Volumes are created in container with root ownership and strict permissions #2630

New issue

Open **carlossg** opened this issue on Nov 26, 2014 · 91 comments

carlossg commented on Nov 26, 2014

The emptyDir volumeMount is owned by root:root and permissions set to 750
hostDir is the same but with 755 permissions

Containers running with a non-root USER can't access the volumes

Related discussion at https://groups.google.com/forum/#!topic/google-containers/D5NdjKFs6Cc
and Docker issue docker/docker#9360

thockin added priority/P2 dependency/docker and removed dependency/docker labels on Nov 26, 2014

thockin commented on Nov 26, 2014 kubernetes member

hostDir should get the same permissions as the existing host entry, though I am not sure we ensure a host direct exists before using hostDir

Part of the problem here is that different containers can run as
different users in the same pod - which user do we create the volume
with? what we really need is a way to tell docker to add supplemental
group IDs when launching a container, so we can assign all containers
in a pod to a common group.

I filed docker/docker#9360

carlossg commented on Nov 26, 2014

Would it be reasonable to add `user` and/or `permissions` option to `volumeMounts` or `emptyDir` to explicitly force it?

thockin commented on Nov 26, 2014 kubernetes member

I don't think that we want that in the API long-term, so I'd rather apply a
hidden heuristic like "chown to the USER of the first container that mounts
the volume" or even "ensure that all VolumeMounts for an emptyDir Volume
have the same USER, else error". Do you think such heuristics would hold?
...

carlossg commented on Nov 26, 2014

That sounds good to me

thockin was assigned by **jbeda** on Nov 30, 2014

thockin commented on Dec 1, 2014 kubernetes member

Labels

Milestone
next-candidate

Assignees
pmorie

22 participants
and others

This is a good starter project

saad-ali assigned saad-ali and unassigned thockin on Dec 12, 2014

saad-ali commented on Dec 13, 2014 kubernetes member

Background
Inside a docker container, the primary process is launched as root by default . And, currently, docker containers can not be run without root privileges (Once docker supports the user namespace, a process inside a container can run as root, and the container root user could actually be mapped to a normal, non-privileged, user outside the container). However, even today, inside a docker container a process can be run under a non-privileged user: the Docker image can create new users and then force docker to launch the entry point process as that user instead of root (as long as that user exists within the container image).

When an external volume is mounted it's permissions are set to ROOT (UID 0), therefore unless the process inside the container is launched as root, it won't have permission to access the mounted directory.

Proposed Workarounds on the Kuberentes side
1. While creating pod, if it requires an EmptyDir volume, before starting containers, retrieve the USER from each container image (introspect JSON for each container image), if any of the containers are launching their main process as non-root, fail pod creation.
2. While creating pod, if it requires an EmptyDir volume, before creating shared volume, Chown it to the USER of the first container that mounts the volume.

- Problems with this approach:
  i. With Kubernetes a pod can contain multiple containers that share a volume, but each container could potentially run their processes with different users inside, meaning even if the owner of a volume was changed, unless the owner was changed to a group that all containers were aware of (and all relevant users were part of), the problem would still exist.
     - Not that big a deal because we could handle it with docker/docker#9360
  ii. Another interesting dimension to the problem is that running CHOWN on a shared volume from outside the containers could fail if the host machine does not have the same user as inside the container (container images can create a new user, that the host is unaware of, and have the entry process run as that user, but since that user does not exist on the host, CHOWN to that user from the host will fail). One work around for this is to share the etc/passwd file between the host and the container, but that is very limiting. Another potential workaround would be for the host to some how reach inside the container during initialization (before the shared volume is mounted), read the USER that the main process will start with and use the image "/etc/passwd" file to map the USER to UID, and CHOWN the shared volume on the host to that UID (CHOWN on the host would only fails if it doesn't find a user *string* because it uses /etc/passwd to find the mapping, but it always succeeds with UIDs because it just sets the uint value directly without any lookup).

Both approaches feel to me like they are breaking a layer of abstraction by having Kubernetes reach into the container to figure out what user the main process would start as, and doing something outside the container with that information. I feel like the right approach would be for the containers themselves to CHOWN any "mounted volumes" during setup (after creating and setting user).

Thoughts?

saad-ali commented on Dec 19, 2014 kubernetes member

@thockin, after talking to some folks, I think **@carlossg**'s approach of explicitly specifying the user in the API would be the cleanest work around. I don't thing we can apply "hidden heuristics" without doing icky violation of abstractions (like reaching in to a container to figure out what username to use and then mounting the container's `/etc/passwd` file to figure out the associated UID).

Proposal to modify the API:

- Extend the API for `EmptyDir`, `GitRepo`, and `GCEPersistentDisk` volumes to optionally specify a unsigned integer UID.
  - If the UID is specified, the host will change the owner of the directory to that UID and set the permissions to `750` (User: `rwx`, Group: `r-x`, World: `---`) when the volume directory is created.
  - If the UID is not specified, the host will not change the owner, but set the permissions to `757` (User: `rwx`, Group: `r-x`, World: `rxw`), i.e. world writable, when the volume directory is created.
  - HostDir volumes would be left untouched, since those directories are not created by Kubernetes.
  - Require UID instead of username string so there are no problems if the user does exist on the host

machine (issue 2.ii above).

Thoughts?

CC: **@bgrant0607**, **@dchen1107**, **@lavalamp**

bgrant0607 added the area/usability label on Dec 19, 2014

thockin commented on Dec 22, 2014 — kubernetes member

I think adding UID to volumes is a hack and redundant. I'd rather we do the right thing and get Docker to support supplemental group IDs.

docker/docker#9360

...

LuqmanSahaf commented on Jan 13, 2015

@saad-ali I think HostDir should not be left untouched. Let's consider this: Hadoop on restart, restores the blocks from the directory it stores data in. If we use emptyDir, the container which restarted will get another directory and the previous data will be lost. And Hadoop requires the permissions and ownership of directory to be set to the user starting Hadoop (hdfs). If HostDir is not allowed to change permissions as per user, then similar use cases to this cannot be achieved. Please, comment.

thockin commented on Jan 14, 2015 — kubernetes member

Define restart? Do you mean the container crashed and came back, or do you mean the machine rebooted and a new pod was scheduled and expects to be able to reclaim the disk space used by the previous pod? Or something else?

...

LuqmanSahaf commented on Jan 14, 2015

@thockin Restart could be anything. It could be after pod failure or container failure. Or the container could be restarted, after changing some configurations (Hadoop needs to be restarted after changes in configs). Does that answer?

LuqmanSahaf commented on Jan 14, 2015

This document mentions that when a pod is unbound, the emptyDir is deleted. In use case of Hadoop, the data might be essential and might be required when another pod of Hadoop comes back (or the container restarts). So, HostDir must be used to persist data even the pod is unbound. But Hadoop requires permissions to be set for the user for the data directory. Hope this explains.

saad-ali commented on Jan 15, 2015 — kubernetes member

With docker/libcontainer#322, docker containers now allow specifying `AdditionalGroups` (supplementary group GIDs). So an updated proposal to handle shared volumes amongst different containers in a pod:

- When creating `EmptyDir`, `GitRepo`, or `GCEPersistentDisk` volumes for a new pod, Kubelet will:
  i. Create a new linux group for the pod on the host machine
     - Group is created with the next available Group ID number (GID)
  ii. Change the group of the new directory (on the host machine) to the newly created group.
  iii. Set the permissions of the new directory (on the host machine) to `770` (User: `rwx`, Group: `rwx`, World: `---`).
  iv. For each docker container, pass in the GID of the new group as `AdditionalGroups` via docker container configs.
     - Still requires docker to support passing `AdditionalGroups` through to libcontainer (docker/docker#9360).
     - May require updating `fsouza/go-dockerclient` to support `AdditionalGroups`

- When creating `HostDir` volumes for a new pod, Kubelet will:
  - Leave the volume untouched, since those directories are not created by Kubernetes.
  - **@LuqmanSahaf**: this is up for debate, but my thinking is that since Kubernetes does not create the HostDir, and since it may contain existing data, Kubernetes should not get in to the business of modifying it. We should leave it up to the creator and maintainer of the HostDir to modify it's ownership or permissions to allow containers to access it.

thockin commented on Jan 15, 2015 — kubernetes member

There's an important distinction between a container restarting and a pod being removed. When a container restarts, the data in a normal emptyDir volume is safe. when a pod is removed, it should be GONE. Leaving Hostdata and expecting it to be there at some later point in time is awkward at best.

All of this is more complicated as soon as user namespaces land.

...

erictune referenced this issue on Jan 15, 2015

Disk space monitoring #3518 — Open

bgrant0607 added the team/UX label on Feb 5, 2015

bgrant0607 modified the milestone: **v1.0** on Feb 6, 2015

bgrant0607 added team/node and removed team/UX labels on Feb 28, 2015

erictune added the area/security label on Mar 17, 2015

brendandburns modified the milestone: **v1.0**, **v1.0-post** on Apr 29, 2015

eatnumber1 commented on May 21, 2015

If I'm understanding the current proposal correctly, I think this is going to create surprising behavior for a certain class of applications.

Many older applications which bind to low (privileged) ports start first as root, then immediately drop privileges to some other user. In such a scenario, the container must be configured to start the application as root, and so the original user (root) would have access to the volume. Once the application calls `setuid(2)` / `seteuid(2)` though, it won't have access anymore. Now the only way to get access to that directory is to modify the container to `chown` the volume before starting the application itself. This is the situation I'm currently in.

Due to this, I'd like to voice another opinion in favor of extending the API to allow explicitly specifying `UID` and `GID` as I don't think the current proposal covers all possible (reasonable) use cases.

This was referenced on Jun 3, 2015

Security contexts and volumes #7925 — Closed

Finalize Kube items openshift/origin#2233 — Closed

smarterclayton commented on Jun 3, 2015

At a minimum the emptydir should use the UID/GID/Labels of the security context (if specified).

pweil- referenced this issue in **openshift/origin** on Jun 4, 2015

WIP: security context constraints #2609 — Closed

pmorie commented on Jun 4, 2015 kubernetes member

> I think adding UID to volumes is a hack and redundant. I'd rather we do the right thing and get Docker to support supplemental group IDs.

+1, but also:

> At a minimum the emptydir should use the UID/GID/Labels of the security context (if specified).

Now that we have security context API in place, I think we should make emptyDir work with the security context of the containers in a pod. One little wrinkle to iron out about this is that volumes are pod-scoped while security contexts are container scoped. I think you will have to look at which containers have the volume mounted and what their security contexts are. If there's a single container that mounts an emptyDir, it's easy -- use the security context of that container. If there are multiple, it gets dicey:

1. Do all containers mounting the emptyDir need to have the same UID?
2. Do the containers have to have the exact same SELinux options?
3. Can we have use cases where there are two different SELinux contexts that both need to use the volume? Can we synthesize the levels for a volume from the labels of different SELinux contexts of different containers that have the same user, role, and type, but different levels?

I think I will probably start prototyping this concentrating on the simple case where there's a single container. I will use the security context of the first container that mounts the volume in the pod spec at first and we can change the strategy for determining the context to use as discussion goes.

@smarterclayton @thockin @erictune @pweil-

thockin commented on Jun 4, 2015 kubernetes member

I don't really like the heuristics here. I acknowledge that I suggested a heuristic but that was half a year ago :)

Other than Docker support not being done yet, why can't we do something like:

1. All volumes get allocated a unique GID (maybe machine-unique or maybe master-allocated?)
2. All volumes are mounted g+srwx
3. All containers in a pod get all volumes in that pod's GIDs in their supplemental groups

Net result should be that all containers in the pod can access all volumes in the pod without restriction, regardless of what UID each container is using.

I don't know anything about SELinux labels, might be a problem. I assert that all volumes in a pod should be available to all containers

Next we have to define "all volumes". emptyDir is pretty obvious. What happens to hostPath mounts that did not exist and were created for this use? Seems reasonable. What about hostPath mounts that existed before this use - no way can we change those. What about things like PDs? Do we run a recursive chown/chgrp/chmod Blech.

@mrunalp for docker support status on supplemental groups.

mrunalp commented on Jun 4, 2015

Waiting for docker/libcontainer#603 to be merged.

pmorie commented on Jun 4, 2015 kubernetes member

> I assert that all volumes in a pod should be available to all containers

Spent a lot of time thinking about this last night, and I agree with you on this point.

thockin commented on Jun 4, 2015 kubernetes member

I don't know if that means we need a pod-level security context or
something, though :)

...

smarterclayton commented on Jun 4, 2015

Although - I may collocate two containers and not want them to share contents (db and web logs) but share a work for.

...

thockin commented on Jun 4, 2015 kubernetes member

I don't find the case of two containers in a pod needing different access
control to a volume to be very compelling. The alternative is to spec a
full security context for volumes and then force the complexity back onto
API users.

On Thu, Jun 4, 2015 at 9:30 AM, Clayton Coleman <notifications@github.com>
wrote:

...

smarterclayton commented on Jun 4, 2015

The argument here seems to be that you don't need intra pod security more complex than "don't mount the same volume into different contexts". I'm ok with a single security context for the pod - just pointing out that if you want to have complex, secure pods, you may want to use user isolation between the containers to secure disk contents.

...

pmorie commented on Jun 4, 2015 kubernetes member

Hrm.

smarterclayton commented on Jun 4, 2015

Although the pod security context is unlikely to work for most real containers once user namespaces land - the UID a container runs as (in user namespaces) is really tied to the container, not the pod. So either that has to be a default security context at the pod level (overridable) or it's instead the volume security context.

...

thockin commented on Jun 4, 2015 kubernetes member

We should share user namespace across the pod.

On Thu, Jun 4, 2015 at 10:32 AM, Clayton Coleman <notifications@github.com>
wrote:

...

smarterclayton commented on Jun 4, 2015

The user namespace of the two containers should probably be in the same range. But the UID of container A and B are not required to be ==, and in many cases you don't want them to be trivially ==, because you may want to "read" the volume but not write it.

...

pmorie commented on Jun 4, 2015 kubernetes member

@smarterclayton

> The user namespace of the two containers should probably be in the same range. But the UID of container A and B are not required to be ==, and in many cases you don't want them to be trivially ==, because you may want to "read" the volume but not write it.

Do you think we could infer this by whether the readOnly flag is set on the VolumeMount?

pweil- commented on Jun 4, 2015 kubernetes member

Some thoughts...

1. When mounting a volume for a container it could inherit the SC of the container if it has nothing set.
2. For a complex case we could spec out the SC for the volume to support ranges of SELinux labels as mentioned before and in this case it would not inherit the SC of the volume
3. For a predefined volume SCs the container's SC would need to be allocated in a manner consistent with the desired security (ie. volume has range `s0:c1.c10` and container has `s0:c1,c2`, GID that has only read, etc) to facilitate custom, complex approaches with fine grained access

smarterclayton commented on Jun 4, 2015

> ----- Original Message -----
>
> > Some thoughts...
> >
> > 1. When mounting a volume for a container it could inherit the SC of the container if it has nothing set.
>
> Hrm - that means a pod with one container would behave differently from a pod with two containers?

...

pweil- commented on Jun 4, 2015 kubernetes member

> Hrm - that means a pod with one container would behave differently from a pod with two containers?

And this is where some of the complexity comes in from being flexible. It is a valid use case to a single pod to have different security contexts for every container in the pod. And likewise, in OpenShift, if the continers make SC requests each container may validate against different SCCs. In that case, it seems like a predefined SC on the volume should be used with container SCs that comply. Inheriting is simply an ease of use feature.

Another idea to throw against the wall - being able to go from a pre-defined SC on a volume (inherited or not) to a RunInRange policy and validating that all container SCs that request the volume will have some sort of access.

pmorie referenced this issue on Jun 5, 2015

Add ability to run E2Es with non-root uids #9299 Open

pmorie commented on Jun 5, 2015 kubernetes member

Security aside -- can we agree that a PR that relaxes the mode is a Good Thing? I would like it if non-root uids could use volumes and junk.

@thockin @smarterclayton

thockin commented on Jun 5, 2015 kubernetes member

I don't know what "relaxes the mode" means - can you be more concrete?

...

pmorie commented on Jun 6, 2015 kubernetes member

My bad. I meant, make emptyDir 0777 instead of 0700.

smarterclayton commented on Jun 6, 2015

At a minimum for us (Paul) we could use the sc of the first container that mounts the volume until we get a broader solution. Users can then control ordering and we'll at least be unbroken.

...

pmorie commented on Jun 6, 2015 kubernetes member

@smarterclayton that's what I was going to next. We still need to fix the non-root UID case when SELinux isn't in play -- which I think we need the 0777 mode to do until we have supplemental groups in docker.

On Fri, Jun 5, 2015 at 6:13 PM, Clayton Coleman <notifications@github.com> wrote:

...

smarterclayton commented on Jun 6, 2015

I do agree that a separate security context for the pod volumes (individual or group) is probably necessary.

...

pmorie commented on Jun 6, 2015 kubernetes member

> I do agree that a separate security context for the pod volumes (individual or group) is probably necessary.

I think so too after having my brain thoroughly pretzelized while thinking through all the permutations of what you would have to handle without one. ➿➿➿

thockin commented on Jun 7, 2015 kubernetes member

For comparison, we are intentionally NOT this flexible for things like RestartPolicy - defining sane semantics for this sort of edge case is just not worth the effort. Is it really worth the effort for security context ?

...

thockin commented on Jun 7, 2015 kubernetes member

Can't we make it 0770 and set the group ID for it and every container in the pod? It's coarse but better than 0777 and closer to where we should go (IMO supplemental GIDs). As far as I see, Security Context does not yet allow a container to set GID...

...

thockin commented on Jun 7, 2015 kubernetes member

I could live with this iff it comes with a giant TODO and docs in the right places.

...

thockin commented on Jun 7, 2015 kubernetes member

I really really want to lean on simple assumptions intra-pod. Adding SC on volumes is not simpler.

...

smarterclayton commented on Jun 7, 2015

Volumes plural, not volumes singular. I'm pretty sure different uids per container is absolutely valid, so we can't guess a uid based on containers alone and be predictable. So we either need a pod default sc or a subset of sc applied to all volumes. If the uid on the directory is wrong for that sc that has other security implications. And labels *have* to match or you get nothing.

So the two options seem to be:

1. Rules based on first container using the volume (first in the pod containers list)
2. Explicit pod level setting

Pod level default sc kind of makes sense, while the first option also makes sense but is order dependent and somewhat implicit.

...

smarterclayton commented on Jun 7, 2015

I agree group is generally useful. It wouldn't work for labels though.

...

thockin-cc commented on Jun 7, 2015

Yeah, I don't know selinux at all (it has only ever given me problems that I don't know how to solve - same as here).

...

smarterclayton commented on Jun 7, 2015

Treat it like 700 - if labels are different you can't see it, if they are you can.

In this context we're probably going to stay simple and say every volume has a single label, and every container has a single label, and we want everything in the pod to have the same label in 99% of cases. Eventually we may want a container to have a different label (which means it can access anything outside of its label, period).

...

pmorie commented on Jun 7, 2015 kubernetes member

Hrm
On Sun, Jun 7, 2015 at 2:25 PM Clayton Coleman <notifications@github.com>
wrote:

...

pmorie commented on Jun 7, 2015 kubernetes member

It doesn't look like docker exposed control of the group at all yet. Am i missing something, **@thockin**? Would the kubelet have to setgid on the container process? Sounds racy to me.

...

pmorie closed this on Jun 7, 2015

pmorie reopened this on Jun 7, 2015

pmorie commented on Jun 7, 2015 kubernetes member

Multiple GH fail, my bad!

thockin commented on Jun 8, 2015 kubernetes member

Hmm, I thought docker allowed setting GID. Damn.

...

pmorie commented on Jun 8, 2015 kubernetes member

@thockin :'-/

...

This was referenced on Jun 8, 2015

Add emtpyDir volumes to ephemeral database templates openshift/origin#2922 Merged

Support emptydir volumes for containers running as non-root #9384 Merged

mrunalp commented on Jun 8, 2015

@thockin **@pmorie** The syntax for setting gid is --user "uid:gid". For e.g.

```
docker run -it --rm --user "1:777" busybox sh
```

pweil- commented on Jun 8, 2015 kubernetes member

> **@thockin** **@pmorie** The syntax for setting gid is --user "uid:gid". For e.g.

@pmorie **@smarterclayton** - we probably want to patch this in to SCs and SCCs then.

smarterclayton commented on Jun 8, 2015

Yes we do.

...

pmorie commented on Jun 8, 2015 kubernetes member

@pweil-

...

thockin-cc commented on Jun 8, 2015

Docker's syntax is atrocious, please don't copy it.

...

pmorie commented on Jun 8, 2015 kubernetes member

@thockin agree, we need a ditinct gid field in security context

On Mon, Jun 8, 2015 at 12:29 PM, thockin-cc <notifications@github.com>
wrote:

...

pweil- commented on Jun 8, 2015 kubernetes member

> Docker's syntax is atrocious, please don't copy it.

Definitely not, we'd add an `RunAsGroup int64` field.

mrunalp commented on Jun 8, 2015

Depending on requirements, you might want to store user and group as strings. They are looked up by default in the passwd and group files. If they aren't found and are numeric then they are converted to uid/gid.

eparis commented on Jun 9, 2015 kubernetes member

So I know it's a wild unreasonable idea, but from a purely correctness technical and layering PoV it seems to me like the right solution would be to just forbid/ignore the USER line in any docker file and make people set the uid of a container somewhere in the kube pod/container declaration. (although that still obviously suffers from the insanity of having to look at /etc/passwd inside the container to interpret the uid of something outside the container if we make this a string)

More seriously though, there doesn't seem to be ANY good solution until we get to user namespaces. Then we can generate a random uid/gid on the outside, chown the emptyDir directory to the uid/gid we picked and just let the container on the inside run as whatever uid/gid it wants. Nothing until we get to that point is anything but an ugly hack.

A relatedish point, just FYI, today (plain) docker is picking a random selinux context for every container. Docker recently accepted a new option `-v /source:/dest:rw,z`. The `z` portion is new. It means to do the equivalent of `chown -R` except it sets the selinux label on all files in the volume to the randomly generated label. They do not have an option to actually `chown -R` and set uid/gid but I know some people want that as well...

Personally, I think putting a security context on the volume and foisting the complexity on the user is the only 'clean' option unless we just ignore it entirely until user namespaces are a reality. One big reason I think this instead of jumping on **@thockin** supplemental gid solution is because there is no selinux analog. selinux is used similarly to uids. You get one. And either it exactly matches or it doesn't. It is not like groups. You can't have more than 1 label. Even with user namespaces, this part is still going to be a PITA...

smarterclayton commented on Jun 9, 2015

> On Jun 8, 2015, at 6:45 PM, Eric Paris ***@***.***> wrote:
>
> So I know it's a wild unreasonable idea, but from a purely correctness technical and layering PoV it seems to me like the right solution would be to just forbid/ignore the USER line in any docker file and make people set the uid of a container somewhere in the kube pod/container declaration. (although that still obviously suffers from the insanity of having to look at /etc/passwd inside the container to interpret the uid of something outside the container if we make this a string)

I agree - we plan to reject images with non numeric usernames in some modes. You can't even trust /etc/passwd anyway, so you're just compensating for lazy image authors.

> More seriously though, there doesn't seem to be ANY good solution until we get to user namespaces. Then we can generate a random uid/gid on the outside, chown the emptyDir directory to the uid/gid we picked and just let the container on the inside run as whatever uid/gid it wants. Nothing until we get to that point is anything but an ugly hack.

People need to write images to a known uid or work on any uid. But all the prep work to get to user namespaces can be done now. And user namespaces doesn't solve the ownership problem because containers can have multiple users, so you still don't know which uid to map to unless the image tells you with a numeric uid.

...

This was referenced on Jun 16, 2015

cannot create directory '/var/jenkins_home/init.groovy.d': Permission denied Closed
jenkinsci/docker#111

Controlling UID/GID of EmptyDir volume #11020 Closed

pmorie commented on Jul 10, 2015 kubernetes member

This topic is relevant to **@jmccormick2001**'s interests

pmorie commented on Jul 10, 2015 kubernetes member

@thockin **@eparis** Is there merit in peeling off a separate issue to discuss this problem for NFS and other !emptyDir volumes?

thockin commented on Jul 10, 2015 kubernetes member

yes

...

pmorie referenced this issue on Jul 15, 2015

Control permissions of non-emptyDir volumes #11319 Open

bgrant0607 removed this from the **v1.0-post** milestone on Jul 24, 2015

jasonbrooks referenced this issue in **projectatomic/atomic-site** on Sep 21, 2015

post on kube/ansible/vagrant #182 Merged

mhausenblas added a commit to mesosphere/time-series-demo that referenced this issue on Oct 28, 2015

`fixes K8S bug` ... cc8051b

mateusz-blaszkowski referenced this issue in **GoogleCloudPlatform/PerfKitBenchmarker** on Oct 29, 2015

Release version 0.23.0 #593 Merged

joshk0 commented on Nov 2, 2015

What is the Kubernetes team recommended workaround for this for the time being? The options I see are as follows:

- Run as root
- Run as root initially and have an entrypoint wrapper that chowns specified directories before dropping permissions (?? would this even work?)

Neither of them seem especially palatable.

I see that a fix was made to chmod emptyDirs to 777 but this isn't in 1.0.x (which is what Google Container Engine, my preferred deployment target, is using.) So it looks like I have to dig in and use one of the above solutions for now.

thockin commented on Nov 2, 2015 kubernetes member

kubernetes v1.1 will be out soon with better support for this case!

@pmorie

...

antoineco commented on Nov 2, 2015

@joshk0 we perform a chown from the containers' entry point (a bash script) and fallback to running our

main process using a normal user using runuser or gosu.

Not ideal but it's the only way to do it on v1.0.

pmorie commented on Nov 3, 2015 kubernetes member

@joshk0 @antoineco We've just introduced API changes that will allow you to specify a supplemental group that owns emptyDir and its derivatives and some block device volumes: #15352

thockin commented on Nov 3, 2015 kubernetes member

Paul,

Do we have a roadmap doc for the evolution of this? We've talked about FSGid being auto-allocated at admission, then maybe being coupled to PVs eventually. It would be nice to be able to see that plan all laid out.

...

pmorie commented on Nov 3, 2015 kubernetes member

@thockin I would love to write one up once I am freed up from finishing stuff for 3.1... I'll make an issue for it.

pmorie referenced this issue on Nov 3, 2015

Create storage roadmap document #16755 Open

antoineco commented on Nov 7, 2015

@pmorie looks good in terms of usability! Thanks a lot for working on this.
Will it also be possible to set a more fine-grained mode on the mountpoint? Right row RW volumes are mounted with 0777 (1.1-beta.1). I have a use case where this causes minor issues:

```
❯ kubectl exec mypod -c logrotate -- ls -ld /var/log/containers/rails/
drwxrwxrwx    2 9999     root          4096 Nov  6 14:35 /var/log/containers/rails/

❯ kubectl exec mypod -c logrotate -- logrotate /etc/logrotate.conf
error: skipping "/var/log/containers/rails/production.log" because parent directory has insecure pe
```

pmorie commented on Nov 7, 2015 kubernetes member

@antoineco Are you using an emptyDir for that volume? I think I will create an issue to change the behavior so that emptyDirs are `chmod a-rwx` if `FSGroup` is specified.

antoineco commented on Nov 7, 2015

Yes, it's an emptyDir volume.

pmorie commented on Nov 7, 2015 kubernetes member

@antoineco Okay, I am packing for a trip next week today, but I will make an issue for this and tag you into it. Do the semantics I mentioned work for you?

antoineco commented on Nov 7, 2015

(Did I hear "KubeCon"? 😄)
What you suggested would work, but you probably meant **o**-rwx, which would set the mode and owner as

follows: `0770 root : FSGroup`. Or did I get everything wrong?

pmorie commented on Nov 7, 2015 — kubernetes member

@antoineco You did :)

I think it articulated things the wrong way -- what I meant was that emptyDir should be `0770 root:fsgroup g+s`.

marcolenzo commented on Nov 12, 2015

@antoineco You suggested as a workaround for 1.0.X to perform a chown from the container's entrypoint. Do you have an example of that?

I can't seem to be able to see the mounted volume when I execute such command as the entrypoint. Is the volume mounted at a later stage?

antoineco commented on Nov 12, 2015

@marcolenzo Nothing fancy, I use a tiny bash script as my entrypoint: `ENTRYPOINT ["/run.sh"]`
This script sets the correct permissions on the shared volume(s) and then starts my service. Example:

```
#!/bin/sh

# reset permissions on log volumes
vol=/var/log/nginx
if [ -d "$vol" -a "$(stat -c '%U' "$vol" 2>/dev/null)" = "root" ]; then
    chown app "$vol"
    chmod o-rwx "$vol"
fi

# startup
echo "+-- starting nginx..."
exec nginx "$@"
```


charles-crain commented on Dec 7, 2015

I noticed that the PR for docker/docker#9360 was merged: docker/docker#10717

Does this mean the group ID solution to this problem can now be implemented? I have had several cases where I had to do what **@antoineco** has had to do, i.e. create little mini startup scripts that keep me from being able to use many 3rd party Docker images as-is.

pmorie commented on Dec 9, 2015 — kubernetes member

@charles-crain There are two ways you can work with supplemental groups via the API now:

1. There's a pod.spec.securityContext.supplementalGroups list which lets you directly specify supplemental groups
2. There's a pod.spec.securityContext.fsGroup field that is a special supplemental group -- if you specify this, block devices and system-generated volumes will be owned by this group and each container will run with this group in their list of supplemental groups

Does that help? Let me know if you need more information.

andrejvanderzee commented on Dec 17, 2015

@pmorie From which version are pod.spec.securityContext.supplementalGroups and pod.spec.securityContext.fsGroup supported? Does it also apply to non-existing hostPaths that are created when the POD starts for the first time?

slack referenced this issue in deis/charts on Dec 28, 2015

Persistance storage for Deis-Lite #50 Closed

charles-crain commented on Jan 4

@pmorie Looking through the commit logs it looks like fsGroup isn't supported until 1.2.0-alpha3 yes?

bgrant0607 added this to the next-candidate milestone on Mar 7

bgrant0607 added team/cluster priority/P1 kind/friction and removed priority/P2 labels on Mar 7

bgrant0607 referenced this issue on Mar 7

Provide possibility to specify the group a container is run as #22179 Open

ndeloof commented on Mar 18

It seems to me this issue is only caused by kubernetes relying on bind mount. When using docker volumes (i.e created using a volume driver, `docker volume` command, or implicit `VOLUME` from Dockerfile) the volume is initialized with content from the docker image, and can be `chwon`ed to match image requirements. Can't kubernetes use volume API to do the same ?

bluemir referenced this issue in bluemir/wikinote on Mar 24

Google Container Engine 배포 #40 Open

jcsirot pushed a commit to jcsirot/jenkins-docker that referenced this issue on Apr 26

`Run jenkins as root in order to workaround kubernetes/kubernetes#2630` 3c444f9

euank added a commit to coreos/kubernetes that referenced this issue on May 18

`rkt: Temporary rkt version upgrade (#2630) ...` f3a9a20

k82cn commented on May 22

@thockin , what's the final decision of this issue? I meet the same when using jenkins & glusterfs: the jenkins is using UID:1000 and glusterfs is mounted by UID:0; the jenkins can not access the FS :(.

```
[root@cdemo01 jenkins]# kc logs jenkins-3tozq
touch: cannot touch ‘/var/jenkins_home/copy_reference_file.log’: Permission denied
Can not write to /var/jenkins_home/copy_reference_file.log. Wrong volume permissions?
```

pmorie commented on May 22 kubernetes member

@andrejvanderzee @charles-crain my sincere apologies, I somehow missed the tags on this thread in the torrent of github alerts. FSGroup is supported as of 1.2.0.

> Does it also apply to non-existing hostPaths that are created when the POD starts for the first time?

Host paths do not support FSGroup or SELinux relabeling, because those could provide an escalation path for a pod to take over a host. However, that said, host paths are not created if they do not exist. Could you be thinking of empty dir? Empty dir volumes do support both FSGroup and SELinux relabel.

@k82 The glusterfs plugin does not support FSGroup at this time.

I am positive we have an issue for controlling the UID of a volume, but I will need to find it.

ndeloof commented on May 22

IIUC Empty Dir Volume are created for Pod but also destroyed when Pod is deleted, so they couldn't be

used for persistant data (typically, jenkins_home considering @k82 scenario).

k82cn commented on May 22

Currently, I'm using `0777` as workaround for demo environment; but a better solution is necessary for production :).

euank added a commit to euank/kubernetes that referenced this issue on May 24

`rkt: Temporary rkt version upgrade (#2630) ...` 08754ee

euank added a commit to euank/kubernetes that referenced this issue on May 24

`rkt: Temporary rkt version upgrade (#2630) ...` 3c007b0

euank added a commit to coreos/kubernetes that referenced this issue on May 25

`rkt: Temporary rkt version upgrade (#2630) ...` 6576863

euank added a commit to coreos/kubernetes that referenced this issue on Jun 1

`Revert "rkt: Temporary rkt version upgrade (#2630)" ...` 983b23d

thockin commented on Jun 3 — kubernetes member

So my reading this issue is largely mitigated, though some plugins, specifically those that are shared like glusterfs, do not support it well. Paul is there a writeup / walkthru of how to use FSGroup?

thockin assigned **pmorie** and unassigned **saad-ali** on Jun 3

pmorie commented on Jun 3 — kubernetes member

There is doc in the form of the proposal, and some doc for security context, but this could probably use better examples.

stapelberg commented 29 days ago

Here's an example for how to use the fsgroup directive:

For the Docker container defined in https://github.com/robustirc/robustirc/blob/master/Dockerfile (which specifies `RUN echo 'nobody:x:99:99:nobody:/:/bin/sh' >> /etc/passwd` and `USER nobody`), the following modification to my kubernetes replicationcontroller config was necessary:

```
--- a/robustirc-node-1.rc.yaml  2016-07-11 22:04:31.795710444 +0200
+++ b/robustirc-node-1.rc.yaml  2016-07-11 22:04:37.815678489 +0200
@@ -14,6 +14,10 @@
     spec:
       restartPolicy: Always
       dnsPolicy: ClusterFirst
+      securityContext:
+        # Specify fsGroup so that the persistent volume is writable for the
+        # non-privileged uid/gid 99, which is used in robustirc's Dockerfile.
+        fsGroup: 99
       containers:
       - name: robustirc
         image: robustirc/robustirc
```

zilman referenced this issue in **cncf/demo** 22 hours ago

ConfigMap backed volumes mounted as root #28 — Open