s cluster and provides useful management features.

## Dashboard

Virtuozzo Storage for Kubernetes provides an easy-to-use dashboard that provides information on the health of pods with the provisioner and snapshot controller, as well as statistics on provisioned volumes and distribution of storage resource utilization by storage class.
The Cluster Setup screen is shown below:
When the cluster is setup in the Storage Management UI (the details can be found in documentation: Cluster Setup), it can be enabled for consumption by the K8s cluster. Kubernetes Storage Provisioners use K8s Secrets to access the storage clusters. It is important to note that secrets are namedpaced objects and are required to create the secret in each namespace where the PVCs are going to be created. To simplify secret distribution, Virtuozzo Storage provides an easy configuration screen in which you can select particular namespaces from which the storage can be accessed.

The Storage Classes management screen is show below:

The Storage Classes Management Screen simplifies the creation of storage classes by eliminating the need to define parameters specific to Virtuozzo Storage in the YAML file. With this screen, the user can select the cluster for the storage class, replication/encoding modes, tiering and standard K8s parameters. The user can also select the default storage class, which is selected for the dynamic provisioning of volumes and policies for snapshot scheduling.
Snapshots

Virtuozzo Storage for Kubernetes is one of the very few storage solutions on the market that offers snapshots for volumes that are deeply integrated with the K8s volume management resource model. It is achieved by the implementation of a custom API resource and controller that follow the K8s API aggregation approach. Snapshots are essential for persistent storage because they provide additional protection of the valuable data, but this is not the only application; Virtuozzo Storage snapshots make it easy to rollback to the previous application state that requires persistent storage, for example restore state of wordpress blog.

With the K8s aggregation APIs introduced in K8s 1.7, Kubernetes now provides an aggregation layer for custom APIs. This layer runs in-process with the kube-apiserver when the new API server is registered, and this aggregation layer proxies calls for additional resources to the specific extension-apiserver, which is usually implemented as a pod running in a cluster.

With Virtuozzo Storage, the pod with the api-server also hosts the controller container that manages the creation/deletion/restoration and scheduling of snapshots.
By implementing snapshots as part of the API leveraging the K8s native aggregation, Virtuozzo enables snapshot management from kubectl, and with the REST API that follows the declarative principals of K8s native APIs.

See the diagram below to review how snapshot resources are related and displayed:

![Diagram]

With Virtuozzo Storage for Kubernetes, the snapshot object is linked to the PVC and not the PV, and there is a reason for it: if the snapshot is linked and associated with the PV itself, it becomes unreachable to users limited by namespace, and therefore unmanageable as a persistent workload as this is a user-facing operation. By associating the snapshot with the PVC, Virtuozzo Storage puts it in the same namespace as the PVC. This puts some limits on the restoration procedure, as only bound PVs (the ones that have a claim associated) can be restored. The restoration procedure requires pods that utilize the PVC to be stopped, but it doesn’t require the recreation of all storage resources like PVCs and PVs.
The creation of snapshots is performed with the spawning of a special worker pod that is deleted after the operation is completed. For every PVC snapshot, this can be created manually at any time:
It is also possible to configure the snapshot policy that targets a specific storage class:

This policy will trigger the creation of snapshots for all PVCs that match target classes every day and will keep up to 5 snapshots before rotation. More details on snapshot management including CLI examples can be found in documentation: Snapshots management.
Conclusion

Containers present significant benefits for supporting cloud-native apps and the transformation to micro-services as they continue to grow in popularity and adoption. Several important challenges associated with managing containers have been addressed by popular orchestration solutions such as Kubernetes. However, deploying production-ready Kubernetes clusters face additional issues. The most noteworthy challenge is determining the right persistent storage solution for containers.

While Kubernetes has an API to integrate with storage systems, it does not provide an actual storage solution itself. Other 3rd party storage solutions that exist on the market also lack a solution to address the persistent storage challenge.

In order to effectively address the challenge of persistent storage, Virtuozzo provides a highly available, software-defined, distributed storage solution with built-in redundancy that provides native support for Kubernetes. It runs on top of commodity hardware, using locally attached hard drives to create storage pools that are available to all servers. This approach makes data highly available, eliminates disk fragmentation, and optimizes overall cluster I/O performance.

To learn more about how Virtuozzo makes it possible to manage persistent data in cloud native environments visit: www.virtuozzo.com/products/virtuozzo-storage