Attendees

Google

Saad Ali <saadali@google.com>
Tim Hockin <thockin@google.com>

RedHat

Bradley Childs <bchilds@redhat.com>
Jan Safranek <jsafrane@redhat.com>
Sami Wagiaalla <swagiaal@redhat.com>
Huamin Chen <hchen@redhat.com>
Paul Morie <pmorie@redhat.com>
Stephen Watt <swatt@redhat.com>

Monday, March 28

Opening Discussion

- Establish Team Goals:
    - Be more collaborative, and not have all the work on one person
    - Good communication and disclosure
- What the teams want:
    - Design doc before code
        - Random person from internet must be able to read design doc and determine the *why* of the feature
    - More tests upstream
        - Enough tests of core features, part of pre-submit
        - Federated testing, give Red Hat keys to run their own tests and feedback into our infra.
            - Red Hat has 1-2 sprint work items that could put messages on commits in GitHub
        - Volume plugin testing.
    - Quick iterative reviews
        - Should grow core SIG beyond Google and Red Hat
            - NetApp EMC meetings
                - EMC
                    - Steve Wong

            - They are not sure how to engage with open source team.
        - When to consider design finished:
            - Only blocker on dynamic provisioning is that we can't do it until we pay down other debt pay down
        - Prioritization
            - Google: Kube release specific
            - Google: will try to increase headcount
            - Red Hat: Get level of anxiety down, by the end of the week, we should be open to dynamic provisioning design.
            - Collectively SIG should think about technical debt that is and is not a blocker to dynamic provisioning.
            - Dynamic Prov higher pri than Containerizing Mount
        - Most important technical debt:
            - Races between the controllers leading to bad states
            - Readability is really bad in some cases
            - Terminal bad states that are not easily recoverable

- https://www.lucidchart.com/documents/edit/78613d06-dbf4-4900-9ee9-63b6a30a9ff1/0
- Agenda
    - Design discussion for three controllers into one
        - Top down charts
        - Bottom up
            - Shows how confusing existing code is
    - If time remaining: review dynamic provisioning
        - Plausible story for FLEX?
    - Discuss Jan's Tests and how to incorporate them in to CI
    - Discuss out-of-tree provisioning

Design for controller consolidation

- Get rid of phase
- Bi-directional PVC to PV pointer
    - Between the two transactions: user sees PV as bound but PVC as unbound
    - Options:
        - Deprecate
            1. Cons: High cost backwards compatibility
                - Could introduce sub-api group version
                - Could do a new resource
        - Expect they may be out of sync and gracefully handle it
    - Sami: Should rely on resource version errors for correctness, if we rely on that we *need* the bidirectional pointer--need two points of failures to prevent too many volumes binding to a single claim or vice versa
- Error messages should be decoupled from PV/PVC

- What we want:
    - Ideally: Single transaction binding from system perspective.
    - If we can't do that then: PV to PVC should happen before PVC to PV binding?
    - Better error messaging
        - If claim can't be bound within a few seconds, an error message should be available to user.
        - Error message: events.
    - Improve UX:
        - Advocate PVC over direct volume
    - Self healing, self-rectifying control loop
- Broken claim
    - Claim points to PV, and PV does not exist, or underlying volume doesn't exist
    - Should not "auto-fix"
- Mis-bound Claim
    - PVC points to PV, PV doesn't point back
    - Is this an error case?
        - Tim: not an exceptional situation, it is a valid user scenario, it might be possible at a later time to fulfill this claim, we should reevaluate it later.
        - Sami: if this is a valid use case then it robs us of the ability to differentiate between controller failing to assign bi-directional pointers vs pre-bound claim
            1. Tim: We need another bit
            2. Sami: we might be able to use UID as that bit
            3. Why not status?
                - If all statuses fields are lost, can you recreate the status?
            4. What if the user sets this new field?
            5. Annotation makes it very clear, what it is, they shouldn't muck with it. Alternate implementation of controller could swap it out.
            6. DECIDED WE WILL USE AN ANNOTATION.
- Pre-bound Claim
    - New claim vs a broken claim: ack bit?
        - To support pre-bound user claim, we need another bit.
    - PV gets created with claim to PVC, PVC with
    - Problem: security, priority.
    - Problem: PVC could get into bad state.
        - Not a guaranteed bind, but semantics should be clear: either bind or error.
    - Let's make it unbroken and not dangerous
    - Automatic recovery:
        - If misbound claim, it is misbound claim
- Status must be recreatable. ClaimRef spec not status.
    - But don't want user to be able to set them?
- Delete inhibition
    - Physical volume or PV object could be deleted from underneath while bound

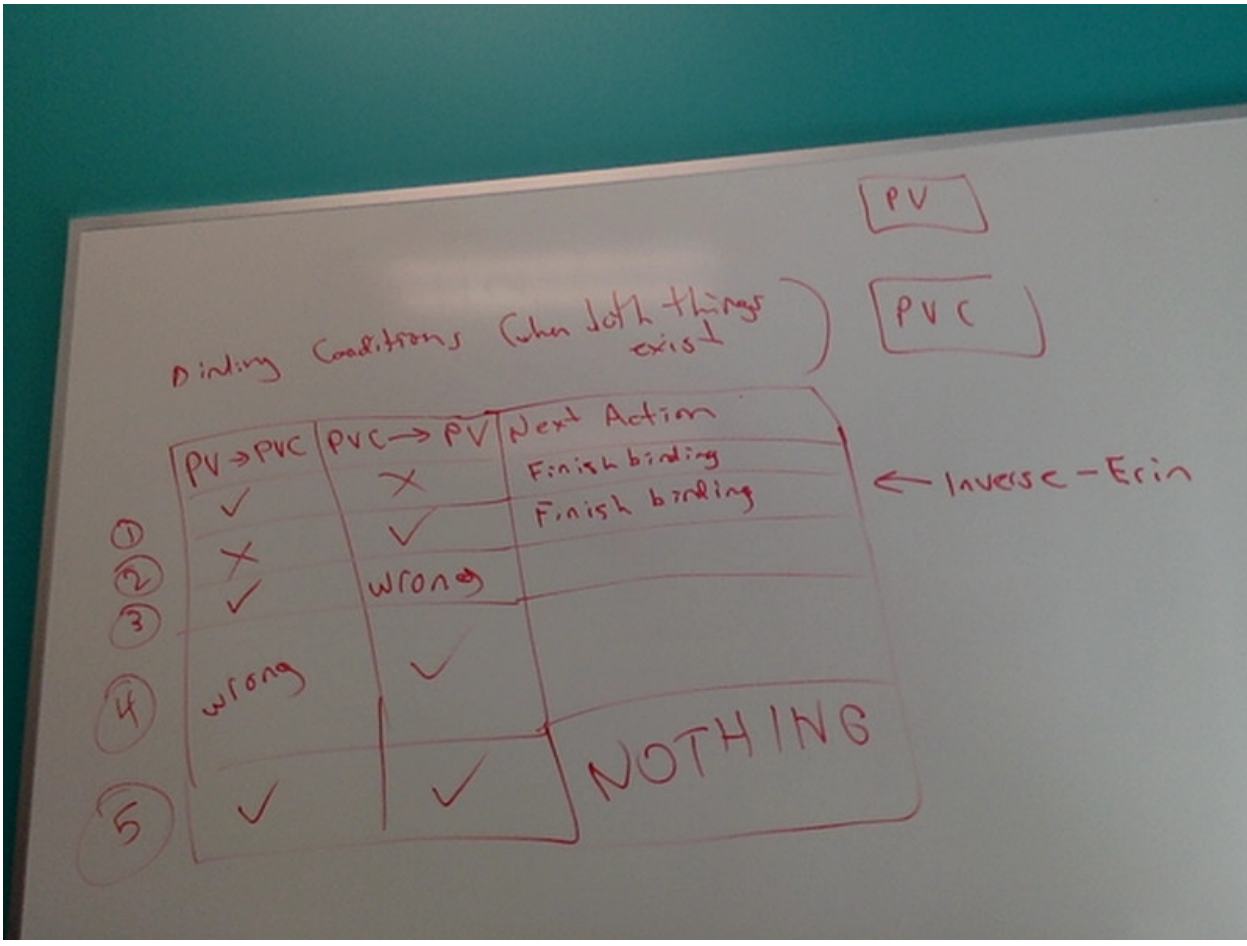
- Tim's pseudo code
    - https://docs.google.com/document/d/1x5AE_tMSDCcFOzOVSyH4e28oo60OwYYFeKdk03XaYFE/edit?ts=56f992f8
    - Should figure out how to move from Phase to conditions
    - Should be able to crash anywhere in the loop and recover without issue
    - "Extra bit" is empty -- claim is bound
    - "Extra bit" is not empty -- claim in not bound
    - Might need another extra bit to store if binding requested by user or by controller
- Do we want to check if direct volume is referencing same volume?
    - No, it'll fail out if a user does this any way
        - It may not be the case for all storage providers: e.g. amazon ebs should only be attached to one node at a time, multiple different containers can mount the same device.
        - ACTION_ITEM (saad, sami): mounter/unmounter code should make sure that a read/write attached volume can not be mounted by multiple pods
- Unwinding a transaction:
    - What if it changes in the middle?
        - As code gets written, error/error undoing
        - All you can do is bail?
    - Sami: we do not need an explicit unwind, next iteration of loop will have enough info to do it.
- Pod PVC PV created out of order, should work and not take forever to bind
- High Level Design
    - ~~Have a periodic fetch that can be triggered both by a timer and any event on PVC or PV object, but is rate limited so that rapid bursts of events are collapsed into a single fetch state of world event~~
    - Watch and periodic list on PVC and PV objects. Add/Update/Delete of on PVC events only trigger that PVC. Events on PV and list trigger all PVC objects.
- Open questions
    - Cleanup
        - What triggers cleanup?
        - What happens if PVC is modified e.g. capacity, etc.?
        - Handling dangling PVs
    - Is there mechanisms to prevent update of some fields in PV and PVC object? Admission controller can be disabled?
        - If so we can use it to:
            1. Block deletion of bound PVC object
                - There is no precedence for this, may not help us much anyway, since the code should handle this anyway.
            2. Block update of all PVC object fields after creation except ClarimRef
            3. Block update of PV object capacity and accessModes fields after creation (allow reclaimPolicy to be changed after creation)

        - Yes. pkg/api/validation/...
    - Do we want to continue support recycle "`VolumeReclaimPolicy`" policy?
    - Multi-access.
        - If read-only many, can multiple PVC be attached to the same PV.

Tuesday, March 29

Continue design for controller consolidation

- Review Tim's psudo code: https://v.etherpad.org/p/f2f-haxx
- When to do roll back?
    - On each iteration to handle controller crash
- Some operations can take much longer to complete (e.g. recycle/scrub), so they should be async.
- findPV match func should "most preferred PV should be the PV that is already pointing to the PVC,"
- Prebound PV/PVC
    - Options 1: "bound by controller" annotation on both PV and PVC
    - Options 2: don't respect it, and implement taints/tolerations in the future.
- We need to worry about a claimRef referencing a volume that doesn't exist vs one that was recently deleted (look at UID)
- Single Pointer Proposal
    - Have only PV have pointer to PVC.
    - Get rid of pointer from PVC to PV
    - Add status to PVC that can be set to "bound" by scanning all PVs
    - What happens if 2 controllers update 2 PVs to the same PVC.
        - Do we need to solve this? Will HA be a problem before transactions are avilable on etcd?
- Let's go with two annotations:
    - Something that was broken vs something that was never done
- Four error scenarios:
    - Pointer up, no pointer down
    - No pointer up, pointer down
    - Pointer up, incorrect pointer down
    - Incorrect pointer up, pointer down

- 

  - Annotation city, handle as many states as we can think of, as we think of more add them to the controller.
  - We can solve this offline, create a table of errors.
- We might not be able to ignore HA, since controllerManager can not be run HA today.
  - But we can still punt on this for now
  - NOT TRUE
- Recycler
  - What triggers clean up?
    - Find PV that doesn't have a PVC
    - 1) PVC deleted event
    - 2) A loop that handles orphaned PVs by monitoring all PVs for claimRefs to deleted PVs.
  - Annotation on PV object that says it's in the process of being recycled.
  - ~~Put a label on the scrubber pod, on controller crash restart, find all PVs with scrubPending and the previously started pod, if no pod present,~~
  - Create a pod in system namespace for scrubbing (with a label so it can be found later), launch a go routine to wait for it to complete. If controller gets restarted, it finds all PVs where PVC does not exist: if the scrubber pod exists, start a go

routine to wait for it to complete, otherwise, start the pod and the go routine to wait for it to complete
    - We need to handle scrubber Pod failed?
    - Delete the PV claim ref so that it can be used.
    - Instrument with events on PV object so it is easy to see what is happening.
    - Keep a counter of recycle attempts in memory. When we loose this counter because of crash, we will start from 1 - in the worst case we will try few times more than necessary.
    - Jan: If a PV never finished binding and PVC is deleted should we still scrub?
        - Yes
- We should add a new "`PersistentVolumeClaimPhase`" that is "Error", indicating that there is nothing the controller can do, requires user intervention.
- Phase is for information only, not for controller to act on. Controller will not take actions on the state.
- How do you differentiate between error state and PVC was deleted?
    - Don't need to?
- Open Question
    - Why does `PersistentVolumeClaimStatus` contain Capacity and `AccessModes`
    - It maybe a hack for OpenShift
    - They are in status they could be potentially stale. But we will still have to update these fields whenever we update claimRef.
- Provisoner
    - What is the trigger?
        - Existing annotation on PVC object
        - Trigger provisioning, when PVC has annotation asking for provisioning.
    - Kick off an async go routine to Provision the volume.
    - Once provisioning completes, we create a PV object with the ClaimRef pointing to the original PVC (therefore it will not be used to fulfill other claims).
    - Then we update the PVC to complete the bi-directional pointers, and the claim is fulfilled. (collapses with the Erin Case)
    - What happens if controller crashes between volume being provisioned and PV object getting created?
        - Can't do anything; no way to up-front label or deterministically name volumes which is applicable across all cloud providers
    - How to prevent a second controller from provisioning for the same PVC?
        - Annotation indicating provisioning pending should handle this
- Sami: We Should formalize our annotations.
    - Starting X, Ending Y, Timestamps
- Non-cloud provisioner
    - Should we split Provisioner recycler, and mounter/attacher/etc code or keep them as one plugin?
- Daniel Smith (lavalamp)
    - Can we add code to prevent deletion of PV object if it points to a PVC?

    - You can verify that the pointer is not present,
    - Finalizer
        - Not yet implemented
        - May be very useful in our designs.
    - Be careful about using annotations they are not versioned, might be locking yourself in for the future.
    - Claim it with a timestamp based mechanism, we make assumption that time skew is not an issue in a number of place
    - Recommend: Time at which the lock expires
        - Maybe necessary for debugging.
    - Dan: “I am ok with deleting PVC but I object to looking at PV”
- Brian Grant
    - Recommends pet set controller creates PVCs and deletes failed PVC
    - Alternate:
        - Provisional bind (mutex) on PVC (claim item from the queue)
        - Bind PV->PVC
        - Bind PVC->PV
        - Release lock
    - Similar to rollback in deployment:
        - Imperative rollback command
        - It ultimately updates your pod template to previous version of pod template
        - Provisional bind, will help three step approach
    - Controller needs master sequence number, an instance of the controller should look at annotation on API objects, and bail if version is newer then it is aware of.
    - Lease expires
    - Pod master election uses annotation with sequence number/time, but no one is annotating individual items and bailing out yet. Similar to internal Lock lease API
    - Where do we put controller manager lock lease objects?
        - Logical place would be the service that is managing it.
        - Config map
        - Third party resource
    - Logical clock!
    - During master election, each controller will update the sequence number
    - Mike Danese wrote a library to do master election.
    - Master election might be higher pri then provisional binding.
    - HA will probably be a goal
        - Active passive (active “fading”)
- HA Master Proposal
    - Config Map, with annotation master seq number.
    - All masters come up and read from config map and write to it. Who ever succeeds is the master, they succeed by writing the next number, who ever wins gets it.

    - The winner master goes out and writes to all PVs that it is working on, your seq number is X.
    - First I write an annotation saying this is a bind being done by seq # X.
    - Other masters will see that the seq is larger than theirs or, if they will try to write because the ref changed
    - Once lease expires, the controllers fight to increment the seq number, winner becomes master.
    -
- Tim: So do we want delete inhibition?
- PVC to deleted PV is the reason we need both annotations.
- Mike Denese
    - Original design principles for leader election
        - Not designed specifically for correctness.
        - Other controllers don't worry about modifying the same objects.
    - No locking over multiple resources.
    - The seq check needs to be done before every write
    - Default lease duration is 15s, renew deadline is 10s
        - If a master can not renew within 10 seconds, it will os.exit
        - Other clients for the leader election will wait 15s before the force elect.
    - Controller manager does leader election already.
    - You can get it down to where you need it by adding a little plumbing

Wednesday, March 30

Continue design for controller consolidation

- Tim: Things we can do to simplify error states:
    - Both PV to PVC links established.
    - We should never have a broken PV to PVC link (but ok PVC to PV link)
        - Can we enforce this?
        - You can create PV with claimRef set or not set.
        - But you can not update (PUT) the PV object after it is created.
        - Add a binding sub-resource to PV, so that it can only be updated via the sub-resource.
        - Add validation to "old way" that you don't change it.
        - Binding concrete operation done through a sub resource.
        - Binding is a sub-resource PV
    - We can't do this because recycling unsets the claimRef
        - But, we could create another sub-resource "unbind" that recycle can use.
- Review Tim's pseudo-code
    - Added both annotations on both PV and PVC object
        - Maybe we can get rid of annotations on PV

        - If we need it, we need to add in the 3rd transaction to update PV
    - Started syncPV pseudo-code
    - Outer loop
        - Saad: a loop that fetches a static state of world and operates on that to reconcile. Each iteration of the loop is triggered by periodic timer or event (can be rate limited).
        - Jan: not comfortable with this
    - Should “repairs” happen in PV or
    - PVC
    - Pseudocode: github.com/pmorie/pv-haxxz
    - ….much pseudocoding occurred…\
- Recycling Revisited
    - Hash deterministic pod name based on PV UID
- Provisioning Revisited
    - How do we prevent leaking volumes if we crash before PV API object is created.

Thursday, March 31

Agenda

- Discuss roadmap
- Write a list of tests we want
- Directed graph with dependencies of deliverables.
- Pmorie: unit/integration testing
- Pseudo-code commenting/finishing

Road map

- Priority list:
    - 1) Attach/detach/mount/unmount redesign
        - 8 weeks maybe
        - Extra head for testing.
    - 2) Binder/recycler/provisioner consolidation
    - 3) Testing
        - Unit test isolated algorithms
        - Integration test: run controller, with “real things” from API server
            - Comes with real api server, mock kubelet
        - E2E tests
    - 4) Dynamic Provisioning v2
        - Two chunks:
            - Classes

                - V2 Provisioning
        - 5) Containerized mount
- Containerized mount
    - Google: nice for extensibility and deployment, but not really critical
    - RH: much easier to run CI for all the volume plugins
    - Google: if someone writes E2E tests for e.g. gluster, first it's awesome, we would argue to put the packages in our images, so we can do it without contanorization
        - What are the most important plugins?
    - Current test holes:
        - We have AWS tests
        - Testing is a pretty big project, how should we handle it?
            - How much can we do with unit testing?
                - You can't unit test something that involves reaching out to amazon, or master and kubelet functionality that depend on each other
        - We will discuss 1.3 testing.
        - Exit criteria for testing:
            - Never done. Should we box it?
            - Good way to measure testing is confidence:
                - "Coverage is not enough".
                - "Do my tests give you X% confidence?"
                - "Test until fear turns to boredom"
    - RH: if we go to PM and ask for a bunch of tests, they'll say we don't care
        - Google: we can be the bad guy
        - Exit criteria: "80% confidence"
- Dynamic provisioning
    - Is this something we'll be able to iterate on fast enough after higher pri features are done.
    - We need to look at it. We have all sort of agreed about the parameterized approach.
    - Should run it by brian grant, since it will set a precedence.
    - We are establishing a pattern that networking at least will follow.
- Paul: We spent a lot of time talking about what happens when two controllers are running and misbehaved rouge controllers.
    - Proposal: provide a facility a shutdown channel, n instances running in a test, start and stop them.
        - Need either chaos monkey or to do it deterministically you need deterministic injection points in the code.
        - Hard to do in such a way that it doesn't add noise
    - Alternate proposal: add a way to optionally do something like this semi-manually as an optional thing in the integration test

Continue design for controller consolidation

- Erin/Clayton Issue:
    - https://github.com/kubernetes/kubernetes/issues/23615#issuecomment-203994582
    - Do we handle both direction?
        - We support whatever the pseudo-code does today
        - If that is not sufficient, we'll add selectors and taints/toleration later
    - Pre binding and recycling is weird, what do we want to happen here?
    - Labels/selectors/taint/toleration is a much better solution
    - If UID is required it is for infrastructure to use.
    - If we support dangling pointer (no UID), it's a user feature.
    - If a reverse selector is sufficient, then we can simplify our design.
    - Erin joined discussion via Hangout
    - Issue is the semantic is a little weird, a pointer pointing to something that doesn't yet exist. What happens when this pre bound PV has a recycle policy?
        - Erin: I think it is valid, if you want to move your configs from dev to production, pointing to a different server, but using the same files, or if you want to detach, name space for bob. But you need to create a whole new new PV
        - Tim: What happens to the original pre bound PV? Do we scrub it and make it available to anyone, or is it still reserved?
        - Erin: aren't we gonna deprecate recycle?
        - Tim: we can't really deprecate it yet.
        - Erin: Maybe it doesn't make sense to restrict from the PV perspective, so I'm undecided?
        - Tim: what if we added a persistent volume claim selector, the default is the selector is empty (any claim satisfies), if you want to reserve a volume for user bob, I will put a claim selector for user equals bob. This applies to the labels of the claims.
        - Erin: can I think about it? It seems excessive to require for user and to coordinate with cluster admin.
        - Tim: we can't guarantee whether PVC or PV comes first, so our implementation can't depend on which comes first.
        - Sami: labels and selectors from the dynamic provisioning is good enough, to increase "specificity" you can make it as generic or specific as you want.
        - Tim: future admission controller that checks every object to make sure that a "user" field is not set, and then k8s sets that field, you can make you PV match against claims that match PVC.
        - Erin: what happens if I want to go to very specific volumes.

        - Tim: I want to create 5 different PVs with specific meanings for a specific user, what is the use case, sounds contrived?
        - Erin: The admin creates 5 different PVs, the user is then able to look at that PV and look at the claim ref
        - Saad: use case cluster admin creates 5 PVs 3 should only be used by eng, 1 of the eng should be able to only for specific user.
        - Erin: claim ref and security context may be able to enable this
        - Tim: Can you explain who uses this?
        - Erin: Open shift team is using it for metrics
        - Erin: they set up a docker registry
        - Brad: why not do it other way around?
        - Erin: they could
        - Tim: if you want to reserve, that someone (the claim) should exist.
        - Tim: proposal: ….
        - Erin: the uid is needed for the binding any way.
        - Tim: claim has to exist before PV.
        - PMorie: will this work with kubectl -f? Will claim always be created before?
        - ...
        - Tim: be concrete on user story, and thoughts on recycle
        - Erin: will do
        - Tim: there is an easy answer and a hard answer, I'll let you come to an answer.
        - PMorie: reflections on Taints: as implemented for pods and nodes, have effects on the enzymatic nature of . taint is a key a value and an effect. One effect is don't allow new pods to schedule. Etc. point is applying this notion to our problem, taint effect could be don't recycle this thing.
        - Tim: The only sane answer is create PVC, if policy is recycle, we recycle, otherwise we have to keep track of if it was pre-bound. On the recycle we better clear the annotations.
        - Tim: Issues with using back selector?
        - Sami: would need to make findSuitablePV aware of them.
        - Tim: if we added a back selector, user has to be aware of it, name your claim or add a label.
    - Conclusion: Leave it the as is, and say we will support the case of writing the namespace and name in a pointer from PV to PVC, respect that, call it a feature, and file a feature request (FR) for reveres claims.
    - Need to get an answer for what to do on recycling.
- Review pseudo-code
    - We should set Phase/Status in pseudo code
- Outer loop
    - Proposal 1: Events are triggers, we have a periodic sync that will fix this.
        - Con: flood of events

        - Pro: This is the way the existing controllers work
    - Proposal 2: Grab state of world, call sync on each object, repeat
    - Hysteresis (period of time you wait after an event), event collapsing.
        - This is an optimization. We should do it the simple way. Get it working and make it fast second.
    - So we will go with proposal 1.
    - Do we want to one PVC changed, to trigger all PVC sync or just that object?
    - Demo problem of PV may not get deleted for up to 10 minutes today, does new design address this?
    - What set does findSuitablePV function look up from?
        - Internal API client cache
- What about AccessMode?
    - It is just another field that the PVC uses for selection, that's it, nothing more then that. There is no enforcment of the AccessMode.
    - It's confusing. But just needs to be better documented.
- PV ordered indexer
    - "`persistentVolumeOrderedIndex`" do we want to keep it?
    - Delete it, we can add it back in if we need it.
- Controller Initialization
    - Controller should not start syncing until "initial sync" from API server is complete.
        - we can check `Controller.HasSynced()` method
    - Should be sufficient to wait for at least one update on PV and one update on PVC
- Annotation for supplemental PV group
    - Question: are we cool with it?
    - Implementation: add an annotation to the PV when the kubelet sees a PV with an annotation on it, it modifies the supplemental group of the pod that
    - Why annotation?
        - Do we want to add a new field?
            - Yes!
        - It is a "beta field" so let's leave it as an annotation.
- Postgres permissions for Azure Plugin (diego.castro issue)
    - Azure storage plugin wants to "pass a UID,GID as mount options, i'm wondering if there's a way to know the pod UID and pass it to the mount program"
    - Postgres Chmod 27 27
    - Bet he's using docker hub image, which expects 27 27, and what he wants to do is pass this mount option.
    - Need to repro and address it. Might be as simple as adding azure volume source with uid/gid.

Test Cases:

- Provisioning
    - E2E, positive: For each volume type that supports provisioning, provision a volume
        - Expected result: 1) PV/PVC objects indicate successful binding, 2) a new physical storage asset is actually created.
    - E2E, positive: A new PVC that requests dynamic provisioning MUST result in a new PV getting created, even if an existing PV could fulfill the claim.
        - Expected result: 1) new PV object and underlying storage asset is created, 2) existing PV object remains unbound
    - E2E, negative: Trying to provision a volume type that has no provisioner
        - Expected result: 1) an event is created indicating no provisioner exists 2) PVC phase should be pending and it should remain unbound 3) if a new provisioner is added the claim will be fulfilled.
    - Integration, negative: Try to provision a volume, but volume plugin provision call fails, and all subsequent retries also fail.
        - Expected result: 1) an event is created for every failed provisioning, 2) the PVC should remain unbound, 3) no new PV object should be created, 4) the controller should retry indefinitely.
    - Integration, positive: Try to provision a volume, but volume plugin provision call fails, a subsequent retry fails, and another retry succeeds.
        - Expected result: 1) an event is created for every failed provisioning, 2) PV/PVC objects indicate successful binding, 3) a new physical storage asset is actually created
- Binding
- Recycling
    - Recycle
        - E2E, positive: For each volume type that specifies a scrubber, create a PV/PVC for a, wait for it to to be created/bound, then delete it by deleting the PVC object.
            - Expected result: 1) verify the PV object is available for further binding, 2) verify the underlying storage asset is scrubbed.
        - *test for already existing scrubber pods*
        - *test for failed scrubber pods*
        - *test that a volume is scrubbed only N times*
    - Delete
        - E2E, positive: For each volume type that supports deletion, provision a volume by creating a PVC, wait for it to to be created/bound, then delete it by deleting the PVC object.
            - Expected result: 1) verify the PV object is deleted, 2) verify the underlying storage asset is deleted.

        - E2E, negative: Trying to delete a volume type that has no deleter
            - Expected result: 1) expect an event 2) PV phase should be error and 3) it should not delete claimRef.
        - Integration, negative: Try to delete a volume, but volume plugin deletion call fails, and all subsequent retries also fail.
            - Expected result: 1) PV becomes phase Released 2) the controller retries indefinitely 3) every failed deletion produces an event
        - Integration, positive: Try to delete a volume, but volume plugin deletion call fails, a subsequent retry attempt also fails, then another retry attempt succeeds.
            - Expected result: 1) verify the PV object is deleted 2) verify the underlying storage asset is deleted 3) every failed deletion produced an event
        - Integration, positive?: Try to delete a volume, but volume plugin deletion call takes long time. Another PV sync happens.
            - Expected result: 1) only one volume plugin deletion call is done 2) the second PV sync does not trigger deletion.
        - Integration, positive: For all volume type that supports deletion: Try to delete a volume, but storage asset has already been deleted [e.g. by previous controller].
            - Expected result: 1) The volume plugin deletions call *succeeds.* 2) verify the PV object is deleted.
        -
    - Retain
        - E2E, positive: Bind a PVC to a PV with ReclaimPolicy: Retain. Delete the PVC.
            - Expected result: 1) the PV phase is Released indefinitely 2) the storage asset exists 3) data on the storage asset are not recycled.
- Attaching
- Detaching
- Mounting
- Unmounting
- Combo
    -
- Individual Plugins
- Concurrency Testing
    - Maybe an integration test that launches multiple controllers and runs them chaos monkey style.
    - Requires more thought on how to do this.

Quota

- Max number of volumes that can be provisioned.

- How/where to enforce this?