# EVE Volume manager

As of this writing we are managing volumes in different parts of the system (zedmanager, domainmgr, downloader, verifier) and the introduction of OCI images has placed some strain on this. Being able to manage the disk space on devices with limited disk space will be hard in the current design.

## Problem statement

As of this writing zedmanager is driving the overall orchestration of the VM images and container filesystems, which makes zedmanager complex and inflexible. For example, the interaction includes

- Zedmanager checking if domainmgr has a current RW image for the <app instance, image> combination, and if so just gives that to domainmgr
- Zedmanager driving the orchestration of verifier and downloader including handling refcounts and race conditions when it comes to images which were verified before a device reboot.
- Zedmanager having rather complex logic for the purge case, when it needs to retain references on the existing images until the new versions have been downloaded and verified.
- The outage time for a purge is not minimal since domainmgr can not create the new volume aka RW disk until the old domU is halted.
- The tracking of space used is hard since it is sprinkled across downloader, verifier, and domainmgr.
- Additional volumes like the CDROM image created for cloud-init is a special case in domainmgr
- Extending this to OCI images is hard

## New overall flow

Client is zedmanager (and baseosmgr for EVE images). It publishes a VolumeConfig which says it desires to have a particular volume in place for an app instance.

The client subscribes to a VolumeStatus, which is updated based on the progression of downloading and verifying blobs and constructing the volume.

When VolumeStatus is completed (for all the volumes needed by an app instance), zedmanager can proceed (to setup networking, run the app instance)

When zedmanager is handling a purge it asks the VolumeMgr to create any volumes which need to change, and when those are complete it can go ahead and reboot the running instance with the new set of volumes.

Volumemgr tracks the RW images in /persist/img and requests download and verification from the downloader and verifier. Volumemgr also creates CDROM volumes for cloud-init.

Thus the steps will be much the same as today for VM images, but with the VolumeMgr responsibility and interfaces between the microservices will be different.

Basically zedmanager will ask volumemgr to provide the needed volumes to start something (A VM or container) using a VolumeConfig. The result in the VolumeStatus will be a RW (could also be RO but that is uncommon) image which will be passed to the runtime via DomainConfig. (A different name might make sense for DomainConfig; it is basically RunConfig).

TBDToday this volume creation for VM images is spread across zedmanager, downloader, verifier and domainmgr. VolumeMgr will do the full lifecycle;

1. Look for existing Volume for this VM (if found return it, unless there are some purge instructions in the VolumeConfig)
2. Look for a verified image; if found create the per-VM RW image using copy
3. If not found, then request a download and a verification.
4. Construct the per-VM RW image.

But in the future I can envision a VM image which is constructed using layers; as long as the file system in the image can be cracked open one could define layers with different parts of the file system. But I don't know if any customer would ever request such a feature - Linux and Windows VMs are likely to be monoliths for 99% of the future use cases.

## Original Structure as of Feb 2020

domainmgr creates the per-VM RW image (does the copy from verified into /persist/img)

zedmanager runs the whole Volume orchestration by interacting with downloader, verifier, and domainmgr to see what is currently available for the VM, and then sending various *Config to get things done.

## Proposed new structure

The new design is to instead have a simpler, more linear structure.

zedmanager sends VolumeConfig to volumemgr, volumgr sends VerifyImageConfig to verifier, verifier sends DownloadConfig to downloader. When the VerifyImageStatus comes back the volumgr will create the RW image and pass it back to zedmanager. Or to minimize changes, have volumemgr send the DownloadConfig to downloader. Latter is shown in the drawing.

This makes it easier to replace all or much of Downloader + Verifier with containerd.

Also, we can further simplify the interaction if VolumeMgr and/or containerd takes responsibility for the already verified objects which are found on boot. The need for a PersistImage* and its refcounting and handshake might go away.

We will split out the resolution of tags (such as nginx:latest) to hashes from the volume and image management by having a new ResolveConfig and ResolveStatus exchange between zedmanager and the downloader. Thus volumemgr will handle objects which has a known hash hence can leverage Content Addressible Storage. Note that a volume isn't identified by the sha itself; there can be multiple VMs or containers using the same blob/image with a particular sha.

## Implementation Approach

1. Define the VolumeConfig and VolumeStatus. Initial cut is the StorageConfig and StorageStatus which are used internally in zedmanager. Might need to be extended to cover the cloud-init volumes.
2. Move the code at and below doInstallProcessStorageEntriesWithVerifiedImage() [zedmanager/updatestatus.go] including all of handleverifier and handledownloader to volumemgr.
3. Move the copy to /persist/img from domainmgr to volumemgr. Remove ImageStatus definition and code.
4. Make zedagent/handlemetrics report images based on VolumeStatus instead of ImageStatus
5. Apply above zedmanager changes to baseosmgr as well
6. Move the creation of the CDROM cloud-init volume from domainmgr to volumemgr [Probably defer this until the object encryption code has been pulled into master since it touches this cloud-init code]

### Subsequent work

1. Explore making VolumeMgr own the /persist/download/ directories when it comes to garbage collecting. Means that PersistImageStatus and Config should no longer be needed for GC and refcount handling.
2. TBD: Do we need some BlobStatus reporting from VolumeMgr to expose the size of the blobs we are storing on the device (separately from the volumes which are created from the blobs)