Kubernetes CSI Volume Expansion using HSPC

Hitachi Storage Plug-in for Containers

Plug-in Development Group
Hitachi Ltd.,
May 2020

Overview

This document introduces how to expand volumes using Kubernetes' CSI Volume Expansion function when Hitachi Virtual Storage Platform is used as storage backend of Kubernetes environment.

Intended audience of this document is IT administrators, system architects, consultants, and sales engineers to assist in planning, designing, and implementing Hitachi storage with container solutions.

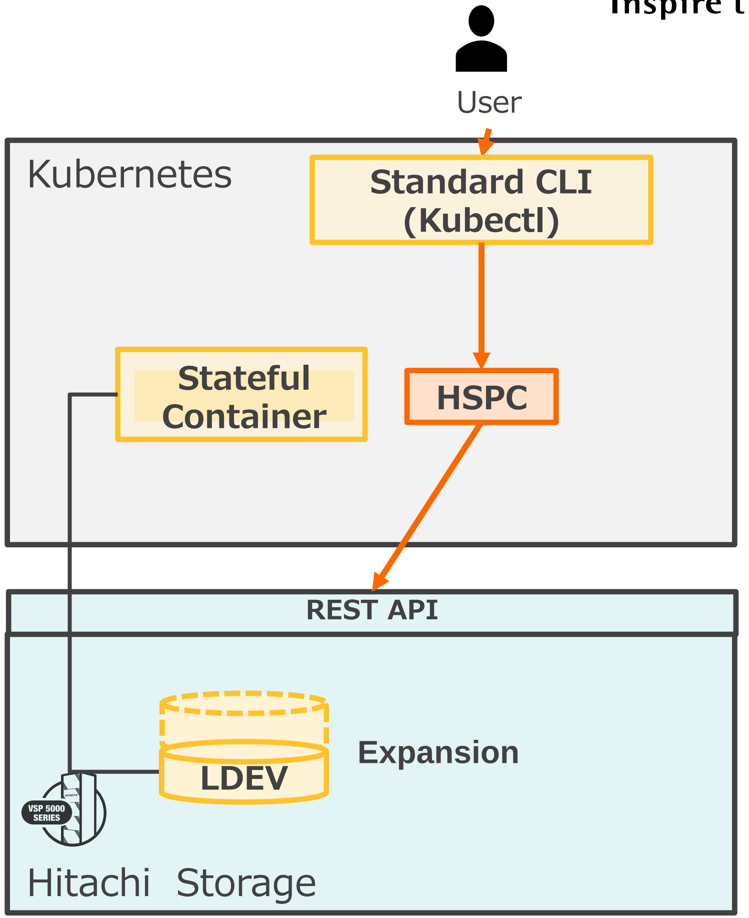
[Note] Kubernetes CSI Volume Expansion is still beta version in Kubernetes 1.16 - 1.18. This document is positioned as Technical Preview.

Table of Contents

1. Use case summary

As more companies are trying to innovate their services or business with the data, both speed and agility for Dev teams to utilize infrastructure resources are becoming important.

Dev teams can flexibly use storage resources by deploying volumes with minimum capacity and expanding later depending on the requirement from applications.



1. Use case summary

The requirements are supported by using Hitachi Storage Plug-in for Containers (HSPC).

This document describes how to expand a persistent volume for a stateful container.

PostgreSQL is used as an example of the stateful container.

- ✓ Users can easily expand an existing volume by editing Persistent Volume Claims (PVC) object.
- ✓ Users do not need to manually interact with the storage backend.
- ✓ Users can increase the size of a volume without delete and recreate.

1. Use case summary

HSPC supports both “Online expansion” and “Offline expansion”.

Online expansion : Expanding Persistent Volumes (PVs) which are being consumed by running Pods.

Offline expansion: Expanding PVs which are not attached to any Pods.

This document will focus mainly on “Online expansion”.

2. Prerequisite

The following describes the prerequisites for using Kubernetes CSI Volume Expansion with HSPC.

1. Enable VSP Program Product Licenses (P.P.)
    - Hitachi Dynamic Provisioning Software (HDP)

2. Install HSPC

    For installation method, please refer to the section “Install Storage Plug-in for Containers for Kubernetes CSI” in HSPC's Quick Reference Guide from Containers - Hitachi Vantara Knowledge.

    [Note] StorageClass and Secret needs to be configured, respectively.

3. Set volume expansion parameters and feature gate

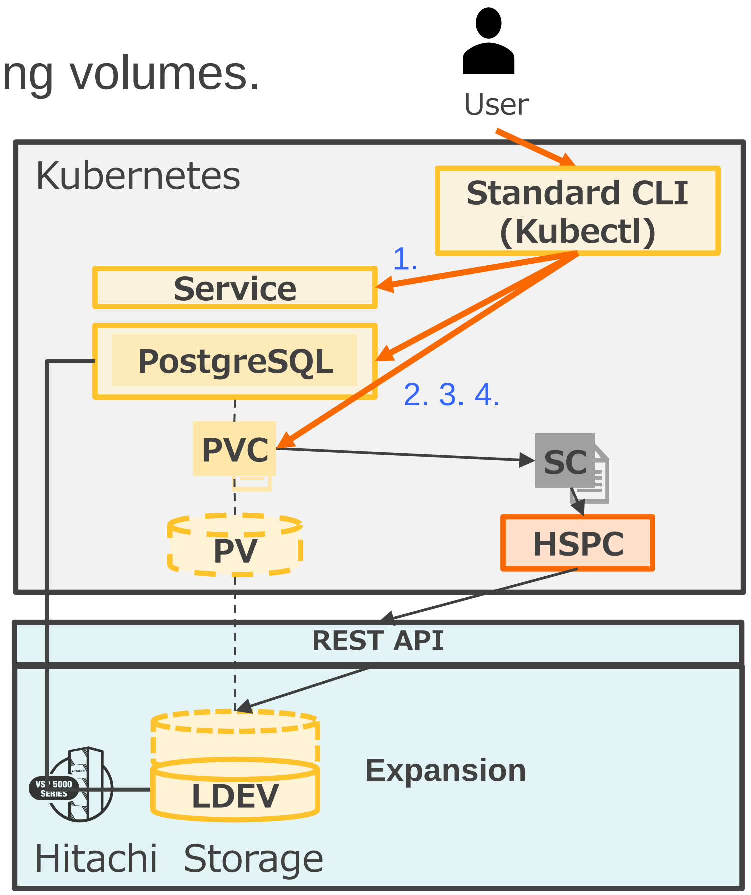
    Refer to appendix for details.

3. Best Practices

Overview

Below is the overview of best practices for expanding volumes.

1. Create a containerized application with Persistent Volume Claims (PVC).
2. Check the size of PVC and filesystem.
3. Change the size of PVC.
4. Check the size of PVC and filesystem.

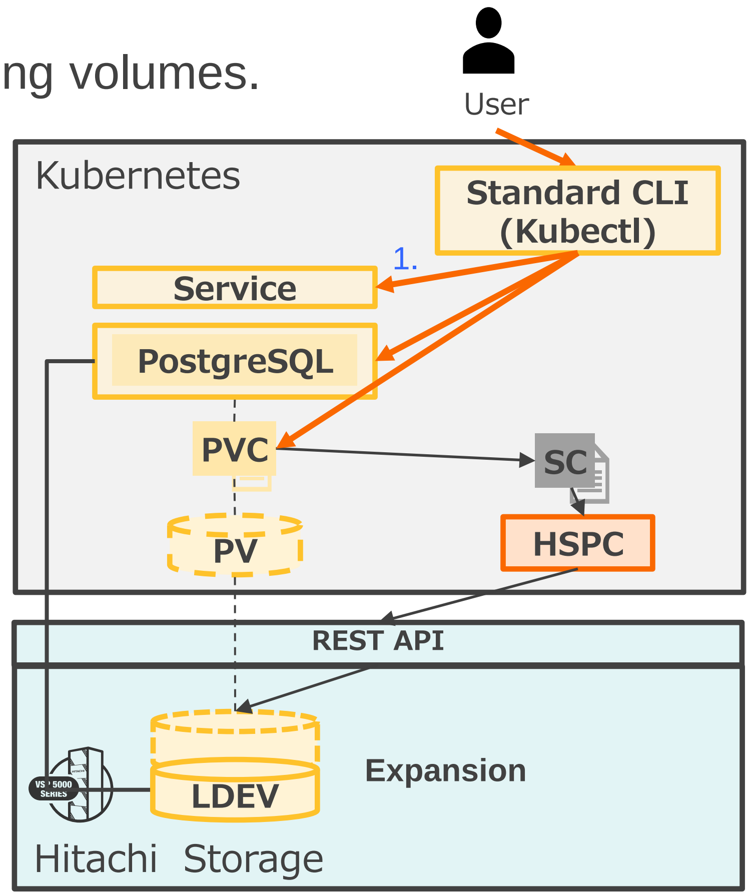


3. Best Practices

Overview

Below is the overview of best practices for expanding volumes.

1. **<u>Create a containerized application with Persistent Volume Claims (PVC).</u>**
2. Check the size of PVC and filesystem.
3. Change the size of PVC.
4. Check the size of PVC and filesystem.



3. Best Practices

1. Create a containerized application with Persistent Volume Claims (PVC).

(1) Create Persistent Volume Claims (hereafter PVC) for PostgreSQL using the manifest file.

postgres.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: postgres-base
    scenario: expansion
  name: postgres-base-svc-for-expansion
spec:
  type: NodePort
  ports:
  - port: 5432
    targetPort: 5432
    nodePort: 30432
  selector:
    app: postgres-base
    scenario: expansion
---
```

PostgreSQL Service

postgres.yaml (continued)

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-base-for-
expansion
  labels:
    app: postgres-base
    scenario: expansion
spec:
  serviceName: "postgres"
  replicas: 1
  selector:
    matchLabels:
      app: postgres-base
      scenario: expansion
```

PostgreSQL StatefulSet

postgres.yaml (continued)

```
  template:
    metadata:
      labels:
        app: postgres-base
        scenario: expansion
```

⋮

Continues to next page

3. Best Practices

1. Create a containerized application with Persistent Volume Claims (PVC).

postgres.yaml (continued)

```
spec:
  containers:
    - name: postgres
      image: postgres:11.3
      imagePullPolicy: "IfNotPresent"
      ports:
        - containerPort: 5432
      env:
        - name: POSTGRES_DB
          value: mydb
        - name: PGDATA
          value: /var/lib/postgresql/data/pgdata
      volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgredb
```

Mounts volume

postgres.yaml (continued)

```
volumeClaimTemplates:
- metadata:
    name: postgredb
    labels:
      scenario: expansion
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: sc-sample
    resources:
      requests:
        storage: 1Gi
```

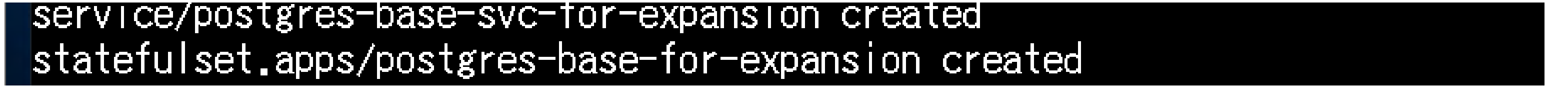
PVC

3. Best Practices

1. Create a containerized application with Persistent Volume Claims (PVC).

(2) Deploy PostgreSQL using the previously created manifest file. By this operation, HSPC dynamically creates volumes to Hitachi storage. This allows the user to persist the data.

```
$ kubectl apply -f postgres.yaml
```



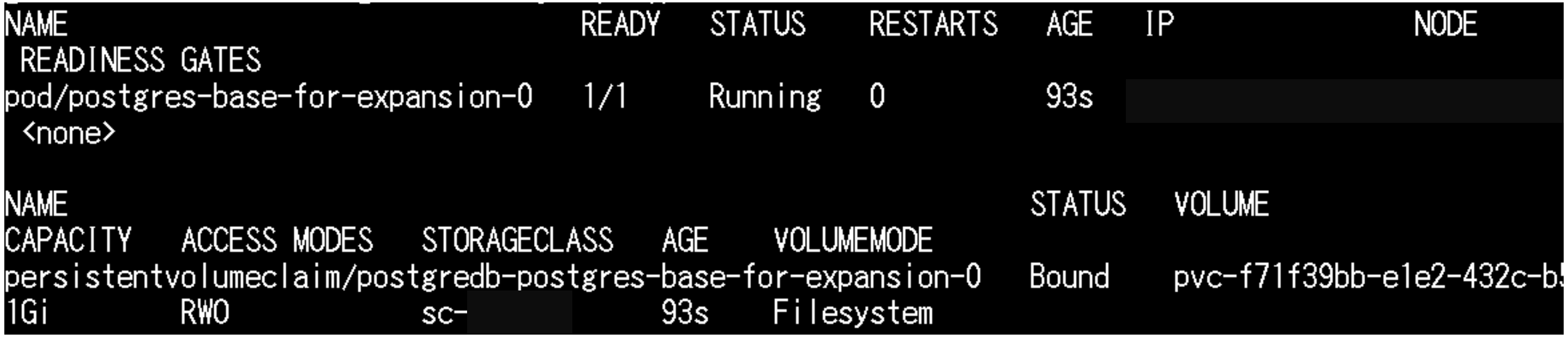
```
service/postgres-base-svc-for-expansion created
statefulset.apps/postgres-base-for-expansion created
```

3. Best Practices

1. Create a containerized application with Persistent Volume Claims (PVC).

(3) Check that the status of the created PVC is “Bound”, and PostgreSQL Pod is “Running”. This means that the Pod has started successfully.

```
$ kubectl get pod,pvc -o wide
```



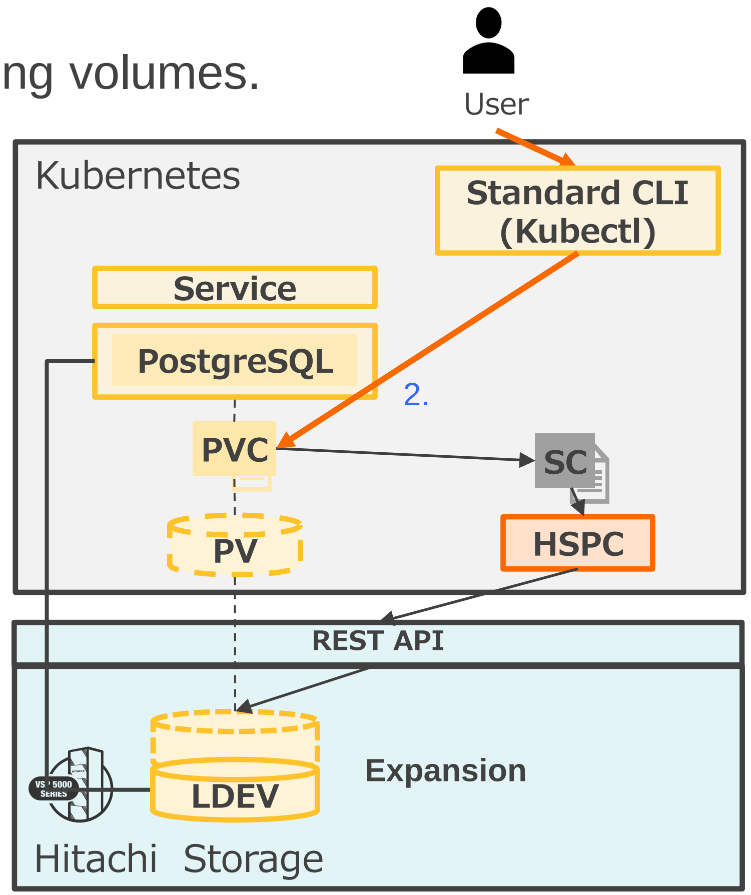
```
NAME                                READY   STATUS    RESTARTS   AGE   IP                 NODE
 READINESS GATES
pod/postgres-base-for-expansion-0   1/1     Running   0          93s   [redacted]
 <none>

NAME                                                            STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE   VOLUMEMODE
persistentvolumeclaim/postgredb-postgres-base-for-expansion-0   Bound    pvc-f71f39bb-e1e2-432c-b
1Gi        RWO            sc-[redacted]      93s   Filesystem
```

3. Best Practices

Overview

Below is the overview of best practices for expanding volumes.

1. Create a containerized application with Persistent Volume Claims (PVC).
2. **<u>Check the size of PVC and filesystem.</u>**
3. Change the size of PVC.
4. Check the size of PVC and filesystem.

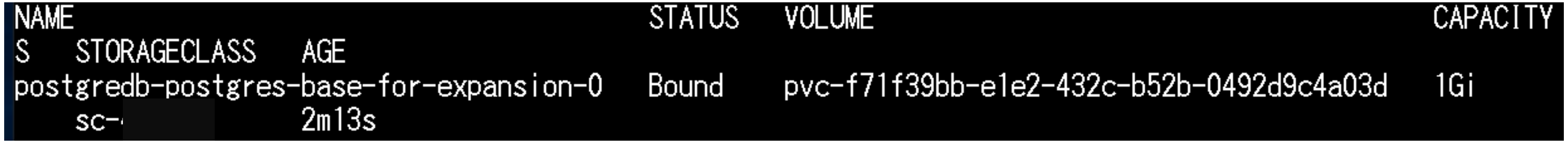


3. Best Practices

2. Check the size of PVC and filesystem.

(1) Check the capacity of PVC.

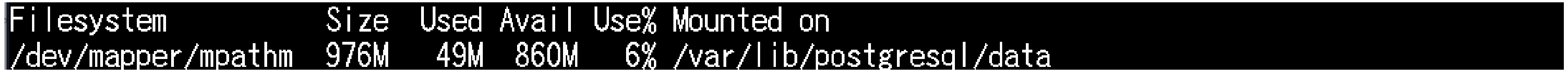
```
$ kubectl get pvc
```



```
NAME                                     STATUS   VOLUME                                     CAPACITY
S   STORAGECLASS   AGE
postgredb-postgres-base-for-expansion-0   Bound    pvc-f71f39bb-e1e2-432c-b52b-0492d9c4a03d   1Gi
    sc-[redacted]         2m13s
```

(2) Check the size of filesystem where Persistent Volume (PV) is mounted.

```
$ kubectl exec -it postgres-base-for-expansion-0 -- df -h /var/lib/postgresql/data
```



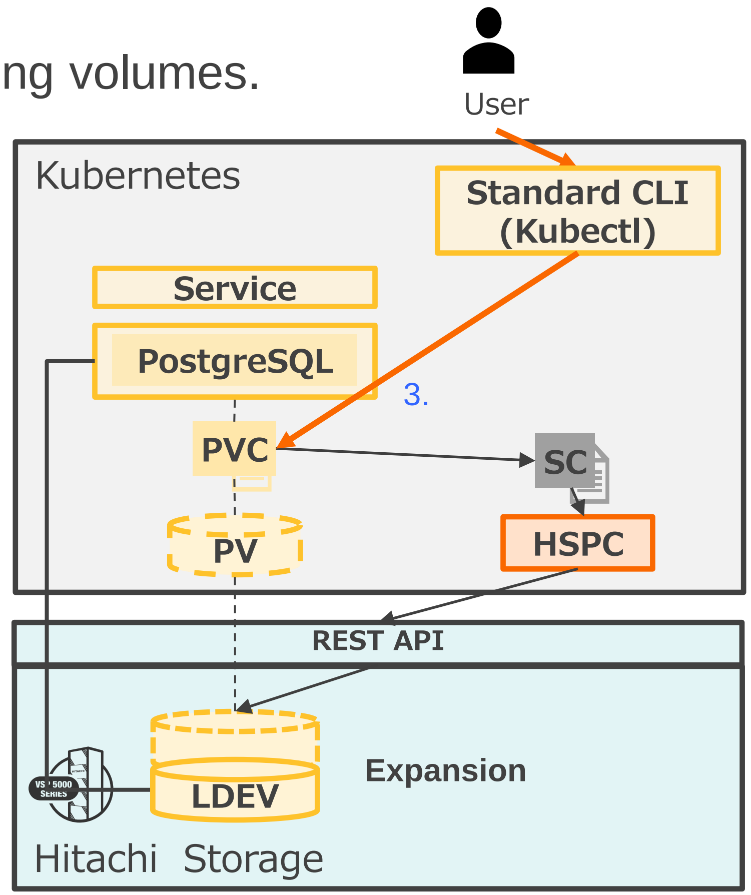
```
Filesystem           Size  Used Avail Use% Mounted on
/dev/mapper/mpathm   976M   49M  860M   6% /var/lib/postgresql/data
```

3. Best Practices

Overview

Below is the overview of best practices for expanding volumes.

1. Create a containerized application with Persistent Volume Claims (PVC).
2. Check the size of PVC and filesystem.
3. **<u>Change the size of PVC.</u>**
4. Check the size of PVC and filesystem.



3. Best Practices

3. Change the size of PVC.

Edit the PVC manifest file.

```
$ kubectl edit pvc postgredb-postgres-base-for-expansion-0
```

PVC manifest file

```
apiVersion: v1
kind: PersistentVolumeClaim
<<Partially omitted>>
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
  storageClassName: sc-sample
  volumeMode: Filesystem
  volumeName: pvc-e6d53358-e675-423d-85e6-
2689f610f5bd
```

PVC manifest file (continued)

```
status:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 2Gi
  phase: Bound
```

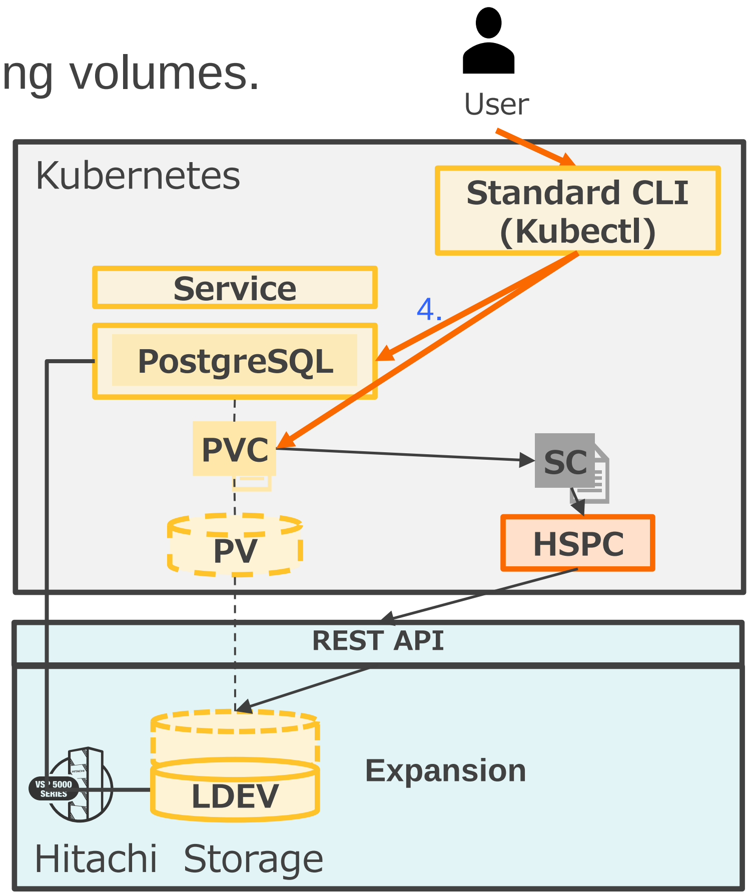
```
persistentvolumeclaim/postgredb-postgres-base-for-expansion-0 edited
```

3. Best Practices

Overview

Below is the overview of best practices for expanding volumes.

1. Create a containerized application with Persistent Volume Claims (PVC).
2. Check the size of PVC and filesystem.
3. Change the size of PVC.
4. **Check the size of PVC and filesystem.**

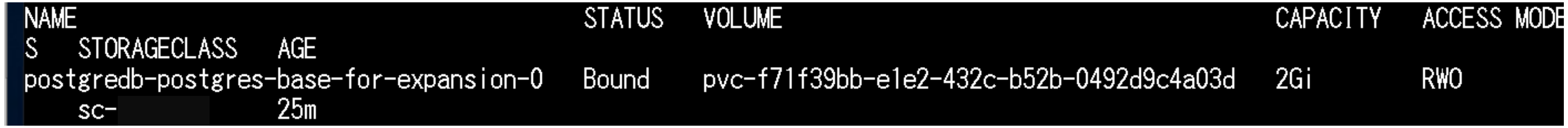


3. Best Practices

4. Check the size of PVC and filesystem.

(1) Check the size of PVC again. Wait until the size is changed to the specified size (in this case, 2Gi).

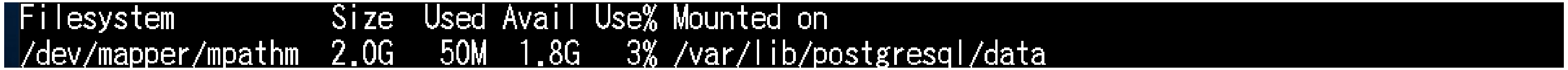
```
$ kubectl get pvc
```



```
NAME                                        STATUS   VOLUME                                     CAPACITY   ACCESS MODE
S   STORAGECLASS   AGE
postgredb-postgres-base-for-expansion-0    Bound    pvc-f71f39bb-e1e2-432c-b52b-0492d9c4a03d   2Gi        RWO
    sc-[redacted]        25m
```

(2) Check the size of filesystem in the pod where PV is mounted.

```
$ kubectl exec -it postgres-base-for-expansion-0 -- df -h /var/lib/postgresql/data
```



```
Filesystem          Size  Used Avail Use% Mounted on
/dev/mapper/mpathm  2.0G   50M  1.8G   3% /var/lib/postgresql/data
```

4. Conclusion

In this document, you have learned how to use Kubernetes CSI Volume Expansion feature using HSPC.

You can save storage resources by starting with small volumes and expanding later as the data grows.

5. Related Links

- Deployment Options for Kubernetes Container Applications on Unified Compute Platform CI with Hitachi VSP Series
- Container Storage Interface (CSI) Driver for Hitachi Virtual Storage Platform Series
- Deploy WordPress and MySQL in Kubernetes using HSPC
- Kubernetes Volume Clone using HSPC (Hitachi Storage Plug-in for Containers)

Appendix

How to set volume expansion parameters and feature gate

HITACHI
Inspire the Next

1. Set the volume expansion parameters for StorageClass

(1) Create a new manifest file for a StorageClass from the existing StorageClass.

```
$ kubectl get sc sc-sample -o yaml > sc.yaml
```

(2) Delete the existing StorageClass.

```
$ kubectl delete -f sc.yaml
```

(3) Edit the parameters in the StorageClass manifest file.

```
$ vi sc.yaml
```

(4) Create the new StorageClass.

```
$ kubectl create -f sc.yaml
```

Make sure the secret name is consistent.

sc.yaml (StorageClass manifest file)

```
<<Partially omitted>>
parameters:
  csiControllerPublishSecretName: secret400130
  <<Partially omitted>>
  csi.storage.k8s.io/controller-expand-secret-name: "secret400130"
  csi.storage.k8s.io/controller-expand-secret-namespace: "default"
provisioner: hspc.csi.hitachi.com
<<Partially omitted>>
allowVolumeExpansion: true
```

How to set volume expansion parameters and feature gate

2. Enable the feature gate

Required for Kubernetes 1.15 and earlier

To set feature gates for a component, such as kubelet, use the --feature-gates flag assigned to a list of feature pairs. Below is an example of steps for setting volume expansion feature gates.

(1) Add the feature gate to kubelet in all nodes.

```
$ vi <config-file-path>/kubelet

$ sysctl restart kubelet
```

e.g. /etc/sysconfig/

kubelet

```
KUBELET_EXTRA_ARGS=--feature-gates=ExpandCSIVolumes=true
```

(2) Add the feature gate to kube-apiserver and kube-controller-manager manifest files.

```
$ vi <manifest-file-path>/kube-apiserver.yaml

$ vi <manifest-file-path>/kube-controller-manager.yaml
```

e.g. /etc/kubernetes/manifests/

kube-apiserver.yaml / kube-controller-manager.yaml

```
<<Partially omitted>>
spec:
  containers:
  - command:    <<Partially omitted>>
  - --feature-gates=ExpandCSIVolumes=true
…
```

Thank You

HITACHI
Inspire the Next