HITACHI
Inspire the Next

# Across Kubernetes Namespace Boundaries:
## Your Volumes Can Be Shared Now!

2023/4/20 @ KubeCon + CloudNativeCon Europe 2023

Takafumi Takahashi, Hitachi Vantara LLC
Masaki Kimura, Hitachi, Ltd.

# Who are we?

HITACHI
Inspire the Next

Name: Takafumi Takahashi
Company: Hitachi Vantara
Github ID: ttakahashi21

- Has been contributing to Kubernetes community by implementing provision volumes from cross-namespace snapshots (KEP-3294).

Name: Masaki Kimura
Company: Hitachi, Ltd.
Github ID: mkimuram

- Has been contributing to Kubernetes community to make Raw Block Volume feature and CSI feature GA.
- Designed and proposed KEP-3294

This session is about details on KEP-3294: Provision volumes from cross-namespace snapshots .

# Disclaimer

HITACHI
Inspire the Next

This session explains designs, specifications, and implementations which are under discussion and development.

Depending on the discussion, they are subject to change.

HITACHI
Inspire the Next

# Contents

HITACHI
Inspire the Next

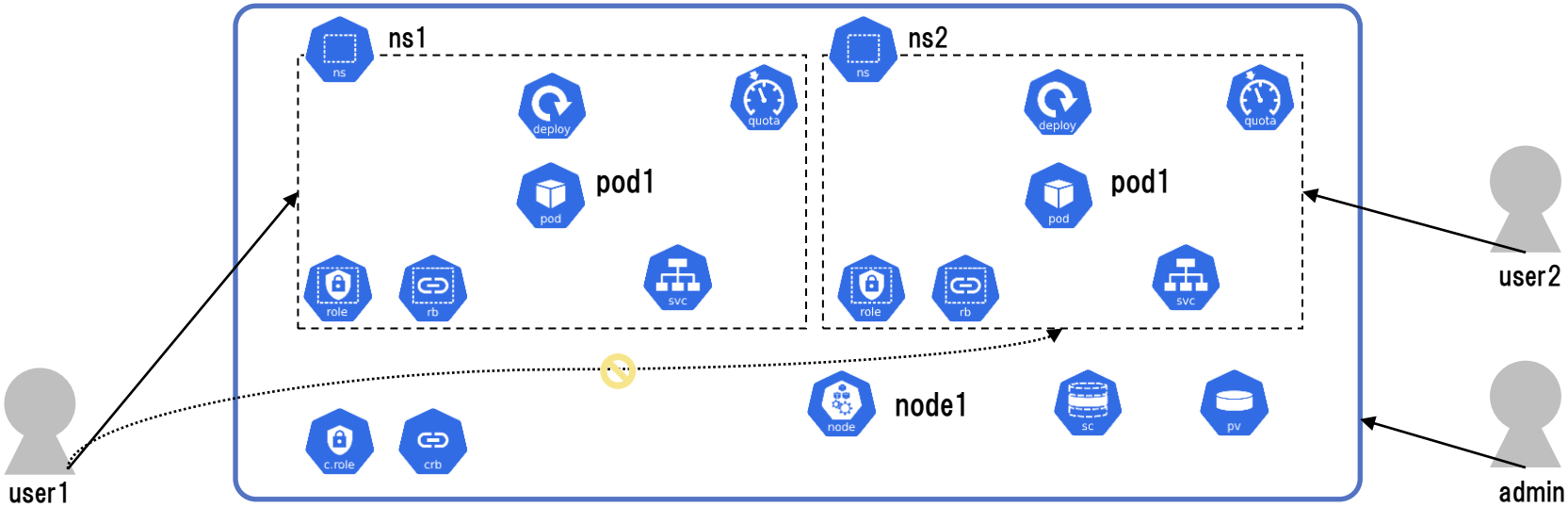
# 1. Namespace and Volume in Kubernetes

## 1-1 Kubernetes Namespace
## 1-2 PersistentVolumeClaim and PersistentVolume
## 1-3 VolumeSnapshot and VolumeSnapshotContent
## 1-4 Creating PVC from VS/PVC

# 1-1 Kubernetes Namespace

HITACHI
Inspire the Next

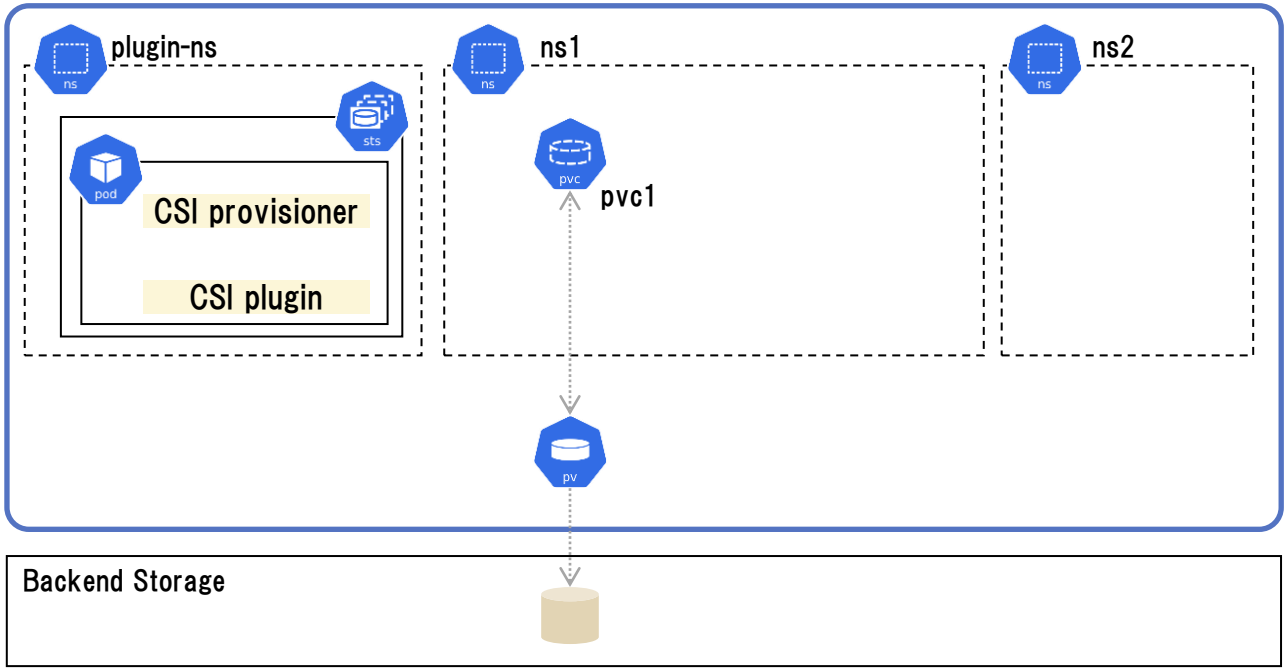
- “In Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster.”
  - Names of resources are unique within a namespace
  - RBAC can be set per namespace
  - Resource quota can be set per namespace
- Kubernetes resource consists of namespaced resource and cluster scoped resource:
  - Namespaced: Pod, Deployment, Service, ResourceQuota, Role, RoleBinding, etc …
  - Cluster-scoped: Node, StorageClass, PersistentVolume, ClusterRole, ClusterRoleBinding, etc …



# 1-2 PersistentVolumeClaim and PersistentVolume (1/2)

HITACHI
Inspire the Next

- Volumes are managed through PersistentVolumeClaim(PVC) and PersistentVolume(PV)
  - PVC is namespaced resource and PV is cluster-scoped resource
  - CSI plugin creates an actual volume specified in a PVC and provisioner creates a PV referenced from the PVC




▪Example of PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

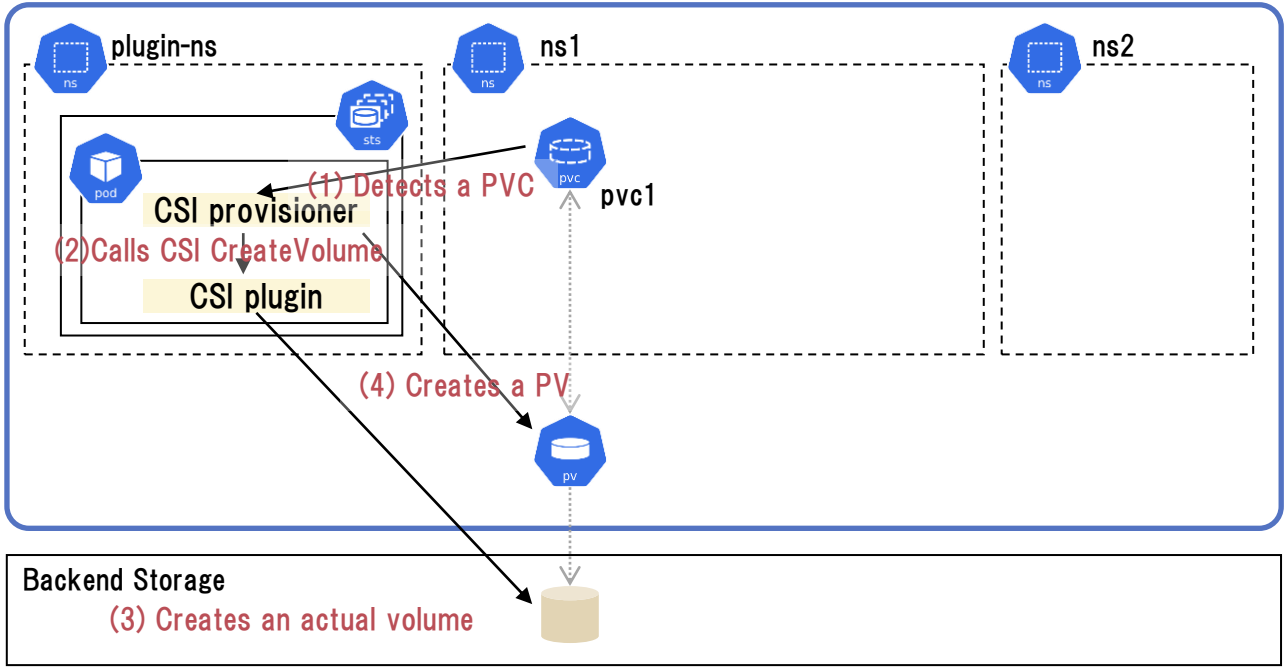
▪Example of PV:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvc-xxxx
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc1
    namespace: ns1
  csi:
    driver: xxxx
    volumeHandle: xxxx
```

# 1-2 PersistentVolumeClaim and PersistentVolume (2/2)

HITACHI
Inspire the Next

- Volumes are managed through PersistentVolumeClaim(PVC) and PersistentVolume(PV)
  - PVC is namespaced resource and PV is cluster-scoped resource
  - CSI plugin creates an actual volume specified in a PVC and provisioner creates a PV referenced from the PVC




▪Example of PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

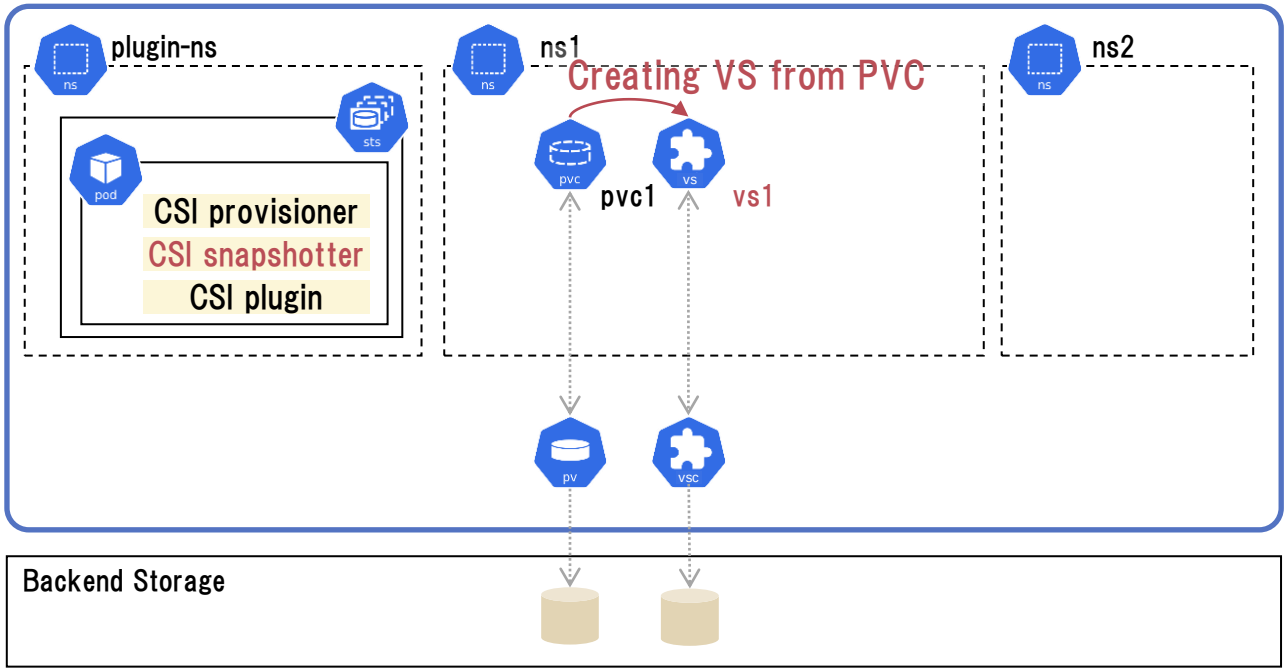
▪Example of PV:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvc-xxxx
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc1
    namespace: ns1
  csi:
    driver: xxxx
    volumeHandle: xxxx
```

# 1-3 VolumeSnapshot and VolumeSnapshotContent (1/2)

HITACHI
Inspire the Next

- Snapshots are managed through VolumeSnapshot(VS) and VolumeSnapshotContent(VSC)
  - VS is namespaced resource and VSC is cluster-scoped resource
  - CSI plugin creates an actual snapshot specified in a VS and snapshotter creates a VSC referenced from the VS




▪Example of VS:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: vs1
  namespace: ns1
spec:
  source:
    persistentVolumeClaimName: pvc1
```

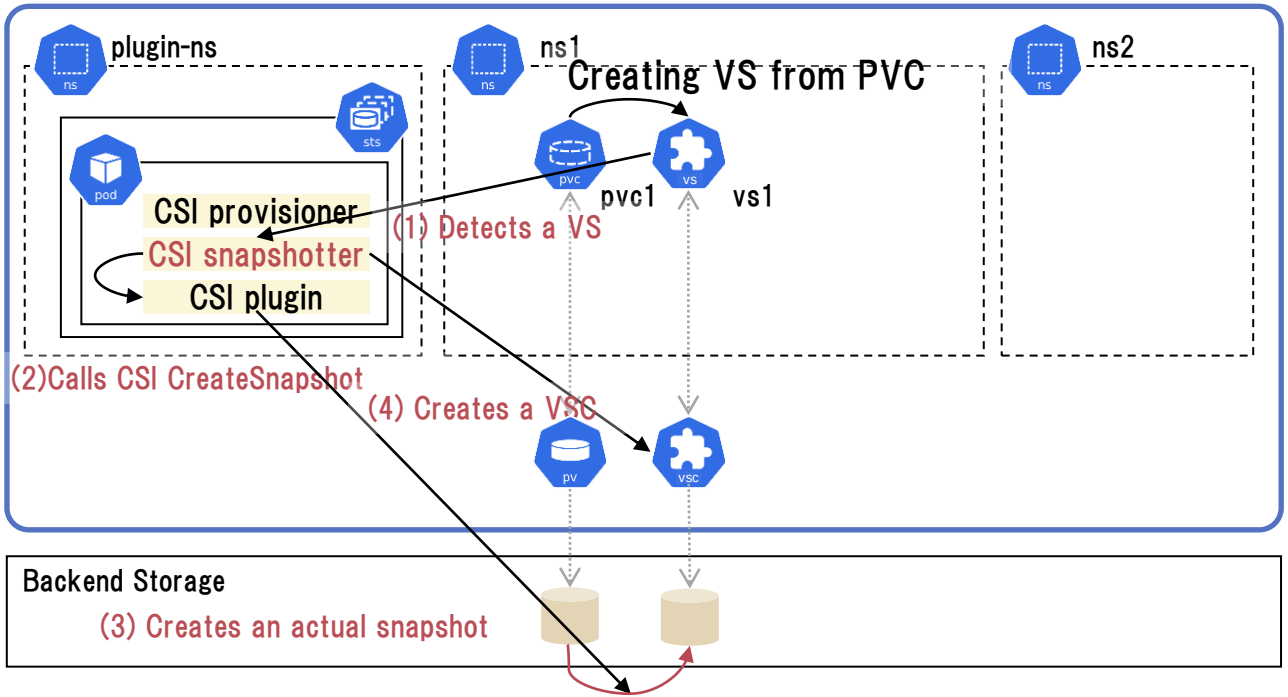
▪Example of VSC

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: snapcontent-xxxx
spec:
  deletionPolicy: Delete
  driver: xxxx
  source:
    volumeHandle: xxxxx
  volumeSnapshotClassName: xxx
  volumeSnapshotRef:
    apiVersion: snapshot.storage.k8s.io/v1
    kind: VolumeSnapshot
    name: vs1
    namespace: ns1
```

# 1-3 VolumeSnapshot and VolumeSnapshotContent (2/2)

HITACHI
Inspire the Next

- Snapshots are managed through VolumeSnapshot(VS) and VolumeSnapshotContent(VSC)
  - VS is namespaced resource and VSC is cluster-scoped resource
  - CSI plugin creates an actual snapshot specified in a VS and snapshotter creates a VSC referenced from the VS




▪Example of VS:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: vs1
  namespace: ns1
spec:
  source:
    persistentVolumeClaimName: pvc1
```

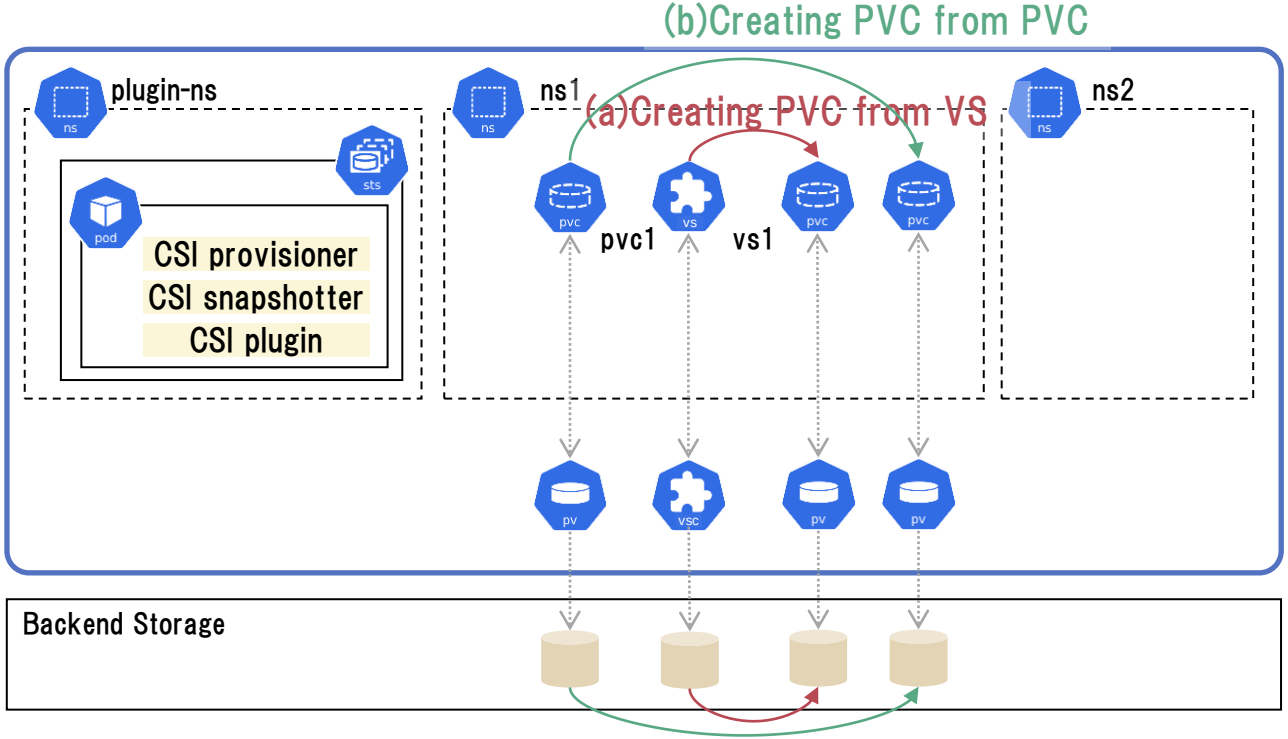
▪Example of VSC

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: snapcontent-xxxx
spec:
  deletionPolicy: Delete
  driver: xxxx
  source:
    volumeHandle: xxxxx
  volumeSnapshotClassName: xxx
  volumeSnapshotRef:
    apiVersion: snapshot.storage.k8s.io/v1
    kind: VolumeSnapshot
    name: vs1
    namespace: ns1
```

# 1-4 Creating PVC from VS/PVC (1/2)

HITACHI
Inspire the Next

- PVC can be created from data sources like existing VS/PVC
  - Data source is specified through DataSource field of PVC
  - Only VS/PVC in the same namespace can be specified as a data source




(a)Creating PVC from VS

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

(b)Creating PVC from PVC

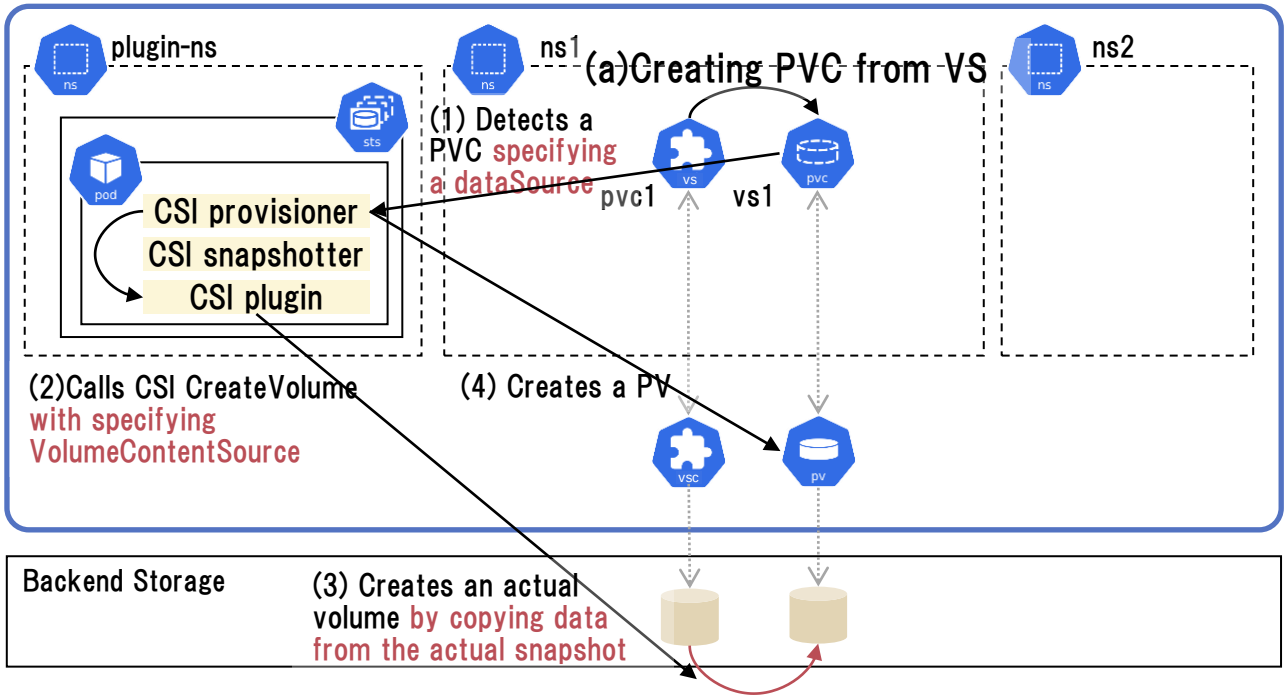
Namespace can't be specified

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: pvc1
```

# 1-4 Creating PVC from VS/PVC (2/2)

HITACHI
Inspire the Next

- PVC can be created from data sources like existing VS/PVC
  - Data source is specified through DataSource field of PVC
  - Only VS/PVC in the same namespace can be specified as a data source




## (a)Creating PVC from VS

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

Namespace can't be specified

HITACHI
Inspire the Next

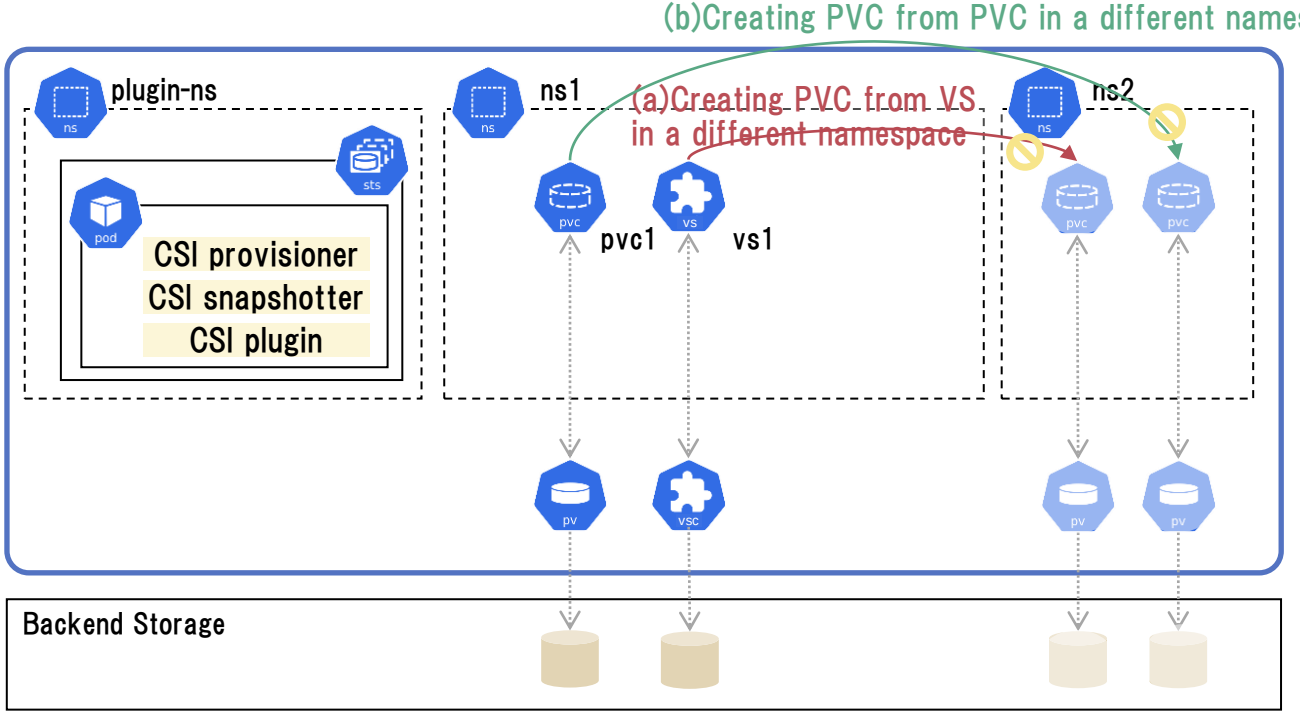
# 2. Issues and Use Cases Around Volumes in Different Namespaces

## 2-1 Issues Around Volumes in Different Namespaces
## 2-2 Use Case Around Volumes in Different Namespaces (1)
## 2-3 Use Case Around Volumes in Different Namespaces (2)

# 2-1 Issues Around Volumes in Different Namespaces

HITACHI
Inspire the Next

- PVC can't be created from an existing VS/PVC in a different namespace
  - Just allowing to copy any volumes in other namespaces causes a security issue
  - On the other hand, there are some use cases that require passing data beyond namespace boundaries



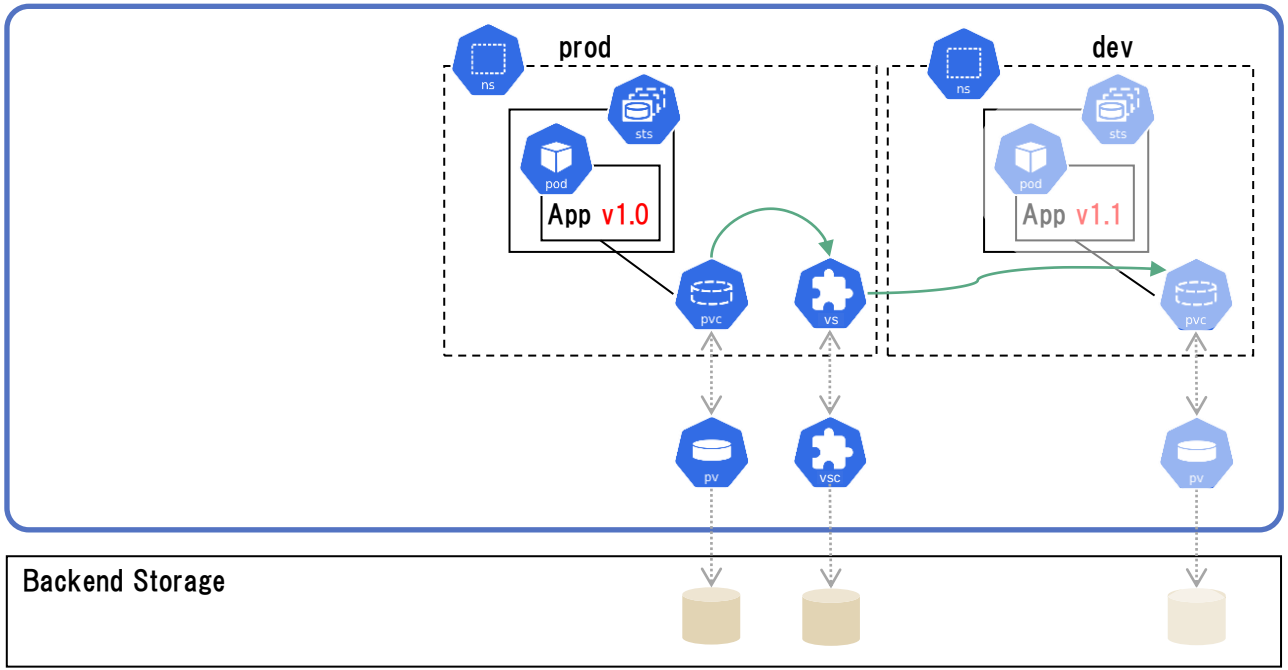

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    apiGroup:
    kind:
    name:
```

Namespace can't be specified

# 2-2 Use Case Around Volumes in Different Namespaces (1)

HITACHI
Inspire the Next

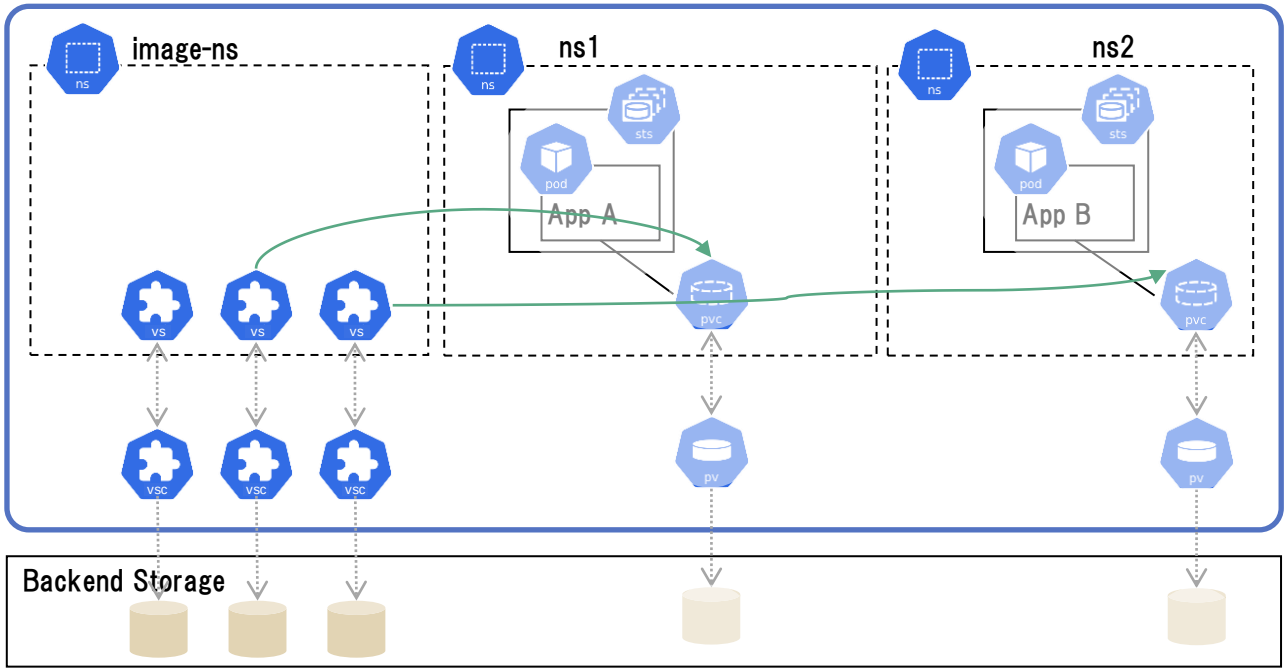
- Copying data from a production namespace to a development namespace
  - For test data in development environment
  - For datasets of machine learning



# 2-3 Use Case Around Volumes in Different Namespaces (2)

HITACHI
Inspire the Next

- Using golden images in one namespace from multiple other namespaces
  - VM image for KubeVirt
  - Common data like initial data and test data



HITACHI
Inspire the Next

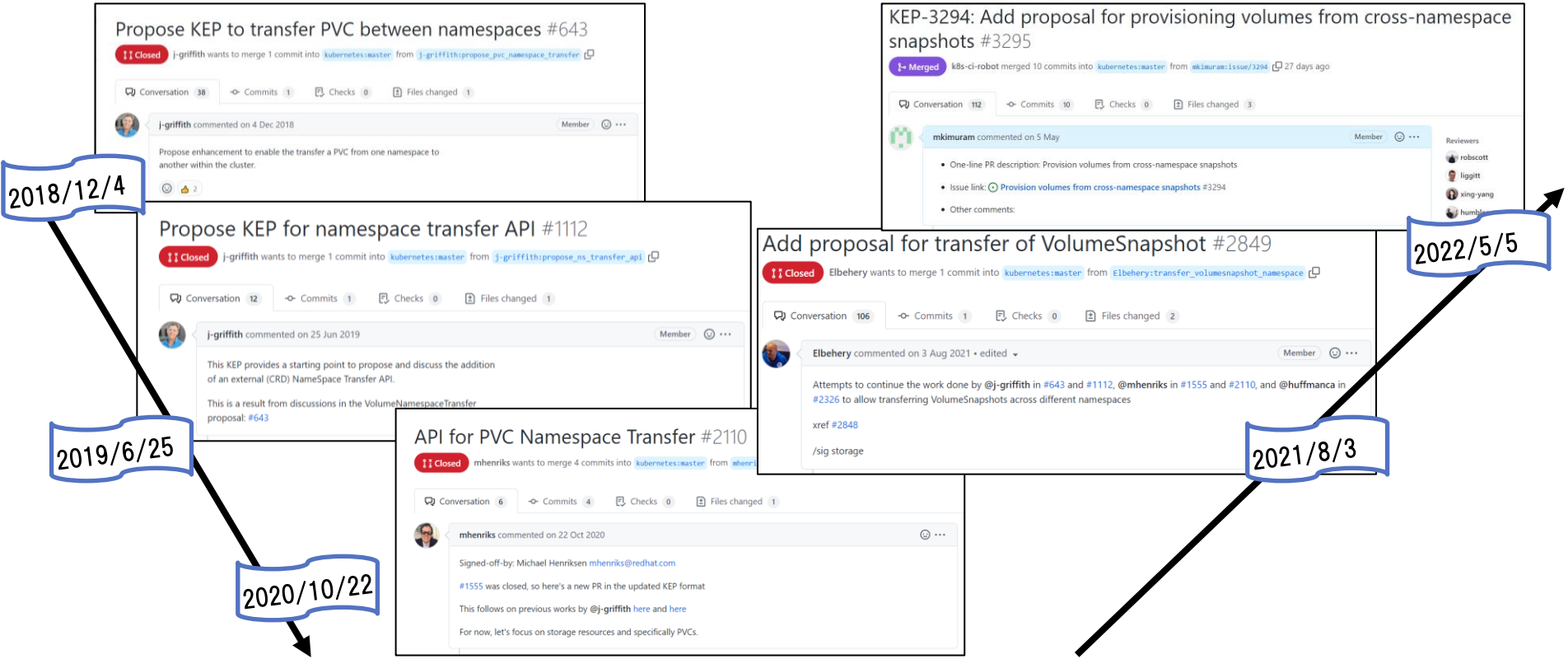
# 3. Long Discussion on the Issue

3-0 History of Discussion in k8s Community
3-1 Summary of the Initial Approach
3-2 Transfer + Clone Approach (API Design)
3-3 Transfer + Clone Approach (Challenges)

# 3-0 History of Discussion in k8s Community

HITACHI
Inspire the Next

- Since 2018, 5 KEPs are opened by 4 authors to discuss for this issue.

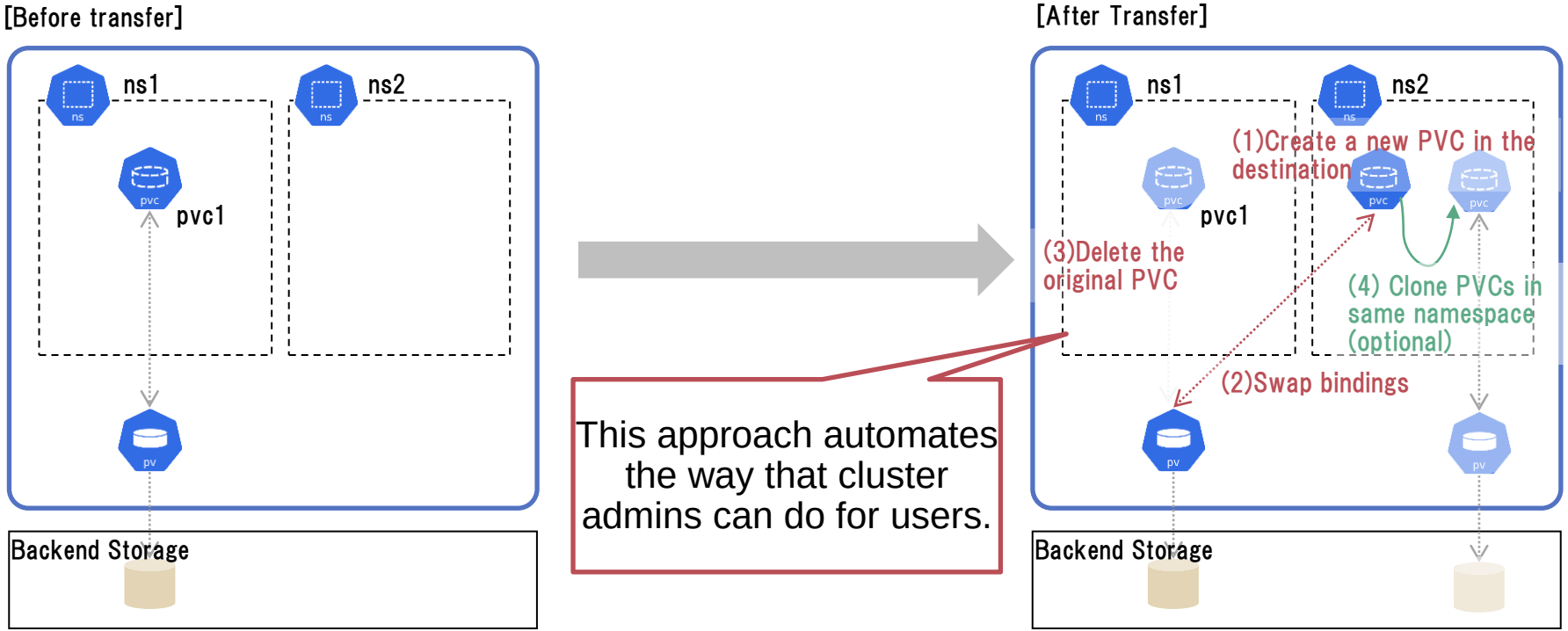



https://github.com/kubernetes/enhancements/pull/643 , https://github.com/kubernetes/enhancements/pull/1112 , https://github.com/kubernetes/enhancements/pull/2110 , https://github.com/kubernetes/enhancements/pull/2849 , https://github.com/kubernetes/enhancements/pull/3295

# 3-1 Summary of the Initial Approach

HITACHI
Inspire the Next

- Initial approach was the transfer of PVC/VS by rebinding the PVC-PV or VS-VSC.
- When copy of PVC/VS is required, clone of PVC/VS is done after transfer



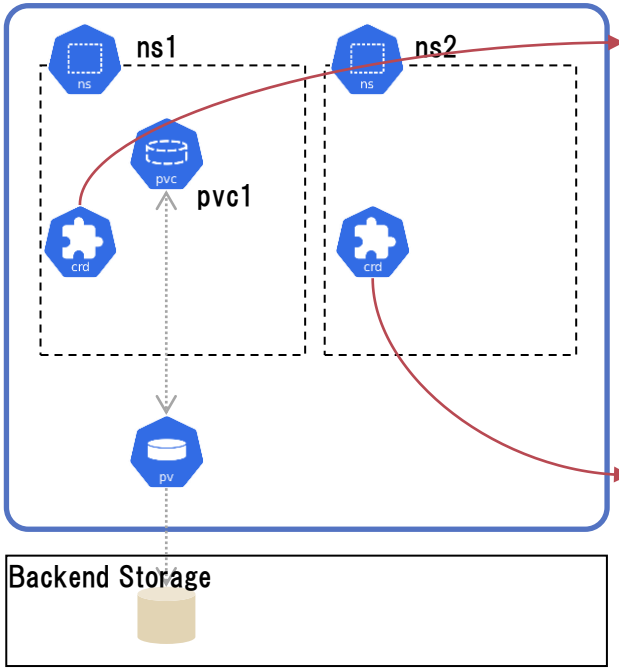

https://github.com/kubernetes/enhancements/pull/643

# 3-2 Transfer + Clone Approach (API Design)

HITACHI
Inspire the Next

- StorageTransferRequest and StorageTransferApproval were planned to be defined.
  - To prevent malicious users from transferring PVCs to any other namespaces

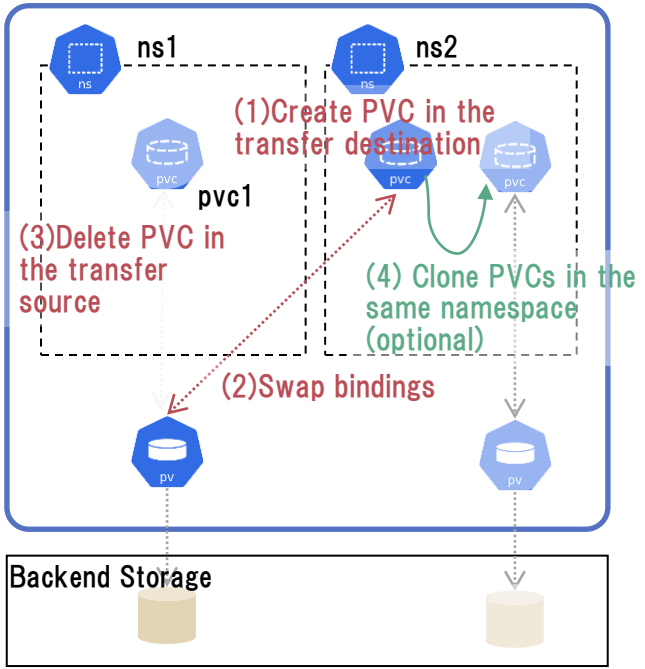
【Before Transfer】




```
apiVersion: v1alpha1
kind: StorageTransferApproval
metadata:
  name: approve-foo
  namespace: ns1
spec:
  source:
    name: pvc1
    kind: PersistentVolumeClaim
  targetNamespace: ns2
  requestName: transfer-foo
```

```
apiVersion: v1alpha1
kind: StorageTransferRequest
metadata:
  name: transfer-foo
  namespace: ns2
spec:
  source:
    name: pvc1
    namespace: ns1
    kind: PersistentVolumeClaim
  approvalName: approve-foo
  targetName: pvc1-moved
```

【After Transfer】

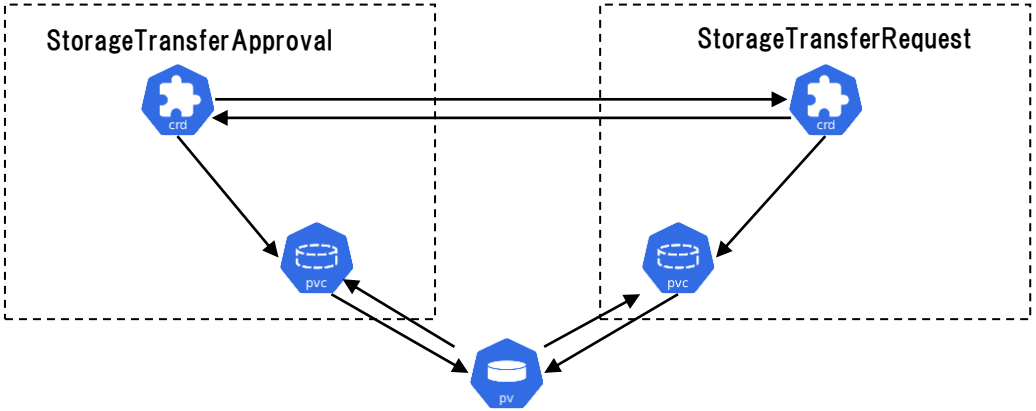



https://github.com/kubernetes/enhancements/pull/2849

# 3-3 Transfer + Clone Approach (Challenges)

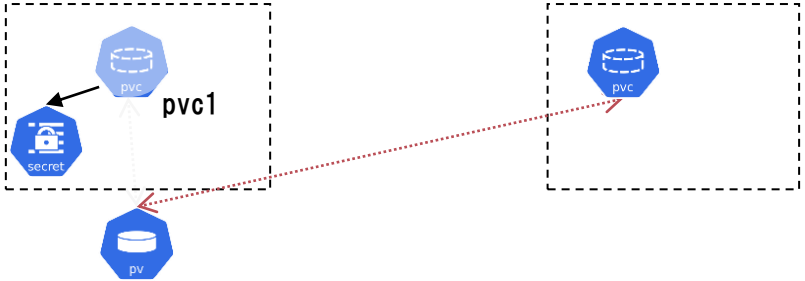
HITACHI
Inspire the Next

1. Difficult to rollback if errors occur or changes happen during the transfer process.




ex) While the same PV is bound to/from both source PVC and destination PVC:
- Deletion/Change/Resize is requested to the source PVC
- Approval/Request CRD is deleted

2. Difficult to handle secrets referenced by PVC/VS
   - Secret that is assumed to be in the same namespace as the PVC cannot be transferred



HITACHI
Inspire the Next

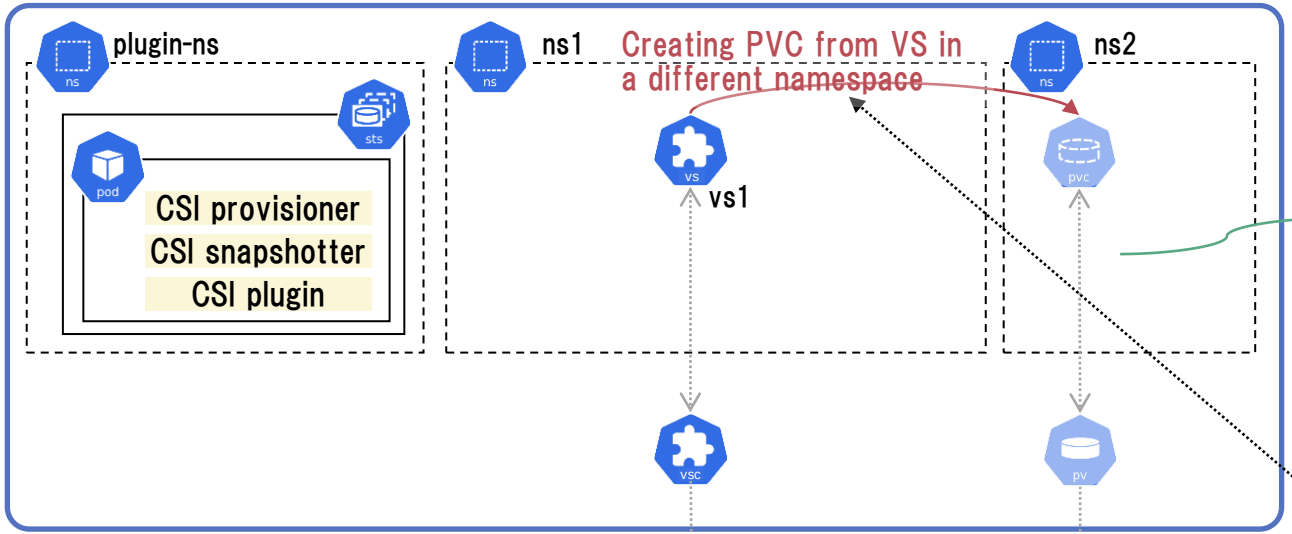
# 4. Current Design and Implementation

4-1 Overview of Current Design
4-2-1 (1) AnyVolumeDataSource (Prerequisite Feature)
4-2-2 (1) Extension of DataSource
4-3-1 (2) ReferenceGrant (Prerequisite Feature)
4-3-2 (2) Grant cross-namespace access
4-4 Behavior
4-5 Development Status

# 4-1 Overview of Current Design

HITACHI
Inspire the Next

Current design utilizes two features that were recently added for another purposes:
(1) Extension of dataSource: by using AnyVolumeDataSource
(2) Grant cross-namespace access: by using ReferenceGrant




```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

(1) Extension of dataSource

(2) Grant cross-namespace access

https://github.com/kubernetes/enhancements/pull/2849#issuecomment-949929595

# 4-2-1 AnyVolumeDataSource (Prerequisite Feature)

HITACHI
Inspire the Next

- AnyVolumeDataSource feature extends data source (alpha in k8s v1.18, beta in k8s v1.24)
  - Existing DataSource field only allows specifying PersistentVolumeClaim and VolumeSnapshot
  - This feature allows any resources to be specified via a newly added DataSourceRef field
  - Copying the data is performed by volume populator for the corresponding resources
  - This feature itself allows only the same namespace to be specified as a data source

【When AnyVolumeDataSource feature is disabled】

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSource:
    apiGroup: hello.example.com
    kind: Hello
    name: example-hello
```

Invalid data source

Other resource than PersistentVolumeClaim/ VolumeSnapshot

【When AnyVolumeDataSource feature is enabled】

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSourceRef:
    apiGroup: hello.example.com
    kind: Hello
    name: example-hello
```

Valid data source

# 4-2-2 (1) Extension of DataSource

HITACHI
Inspire the Next

- To allow specifying a namespace as a data source of a PersistentVolumeClaim, namespace alpha field is added to DataSourceRef

**Add namespace field to dataSourceRef**

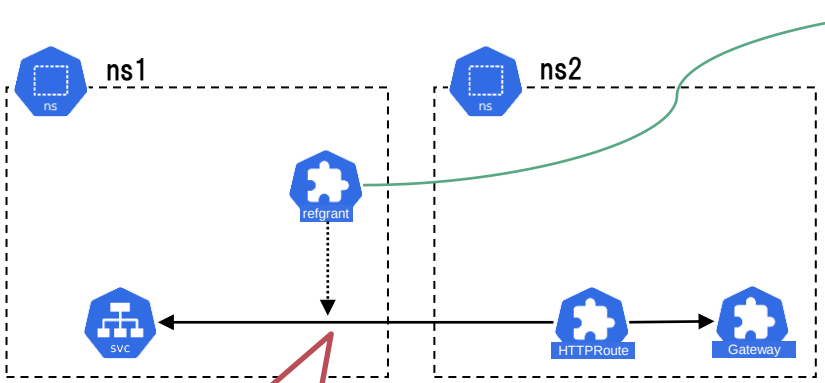
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns2
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSourceRef:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
    namespace: ns1
```

- For compatibility purpose, existing DataSource field doesn't accept namespace. Therefore, users are required to pass namespace via DataSourceRef field.

# 4-3-1 (2) ReferenceGrant (Prerequisite Feature)

HITACHI
Inspire the Next

- ReferenceGrant is added in Gateway API (alpha in April 2021, beta in July 2022)
  - To allow Gateway API to access to the resource across namespaces
  - ex) To allow HTTPRoute in Gateway API to access to Service in a different namespace

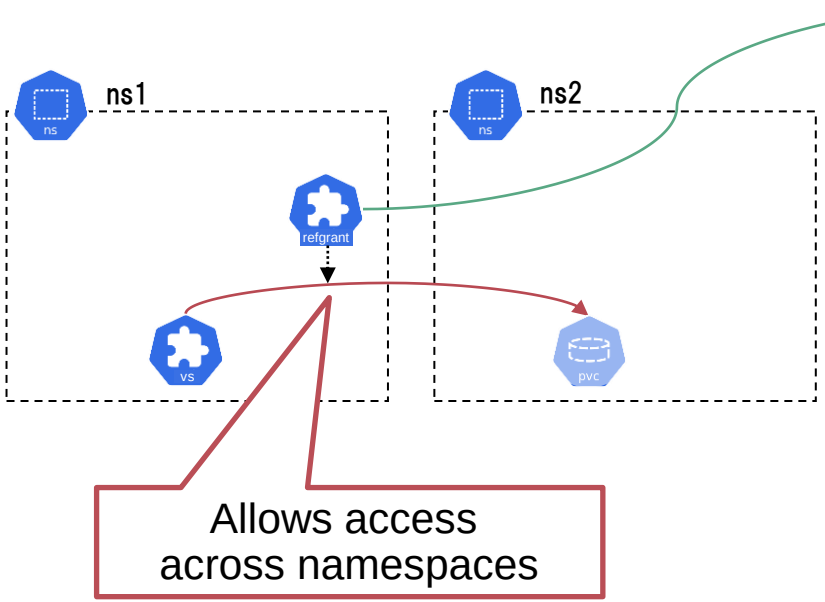



```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: ReferenceGrant
metadata:
  name: allow-prod-traffic
  namespace: ns1
spec:
  from:
  - group: gateway.networking.k8s.io
    kind: HTTPRoute
    namespace: ns2
  to:
  - group: ""
    kind: Service
```

# 4-3-2 (2) Grant cross-namespace access

HITACHI
Inspire the Next

- ReferenceGrant is used to allow specifying VS/PVC across namespace as dataSource of PVC




```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: ReferenceGrant
metadata:
  name: foo-grant
  namespace: ns1
spec:
  from:
  - group: ""
    kind: PersistentVolumeClaim
    namespace: ns2
  to:
  - group: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

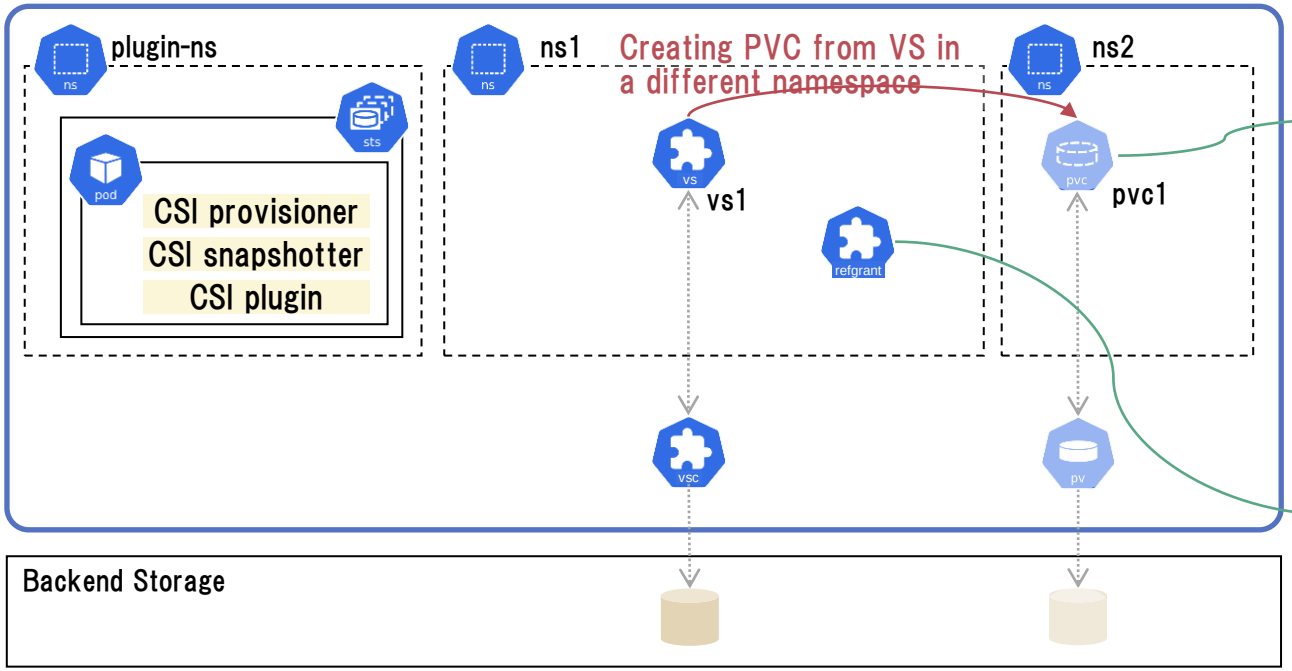
- ReferenceGrant is planned to be moved to new sig-auth API group in k8s 1.28

https://github.com/kubernetes/enhancements/issues/3766

# 4-4 Behavior (1/2)

HITACHI
Inspire the Next

(1) Extension of dataSource: by specifying namespace via spec.dataSourceRef.namespace alpha field
(2) Grant cross-namespace access: by using ReferenceGrant




```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns2
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSourceRef:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
    namespace: ns1
```

Namespace can be specified

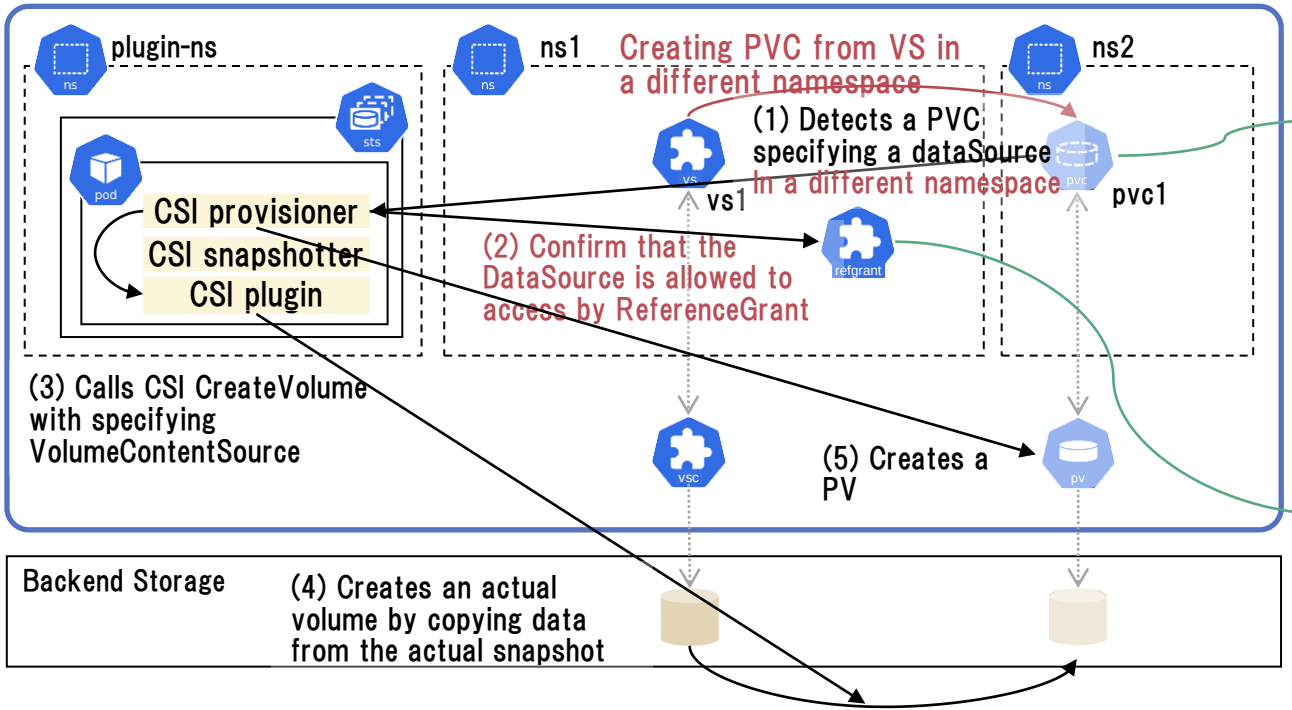
```
apiVersion: gateway.networking.k8s.io/v1be
kind: ReferenceGrant
metadata:
  name: foo-grant
  namespace: ns1
spec:
  from:
  - group: ""
    kind: PersistentVolumeClaim
    namespace: ns2
  to:
  - group: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

Allow access across namespace

# 4-4 Behavior (2/2)

HITACHI
Inspire the Next

(1) Extension of dataSource: by specifying namespace via spec.dataSourceRef.namespace alpha field
(2) Grant cross-namespace access: by using ReferenceGrant




```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
  namespace: ns2
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSourceRef:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
    namespace: ns1
```

Namespace can be specified

```yaml
apiVersion: gateway.networking.k8s.io/v1be
kind: ReferenceGrant
metadata:
  name: foo-grant
  namespace: ns1
spec:
  from:
  - group: ""
    kind: PersistentVolumeClaim
    namespace: ns2
  to:
  - group: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: vs1
```

Allow access across namespace

# 4-5 Development Status

HITACHI
Inspire the Next

- Alpha in k8s v1.26
- Release Plan:
  - Beta: v1.29
  - Stable: v1.31
- Scope:
  - Provision of PVC from cross namespace data source (No transfer of PVC/VolumeSnapshot)
  - Supported data source:
    - v1.26: VolumeSnapshot, PersistentVolumeClaim
    - planning: AnyVolumeDataSource
- Feature Gate:
  - Name: CrossNamespaceVolumeDataSource
  - Required components:
    - Kubernetes(controller)
    - CSI external-provisioner

HITACHI
Inspire the Next

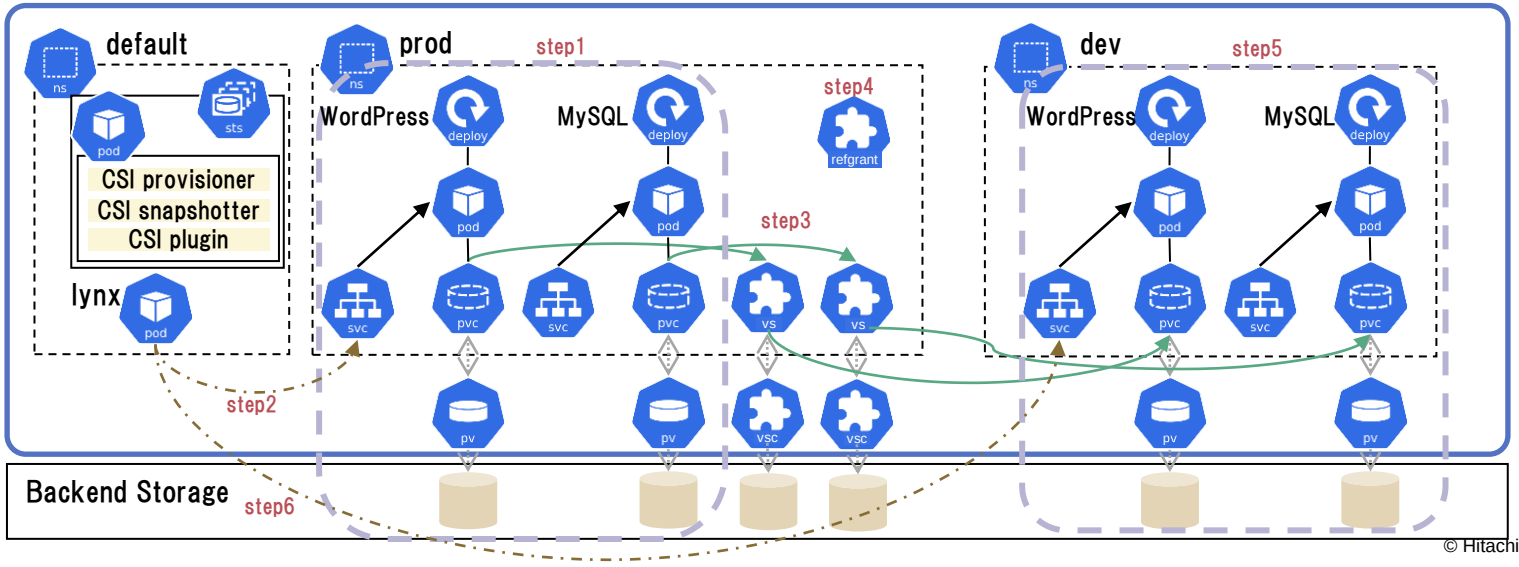
# 5. Demo

5-1 Overview
5-2-0 Check Demo Configuration
5-2-1 Deploying Prod WordPress
5-2-2 Accessing to Prod WordPress (Creating data)
5-2-3 Taking Snapshots
5-2-4 Creating ReferenceGrant
5-2-5 Creating Dev Volumes from Prod Snapshots and Using them
5-2-6 Accessing to Dev WordPress (Checking data)

# 5-1 Overview

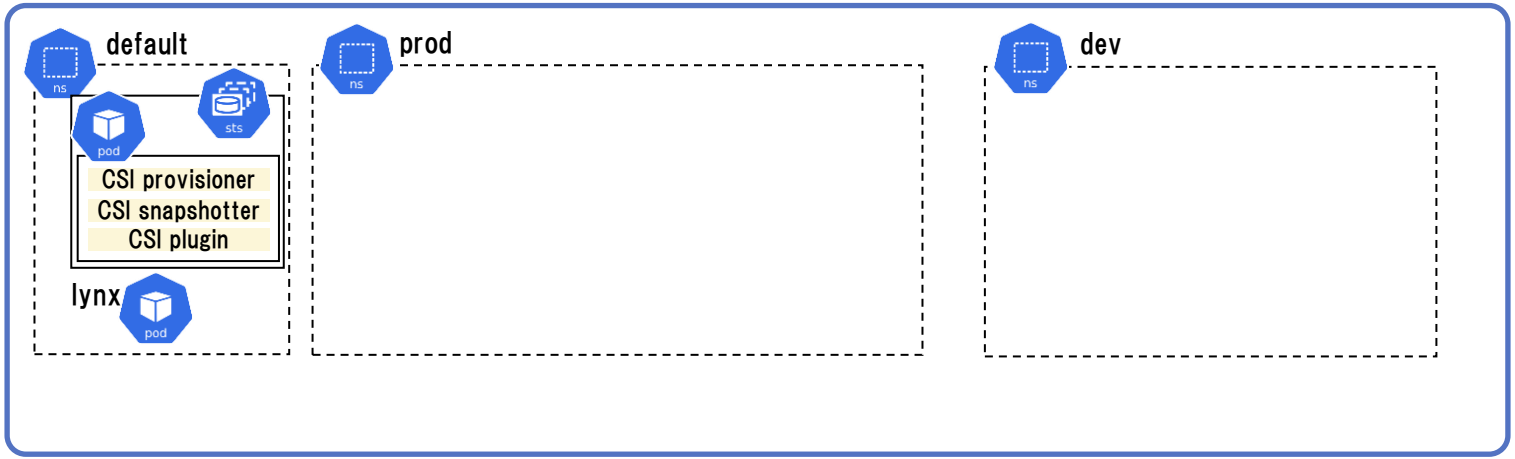
- Demo configuration
  - k8s v1.26.0 + external-provisioner (v3.4.0)
- Demo scenario
  - Creating snapshots from PVCs for WordPress in prod namespace
  - Creating PVCs from the snapshots and using them for WordPress in dev namespace

1. Deploying WordPress in prod
2. Accessing to the WordPress (Creating data)
3. Taking snapshots
4. Creating a ReferenceGrant
5. Creating volume from the snapshots in dev and using them from the WordPress in dev
6. Accessing to the WordPress (Checking data)



# 5-2-0 Check Demo Configuration (1/2)

- k8s v1.26.0 + external-provisioner (v3.4.0) are deployed
  - CrossNamespaceVolumeDataSource Feature Gate is enabled
- 2 namespaces, prod and dev, are created
- Lynx pod is deployed in default namespace for accessing to WordPress



# 5-2-0 Check Demo Configuration (2/2)

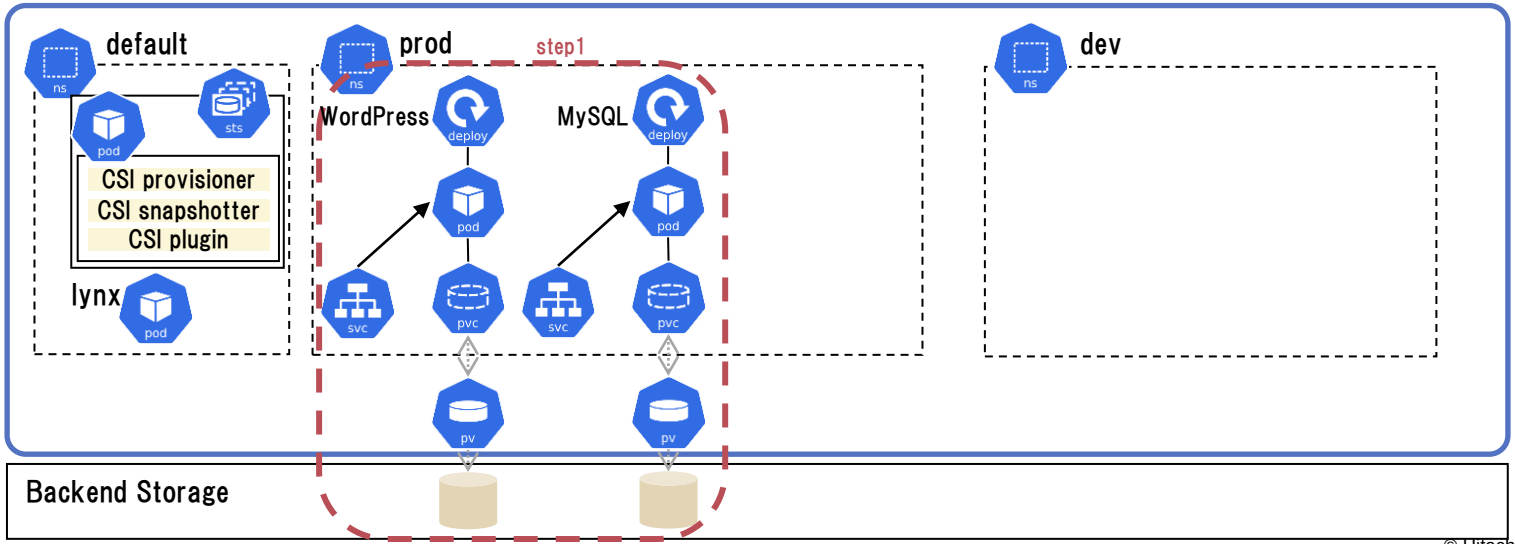
HITACHI
Inspire the Next

```
$
```

# 5-2-1 Deploying Prod WordPress (1/2)

- Deploy WordPress in prod namespace by using the steps described in Kubernetes official document
  Example: Deploying WordPress and MySQL with Persistent Volumes | Kubernetes
- Configuration:
  - WordPress: 1 pod with 1 PVC is managed by 1 Deployment.
  - MySQL : 1 pod with 1 PVC is managed by 1 Deployment.

  Note that the PVCs are empty when deployed.



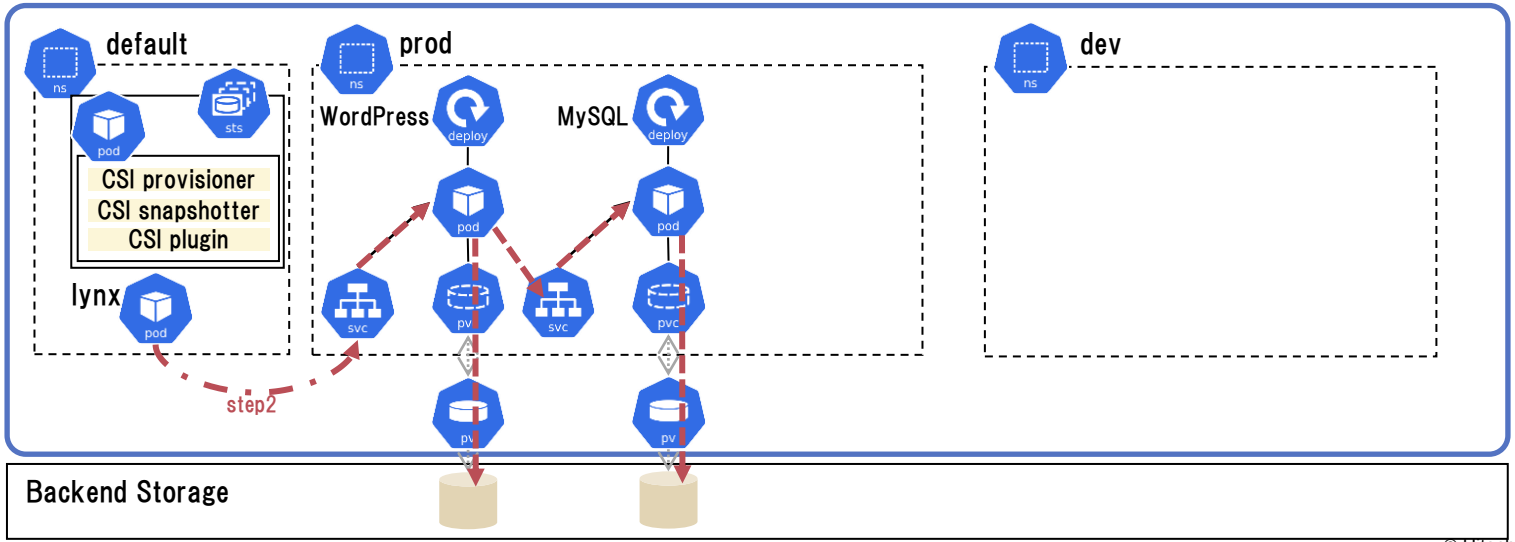
# 5-2-1 Deploying Prod WordPress (2/2)

HITACHI
Inspire the Next

```
$
```

# 5-2-2 Accessing to Prod WordPress (Creating data) (1/3)

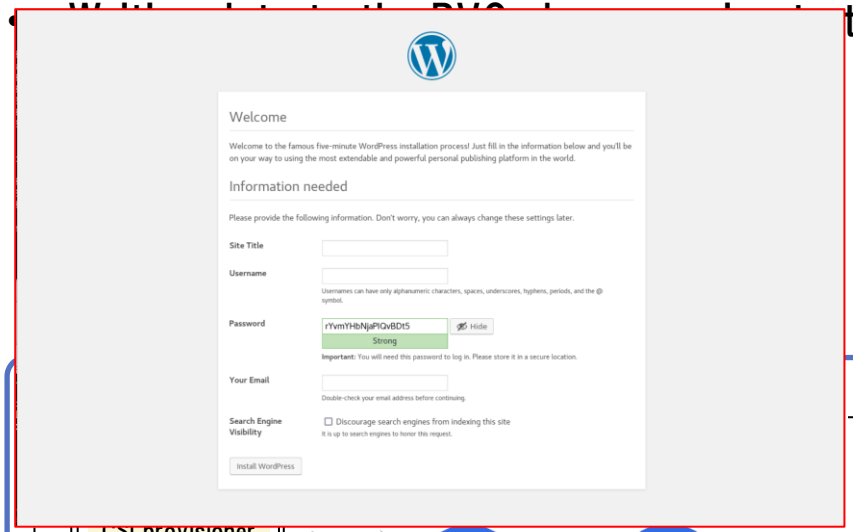
- Writing data to the PVCs by accessing to the WordPress



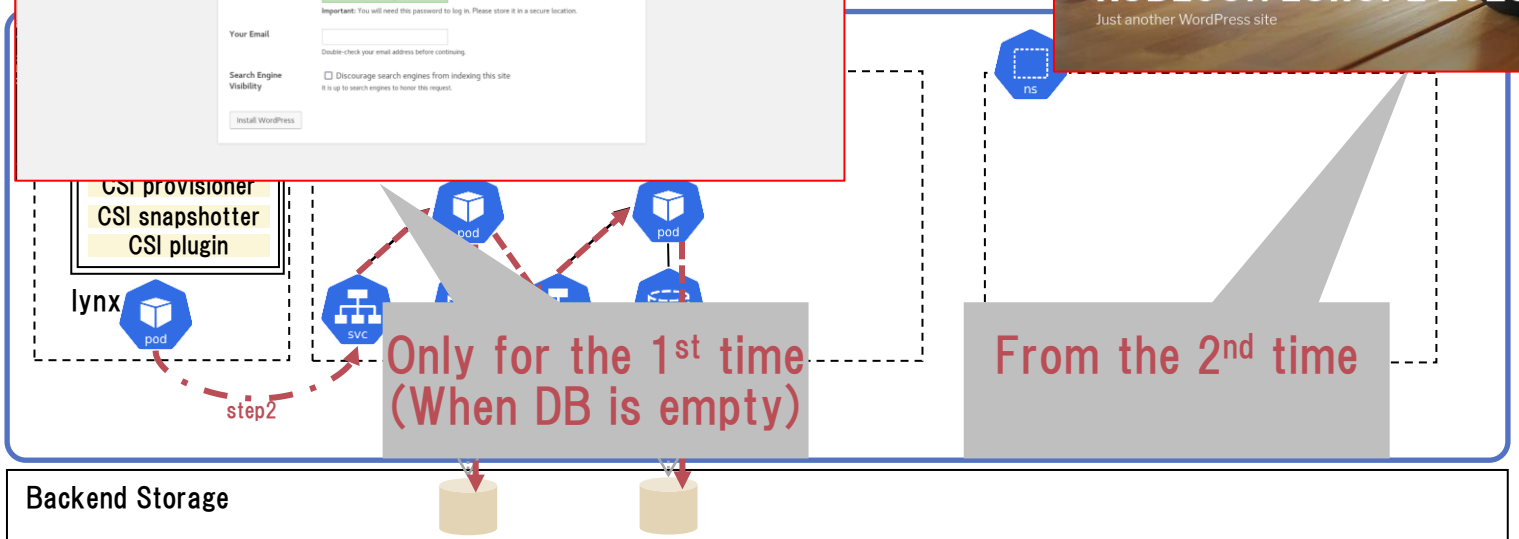
# 5-2-2 Accessing to Prod WordPress (Creating data) (2/3)

HITACHI
Inspire the Next

- With data to the PVC, access to the WordPress







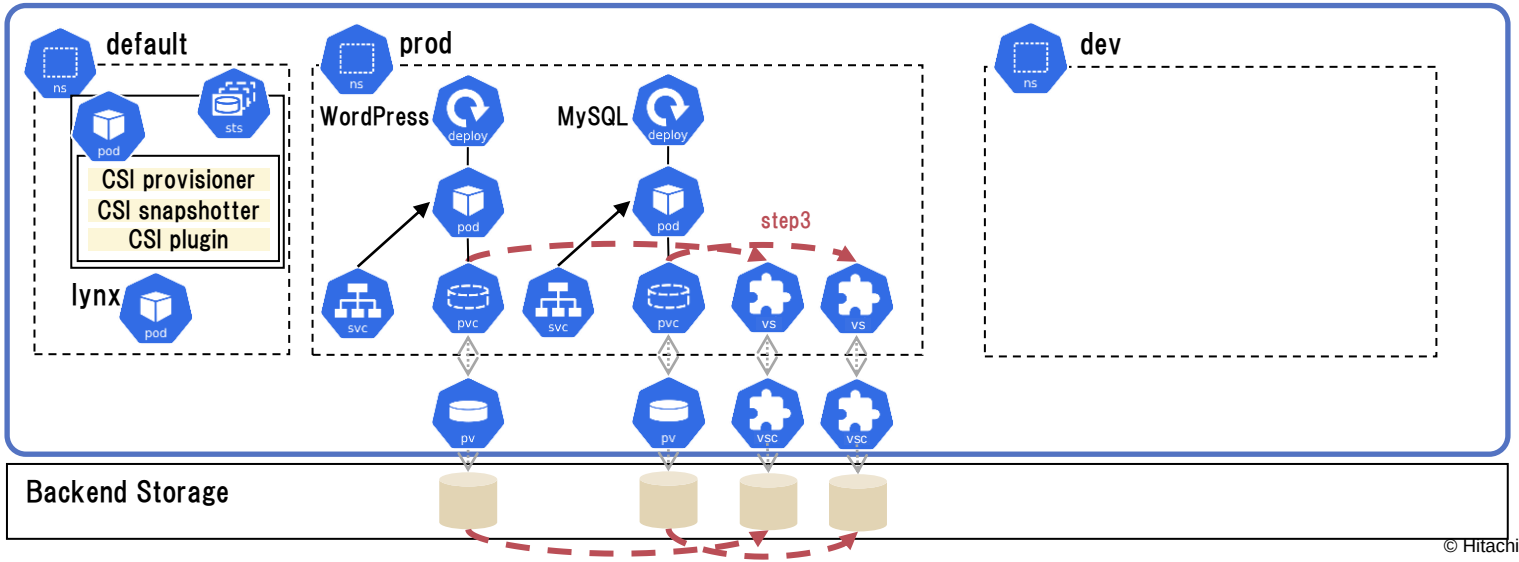
# 5-2-2 Accessing to Prod WordPress (Creating data) (3/3)

HITACHI
Inspire the Next

```
$
```

# 5-2-3 Taking Snapshots (1/2)

- Taking snapshots from PVCs for both WordPress and MySQL in prod namespace



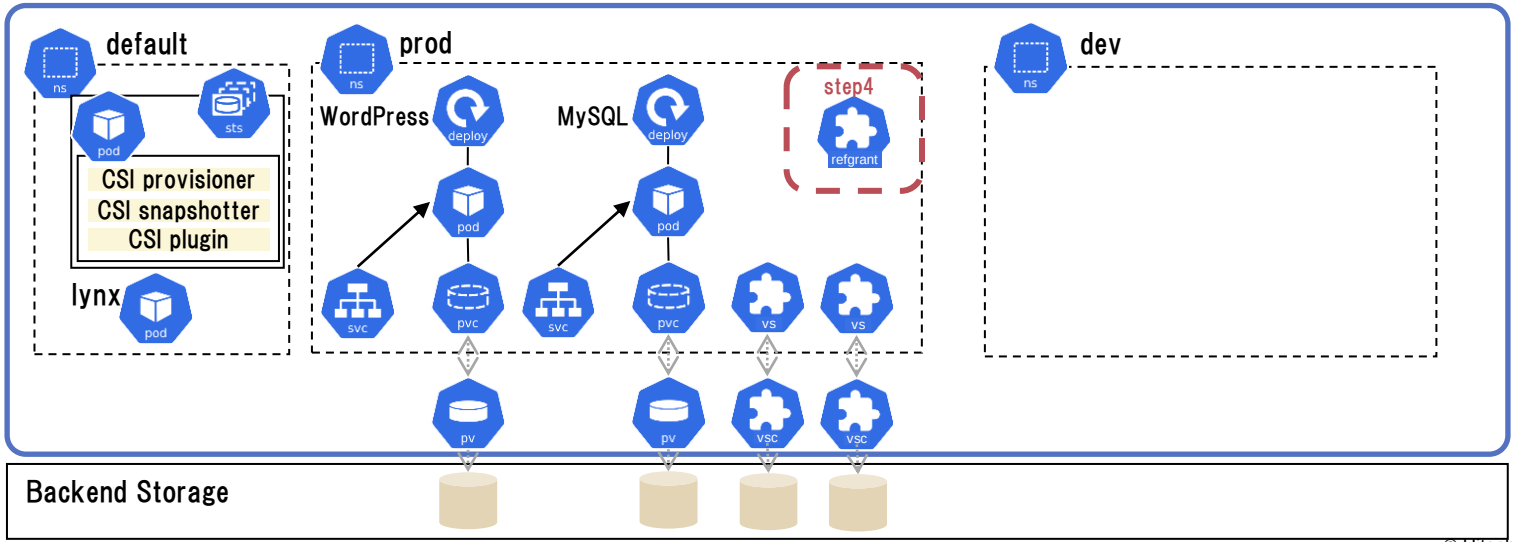
# 5-2-3 Taking Snapshots (2/2)

HITACHI
Inspire the Next

```
$
```

# 5-2-4 Creating ReferenceGrant (1/2)

- Creating a ReferenceGrant for allowing access to the snapshots in prod namespace from the PVCs in dev namespace



# 5-2-4 Creating ReferenceGrant (2/2)

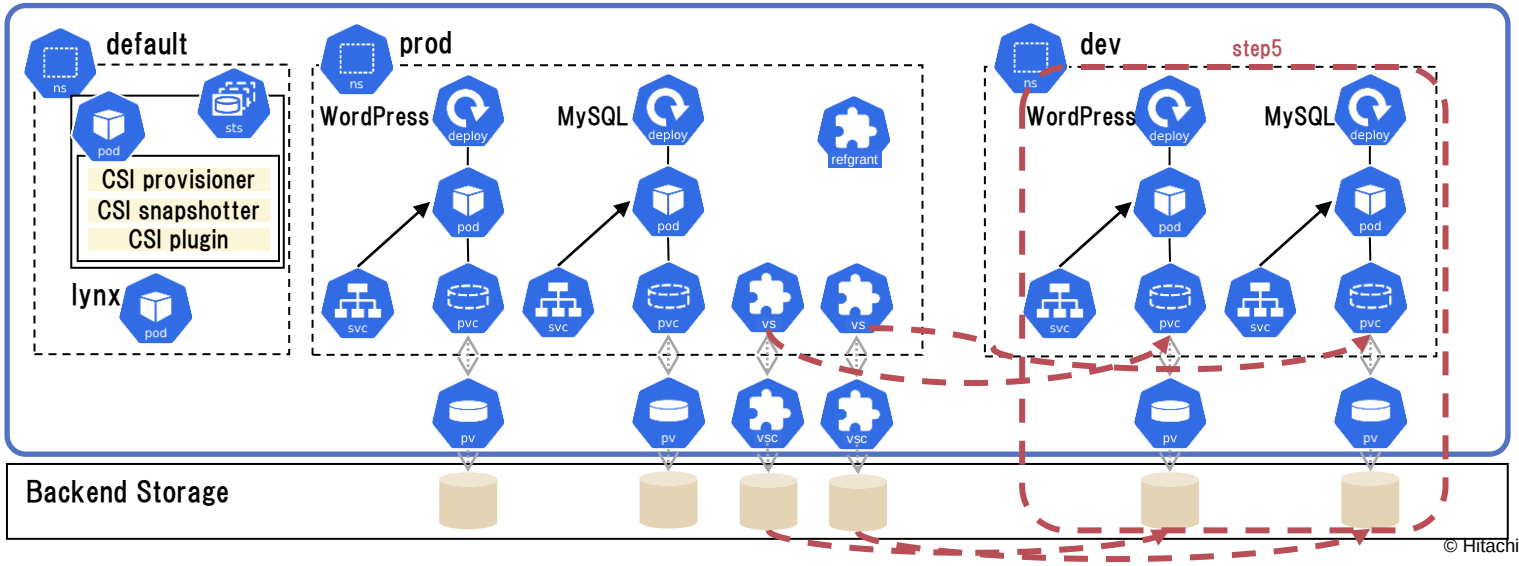
HITACHI
Inspire the Next

```
$
```

# 5-2-5 Creating Dev Volumes from Prod Snapshots and Using them

HITACHI
Inspire the Next

- Deploy WordPress in dev namespace
  - PVCs used from the WordPress are provisioned from the snapshot in prod namespace (PVCs are created with specifying the snapshots in the DataSourceRef field)



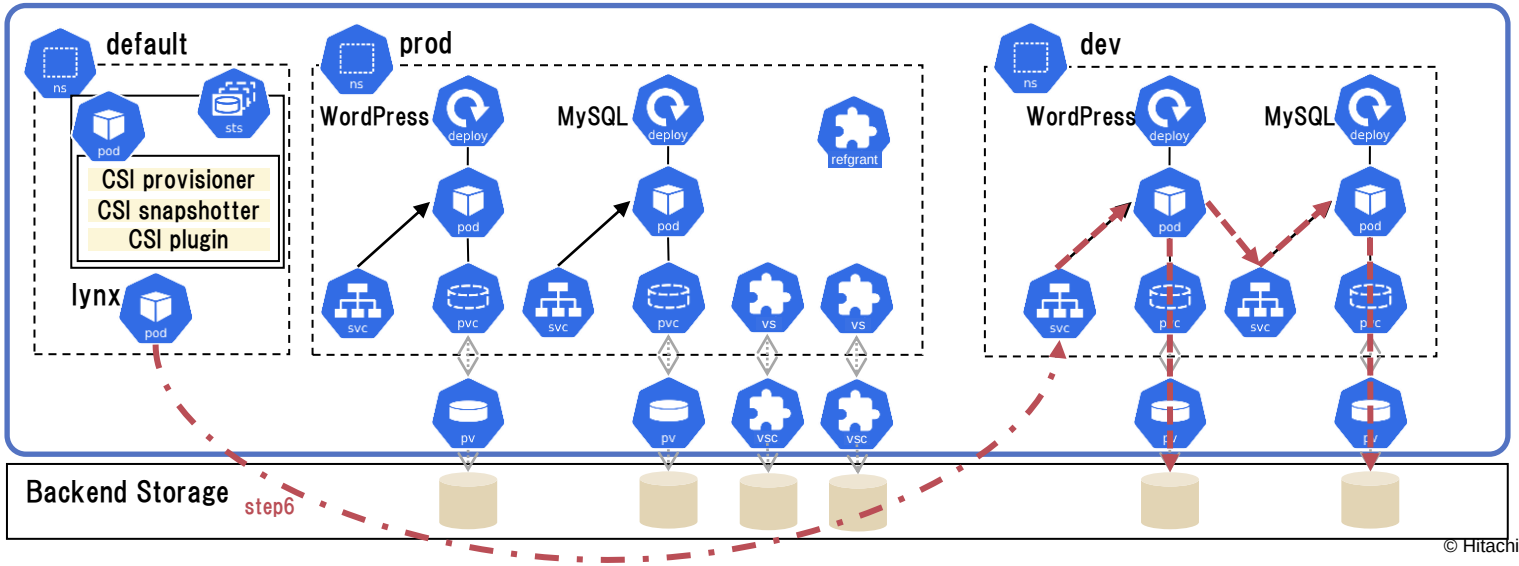
# 5-2-5 Creating Dev Volumes from Prod Snapshots and Using them

HITACHI
Inspire the Next

```
$
```

# 5-2-6 Accessing to Dev WordPress (Checking data) (1/2)

- Checking that data in dev PVCs are copied from prod snapshots by accessing dev WordPress (“KubeCon Europe 2023” should be displayed on the first access, instead of the initialization page).



# 5-2-6 Accessing to Dev WordPress (Checking data) (2/2)

HITACHI
Inspire the Next

```
$
```

HITACHI
Inspire the Next

# 6. Conclusion

## 6-1 Conclusion
## 6-2 Reference

# 6-1 Conclusion

HITACHI
Inspire the Next

- Issue: In k8s 1.25 or prior,
  - Data in production namespace can't be copied to development namespace for testing
  - Golden images in one namespace can't be shared from other namespaces

- Resolution: In k8s 1.26 or later, PVCs can be provisioned from PVCs or snapshots in a different namespace by using CrossNamespaceVolumeDataSource feature (Alpha in k8s 1.26)
  - To enable the feature (Only required during Alpha):
    - CrossNamespaceVolumeDataSource feature gate is required to be enabled for both CSI provisioner and k8s controllers
  - To utilize the feature:
    - Creating a ReferenceGrant to allow accessing to a data source from a PVC
    - Creating a PVC by specifying dataSourceRef with namespace

- Call to action: Please give your feedback on this feature to SIG Storage!

# 6-2 Reference

HITACHI
Inspire the Next

- KEP:
  - KEP-3294: Provision volumes from cross-namespace snapshots
  - Old KEP discussions:
    - Propose KEP to transfer PVC between namespaces
    - Propose KEP for namespace transfer API
    - API for PVC Namespace Transfer
    - Add proposal for transfer of VolumeSnapshot
- Documents:
  - Concept: Persistent Volumes | Kubernetes
  - Blog: Kubernetes v1.26: Alpha support for cross-namespace storage data sources | Kubernetes

# Trademarks

HITACHI
Inspire the Next

- Kubernetes is a registered trademark of The Linux Foundation.
- WordPress is a registered trademark of WordPress Foundation.
- Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates.
- Other company, product, or service names may be trademarks or service marks of others.

HITACHI
Inspire the Next

# END

## Across Kubernetes Namespace Boundaries:
## Your Volume Can Be Shared Now!

2023/4/20 @ KubeCon + CloudNativeCon Europe 2023

Takafumi Takahashi, Hitachi Vantara LLC
Masaki Kimura, Hitachi, Ltd.


Hitachi Social Innovation is
POWERING GOOD

KubeCon CloudNativeCon Europe 2023

Please scan the QR Code above
to leave feedback on this session