


KubeCon | CloudNativeCon

North America 2019

Storage on Kubernetes

Learning From Failures

Hemant Kumar, Jan Šafránek
Red Hat

Agenda

- Data loss.
- Security issues.
- Data corruption.
- Attach/detach issues.
- Open issues.

Data lost during migration

Data lost during migration

What happened?

1. User moves PV an PVC objects from "testing" to "production" clusters.
   - On the testing cluster:

```
$ kubectl get pv -o yaml > pvs.yaml
$ kubectl get pvc -o yaml > pvcs.yaml
```

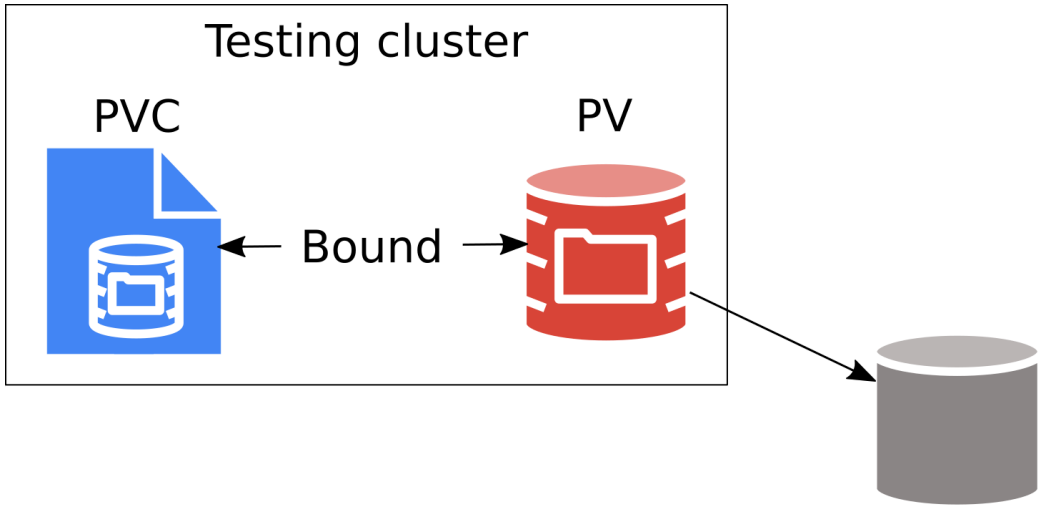
   - On the production cluster:

```
$ kubectl apply -f pvs.yaml
$ kubectl apply -f pvcs.yaml
```

2. **Kubernetes deletes PV and the volume in storage backend.**

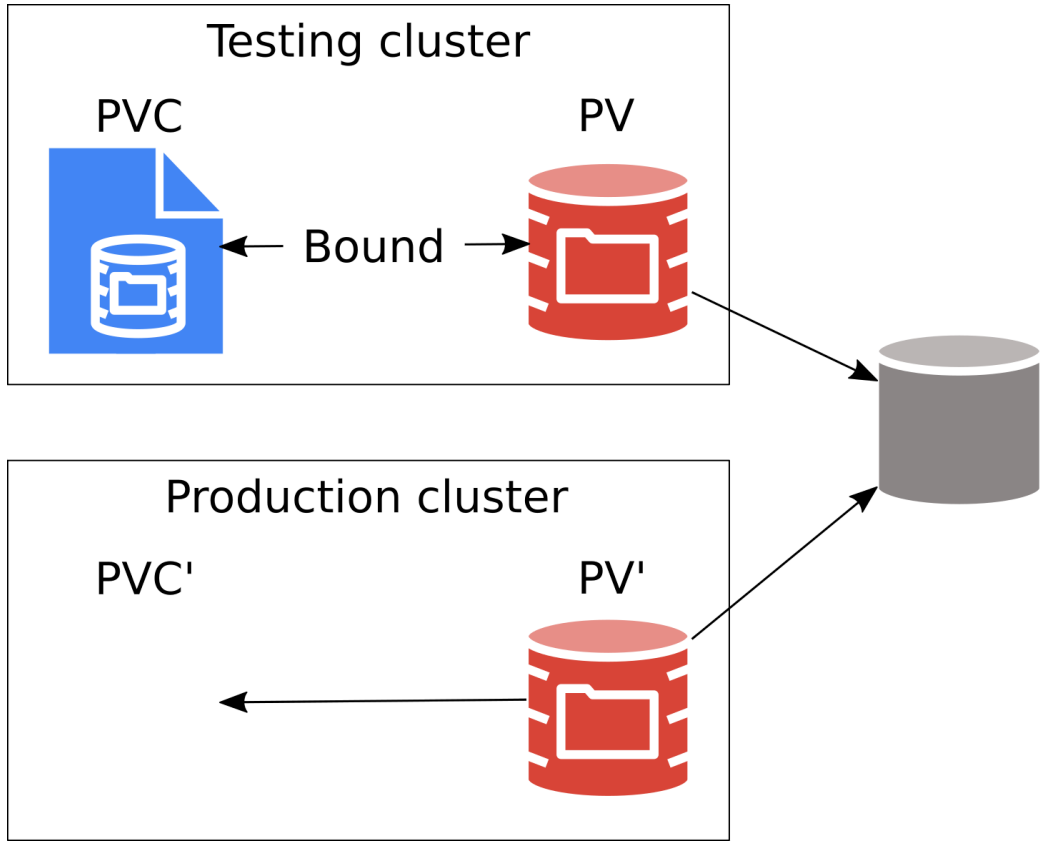
Data lost during migration

Why?



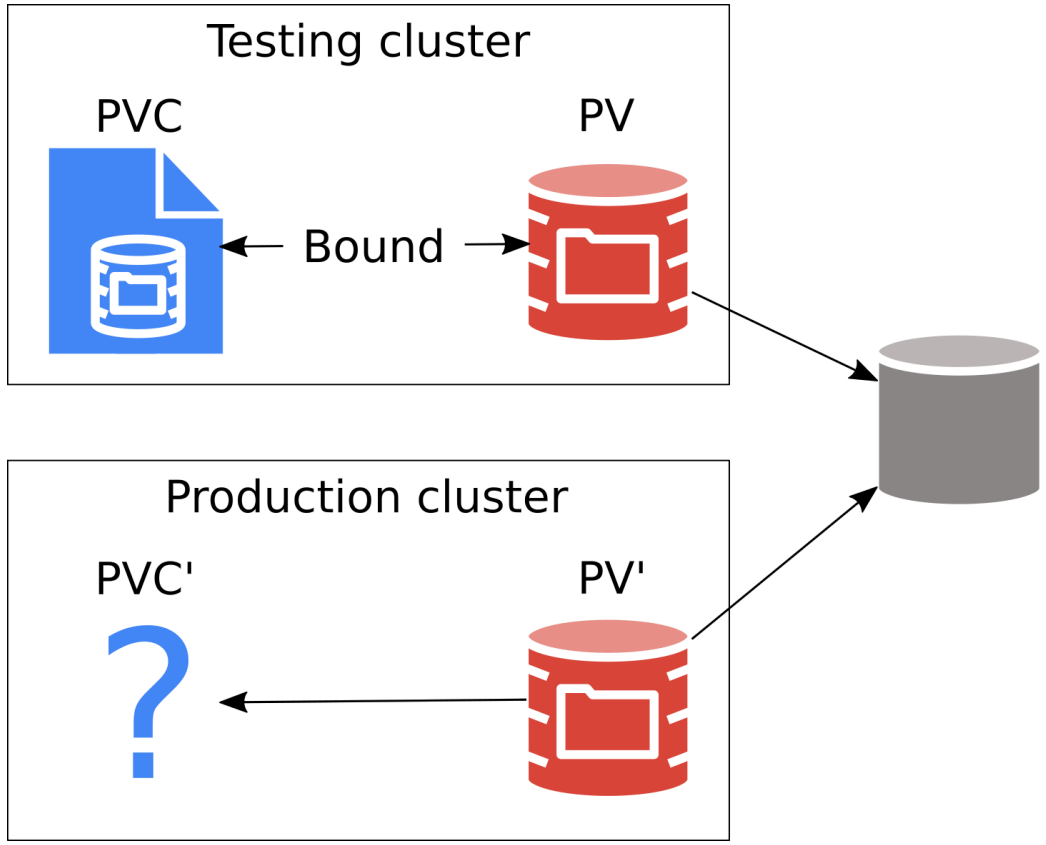
Data lost during migration

Why?



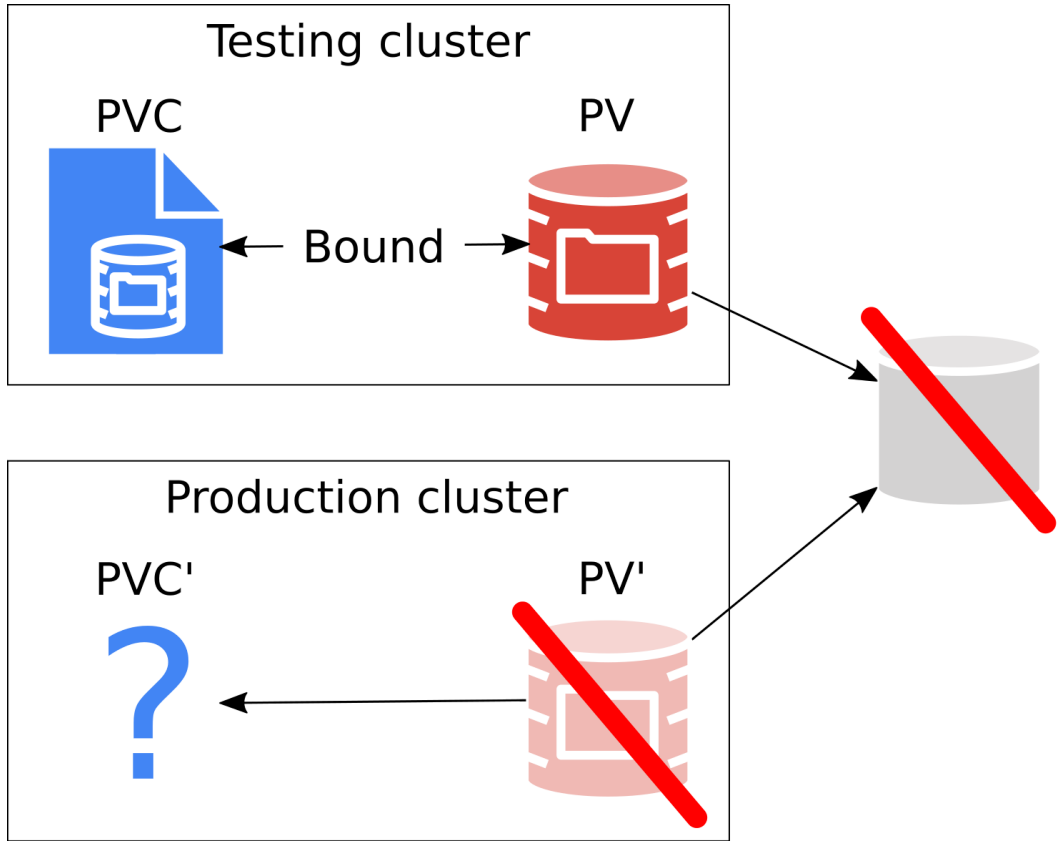
Data lost during migration

Why?



Data lost during migration

Why?



Data lost during migration

It's not a bug, it's a feature!

- Do regular backups!
- Use dedicated tools for migration, such as Ark / Velero.
  - *How to Backup and Restore Your Kubernetes Cluster - Annette Clewett & Dylan Murray, Tuesday 4:25pm.*
- Do not mess up with PVs/PVCs.

Data lost during migration

It's not a bug, it's a feature!

- Do regular backups!
- Use dedicated tools for migration, such as Ark / Velero.
  - *How to Backup and Restore Your Kubernetes Cluster - Annette Clewett & Dylan Murray, Tuesday 4:25pm.*
- Do not mess up with PVs/PVCs.
- But if you want to…
  - Use `Retain` reclaim policy.
  - Sanitize PVCs and PVs before restoring them.
    - Clean `pv.spec.claimRef.UID`.
    - Clean Kubernetes annotations on PV/PVC.

Data lost during migration

It's not a bug, it's a feature!

- Do regular backups!
- Use dedicated tools for migration, such as Ark / Velero.
  - *How to Backup and Restore Your Kubernetes Cluster - Annette Clewett & Dylan Murray, Tuesday 4:25pm.*
- Do not mess up with PVs/PVCs.
- But if you want to…
  - Use `Retain` reclaim policy.
  - Sanitize PVCs and PVs before restoring them.
    - Clean `pv.spec.claimRef.UID`.
    - Clean Kubernetes annotations on PV/PVC.

Lessons learned:

- Education.
- Better documentation.

Volumes are recycled while they are used by pods

Volumes are recycled while they are used by pods

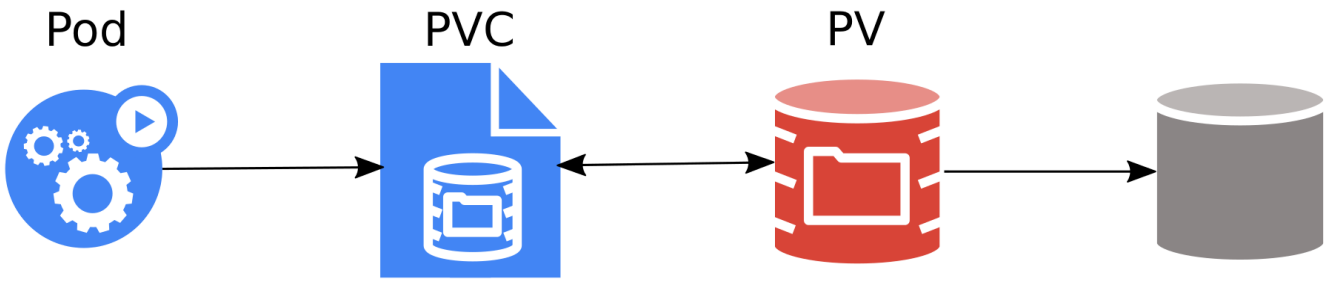
What happened?

- User deletes PVC while it's still used by a pod.
- All data on the volume are wiped.

Volumes are recycled while they are used by pods

Why?

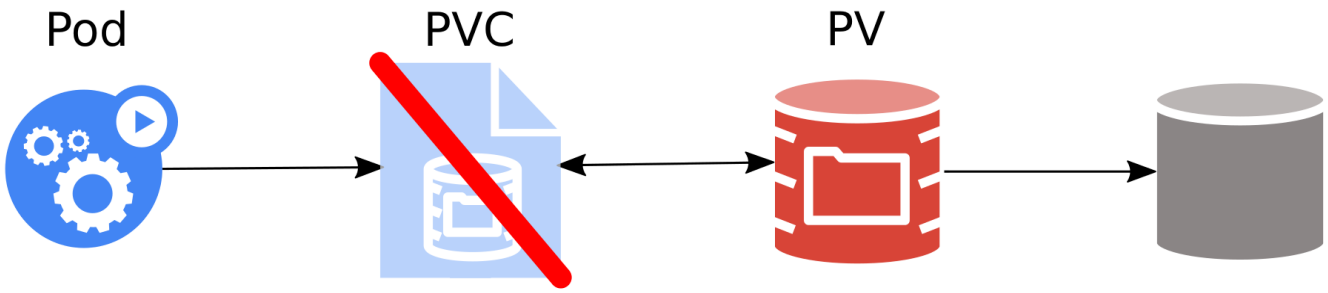
- Kubernetes has no referential integrity.



Volumes are recycled while they are used by pods

Why?

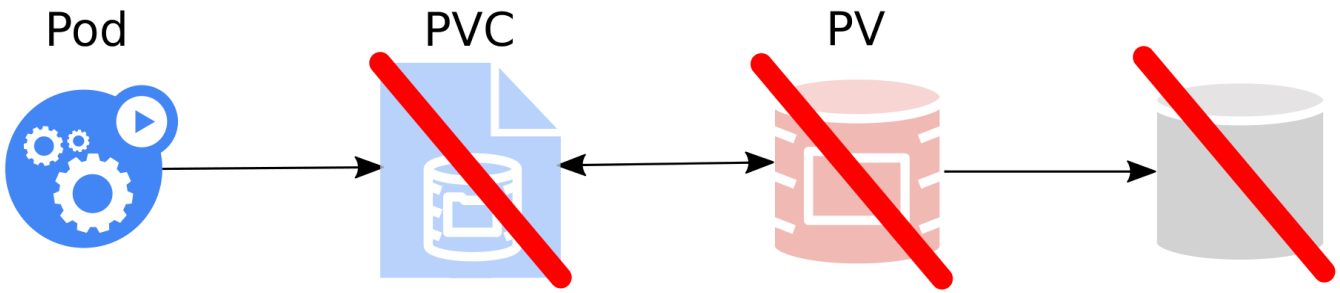
- Kubernetes has no referential integrity.



Volumes are recycled while they are used by pods

Why?

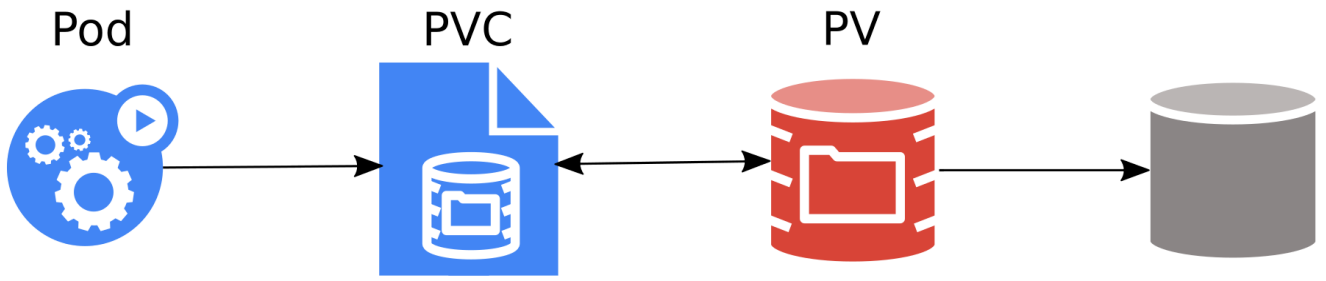
- Kubernetes has no referential integrity.



Volumes are recycled while they are used by pods

How we fixed it?

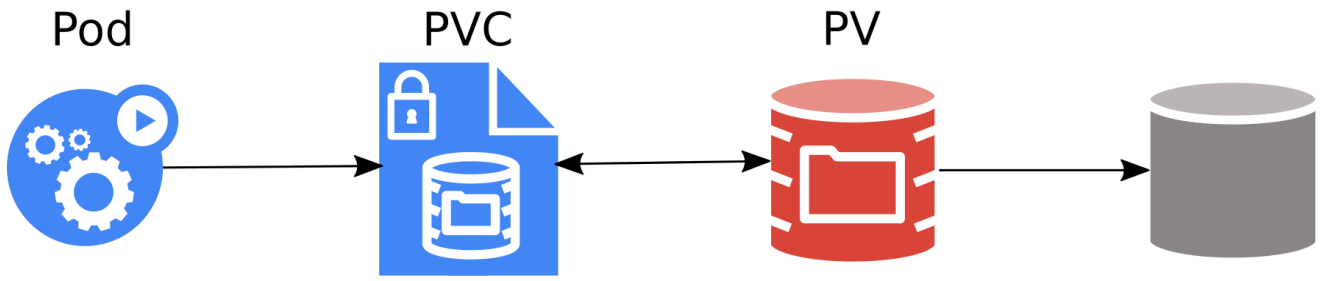
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

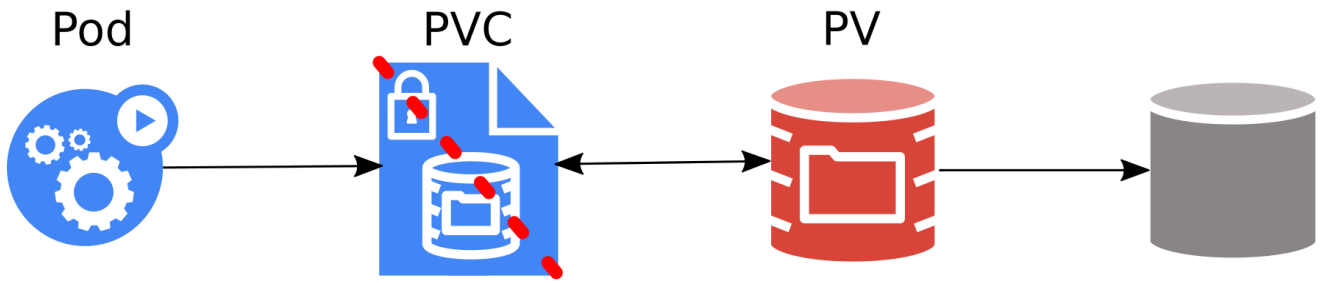
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

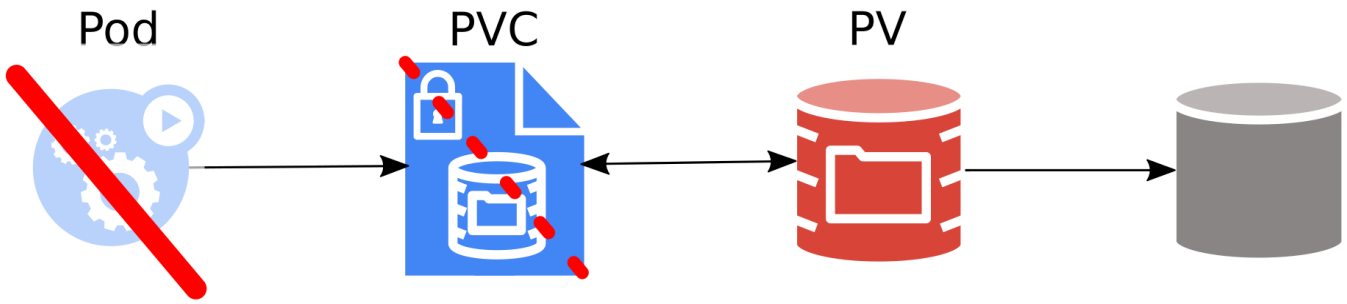
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

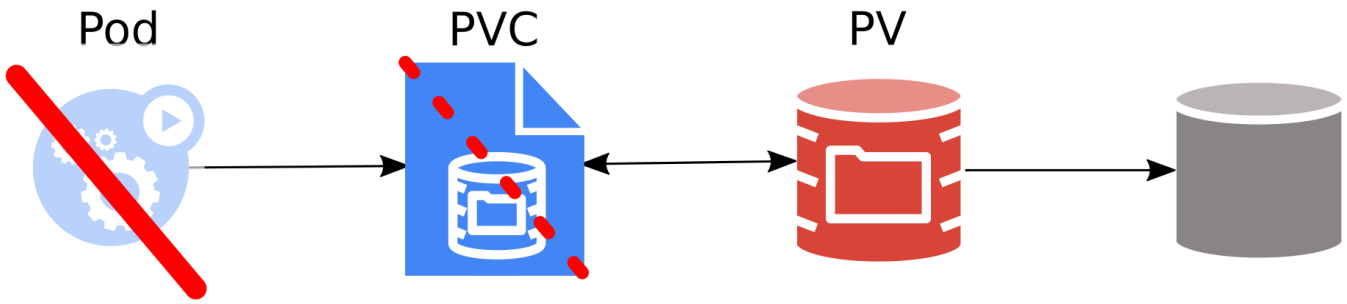
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

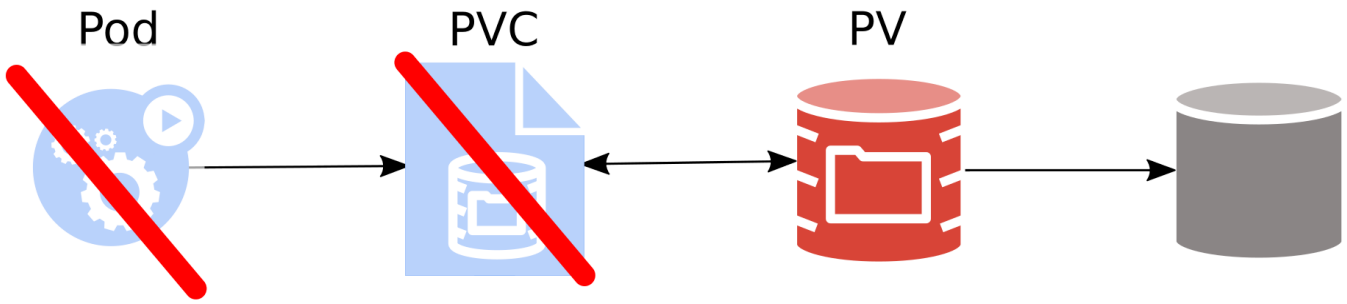
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

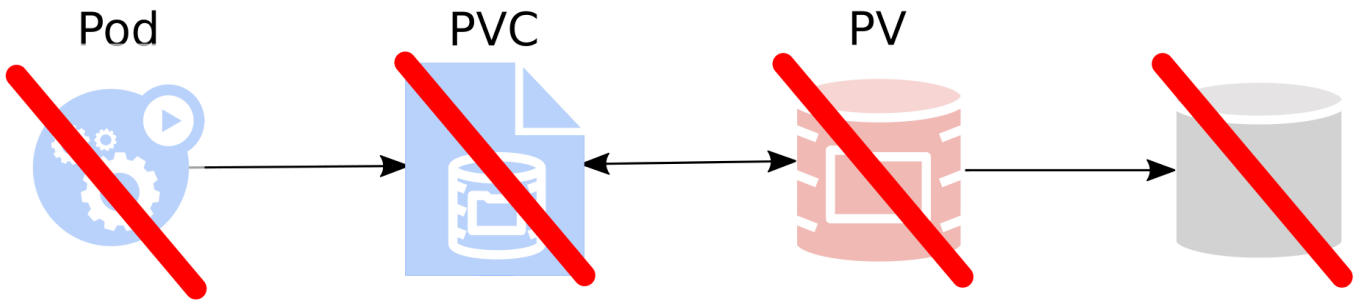
- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Volumes are recycled while they are used by pods

How we fixed it?

- Using `Finalizers`.
- StorageInUseProtection admission plugin and controller.



Data on `PersistentVolume` wiped after kubelet restart

Data on PV wiped after kubelet restart

What happened?

- Kubelet is offline and a running pod is deleted in the API server.
- Newly (re)started kubelet deletes all data on a volume that the pod used.

Data on PV wiped after kubelet restart

What happened?

- Kubelet is offline and a running pod is deleted in the API server.
- Newly (re)started kubelet deletes all data on a volume that the pod used.

Why?

- Newly (re)started kubelet does not see the pod in API server.
  - kubelet did not unmount the volume.
  - Orphan directory scan removed all files in presumably empty pod directory.

Data on PV wiped after kubelet restart

How we fixed it?

- Review all `os.RemoveAll` in Kubernetes.
  - Never delete orphan directories across filesystem boundary.

Data on PV wiped after kubelet restart

How we fixed it?

- Review all `os.RemoveAll` in Kubernetes.
  - Never delete orphan directories across filesystem boundary.
- Introduce *reconstruction*.
  - Scan `/var/lib/kubelet` on kubelet start and reconstruct caches.

Data on PV wiped after kubelet restart

How we fixed it?

- Review all `os.RemoveAll` in Kubernetes.
  - Never delete orphan directories across filesystem boundary.
- Introduce *reconstruction*.
  - Scan `/var/lib/kubelet` on kubelet start and reconstruct caches.

Lessons learned

- Introduced `[Distuptive]` tests for kubelet restart.

Data on `PersistentVolume` wiped after kubelet restart *again*

Data on PV wiped after kubelet restart *again*

What happened?

- A directory on the root disk used as local volume wiped out.
- Same scenario as above.

Data on PV wiped after kubelet restart *again*

What happened?

- A directory on the root disk used as local volume wiped out.
- Same scenario as above.

Why?

- Root disk used as a local volume does not introduce filesystem boundary.
- The local volume was used with `SubPath` feature.

Data on PV wiped after kubelet restart *again*

What happened?

- A directory on the root disk used as local volume wiped out.
- Same scenario as above.

Why?

- Root disk used as a local volume does not introduce filesystem boundary.
- The local volume was used with `SubPath` feature.

How we fixed it?

- Check for SubPath volumes before removing orphan directories.

Data on PV wiped after kubelet restart *again*

What happened?

- A directory on the root disk used as local volume wiped out.
- Same scenario as above.

Why?

- Root disk used as a local volume does not introduce filesystem boundary.
- The local volume was used with `SubPath` feature.

How we fixed it?

- Check for SubPath volumes before removing orphan directories.

Lessons learned

- Introduce `[Disruptive]` tests for kubelet restart with `SubPath`.

CVE-2017-1002101

Subpath volume mount handling allows arbitrary file access in host filesystem

CVE-2017-1002101

What happened?

"Subpath volume mount handling allows arbitrary file access in host filesystem"

A pod can get access to full host filesystem, including:

- Container runtime socket.
- Any Secrets present on the node.
- Any Pod volume present on the node.
- ...

CVE-2017-1002101

What happened?

"Subpath volume mount handling allows arbitrary file access in host filesystem"

A pod can get access to full host filesystem, including:

- Container runtime socket.
- Any Secrets present on the node.
- Any Pod volume present on the node.
- ...

Why?

- Symlinks created *in a pod* were evaluated *outside of the pod*.

CVE-2017-1002101

How we fixed it?

KubeCon NA 2018: How Symlinks Pwned Kubernetes (And How We Fixed It) - Michelle Au, Google & Jan Šafránek, Red Hat.

CVE-2017-1002101

How we fixed it?

KubeCon NA 2018: How Symlinks Pwned Kubernetes (And How We Fixed It) - Michelle Au, Google & Jan Šafránek, Red Hat.

Lessons learned

- Don't trust user.
- Containers can introduce security issues not seen before.
- Kubernetes Security Response Team (aka Product Security Committee) works and is helpful.

Corrupted filesystem on ReadWriteOnce volumes

Corrupted filesystem on ReadWriteOnce volumes

Story of two bugs, two years apart:

- Nobody wants this in their Kernel logs

```
[2480314.265276] XFS (dm-43): Unmounting Filesystem
[2480314.543698] device-mapper: ioctl: remove_all left 68 open device(s)
[2480342.623544] XFS (dm-7): Metadata corruption detected at xfs_inode_buf_verify
[2480342.623703] XFS (dm-7): Unmount and run xfs_repair
[2480342.623786] XFS (dm-7): First 64 bytes of corrupted metadata buffer:
```

Corrupted filesystem on ReadWriteOnce volumes

Story of two bugs, two years apart:

- Nobody wants this in their Kernel logs

```
[2480314.265276] XFS (dm-43): Unmounting Filesystem
[2480314.543698] device-mapper: ioctl: remove_all left 68 open device(s)
[2480342.623544] XFS (dm-7): Metadata corruption detected at xfs_inode_buf_verify
[2480342.623703] XFS (dm-7): Unmount and run xfs_repair
[2480342.623786] XFS (dm-7): First 64 bytes of corrupted metadata buffer:
```

- Kubernetes/Openshift version - 1.10/3.10

Corrupted filesystem on ReadWriteOnce volumes

Story of two bugs, two years apart:

- Nobody wants this in their Kernel logs

```
[2480314.265276] XFS (dm-43): Unmounting Filesystem
[2480314.543698] device-mapper: ioctl: remove_all left 68 open device(s)
[2480342.623544] XFS (dm-7): Metadata corruption detected at xfs_inode_buf_verify
[2480342.623703] XFS (dm-7): Unmount and run xfs_repair
[2480342.623786] XFS (dm-7): First 64 bytes of corrupted metadata buffer:
```

- Kubernetes/Openshift version - 1.10/3.10
- Volume type: Fiber channel

Corrupted filesystem on ReadWriteOnce volumes

Story of two bugs, two years apart:

- Nobody wants this in their Kernel logs

```
[2480314.265276] XFS (dm-43): Unmounting Filesystem
[2480314.543698] device-mapper: ioctl: remove_all left 68 open device(s)
[2480342.623544] XFS (dm-7): Metadata corruption detected at xfs_inode_buf_verify
[2480342.623703] XFS (dm-7): Unmount and run xfs_repair
[2480342.623786] XFS (dm-7): First 64 bytes of corrupted metadata buffer:
```

- Kubernetes/Openshift version - 1.10/3.10
- Volume type: Fiber channel
- Reported on: November 2017

Corrupted filesystem on ReadWriteOnce volumes

- And neither we want this in our Kernel logs

```
Aug 26 22:34:57.001029 ip-10-0-6 kernel: XFS (rbd0): Metadata corruption detected at xfs_dir
Aug 26 22:34:57.001213 ip-10-0-6 kernel: XFS (rbd0): Unmount and run xfs_repair
Aug 26 22:34:57.001342 ip-10-0-6 kernel: XFS (rbd0): First 128 bytes of corrupted metadata
```

Corrupted filesystem on ReadWriteOnce volumes

- And neither we want this in our Kernel logs

```
Aug 26 22:34:57.001029 ip-10-0-6 kernel: XFS (rbd0): Metadata corruption detected at xfs_dir
Aug 26 22:34:57.001213 ip-10-0-6 kernel: XFS (rbd0): Unmount and run xfs_repair
Aug 26 22:34:57.001342 ip-10-0-6 kernel: XFS (rbd0): First 128 bytes of corrupted metadata
```

- Kubernetes/Openshift version - 1.14/4.2

Corrupted filesystem on ReadWriteOnce volumes

- And neither we want this in our Kernel logs

```
Aug 26 22:34:57.001029 ip-10-0-6 kernel: XFS (rbd0): Metadata corruption detected at xfs_dir
Aug 26 22:34:57.001213 ip-10-0-6 kernel: XFS (rbd0): Unmount and run xfs_repair
Aug 26 22:34:57.001342 ip-10-0-6 kernel: XFS (rbd0): First 128 bytes of corrupted metadata
```

- Kubernetes/Openshift version - 1.14/4.2
- Volume type: Ceph-RBD via CSI/Rook

Corrupted filesystem on ReadWriteOnce volumes

- And neither we want this in our Kernel logs

```
Aug 26 22:34:57.001029 ip-10-0-6 kernel: XFS (rbd0): Metadata corruption detected at xfs_dir
Aug 26 22:34:57.001213 ip-10-0-6 kernel: XFS (rbd0): Unmount and run xfs_repair
Aug 26 22:34:57.001342 ip-10-0-6 kernel: XFS (rbd0): First 128 bytes of corrupted metadata
```

- Kubernetes/Openshift version - 1.14/4.2
- Volume type: Ceph-RBD via CSI/Rook
- Reported on: August 2019

Corrupted filesystem on ReadWriteOnce volumes

What happened?

- Same volume could be temporarily mounted on more than one node.

Corrupted filesystem on ReadWriteOnce volumes

What happened?

- Same volume could be temporarily mounted on more than one node.

How do we fix it?

- Storage Provider should fix it.
- Enforce AccessModes.

Corrupted filesystem on ReadWriteOnce volumes

So what are AccessModes?

Corrupted filesystem on ReadWriteOnce volumes

So what are AccessModes?

- ReadWriteOnce
- ReadWriteMany
- ReadOnlyMany

Corrupted filesystem on ReadWriteOnce volumes

So what are AccessModes?

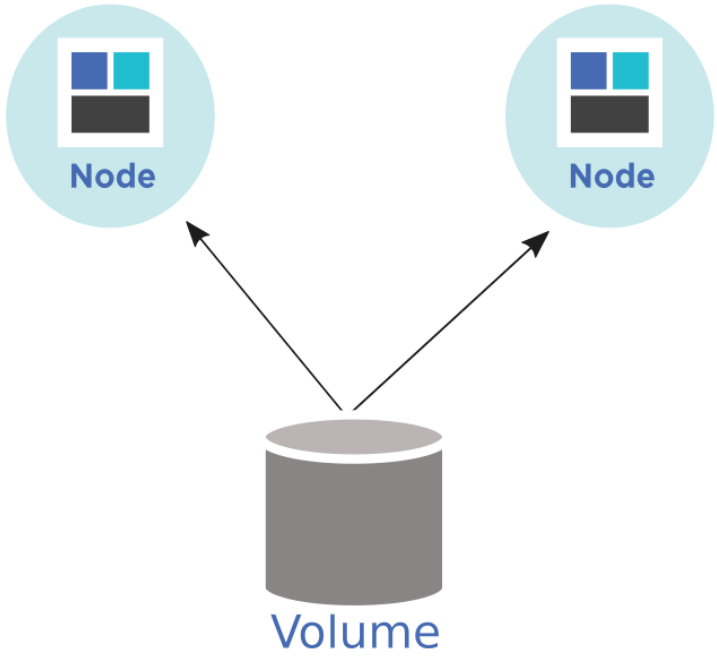
- ReadWriteOnce
- ReadWriteMany
- ReadOnlyMany

You can request a volume of specific AccessMode while creating a PVC:

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

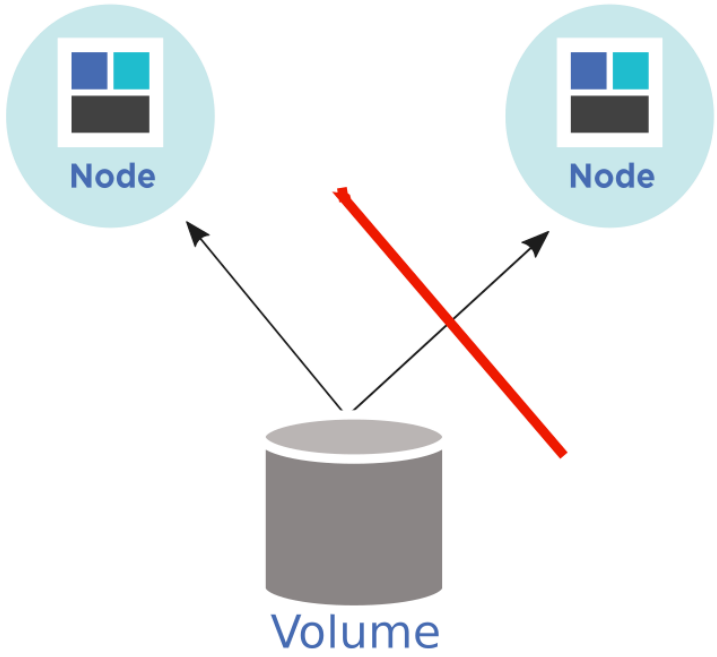
Corrupted filesystem on ReadWriteOnce volumes

Kubernetes did not enforce AccessModes at all until version 1.7/1.8



Corrupted filesystem on ReadWriteOnce volumes

Kubernetes did not enforce AccessModes at all until version 1.7/1.8



Corrupted filesystem on ReadWriteOnce volumes

But those two bugs are newer - 1.10 and 1.14!

Corrupted filesystem on ReadWriteOnce volumes

Limitations of AccessMode enforcement in Kubernetes

Corrupted filesystem on ReadWriteOnce volumes

Limitations of AccessMode enforcement in Kubernetes

- It only works for volume types that are Attachable.

Corrupted filesystem on ReadWriteOnce volumes

Limitations of AccessMode enforcement in Kubernetes

- It only works for volume types that are Attachable.
- It does not prevent 2 pods from using same volume on same node.

Corrupted filesystem on ReadWriteOnce volumes

Limitations of AccessMode enforcement in Kubernetes

- It only works for volume types that are Attachable.
- It does not prevent 2 pods from using same volume on same node.
- It is based on cached volume state in controller-manager.

Corrupted filesystem on ReadWriteOnce volumes

Attachable volumes:

- AWS EBS
- OpenStack Cinder
- GCE PD
- vSphere disks
- CSI volume that does have `PUBLISH_UNPUBLISH_VOLUME` capability.

Corrupted filesystem on ReadWriteOnce volumes

Attachable volumes:

- AWS EBS
- OpenStack Cinder
- GCE PD
- vSphere disks
- CSI volume that does have `PUBLISH_UNPUBLISH_VOLUME` capability.

Volume types which are not attachable:

- iSCSI
- Ceph-RBD
- Fiber Channel
- CSI volume that does not have `PUBLISH_UNPUBLISH_VOLUME` capability.

Corrupted filesystem on ReadWriteOnce volumes

Fix for non-attachable volumes(in-tree)

Corrupted filesystem on ReadWriteOnce volumes

Fix for non-attachable volumes(in-tree)

- Implement a dummy `Attach` and `Detach` interface which is basically a NOOP for `iSCSI`, `FC` and `Ceph-RBD`.

Corrupted filesystem on ReadWriteOnce volumes

Fix for non-attachable volumes(in-tree)

- Implement a dummy `Attach` and `Detach` interface which is basically a NOOP for `iSCSI`, `FC` and `Ceph-RBD`.
- This would basically turn non-attachable volume types into attachable.

Corrupted filesystem on ReadWriteOnce volumes

Fix for non-attachable volumes(in-tree)

- Implement a dummy `Attach` and `Detach` interface which is basically a NOOP for `iSCSI`, `FC` and `Ceph-RBD`.
- This would basically turn non-attachable volume types into attachable.
- It will ensure that volume is made available on a node via control-plane attach/detach controller and not directly.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.
  - Pushes the problem back to storage provider from Kubernetes.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.
  - Pushes the problem back to storage provider from Kubernetes.
  - May not be possible in some cases where an off-the-shelf storage solution is deployed.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.
  - Pushes the problem back to storage provider from Kubernetes.
  - May not be possible in some cases where an off-the-shelf storage solution is deployed.
- External-Attacher CSI sidecar can support NOOP attach/detach of volumes which don't have `PUBLISH_UNPUBLISH_VOLUME` capability.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.
  - Pushes the problem back to storage provider from Kubernetes.
  - May not be possible in some cases where an off-the-shelf storage solution is deployed.
- External-Attacher CSI sidecar can support NOOP attach/detach of volumes which don't have `PUBLISH_UNPUBLISH_VOLUME` capability.
  - Ensure that external-attacher is running even if CSI driver does not support attach/detach.

Corrupted filesystem on ReadWriteOnce volumes

Recommendations for CSI Volumes

- Whenever possible implement strong control-plane based fencing for publishing volumes to a node.
  - Pushes the problem back to storage provider from Kubernetes.
  - May not be possible in some cases where an off-the-shelf storage solution is deployed.
- External-Attacher CSI sidecar can support NOOP attach/detach of volumes which don't have `PUBLISH_UNPUBLISH_VOLUME` capability.
  - Ensure that external-attacher is running even if CSI driver does not support attach/detach.
  - Do not disable attach/detach from `CSIDriver` object.

Volumes not attached / detached on AWS

Volumes not attached / detached on AWS

What happened?

- AWS EBS volume was *attaching* / *detaching* forever.
- Very hard to reproduce.

Volumes not attached / detached on AWS

Kubernetes AWS cloud provider device allocator

- Re-using a device that was just released can lead to volume *attaching* forever.
  - LRU of free device names.

Volumes not attached / detached on AWS

Kubernetes AWS cloud provider device allocator

- Re-using a device that was just released can lead to volume *attaching* forever.
  - LRU of free device names.
- Node is unusable after force-detach.
  - Don't force-detach volumes on AWS!
  - Tainting nodes where attach times out.

Volumes not attached / detached on AWS

Eventual consistency

Why?

- Volume is detached, but AWS says it's attached.
- Volume is attached, but AWS says it's detached.
- Can go back in time.
  - `detaching`
  - `detached`
  - `detaching`

Volumes not attached / detached on AWS

Eventual consistency

Why?

- Volume is detached, but AWS says it's attached.
- Volume is attached, but AWS says it's detached.
- Can go back in time.
  - `detaching`
  - `detached`
  - `detaching`

How we fixed it?

- Uncertain attach state.

Volumes not attached / detached on AWS

Eventual consistency

Why?

- Volume is detached, but AWS says it's attached.
- Volume is attached, but AWS says it's detached.
- Can go back in time.
  - `detaching`
  - `detached`
  - `detaching`

How we fixed it?

- Uncertain attach state.

We still love AWS!

Open Issues

Recursive `chown`

```
$ kubectl explain pod.spec.securityContext.fsGroup

FIELD:    fsGroup <integer>

DESCRIPTION:
     A special supplemental group that applies to all containers in a pod. Some
     volume types allow the Kubelet to change the ownership of that volume to be
     owned by the pod [...]
```

- kubelet does recursive `chown` to set ownership of **all** files on the volume.
  - Slow on large volumes.
- Design in progress.
  - Take shortcuts? Some files may have wrong owner.
  - Make `chown` optional? Requires API change.
  - Use overlay FS? Requires the overlay installed on nodes.

Detaching volumes from shutdown nodes

- Kubernetes will not automatically detach volumes from nodes which have been shutdown.
  - Kubernetes does evict Pods from shutdown nodes automatically.
  - Replacement Pods on new nodes may not be able to start if they are using Persistent volumes.

Detaching volumes from shutdown nodes

Kubernetes will not detach volumes from shutdown nodes

- Pods on shutdown node do not automatically get deleted and stay in "unknown" state.
- Kubernetes does not detach volumes from Pods in "unknown" state.

Detaching volumes from shutdown nodes

How do we recover from it?

- On cloudprovider managed clusters such as AWS, GCE - running a cluster in Autoscaling group will cause a shutdown node to be deleted and replaced.
  - Volumes are automatically detached from a deleted node.
- For bare-metal clusters or cloudproviders that don't allow easy replacement of a node, this is a bigger problem.
  - An external controller can monitor for shutdown nodes and force delete pods in "unknown" state from those nodes.
- Kubernetes community is working on a design consensus that should solve this for good.
  - Add node shutdown KEP

`EmptyDir` volumes share I/O

- `EmptyDir` shares I/O bandwidth with the system and all other pods.
- Rogue pod may trash I/O performance for the others.

AWS EBS encrypted volumes occasionaly do not mount

- Sometimes newly created encrypted EBS volumes are not zeroed.
- Kubernetes does not overwrite existing data.

Summary

- Fixing bugs is never ending process.

Summary

- Fixing bugs is never ending process.
- Still learning from our failures.
  - Huge e2e test matrix.

Summary

- Fixing bugs is never ending process.
- Still learning from our failures.
  - Huge e2e test matrix.
- Kubernetes does not loose data *most* of the time.
  - Unless users ask for it.

Summary

- Fixing bugs is never ending process.
- Still learning from our failures.
  - Huge e2e test matrix.
- Kubernetes does not loose data *most* of the time.
  - Unless users ask for it.
- Still amazed by user creativity.


Questions?

Junkyard

Not fixable issues

`PersistentVolumeClaim` naming

- `Pod` is not `CPUAndMemoryClaim`.

`PersistentVolumeClaim` naming

- `Pod` is not `CPUAndMemoryClaim`.* `Service` is not `LoadBalancerClaim`.

`PersistentVolumeClaim` naming

- `Pod` is not `CPUAndMemoryClaim`. *`Service`* *is not* *`LoadBalancerClaim`*. `Volume` is `PersistentVolumeClaim` ???

`PersistentVolumeClaim` naming

- `Pod` is not `CPUAndMemoryClaim`. *`Service`* *is not* *`LoadBalancerClaim`*. `Volume` **is** `PersistentVolumeClaim` ???

"Fixed" in `VolumeSnapshot` & `VolumeSnapshotContent`.

AccessModes

- `ReadWriteOne`, `ReadWriteMany`, `ReadOnlyMany`
- Enforced only lightly in A/D controller!
  - Multiple pods can still use single `ReadWriteOne` volume on the same node.
- Fix would break behavior.

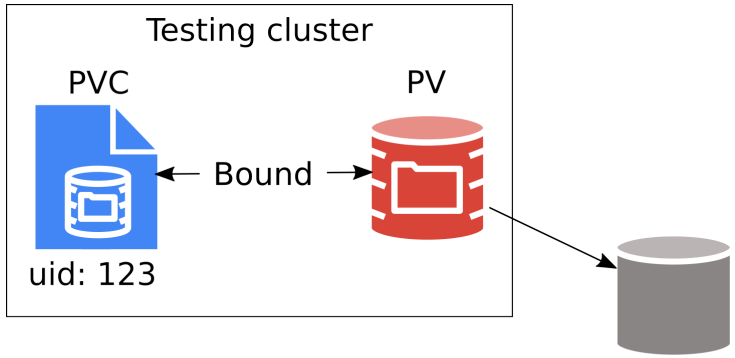
Volume reconstruction

TODO: remove? It's covered in one of the fixed issues.

- kubelet reconstructs caches from `/var/lib/kubelet/pods`.
  - TODO: add example?
  - Mostly works and is actively supported!
- There should be a real database / checkpointing.
  - Current kubelet checkpoints do not include PVCs / PVs.

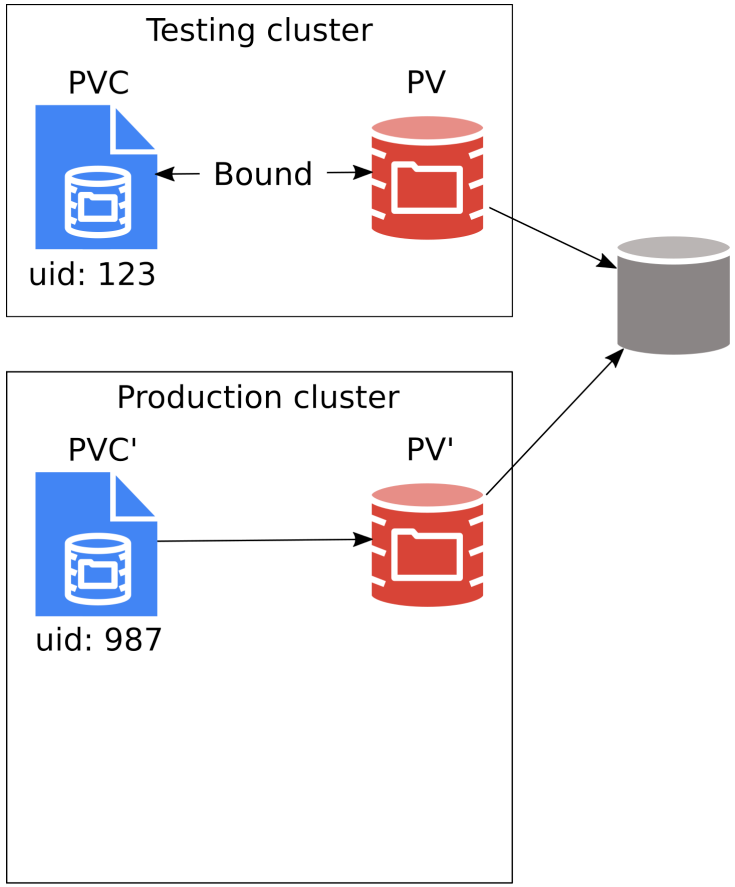
Data lost during migration

PVC first



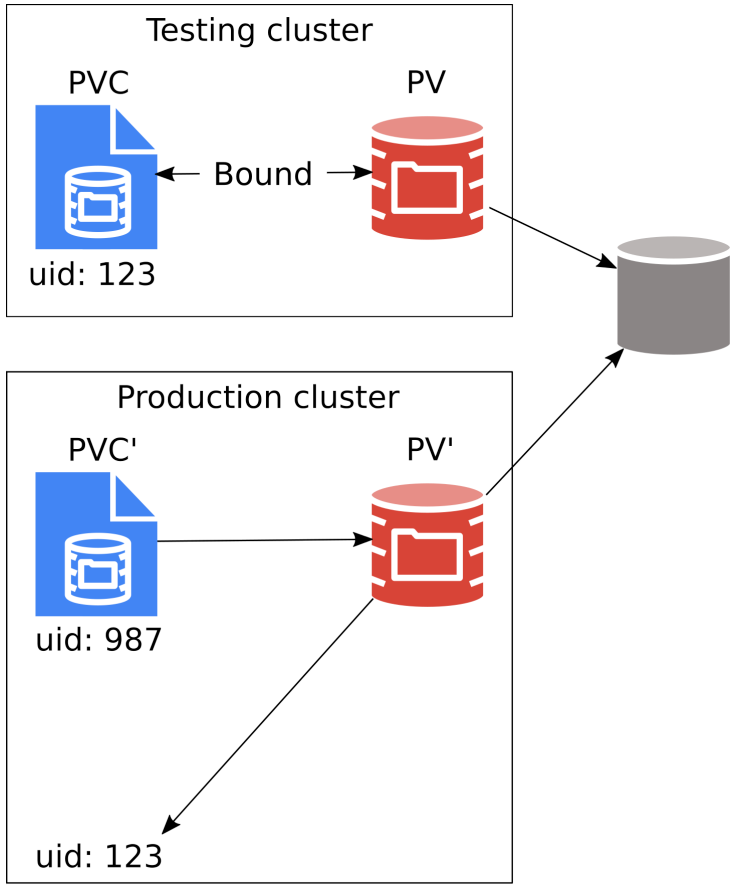
Data lost during migration

PVC first



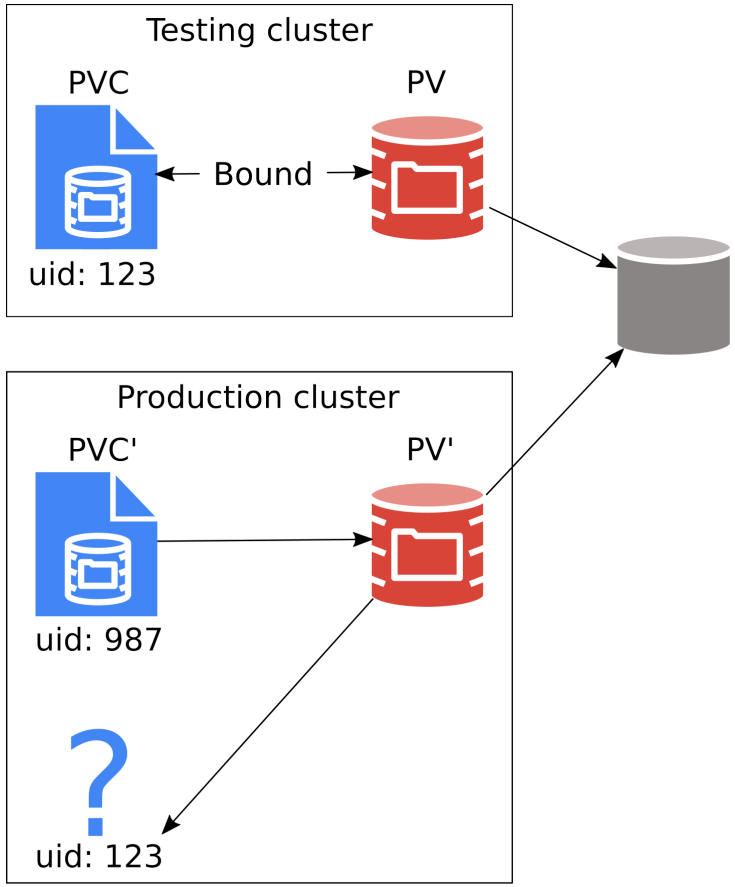
Data lost during migration

PVC first



Data lost during migration

PVC first



Data lost during migration

PVC first