KubeCon | CloudNativeCon

North America 2017

Raw Block Volume in Kubernetes

Mitsuhiro Tanino, Principal Software Engineer, *Hitachi Vantara*

Agenda

- Who am I
- Background
- Raw Block Volume Support
- Usage of Raw Block Volumes
- Implementation deep dive
- Future Work

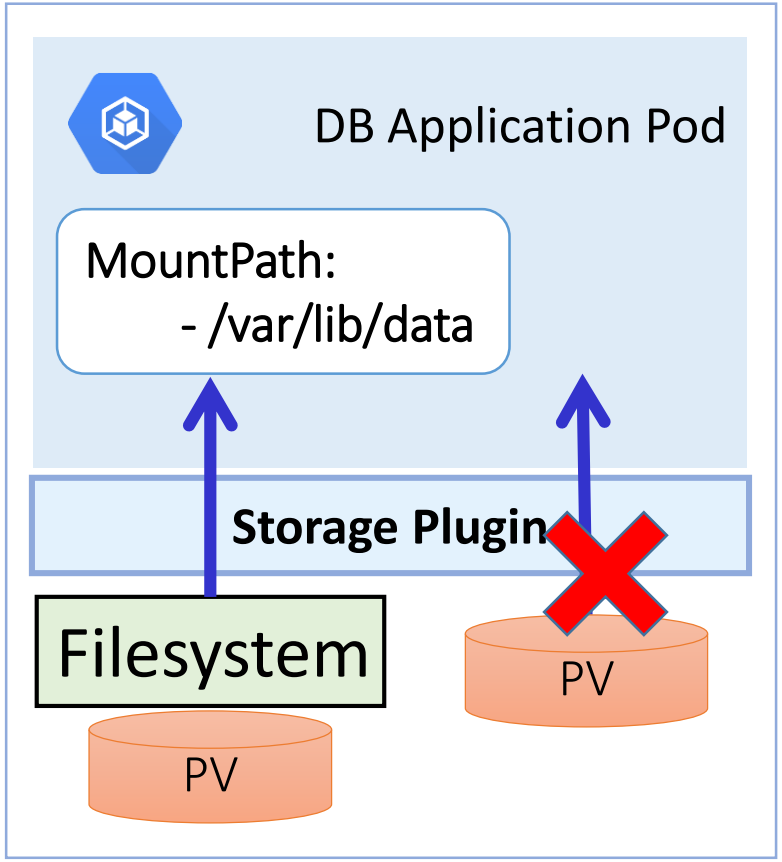
※Slide is available from Kubecon’s sched site.

Who am I

- Software engineer at Hitachi Vantara(previously Hitachi Data Systems)
- Linux support service team for 8 years, especially cpu, memory and timer related area.
- Contributed OpenStack Cinder project for three years. I’ve done many bugfixes and added few features.
- From this year, I started to contribute Kubernetes sig-storage, I’m trying to enhance reliability and stability for iSCSI and FC drivers and also contributing Block Volumes Support feature development.

Background

- By using Kubernetes storage feature, user can create/delete persistent volume with their pod.
- When pod with volume is created, storage plugin creates a volume and create a filesystem on top of it. Therefore, currently these volumes must be consumed via filesystem inside the pod.
- This means that user can’t consume raw block volume directly from their applications without filesystem.



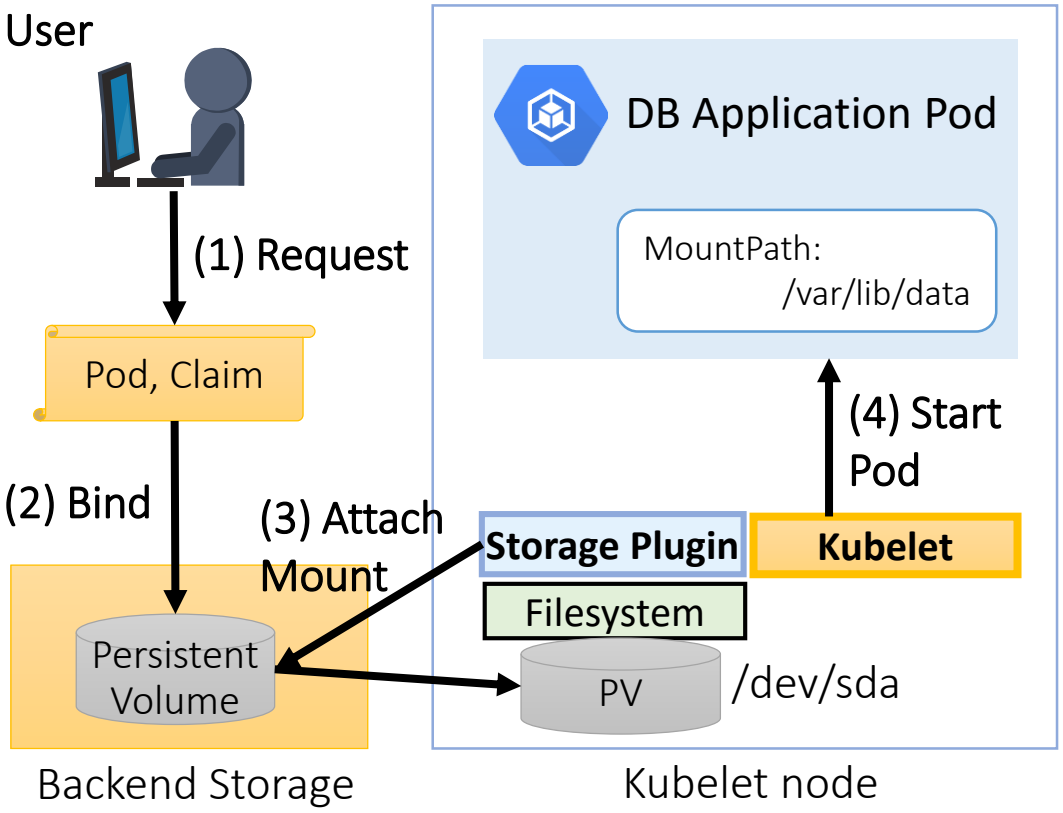
Background

- Benefits of raw block volume are;
  - Enable users to choose appropriate type of volume for their applications such as ***'MariaDB', 'HiRDB'(*1)***, etc.
  - Provide consistent I/O performance and low latency compared to filesystem volume, especially enterprise '***Fibre Channel', 'iSCSI'*** storage and ***'Local SSD'*** disk are suitable for raw block volume use-case.
  - At production system, raw block is essential functionality.
  - In addition, Container Storage Interface(CSI) defines to support both file volume and block volume capabilities.(CSI plugin is alpha at v1.9)

*1: HiRDB: Hitachi's RDBMS

Problem

- **Current workflow: Pod with Filesystem Persistent Volume**



(1) User requests a Pod with Persistent Volume Claim

(2) PV-Binder binds requested PVC and pre-provisioned Persistent Volume

(3) Storage plugin attaches PV to Kubelet node. PV is recognized as /dev/sda on node. Also ***plugin creates filesystem on top of /dev/sda if not exist,*** then mount it to root FS on kubelet node,

(4) Kubelet starts an application Pod. During pod creation, ***the mount point is passed into container's mountPath: /var/lib/data***

Problem

- **Current workflow: Pod with Filesystem Persistent Volume**

Following components can't handle block type of volume.
They only support Filesystem type volume at Kubernetes v1.8.

- Persistent Volume, Persistent Volume Claim
- kube-apiserver, kube-controller-manager, kubelet
- Storage Plugin

Problem

- Today's workarounds
  - When user creates a pod with privileged option, all block devices on kubelet node are exposed to a container.
  - In order to use privileged option, it requires 'ALLOW_PRIVILEGED=true' for Kubernetes cluster.
  - This isn't recommended way since privileged container may cause serious security problem.

```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  containers:
    - name: privileged-container
      # The container definition
      # ...
      securityContext:
        privileged: true
```

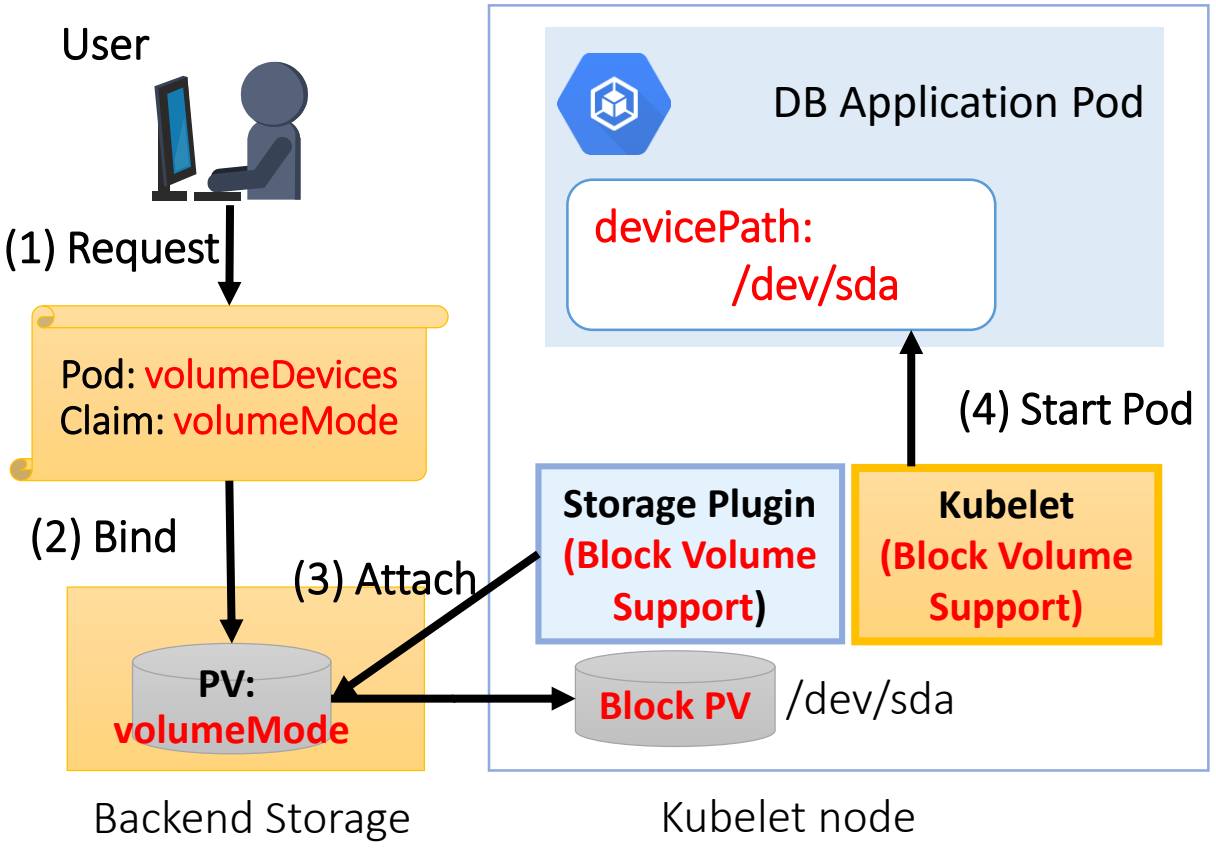
Raw Block Volume Support

New feature for solution

- Raw Block Volume Support (v1.9, alpha feature)
  - New API support
    - ‘***volumeMode’*** API for ‘***PV’*** and ‘***PVC’***
    - ‘***volumeDevices’*** API for ‘***Pod definition’***
  - Plugin support
    - ***‘Fibre Channel plugin’*** supports raw Block Volume as a reference driver(Only support pre-provisioned PV)
  - Dynamic Provisioning support
    - Spec is under discussion.
    - But we confirmed our FC external-provisioner (not OSS) works together with updated FC plugin.

New feature for solution

- **New workflow: Pod with Block Persistent Volume**



(1) User requests a Pod with ***volumeDevices***, and PVC with ***volumeMode = Block***

(2) PV-Binder binds requested PVC and pre-provisioned PV with ***volumeMode = Block***

(3) Storage plugin attaches the PV to Kubelet node. PV is recognized as ***/dev/sda*** on node. ***Plugin doesn't create filesystem.***

(4) Kubelet starts a DB application Pod. During pod creation, the ***Block PV is passed to the Pod as /dev/sda*** (or any user defined devicePath)

Usage of Raw Block Volumes

Configuration for services

- Enable feature-gates
  - Block Volume is '***alpha'*** feature at v1.9
  - Admin needs to configure ***'--feature-gates=BlockVolume=true'*** for services('kubelet', 'kube-api-server', 'kube-controller-manager', etc) to enable BlockVolume
- Without using feature-gates definition, volumeMode and volumeDevices API fields are not accepted.

Block Volume Definition

- **Persistent Volume definition**

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: block-pv001
spec:
 capacity:
   storage: 1Gi
 accessModes:
   - ReadWriteOnce
volumeMode: Block
 persistentVolumeReclaimPolicy: Retain
 fc:
   targetWWNs: ['28000001ff0414e2']
   lun: 0
```

- The '***volumeMode***' API filed supports two volume modes
  - ***Filesystem***
  - ***Block***
- Admin can define expected usage of volume through volumeMode
- If volumeMode is not specified, the PV is treated as filesystem volume as same as existing behavior

Block Volume Definition cont.

- **Persistent Volume Claim definition**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: block-pvc001
spec:
 accessModes:
   - ReadWriteOnce
volumeMode: Block
 resources:
   requests:
     storage: 1Gi
```

- The '***volumeMode***' API field supports two volume modes
  - ***Filesystem***
  - ***Block***
- User can define an expected mode of volume through PVC's volumeMode.
- If volumeMode is not specified, PVC requests a volume with volumeMode=Filesystem as same as existing behavior.

Block Volume Definition cont.

- **Pod definition**

```
apiVersion: v1
kind: Pod
metadata:
 name: blockvolume-pod
spec:
 containers:
   - name: blockvolume-container
     # The container definition
     # ...
     volumeDevices:
       - name: data
         devicePath: /dev/xvda
 volumes:
   - name: data
     persistentVolumeClaim:
       claimName: block-pvc001
       readOnly: false
```

- The '***volumeDevices***' API field is supported to define block volume in pod.
  - ***'name'*** is volumes name.
  - ***'devicePath'*** is a device path inside container.
- '***readOnly***' parameter can be define in Pod definition. If this parameter is true, The volume is passed as read only block volume.
- These two combination of parameters are available.
  - ***volumeDevices*** and ***volumeMode*** '***Block***'
  - ***volumeMounts*** and ***volumeMode*** '***Filesystem***'

VolumeMode binding rule for pre-provisioned PV

- To support raw block volume, we added new binding rule into persistent volume controller.
- There are three conditions of volumeMode, unspecified, Filesystem and Block.
- Unspecified is handled as Filesystem to keep compatibility with existing behavior
- If a user requests a raw block volume through the PVC.volumeMode field, it can only bind to PV which has same volumeMode.

  'pvc.volumeMode' == 'pv.volumeMode'.

#	PV volumeMode	PVC volumeMode	Result
1	unspecified	unspecified	BIND
2	unspecified	Filesystem	BIND
3	unspecified	Block	NO BIND
4	Filesystem	unspecified	BIND
5	Filesystem	Filesystem	BIND
6	Filesystem	Block	NO BIND
7	Block	unspecified	NO BIND
8	Block	Filesystem	NO BIND
9	Block	Block	BIND

Quick glance of block PV/PVC

- New field ‘VolumeMode’ is shown in ‘describe PV/PVC’
- volumeMode is shown only if feature-gate is enabled.

```
% kubectl describe pv/block-pv001
Name:            block-pv001
Labels:          <none>
Annotations:     pv.kubernetes.io/bound-by-controller=yes
StorageClass:    fast
Status:          Bound
Claim:           default/block-pvc001
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Block
Capacity:        1Gi
Message:
Source:
 # The storage definition
 # ...
```

```
% kubectl describe pvc/block-pvc001
Name:          block-pvc001
Namespace:     default
StorageClass:  fast
Status:        Bound
Volume:        block-pv0001
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed=yes
               pv.kubernetes.io/bound-by-controller=yes
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Block
Events:        <none>
```

Quick glance of running pod with block volume

- New field ‘Devices’ is shown in ‘describe pod’
- This shows devicePath inside the container and volume name

```
% kubectl describe po/blockvolume-pod
Name:         block-pod
Containers:
    # The container definition
    # ...
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-kdjq8 (ro)
    Devices:
      /dev/xvda from data
Volumes:
  data:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  block-pvc001
    ReadOnly:   false
```

Quick glance of running pod with block volume

- Device ‘/dev/xvda’ is created inside the pod.
- User can issue I/O to block device directly inside the pod.

```
% kubectl exec -it blockvolume-pod -- ls -la /dev/
total 0
drwxr-xr-x  5 root root     400 Oct  8 22:23 .
drwxr-xr-x 21 root root     242 Oct  8 22:23 ..
...
crw-rw-rw-  1 root root  1,   9 Oct  8 22:23 urandom
brw-rw----  1 root disk  7,   2 Oct  8 22:23 xvda
crw-rw-rw-  1 root root  1,   5 Oct  8 22:23 zero

% kubectl exec -it blockvolume-pod -- dd if=/dev/zero of=/dev/xvda bs=1024k count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.188629 s, 556 MB/s
```

Implementation deep dive

Implementation deep dive

- **Kubernetes Volume Plugin interface**
- **Directories under kubelet node**
  - Pod volume directory
  - Plugin directory
- **Avoid silent volume replacement**

Kubernetes Volume Plugin interface

- **Implement golang interfaces**
  - Mounter / Unmounter (basic function to mount / unmount volume)

  - **Optionally**
    - Attacher / Detacher (if plugin has cloud provider)
    - Provisioner / Deleter (if plugin provides dynamic provisioning)
    - Recycler (if plugin provides volume scrubbing/recycling feature)
    - **BlockVolumeMapper**
    - **BlockVolumeUnmapper**

New interfaces

Kubernetes Volume Plugin interface

- **Mounter/Unmounter interface (filesystem volume)**
  - Make data source(volume, block device, network share, etc) available as a directory on kubelet node's root Filesystem directory.
  - Then volume is mounted into container by kubelet's container runtime interface
  - Methods always called from kubelet node(kubelet binary)
- **Methods**
  - SetUpAt(…) : Attach and Mount volume to kubelet node
  - TearDownAt(…) : Unmount and Detach volume from kubelet node
  - ..

Kubernetes Volume Plugin interface

- **BlockVolumeMapper/Unmapper interface (block volume)**
  - Make data source(volume, block device) available as block device on kubelet node.
  - Then the block device is passed into container by container runtime interface on kubelet node.
  - Methods always called from kubelet node(kubelet binary)
- **Methods**
  - SetUpDevice(…) : Attach volume to kubelet node…(*1)
  - TearDownDevice(…) : Detach volume from kubelet node…(*1)

(*1): If a plugin has attacher/detacher interface, these methods could be no operation because attach and detach operations are done by pv-controller.

Implementation deep dive

- **Kubernetes Volume Plugin interface**
- **Directories under kubelet node**
  - Pod volume directory
  - Plugin directory
- **Avoid silent volume replacement**

Directories under kubelet node

- **Role of directories under kubelet node**
  - Pod volume directory
    - This directory is used to mount a formatted volume per pod, then this directory is mounted into container.
  - Plugin directory
    - This directory is used to store plugin's specific configurations, and also formatted volume is mounted on this directory.
    - ex. iSCSI plugin and RBD plugin store their connection information to the plugin directory.

Using these information, cluster admin can find which block device is used on which application pod for trouble shooting.

Directories under kubelet node

- However, about the block volume case, the volume isn’t formatted. Therefore they can’t be mounted to these directories.
- Instead, kubelet stores symbolic link which is associated to block device(such as /dev/sdX) to these directories.
- Admin can find physical block device corresponding to PV/PVC by checking the Symbolic Link.

Pod volume directory on kubelet node with FC plugin

- **Mounter/Unmounter**
- Each pod has own pod volume directory.
  - */var/lib/kubelet/pods/{pod uuid}/kubernetes.io~fc/*
- Volume is formatted and mounted under volumeName dir.
  - */var/lib/kubelet/pod/{pod uuid}/kubernetes.io~fc/{pvName}/*
- If a pod has multiple volumes, multiple volume directories are created under pod volume directory.

- **BlockVolumeMapper/Unmapper**
- Each pod has own pod volume directory.
  - */var/lib/kubelet/pods/{pod uuid}/**volumeDevices**/kubernetes.io~fc/*
- Symbolic link associated to block device such as /dev/sdX is stored under pod volume directory.
  - */var/…/**volumeDevices**/kubernetes.io~fc/**{pvName symlink} -> /dev/sdb***
- If a pod has multiple volumes, multiple symbolic links with volume name are stored under the directory

Admin can find physical block device by checking **Mount Point**

Admin can find physical block device by checking the **Symbolic Link**

Plugin directory on kubelet node with FC plugin

- **Mounter/Unmounter**
- Each volume has own plugin directory.
  - */var/lib/kubelet/plugins/kubernetes.io/fc/{wwn-lun-0}/*
- Volume is formatted and mounted under plugin dir.
  - */var/lib/kubelet/plugins/kubernetes.io/fc/{wwn-lun-0}/*
- Even ff multiple pods use a same volume, only one plugin directory per volume is created.

- **BlockVolumeMapper/Unmapper**
- Each volume has own plugin directory.
  - */var/lib/kubelet/plugins/kubernetes.io/fc/**volumeDevices**/{wwn-lun-0}/*
- Symbolic link associated to block device such as /dev/sdX is stored under plugin directory.
  - */var/…/fc/**volumeDevices**/wwn-lun-0/**{pod uuid symlink} -> /dev/sdb***
- If multiple pods use a same volume, ***multiple symbolic links*** with pod uuid name are stored under the directory

Implementation deep dive

- **Kubernetes Volume Plugin interface**
- **Directories under kubelet node**
  - Pod volume directory
  - Plugin directory
- **Avoid silent volume replacement**

Avoid block volume silent replacement

- **Problem**
  - We found that container runtime doesn’t take a lock to attached block volume even if container is online. Therefore volume is possibly replaced silently to another volume even if application is issuing I/O to the block volume inside the container.

- **Solution**
  - We added a logic to open device file of Block volume such as /dev/sdX via **loopback device**. This takes device lock using file-descriptor, then user can avoid device silent replacement.
  - Cluster admin may notice that there are many loopback devices on kubelet node if user uses raw block volumes.

Future work

Future work

- Dynamic provisioning for raw block volumes, Need to finalize spec at v1.10
- Add e2e test cases
- Raw block support for remaining volume plugins:
  - Local volumes
  - GCE PD
  - AWS EBS
  - iSCSI and iSCSI external provisioner
  - RBD
  - Gluster
  - ....

Your contribution is welcome!

Wrap-up

- Raw Block Volume is introduced to v1.9 as alpha feature.
- New API field
  - '***volumeMode'*** API for '***PV'*** and '***PVC'*** **(Filesystem/Block)**
  - '***volumeDevices'*** API for '***Pod definition***' **(name and devicePath)**
- Plugin support
  - ***'Fibre Channel plugin'*** supports raw Block Volume as a reference driver
- New binding rule
  - PVC and PV are bound if 'pvc.volumeMode' == 'pv.volumeMode'.
- Dynamic Provisioning support is under discussion

References

- Spec
  - (#1265) Block Volume Support Spec…… [merged]
- API, Controller, Kubelet, etc changes
  - (#50457) API Change…….. [merged]
  - (#53385) VolumeMode PV-PVC Binding change…… [merged]
  - (#51494) Container runtime interface change, …….. [merged]
    volumemanager changes, operationexecutor changes
  - (#55112) Block volume: Command line printer update... [merged]
- - Block Volume support for Plugins
  - (#51493) FC plugin update…….. [merged]
  - ***(#54752) iSCSI plugin update……..*** ***-> ongoing, v1.10***
  - ***(#55899) AWS plugin update……..*** ***-> ongoing, v1.10***
  - ***(#56651) RBD plugin update……..*** ***-> ongoing, v1.10***
  - ***Local plugin update……..*** ***Spec update merged***

Disclaimer

- Kubernetes is a registered trademark of The Linux Foundation.
- MariaDB is a trademark of the MariaDB Foundation.
- GCE is a trademark of the Google Inc.
- Amazon Web Services is a trademark of Amazon.com, Inc. or its affiliates in the United States and/or other countries.
- Gluster is a trademark of the Red Hat, Inc.
- All other trademarks are the property of their respective owners.

HITACHI

Inspire the Next

Appendix: New interface for plugin: implementation matrix

#	Interface	Method	Summary
1	BlockVolumeMapper	SetUpDevice()	Attach volume to kubelet node
2	BlockVolumeUnmapper	TearDownDevice()	Detach volume from kubelet node
3	BlockVolumePlugin	NewBlockVolumeMapper()	Create blockVolumeMapper object
4		NewBlockVolumeUnmapper()	Create blockVolumeUnmapper object
5		ConstructBlockVolumeSpec()	Create volumeSpec
6	BlockVolume	GetGlobalMapPath()	Get global map path
7		GetPodDeviceMapPath()	Get pod device map path