WHITE PAPER

DEPLOYING STATEFUL CONTAINERS WITH KUBERNETES

PAVILION AND NVMe-oF

pavilion data

Table of Contents

1. INTRODUCTION..... 2
2. KUBERNETES..... 2
3. KUBERNETES AND PAVILION DATA..... 3
4. PERSISTENT, STATEFUL K8s..... 3
5. DEPLOYING KUBERNETES WITH PAVILION..... 6
6. SETTING UP THE KUBERNETES CLUSTER TO UTILIZE PAVILION STORAGE..... 7
7. PROVISIONING STORAGE FOR CONTAINERS..... 10
8. DEPLOYING A CONTAINERIZED APPLICATION..... 14
9. SUMMARY..... 15

INTRODUCTION

Pavilion Data is an industry leader in NVMe-Over-Fabrics (NVMe-oF) and the only vendor to deliver a Hyper-Parallel Flash Array purpose-built for organizations embracing modern deployment models and applications for digital transformation. Containers are one such deployment model that provides superior operational efficiencies. However, containers were traditionally designed as ephemeral resources for development and test. As container benefits like the separation of infrastructure from application becomes obvious, there is a natural desire to move from experiment to production. However, implementing containers in production creates new challenges for data storage at scale.

In this whitepaper, we discuss how Pavilion's NVMe-oF Storage Array can help organizations migrate from Virtual Machines (VMs) and non-persistent containers to a Composable, Disaggregated Infrastructure (CDI) where persistent containers and stateful applications are readily available and deployed so that the ever-changing requirements of product workloads. This means that compute, network, and storage resources all scale independently to meet a diverse set of application requirements.

KUBERNETES

Kubernetes (commonly stylized as k8s) is an open-source container-orchestration system for automating application deployment, scaling, and management. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It works with a range of container tools, including Docker. Many cloud services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service.

Containers as a service (CaaS) is a cloud service model that allows users to manage and deploy containers, applications, and clusters through container-based virtualization. CaaS is highly useful to IT departments and developers in building secure and scalable containerized applications. To fully deliver on the promise of CaaS, application response time, scalability and availability must be as good as bare metal application implementations.

As NVMe SSDs become ubiquitous with constant improvements in capacity and cost, they are rapidly becoming the defacto performance storage tier for modern data centers. However, deployment models vary widely, and traditional approaches like Direct-Attached Storage (DAS) or legacy All-Flash Arrays have significant limitations, especially as drives become larger and more performant. For K8's, a fundamental rethink of storage infrastructure offers big payoffs.

KUBERNETES AND PAVILION DATA

Using Pavilion Data's Hyper-Parallel Flash Array, K8's can be deployed as stateful, persistent resources that can be fine-tuned for storage performance, capacity, high-availability, and ease of management.

Using functions inherent to the Pavilion system, it is possible to define Persistent Volumes with Persistent Volume Claims that have specific Service Level Agreements for storage read/write bandwidth, IOPS, thin provisioned capacity for non-disruptive growth and can be orchestrated with zero-footprint snapshots allowing K8 portability between server nodes for data sharing, backup and archival.

PERSISTENT, STATEFUL K8s

Persistent volumes and persistent volume claims let containerized apps use external storage without changes to the codebase. According to Gartner Group, by 2022, more than 75% of global organizations will be running containerized applications in production[1]. In order to effectively maximize the potential for containerized applications, persistent storage must address three fundamental requirements:

1. Manageability
2. Availability
3. Scalability

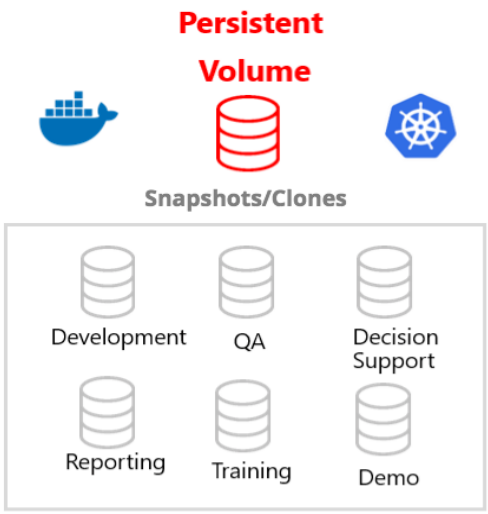
MANAGEABILITY

Persistent containers should adhere to common best practices for manageability. Storage Area Network technology has proven to be the most widely accepted approach to making snapshots, replicating and performing data protection. But SANs, especially fibre channel-based implementations are laden with unnecessary latency and carry the baggage of support for legacy hard disk protocols like SATA and SAS which have no place in modern application deployment. NVMe-oF provides a very low latency disaggregation methodology to give storage arrays SAN-like management features without the quagmire of legacy systems. Pavilion Data's Hyper-Parallel Flash Array offers manageability benefits like:

- Self-extracting flex Kubernetes volume plug-in
- Application proxy to manage multiple arrays
- Zero-footprint Snapshots and clones

1 Gartner Group - Top Emerging Trends in Cloud-Native Infrastructure ID G00385619, May 28, 2019

Zero-footprint snapshots are particularly valuable for running containerized applications. Administrators can orchestrate movement of snaps and clones to any host across any standard Ethernet or Infiniband network. These clones are not restricted to the pod where the master volume is being used. This same snapshot process allows for isolation of container backups at a volume, pod or cluster level.

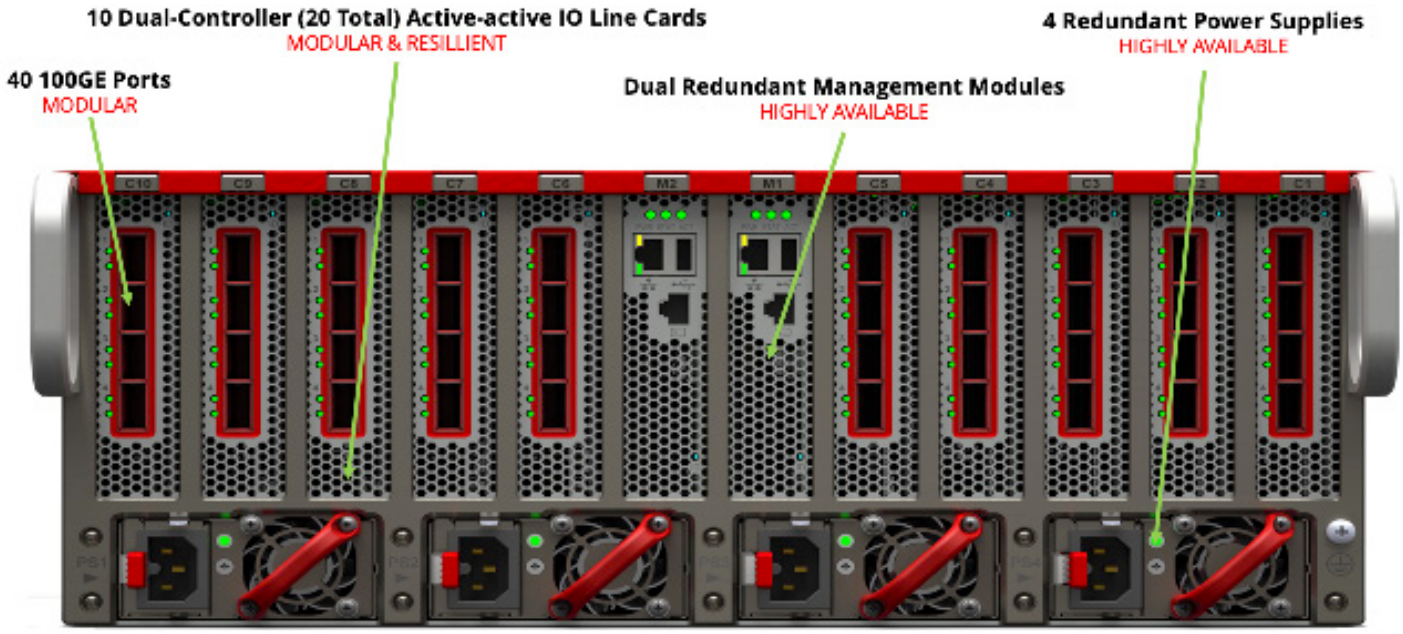


Persistent volume containers can now be portable to any team in the organization without concern for host location or storage capacity consumption on the array

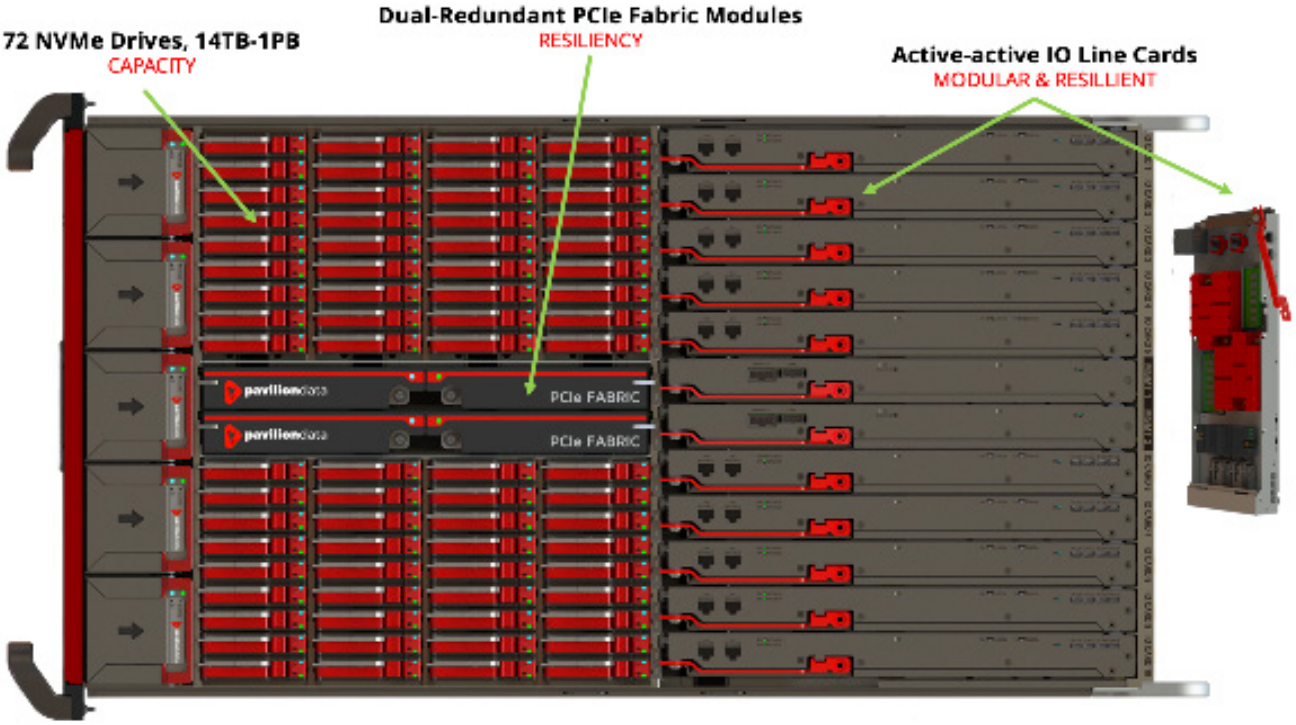
Pavilion Data also provides thin provisioning to allow containers to grow and shrink without disruption to production workloads. This capability is not readily available with traditional direct-attached storage approaches.

AVAILABILITY

Direct-attached storage leaves containerized applications with a single point of failure (server and storage.) Pavilion makes it possible to deliver shared storage for persistent containers with the same performance as locally-attached SSDs for large-scale distributed applications. The Pavilion Array is designed from the ground up with key availability features in order to support maximum application uptime in cloud-scale environments.



Pavilion supports multi-path IO with the NVMe-Over-Fabrics protocol. HA is provided for multiple failure points, including port, path, NIC, and controllers using Pavilion's Active-Active controllers with MPIO. This will work in either a direct-connect scenario, or through a switch.



With Pavilion Data's Hyper-Parallel Flash Array, every component is at least dual redundant, including network ports, SSDs, internal PCIe fabric, IO line cards, supervisor modules, power supplies, and fans. Likewise, all components in the chassis are hot-swappable for maximum serviceability, including SSDs, IO line cards, supervisor modules, PCIe fabrics, fans, and power supplies.

Unlike DAS where a server and its disks are a single point of failure, Pavilion implements data volumes provisioned from a drive group containing up to 18 NVMe SSDs in a RAID-6 configuration. This ensures that up to two drives can fail without interrupting application access to data. The entire system contains up to 4 zones of media, each with its own independent RAID group.

SCALABILITY

Pavilion's NVMe-oF is a 4 Rack-Unit chassis that delivers high performance at rack-scale. With up to 72 Standard-Format 2.5" U.2 NVMe SSDs, up to 20 Active-Active storage controllers, and up to 40 100 GbE network ports the design offers unparalleled density in a storage system in terms of both capacity and performance.

Storage controllers and network ports are delivered on modular "IO Line Cards" which can be added (independent of capacity) as-needed to scale performance and connectivity. Each IO Line card has two independent storage controllers, each with its own memory and copy of the operating system. The IO Line Cards connect to the NVMe drive array through an internal, multi-terabit PCIe switched network.

In a nutshell, we offer the performance, agility and cost factors of Direct Attached SSD/NVMe with all the operational benefits and economies of Shared Storage resources. Some of the most important scalability requirements that the array meets are:

- Delivers 120s of GB/s of bandwidth and 20million IOPS in 4U or less, effectively powering racks of
- clustered servers with shared storage and allowing DAS SSDs to be removed from the servers
- Offers up to 920 TB Usable in a single 4U System.
- Allows Performance and Capacity to be scaled independently within the same chassis

DEPLOYING KUBERNETES WITH PAVILION

Pavilion provides Kubernetes integration, allowing for containerized applications to dynamically allocate shared storage resources served from a centralized storage array over the network.

Pavilion's array can be deployed by connecting it to Kubernetes application servers by a common Ethernet network, using the NVMe-oF block storage protocol. The Ethernet network can be RDMA-capable, or basic TCP.

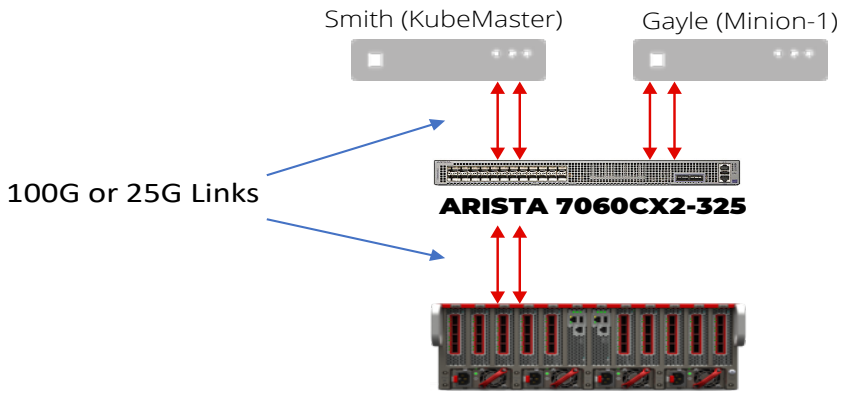
PERFORMANCE

Pavilion provides high-bandwidth, low latency storage for Kubernetes. A Pavilion system can produce 120 GB/s of read bandwidth, and 20 million 4K Read IOPS. The configuration outlined in this paper is not large enough to generate this kind of throughput given that it is a small example. To obtain this type of performance, a large-scale Kubernetes environment needs to be deployed. A Pavilion System can support up to 4096 containers.

The overall throughput mentioned above can be spread across all of the containers hosted by the chassis at any given time. In addition, latencies of single IOs inside a container can be as low as 40 microseconds.

KUBERNETES STORAGE PROVISIONING STEPS

In this example, a 2-node cluster is set up that looks like this:



A Kubernetes cluster can be created using multiple nodes. One node is the master (KubeMaster) and the other nodes are called Minions. In this example, the host 'Smith' is the master, and the host 'Jones' is a minion.

1. Install the Pavilion FlexVolume Plugin on each node of the cluster
2. Install the PVLCTL Management Utility on the Master node
3. Create a volume of the desired size on the Pavilion chassis.
4. Create a Persistent Volume (PV) specification file, which has all the mapping data required to map storage to the volume created in Step 1.
5. Create a Persistent Volume Claim (PVC) specification file which maps to the PV specification file defined in Step 2.
6. Once the PVC bound to the PV, the cluster nodes can create a containerized application which can access the storage using the Persistent Volume (PV). This is done by instantiating the containerized application, mapping to the PVC.
7. The required container image/s are then pulled from the container repository.
8. Once the container is launched, the Pavilion FlexVolume pvlnvmefv plugin is invoked.
9. On successful completion of the discovery of the underlying target volume, the block device is formatted to the desired file system in the PV spec, and appropriately mounted.

SETTING UP THE KUBERNETES CLUSTER TO UTILIZE PAVILION STORAGE

The steps to follow to set up the Kubernetes cluster and leverage Pavilion storage are listed below:

1. A two-node cluster, Smith and Gayle, are set up. The node information is listed below:

```
[root@smith data]# kubectl get nodes -o wide

NAME   STATUS ROLES  AGE
VERSION       CONTAINER-RUNTIME   VERSION       INTERNAL-IP   EXTERNAL-IP   OS-IMAGE      KERNEL-
gayle.pds.local     Ready         <none> 21d 3.10.0-862.el7.x86_64  docker://1.13.1     v1.14.2
172.25.50.50  <none> CentOS Linux 7 (Core)
smith.pds.local     Ready         master 21d 3.10.0-693.el7.x86_64  docker://1.13.1     v1.14.1
172.25.50.48  <none> CentOS Linux 7 (Core)
```

2. Instructions for deploying and configuring the Pavilion FlexVolume Plugin are listed below. This needs to be done on all of the nodes in the Kubernetes cluster. This shows the details for the host 'Smith'. Obtain the `kubernetes_flexvol_ worker_nodes_pavilion_setup 2.x.x.x.tar.gz`

a) Define the protocol used to communicate with the Pavilion system over the Ethernet network. The choices are RDMA (Block), TCP(Block), or NFS(File).

b) Select the host ports that will be used to communicate with the Pavilion system.

```
[root@smith pvl-fv-plugin]# ./pvl-k8s-fv-plugin Creating directory pvl-fv-self-extractable-tar
Verifying archive integrity... All good.
Uncompressing PavilionData Kubernetes FlexVolume Plugin......
================================================================================ PAVILION DATA KUBERNETES
FLEX-VOLUME CONFIGURATION (v1.0.0.dev7)
================================================================================
******************************************************************** Running system pre-check
to determine presence of required utils
******************************************************************** SUCCESS: : All required
system utilities and system commands present.
********************************************************************  SUCCESS:: PavilionData
Kubernetes FlexVolume Plugin has been deployed successfully. Further start the plugin configuration
...
********************************************************************************
USAGE INSTRUCTIONS AND NOTES
1 Define Initiator network paths for accessing Volumes on Pavilion Data Chassis using either over NFS
or on NVME over RDMA,or NVME over TCP transport.
2. Each NW interface configuration can involve one or more NW ports depending on application HA
requirements.
3. It is recommended that all the K8s nodes configured to access the the volumes on Pavilion Data
Chassis have the same subnet configuration.
4. Only IPv4 Ports available for external communication can be setup
for further configuration.
********************************************************************************

================================================================================ NETWORK CONFIGURATION FOR
ACCESSING PAVILION DATA CHASSIS
================================================================================
K8s Nodes can communicate to Volumes on Pavilion Data Chassis either over FS or on NVME over RDMA
Transport or NVME over TCP Transport.
1. RDMA
2. TCP
3. NFS
Please select the Transport Protocol:[1/2/3]1
```

```
Select all NW Ports on this Host through which Volumes on Pavilion Data Chassis shall be accessed.

Access Chassis over NW Port      eth0   [Y/N]N
Access Chassis over NW Port      eth4   [Y/N]Y
Access Chassis over NW Port      eth5   [Y/N]N
Access Chassis over NW Port      docker0      [Y/N]N
Access Chassis over NW Port      weave  [Y/N]N

Logical network configs need to be defined further.
**************************************************************************
Enter Logical Name for NW Configuration : demo-nw Auto-configuring, as only single port selected
**************************************************************************
Preferred List selection
Only one logical config enabled; using that for further config.
**************************************************************************
SUCCESS: NW config for Pavilion Flex Volume plugin completed successfully
**************************************************************************
```

3. Installation of PVLCTL management utility

This utility will create the underlying storage volumes on the Pavilion chassis, which are leveraged by containerized applications. This will also facilitate taking snapshots and clones so that they can be used to provision new containers.
Obtain the `kubernetes_pvlctl_master_node_pavilion_setup 2.3.0.0.tar.gz` bundle from Pavilion support team. Extract it and install it as follows.

```
[root@smith pvlctl-util]# ./pvl-k8s-pvlctl-plugin Creating directory pvl-pvlctl-self-extractable-tar
Verifying archive integrity... All good.
Uncompressing Pavilion Data Kubernetes pvlctl Plugin.............................
========================================================================== PAVILION DATA KUBERNETES
pvlctl CONFIGURATION (v1.0.0.dev3)
==========================================================================
************************************************************************** Running system pre-check
to determine presence of required utils
************************************************************************** SUCCESS: All required
system utilities and system commands present.
************************************************************************** SUCCESS: Pavilion Data
Kubernetes pvlctl Plugin deployed successfully.
**************************************************************************
```

PROVISIONING STORAGE FOR CONTAINERS

Once the cluster is set up with the Pavilion FlexVolume plugin and the PVLCTL management utility, storage can be provisioned and leveraged by containerized applications. The high level steps are:

a) Create a Pavilion Volume
b) Create a Kubernetes PersistentVolume (PV) mapped to the Pavilion Volume
c) Create Kubernetes PersistentVolumeClaim (PVC) mapped to the Persistent Volume

Note that once installed, the pvlctl executable can be found in the /etc/pavilion directory on the master node. The syntax for the PVLCTL is listed below.

```
[root@smith pavilion]# pwd
/etc/pavilion
[root@smith pavilion]# ./pvlctl
NAME:
pvlctl - Pavilion Kubernetes CLI Utility

USAGE:
pvlctl [global options] command [command options] [arguments...]

VERSION:
1.0.0.dev8

COMMANDS:
create Creates a new storage entity and corresponding PV, PVC Claim Entity [IN] Pavilion Storage
Spec File

[OUT] Status, Message (PVC if successful)
view   View storage properties associated with PVL PVC [IN] Pavilion PVC Name
[OUT] Status, Message
delete Deletes a PVL PVC Claim and associated PV with linked storage entity [IN] Pavilion PVC Name
[OUT] Status, Message
help, h Shows a list of commands or help for one command

GLOBAL OPTIONS:
--help, -h    show help
--version, -v print the version
```

Create Pavilion Volume and Kubernetes logical volume objects (PV, PVC) using the PVLCTL utility ‘create’ command.

Once the cluster is set up with the Pavilion FlexVolume plugin and the PVLCTL management utility, storage can be provisioned and leveraged by containerized applications. The high level steps are:

a) Create a Pavilion Volume
b) Create a Kubernetes PersistentVolume (PV) mapped to the Pavilion Volume
c) Create Kubernetes PersistentVolumeClaim (PVC) mapped to the Persistent Volume

Note that once installed, the pvlctl executable can be found in the /etc/pavilion directory on the master node. The syntax for the PVLCTL is listed below.

```
[root@smith pavilion]# pwd
/etc/pavilion
[root@smith pavilion]# ./pvlctl

[root@smith pavilion]# pwd
/etc/pavilion
[root@smith pavilion]# ./pvlctl NAME:
pvlctl - Pavilion Kubernetes CLI Utility
USAGE:
pvlctl [global options] command [command options] [arguments...]
VERSION:
1.0.0.dev8
COMMANDS:
create Creates a new storage entity and corresponding PV, PVC Claim Entity [IN] Pavilion Storage
Spec File
[OUT] Status, Message (PVC if successful) view View storage properties associated with PVL PVC
[IN] Pavilion PVC Name [OUT] Status, Message
delete Deletes a PVL PVC Claim and associated PV with linked storage entity [IN] Pavilion PVC Name
[OUT] Status, Message
help, h Shows a list of commands or help for one command
GLOBAL OPTIONS:
--help, -h    show help
--version, -v print the version
```

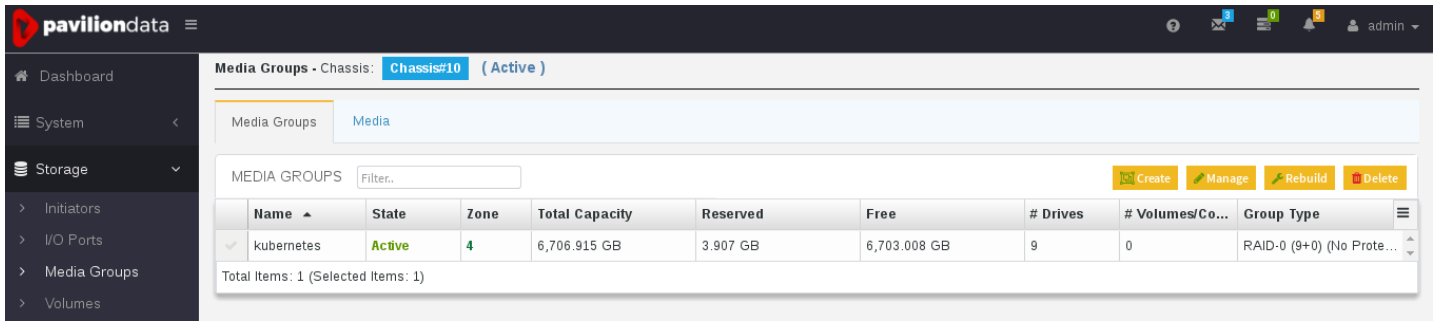
Create a Pavilion Volume and Kubernetes logical volume objects (PV, PVC) using the PVLCTL utility 'create' command.

```
[root@smith pavilion]# /pvlctl create --help NAME:
pvlctl create - Creates a new storage entity and corresponding PV, PVC Claim Entity [IN] Pavilion
Storage Spec File
[OUT] Status, Message (PVC if successful)
USAGE:
pvlctl create [arguments...]
```

Prepare Volume Creation Spec File using supplied volume creation template. The parameters of the volume will need to be specified as part of this step. The parameters are outlined below.

```
[root@smith pavilion]# cp dynamic_vol_creation/volume-template.yaml demo-volume.yaml [root@smith
pavilion]# cat demo-volume.yaml
apiVersion: v1 kind: pavilion metadata:
storageEntity: Volume spec:
storageProps: fsType: “ext4” options:
chassis: “172.25.50.41”
chassis_version: “2.0” uid: “operator”
pwd: “b30cfbf4765c76fba16840a6d692c03f6d0040da9705e7452db252b9a818031f” pvlvolsizeingb: “111”
pvlvolname: “demo-vol”
pvlmgname: “kubernetes”
```

For this example, a Pavilion Media group named “kubernetes” is being used to create volumes using the specification listed above. This media group needs to exist when the PVLCTL utility is invoked. The Pavilion GUI displaying the media group appears below.



Create a Persistent Volume using the defined volume specification.

```
[root@smith pavilion]# ./pvlctl create ./demo-volume.yaml SUCCESS: v-demo-vol-gb00001004bbf9110
{“status”:”Success”,”message”:”v-demo-vol-gb00001004bbf9110”}
```

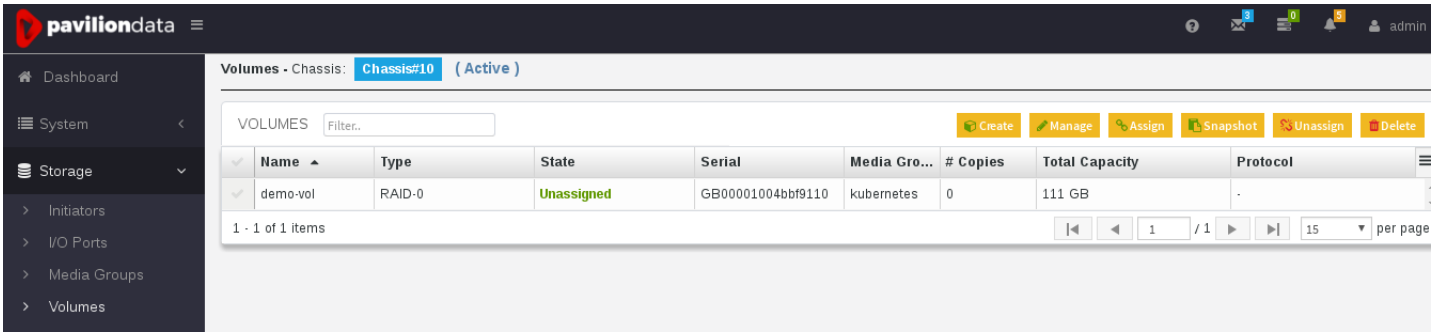
The command above successfully created a volume on the Pavilion chassis, using the required media group and also created the mapped PV and PVC objects which are now ready to be consumed by containerized applications.

The kubectl utility can be now used to view the PV and PVC objects that have been created.

```
[root@smith pavilion]# kubectl get pv -o wide
```

```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
v-demo-vol-gb00001004bbf9110               111Gi      RWX            Retain           Bound    default/v-demo-vol-
b00001004bbf9110   pavilion                                4m29s
[root@smith pavilion]# kubectl get pvc -o wide
NAME                                       STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
AGE
v-demo-vol-gb00001004bbf9110 4m33s         Bound    v-demo-vol-gb00001004bbf9110               111Gi      RWX            pavilion
```

The created volume is also displayed in the Pavilion UI as follows.



Similar to “Create Volume”, you can also execute “Create snapshot”, “Create Clone” functionality using the PVLCTL by using the
following templates provided with the utility:
snapshot-template.yaml : To create snapshot of pvl volume and attach to POD clone-template.yaml : To create clone of pvl-volume snapshot and attach to POD
A Pavilion Volume can also be deleted easily using the PVLCTL utility, which also removes the associated PV and PVC objects. Syntax and usage are outlined below.

```
[root@smith pavilion]# ./pvlctl delete --help NAME:
pvlctl delete - Deletes a PVL PVC Claim and associated PV with linked storage entity [IN]
Pavilion PVC Name
[OUT] Status, Message
USAGE:
pvlctl delete [arguments...]
```

DEPLOYING A CONTAINERIZED APPLICATION

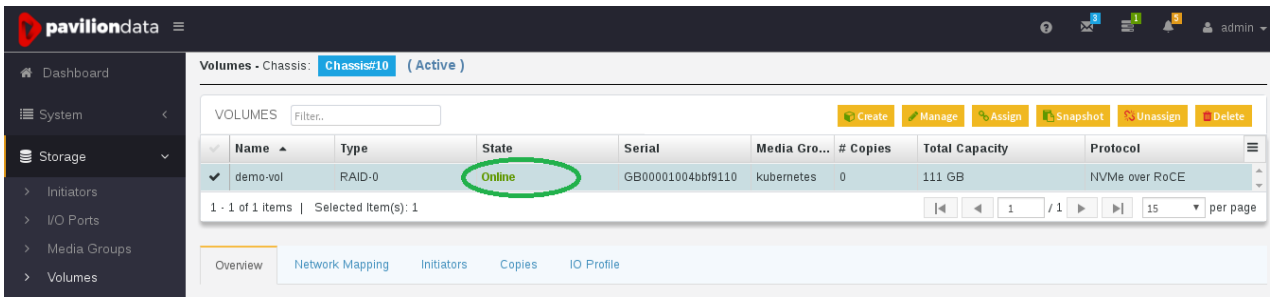
A sample NGINX POD is illustrated in this section, which leverages Pavilion Storage as created above. Here claimName is the
PVC object name given by "kubectl get pvc" displayed in the above example.

```
[root@smith pavilion]# cat nginx_sample_app.yaml apiVersion: v1
kind: Pod metadata:
name: nginx-pvl namespace: default
spec:
containers:
name: nginx-pvl image: nginx volumeMounts:
name: footest
mountPath: /pvl_vnvme_mnt ports:
containerPort: 80 volumes:
name: footest persistentVolumeClaim:
claimName: v-demo-vol-gb00001004bbf9110
```

Deploy a Kubernetes NGINX POD using the kubectl utility:

```
[root@smith pavilion]# kubectl create -f nginx_sample_app.yaml pod/nginx-pvl created
[root@smith pavilion]# kubectl get pv -o wide
NAME                 CAPACITY     ACCESS MODES  RECLAIM POLICY      STATUS
CLAIM STORAGECLASS   REASON AGE
v-demo-vol-gb00001004bbf9110      111Gi  RWX    Retain Bound  default/v-demo-vol- gb00001004bbf9110
pavilion                    152m
[root@smith pavilion]# kubectl get pvc -o wide
NAME   STATUS VOLUME CAPACITY     ACCESS MODES  STORAGECLASS
AGE
v-demo-vol-gb00001004bbf9110      Bound  v-demo-vol-gb00001004bbf9110      111Gi  RWX    pavilion 152m
[root@smith pavilion]# kubectl get pods -o wide
```

Note Volume Display in the Pavilion UI now shows the volume is 'online'.



```
Using the Direct Volume definition in the POD itself
[root@smith pavilion]# cat nginx-pvl-fv.yaml apiVersion: v1
kind: Pod metadata:
name: nginx-pvl namespace: default
spec:
containers:
name: nginx-pvl image: nginx volumeMounts:
name: test
mountPath: /pvl_vnvme_mnt ports:
containerPort: 80
volumes:
name: test
flexVolume:
driver: “pavilion/pvlnvmefv” fsType: “ext4”
options:
chassis: “172.25.50.41”
chassis_version: “2.0” uid: “operator”
pwd: “b30cfbf4765c76fba16840a6d692c03f6d0040da9705e7452db252b9a818031f” pvlvolname: “manual”
pvlvolsizeingb: “111” pvlmgname: “demo-vol” pvlvolsecret: “”
```

Deploy the NGINX POD with Flex Volume info embedded in its definition

```
(root@smith pavilion]# kubectl create -f
nginx-pvl-fv.yaml pod/nginx-pvl created
```

Verify Pavilion Volume based Persistent Volume is attached to container. Execute below command on K8s node to check mount points within that particular POD name.

```
[root@smith pavilion]# kubectl exec -it nginx-pvl mount | grep pvl
/dev/nvme0n1 on /pvl_vnvme_mnt type ext4 (rw,relatime,data=ordered)
```

SUMMARY

Deploying K8s with Pavilion Data offers a simple, scalable and reliable method to modernize traditional workloads and create new solutions with scale-out applications. Our current approach delivers scalability and storage management features that cannot be achieved with traditional DAS or All-Flash Arrays. Of course, we are always striving to set the bar higher. Please contact your sales representative for our newest advances in Kubernetes and Containers.