The AuriStorFS KMOD/CSI Special Resource

Gerry Seidman
gerry@auristor.com

February 7, 2022

Agenda

- Understanding NFD, SRO and CSI in the context of AuriStorFS
- Discussion about AuriStorFS Cache Manager on OpenShift nodes
- Approaches for using Kubernetes Objects to mount CSI Volumes

Components Comprising SRO/CSI

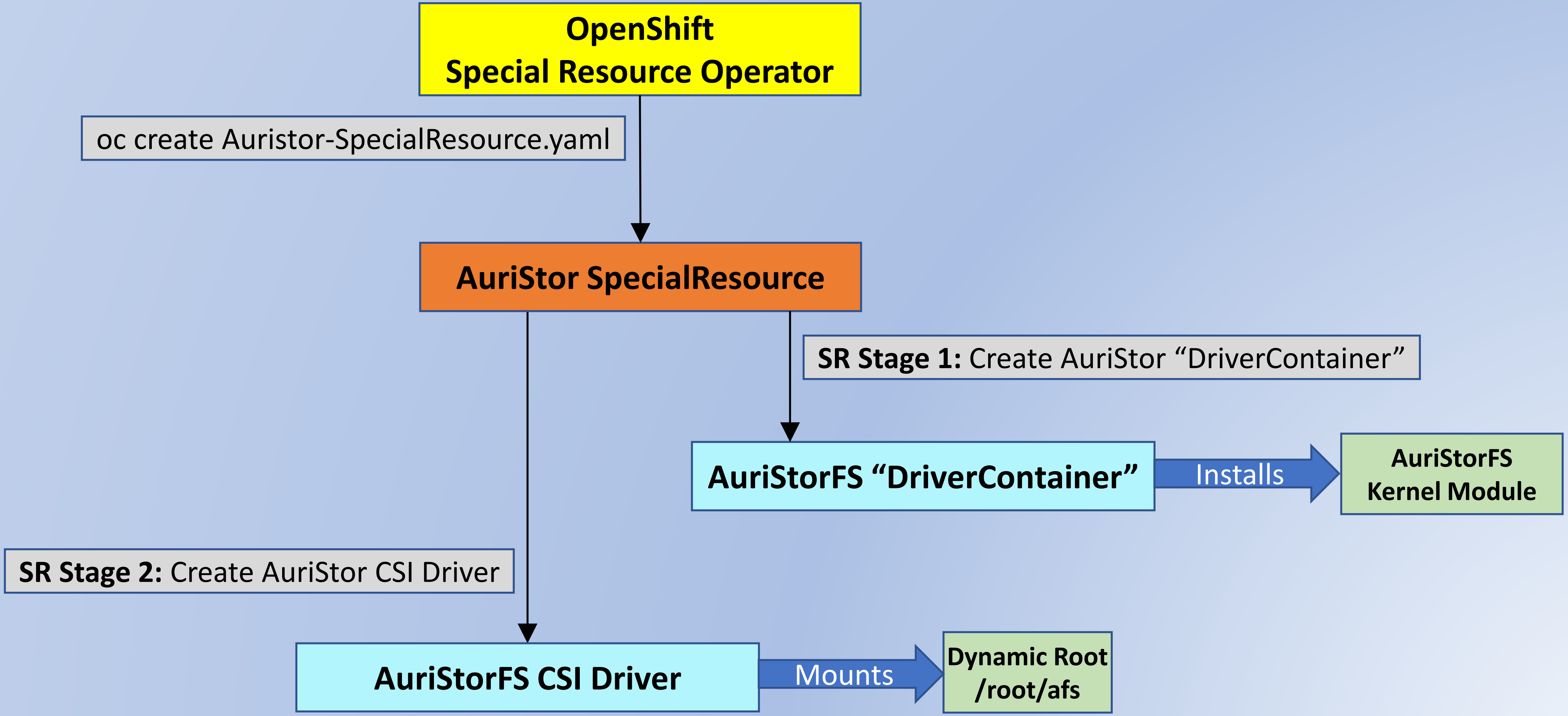
- Node Feature Discovery Operator (NFD)
  - OpenShift operator to publish Features such as Kernel Version

- Special Resource Operator (SRO)
  - OpenShift Operator to manage “Special Resources”
    - Installing/maintaining Vendor Kernel Modules

- AuriStorFS Special Resource (SR)
  - Installs/Maintains AuriStorFS Kernel Module
  - Installs/Maintains AuriStorFS CSI Driver

- AuriStorFS CSI Driver (CSI)
  - Allows Kubernetes Pods to mount AuriStorFS Volumes as Kubernetes Volumes

SRO Manages State Transitions for AuriStor SR


OpenShift
Special Resource Operator
oc create Auristor-SpecialResource.yaml
AuriStor SpecialResource
SR Stage 1: Create AuriStor "DriverContainer"
AuriStorFS "DriverContainer"
Installs
AuriStorFS
Kernel Module
SR Stage 2: Create AuriStor CSI Driver
AuriStorFS CSI Driver
Mounts
Dynamic Root
/root/afs

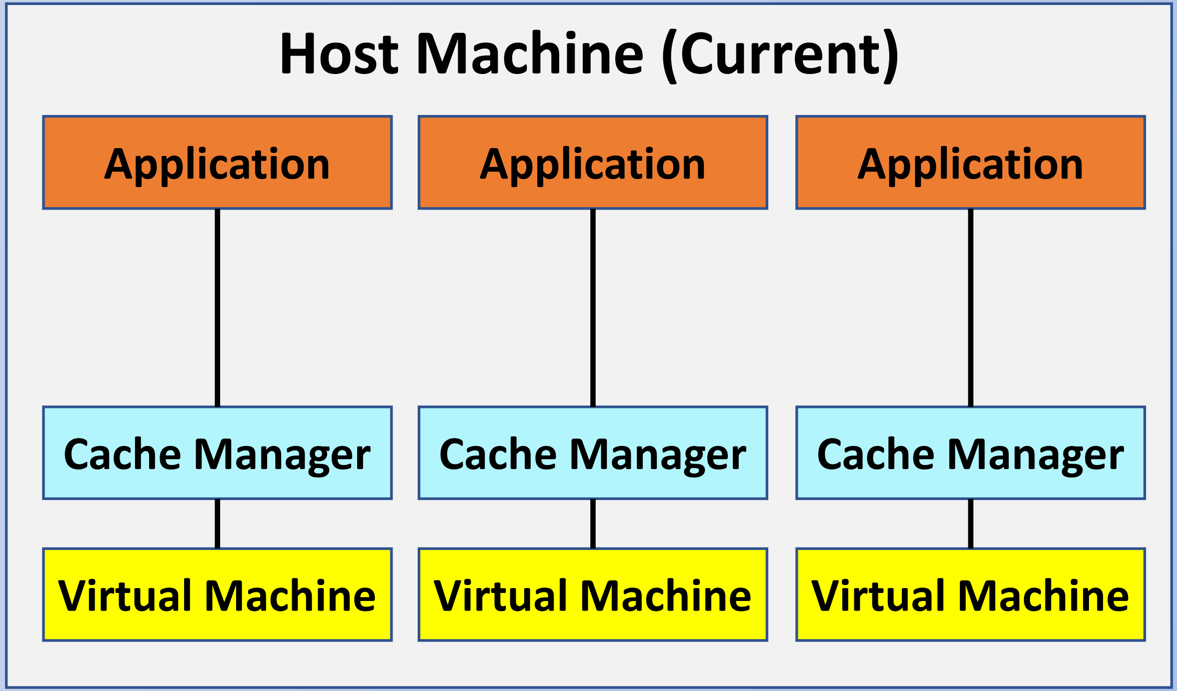
AURISTOR® THE GLOBAL NAMESPACE FILE SYSTEM

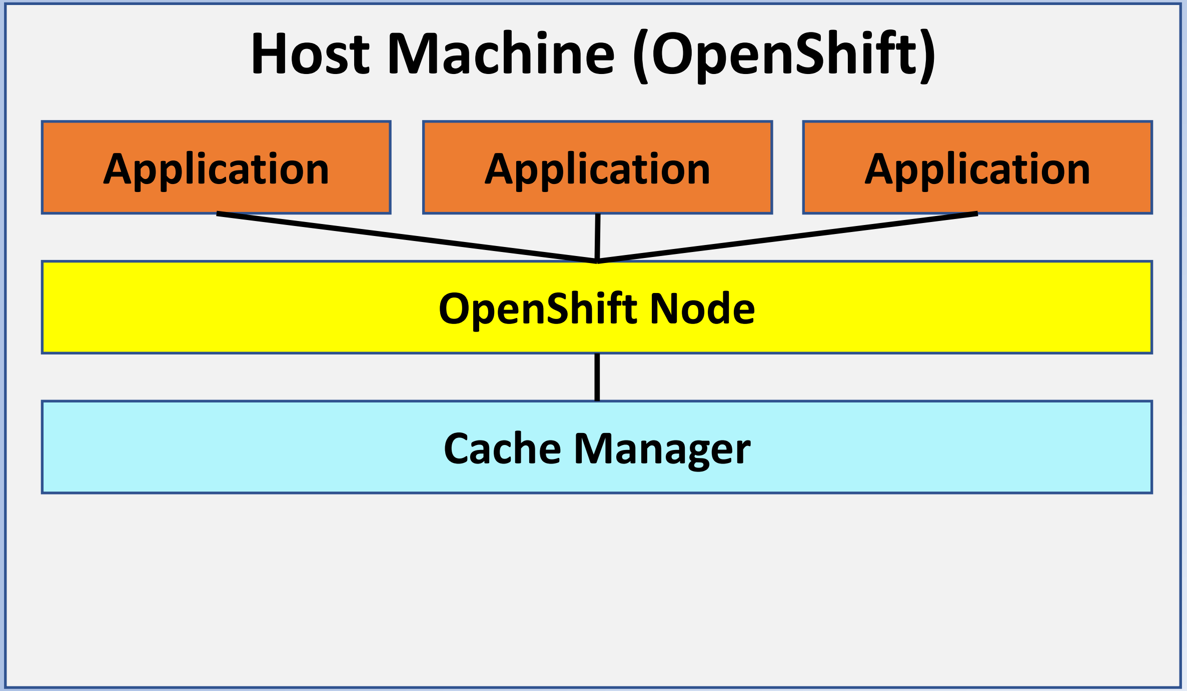
Installing NFD and SRO

- See OpenShift NFD Documentation
  - https://docs.openshift.com/container-platform/4.8/scalability_and_performance/psap-node-feature-discovery-operator.html#installing-the-node-feature-discovery-operator_node-feature-discovery-operator

- See OpenShift SRO Documentation
  - https://docs.openshift.com/container-platform/4.9/hardware_enablement/psap-special-resource-operator.html#installing-special-resource-operator

Virtual Machines vs OpenShift Node



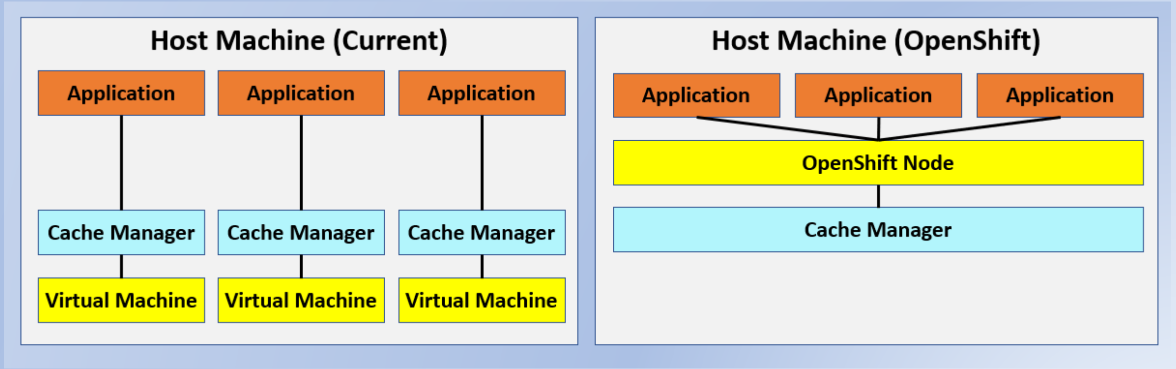


Conversation on Reduced Number of CMs

- How many VMs are going to be consolidated into a single OpenShift node?
  - There will be a reduction in the total number of Cache Managers

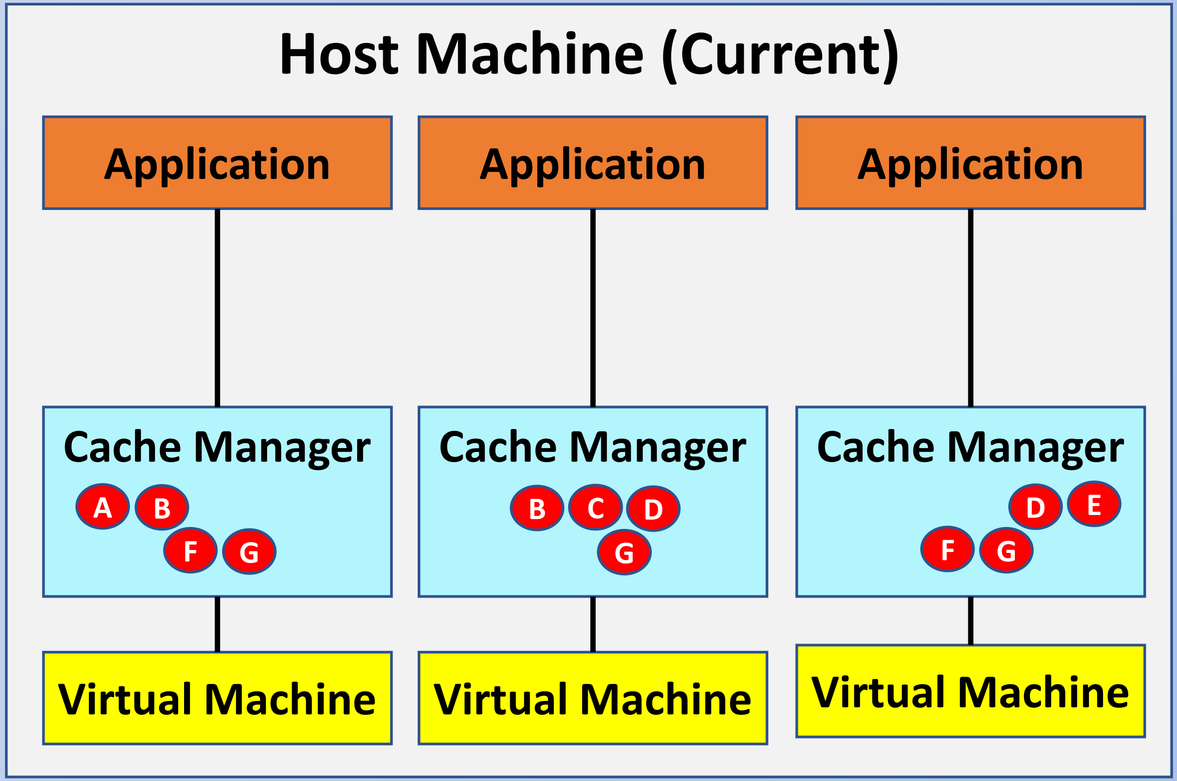
- There will be a decrease the number of clients that the volume location and file servers interact with.

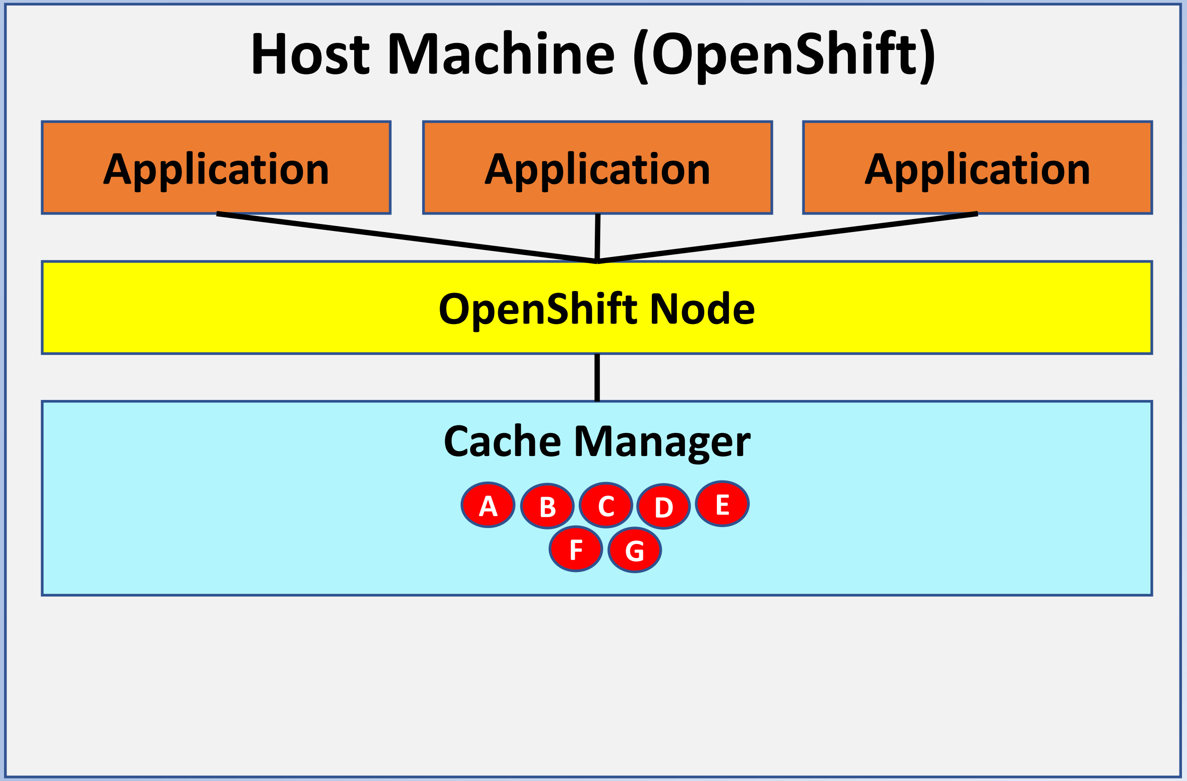
- Would reduce the number of data copies if multiple pods read the same data.



- OpenShift node cache manager will likely need to manage more files and more data then a single VM hosted cache manager.
  - Each OpenShift cache manager will manage more simultaneous rx connections/calls.

Discussion Regarding Disk Cache Size





Each Cache should be large enough to hold all the files used by the application running on that VM

The Shared Cache should be large enough to hold the union of all files used by the applications on that Machine

Note: The Disk Cache Size does not need to be the same on each Machine

Configuring Cache Manager with SRO

- Cache Manager Disk Storage
  - Dedicated Partition (Recommended)
  - Root File System

- yfs-client.conf
  - Same Configuration per OpenShift node?
  - In a Kubernetes ConfigMap

- cellServDb.conf
  - In a Kubernetes ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: yfs-client
data:
  yfs-client.conf: |-
    include /usr/share/yfs/cellservdb.conf

    [defaults]
      thiscell = your-file-system.com

    [afsd]
        blocks = 100000
        debug = yes
        # memcache = yes
        nomount = yes
```

AuriStor Volume from Inside Running Pod

```
$ oc exec -it pvx -- /bin/sh

sh-4.4# ls /data
In-GERKO  out  out2

sh-4.4# findmnt /data
TARGET SOURCE                          FSTYPE     OPTIONS
/data  AFS[/.@mount/auristor.io/gerko] auristorfs rw,relatime,context=system_u:object_r:nfs_t:s0

sh-4.4# fs examine /data
File /data (1048830.1.1) contained in volume 1048830
Volume status for vid = 1048830 named gerko
Current disk quota is 1000000
Current blocks used are 5
The partition has 11074240 blocks available out of 15718400
```

- Note that, if present in container, AuriStorFS Client Tools work inside Pods

Pod Volume ⬄ AuriStorFS Volumes

```
kind: Pod
apiVersion: v1
metadata:
  name: pvx
  namespace: demos
spec:
  terminationGracePeriodSeconds: 0
  containers:
    - name: simple
      image: alpine
      command: [ "sleep", "1000"]
      volumeMounts:
      - mountPath: "/data"
        name: data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: pvc-pvx
```

Statically Provisioned PersistentVolumes

```
kind: Pod
apiVersion: v1
metadata:
  name: pvx
  namespace: demos
spec:
  terminationGracePeriodSeconds: 0
  containers:
    - name: simple
      image: alpine
      command: [ "sleep", "1000"]
      volumeMounts:
      - mountPath: "/data"
        name: data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: pvc-pvx
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-pvx
  namespace: demos
spec:
  volumeName: pv-pvx
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 5Mi   # Ignored
  selector:
    matchLabels:
      volume: pv-pvx
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-pvx
  labels:
    volume: pv-pvx
spec:
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  capacity:
    storage: 5Mi    # IGNORED
  csi:
    driver: auristorfs.csi.auristor.com
    volumeHandle: gerko-pvx
    volumeAttributes:
      cell-name:  "auristor.io"
      volume-name: "gerko"
```

Note: PersistentVolumes are NOT Namespaced

Namespace Caveat

- PersistentVolumes are NOT Namespaced

- PersistentVolumeClaims ARE namespaced objects
    - PV/PVC Binds are exclusive

- Therefore
    - For AuriStorFS Volumes used in multiple namespaces
        - A PVC Object will be needed for AuriStor Volume used in each Namespace
        - A Separate statically allocated PersistentVolume Object will also be needed
            - With a globally unique name

Dynamically Provisioned Using StorageClasses

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: pvx
  namespace: demos
spec:
  terminationGracePeriodSeconds: 0
  containers:
    - name: simple
      image: alpine
      command: [ "sleep", "1000"]
      volumeMounts:
      - mountPath: "/data"
        name: data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: pvc-pvx
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-scx
  namespace: demos
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Mi   # IGNORED
  storageClassName: storage-class-scx
```

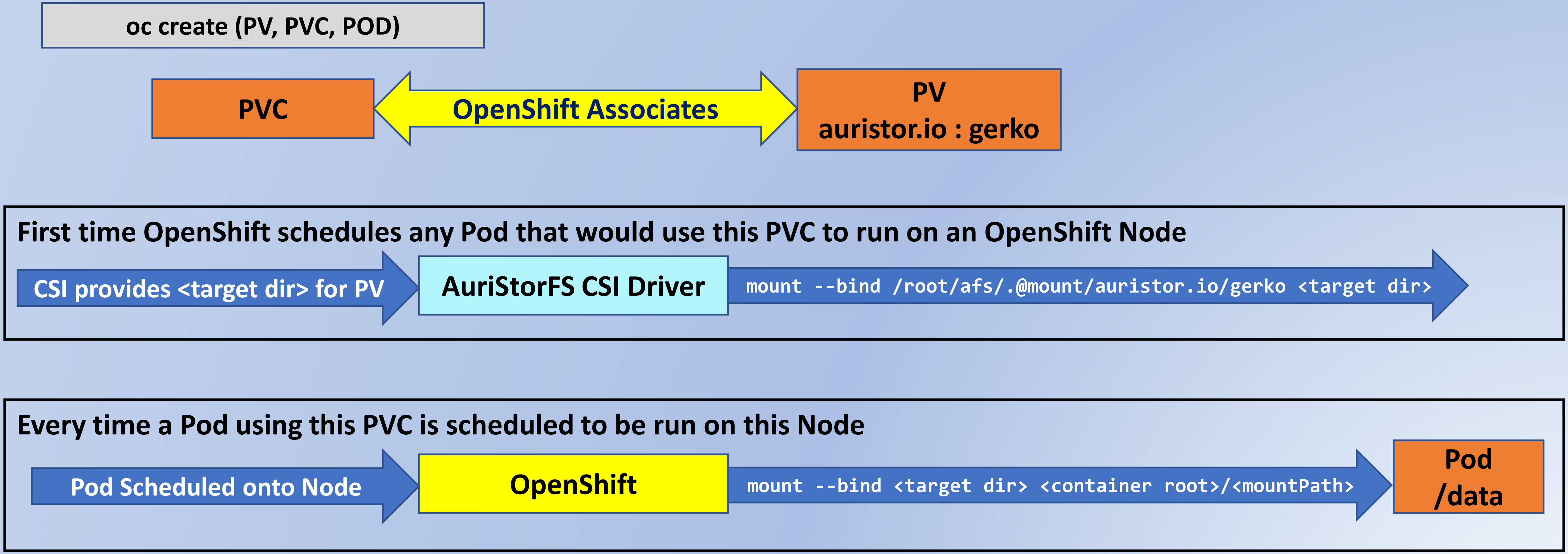
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-scx
provisioner: auristorfs.csi.auristor.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
parameters:
  cell-name:  "auristor.io"
  volume-name: "gerko"
```

Note: StorageClasses are NOT Namespaced

Ephemeral Volumes

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: evx
  namespace: demos
spec:
  terminationGracePeriodSeconds: 0
  containers:
    - name: simple
      image: alpine
      command: [ "sleep", "1000"]
      volumeMounts:
      - mountPath: "/data"
        name: data
  volumes:
    - name: data
      csi:
        driver: auristorfs.csi.auristor.com
        volumeAttributes:
          cell-name:  "auristor.io"
          volume-name: "gerko"
```

OpenShift and AuriStorFS Volume Mount Internals



oc create (PV, PVC, POD)

PVC

OpenShift Associates

PV
auristor.io : gerko

First time OpenShift schedules any Pod that would use this PVC to run on an OpenShift Node

CSI provides <target dir> for PV

AuriStorFS CSI Driver

```
mount --bind /root/afs/.@mount/auristor.io/gerko <target dir>
```

Every time a Pod using this PVC is scheduled to be run on this Node

Pod Scheduled onto Node

OpenShift

```
mount --bind <target dir> <container root>/<mountPath>
```

Pod
/data

AURISTOR®
THE GLOBAL NAMESPACE FILE SYSTEM

Options for Mapping Pod Mountpoints to AFS Volume

Method	Pros	Cons
Static PersistentVolumes (PV)	- Only Administrator Objects (PV) have AuriStorFS volume details/parameters	- PV Objects must be created - One PV necessary for each Namespaces
Dynamic PersistentVolumes (StorageClasses)	- Only Administrator Objects (StorageClass) have AuriStorFS volume details/parameters - A single StorageClass Object can be used for all Namespaces - PV Objects are dynamically created for each PVC by the CSI Driver	- Must go through the CSI Driver Controller 'createVolume' phase - Slightly slower Volume Mounting
Ephemeral Volumes	- No need for PVC, PV or StorageClass Kuberenetes Objects - Slightly faster Volume Mounting	- Not Administrator Controlled - User's Pod Objects must have AuriStorFS volume details/parameters

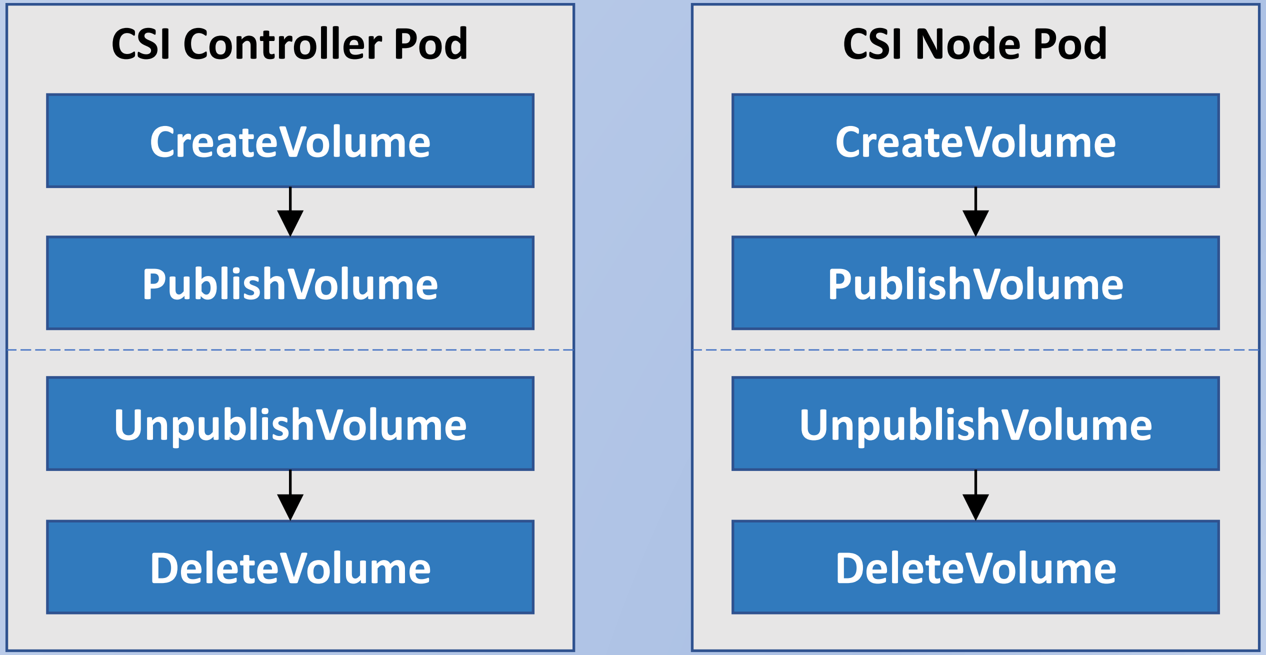
- For more information:
  - Static: https://kubernetes.io/docs/concepts/storage/persistent-volumes/
  - Dynamic: https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/
  - Ephemeral: https://kubernetes.io/docs/concepts/storage/ephemeral-volumes/

Understanding CSI Constituent Pods

- CSI Controller Pod (one primary per cluster)
  - One set of calls per PV
  - Management of OpenShift PV Objects
  - Optional: Pre-validation that the AFS Volume exists/online (pre-fail condition)
    - Pre-Fail that would prevent scheduling Pods that would try to use Volume

- Node Pod (one per node in cluster)
  - One set of calls per Node per PV
    - Only when first Application Pod using volume is scheduled onto OpenShift Node
  - Performs the actual mounting of the AFS Volume for on that node
  - Optional: Pre-validation that the AFS Volume exists/online (pre-fail condition)
    - Pre-Fail that would prevent starting of Pods that would try to use Volume

Understanding CSI/PV Lifecycle Interaction



CSI Call	Semantics
Controller.CreateVolume	Ready for AuriStorFS CSI Driver to create PV Object
Controller.PublishVolume	Ready to attach PV for use by a node
Node.StageVolume	Ready for First Pod Consumer for PV on this Node
Node.PublishVolume	Perform Mount for a specific Pod Consumer of PV

CSI Lifecycle for the Volume Mountpoint Options

CSI Call (6,7)	Static PV	Dynamic PV(8)	Ephemeral Volume (4,5)
CSI.CreateVolume (1,2)		X	
CSI.PublishVolume (1)	X	X	
Node.StageVolume (3)	X	X	
Node.PublishVolume (3)	X	X	X

(1): Optional check of exists/online status upon creation of PV may occur here

(2): 'CreateVolume' means CSI will create PV Object, not AuriStorFS Volume Creation (unless explicitly requested)

(3): Optional check of exists/online status upon PV scheduled for Pod may occur here

(4): No exist/online validation possible with Ephemeral Volumes

(5): No PV objects are created for Ephemeral Volumes. PV objects are useful for dashboards/monitoring

(6): Lifecycle and especially option validations effect Pod time to readiness

(7): Logging occurs with each CSI call that is made

AURISTOR®
THE GLOBAL NAMESPACE FILE SYSTEM

Questions?

Gerry Seidman
gerry@auristor.com