
OmniTI

Sandboxing OpenZFS on Linux

Albert Lee kernel engineer

trisk@omniti.com

In the beginning

Zones (née Kevlar)

- Introduced in Solaris 10

Motivations (Price & Tucker, 2004):

- Consolidation of existing server workloads which may require dedicated named resources
- Isolation from privilege escalation and resource exhaustion in potentially compromised environments
- FreeBSD jails and Linux-VServer precedence with low resource overhead, how can we make it simple for administrators and software developers?

Some of this will sound familiar

- Zones encapsulate named resources including filesystem, network interfaces, process tree, IPC
- All process privileges are interpreted in zone security context
- Fine-grained resource accounting and limits
- Initial *global zone* has traditional semantics
- Userspace tools configure and deploy *non-global zones*
- Persistent configuration and identity
- Stateful, with well-defined transitions
- APIs allow delegation of resource administration to non-global zones and integration with software deployment

Enter ZFS

- Zones could inherit filesystems from the global zone via loopback mount
- With introduction of ZFS, datasets could be *delegated* to zones
- Selected datasets and their children visible to processes in zone
- Delegation allows full use of *zfs(1M)* administrative operations

Limitations of delegation

- Changes to dataset list require zone restart
- Access to a dataset cannot be shared between zones

This is great, but what about Linux?

When ZFS was ported to Linux by Behlendorf (hi!) and others:

- Most code for zone integration was preserved
- But stubbed to always believe it was running in the global zone
- This was a good idea at the time

Linux has native containers now, right?

Well, sort of

- Composition of primitives for resource isolation and management

Linux namespaces

- Objects representing named resources, can be nested
- Processes can be created in a specified namespace, or moved after creation
- Namespaces for each named resource type: mount, net, ipc, uts…
- User namespaces (3.12+) translate UID/GID ranges inside the namespace

The other bits

Linux control groups (*cgroups*)

- Resource control and accounting of process trees
- Provides interface to create hierarchy of resource constraints and monitor usage
- Interestingly, trees for each resource don't have to match actual process relationships or trees for other resources

The problem

Linux containers are thus an elaborate illusion painstakingly created from:

- A set of namespaces to isolate the resources expected by applications
- A filesystem image
- A set of cgroups to provide resource controls

Of course, these days there's tooling (LXC, Docker, rkt) to maintain this illusion and make this all palatable, but

- There's no first-class object that represents a container
- How can ZFS tell we are trying to manage it from inside a container?

Our solution

Datasets are a unique type of named resource

- So, begrudgingly, we introduce yet another namespace
- Let's call it *dataset* for now, because maybe it could be useful for more than ZFS

That zones integration sure is convenient

- Most of the entry points in ZFS where dataset visibility and process privileges are checked for zones make perfect sense for this
- Implement the zones API in the SPL (Solaris Porting Layer)
- Pretend we have a zone object and associate it with this namespace

datasetns, first pass

- Initial namespace is mapped to global zone, for simplicity
- All other namespaces are treated as non-global zones
- Each zone object keeps a list of delegated datasets
- Lists can be managed on the fly, unlike zones

To start a process in a new namespace:

*clone(..., **CLONE_NEWDATASET**, ...)*

- Initial list will always be empty
- Must populate list of datasets from initial namespace with writes to */proc/$pid/ns/dataset*
- Datasets or their children cannot be delegated to more than one namespace

In action

```
root@host$ zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
mistify                     127M  9.66G  10.1M  /mistify
mistify/data                 19K  9.66G    19K  /mistify/data
mistify/demo                 19K  9.66G    19K  /mistify/demo
mistify/images             19.5K  9.66G  19.5K  /mistify/images
mistify/private             117M  3.89G   117M  /mistify/private
root@host$ ls -l /proc/self/ns/dataset
lrwxrwxrwx 1 root root 0 Sep 15 07:40 dataset -> dataset:[4026531835]

root@container$ ls -l /proc/self/ns/dataset
lrwxrwxrwx 1 root root 0 Sep 15 07:40 dataset -> dataset:[4026532151]
root@container$ zfs list
no datasets available
root@container$ echo $$
394

root@host$ echo 'mistify/demo' > /proc/394/ns/dataset

root@container$ zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
mistify                   127M  9.66G  10.1M  /mistify
mistify/demo               19K  9.66G    19K  /mistify/demo
root@container$ zfs create mistify/demo/blahblah
root@container$ zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
mistify                   127M  9.66G  10.1M  /mistify
mistify/demo               38K  9.66G    19K  /mistify/demo
mistify/demo/blahblah      19K  9.66G    19K  /mistify/demo/blahblah
```


OmniTI

datasetns, second pass

- Initial list is empty but allows full access to all datasets
- List is now inherited by new namespaces so datasets can be
- Processes manage the datasets allowed for their own namespace with writes to */proc/self/ns/dataset*
- Once list is populated, only removal of entries is allowed to prevent escalation

In action

```
root@host$ zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
mistify                     127M  9.66G  10.1M  /mistify
mistify/data                 19K  9.66G    19K  /mistify/data
mistify/demo                 19K  9.66G    19K  /mistify/demo
mistify/images             19.5K  9.66G  19.5K  /mistify/images
mistify/private             117M  3.89G   117M  /mistify/private
root@host$ ls -l /proc/self/ns/dataset
lrwxrwxrwx 1 root root 0 Oct 09 11:49 dataset -> dataset:[4026531835]

root@container$ ls -l /proc/self/ns/dataset
lrwxrwxrwx 1 root root 0 Oct 09 11:49 dataset -> dataset:[4026532151]
root@container$ zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
mistify                     127M  9.66G  10.1M  /mistify
mistify/data                 19K  9.66G    19K  /mistify/data
mistify/demo                 19K  9.66G    19K  /mistify/demo
mistify/images             19.5K  9.66G  19.5K  /mistify/images
mistify/private             117M  3.89G   117M  /mistify/private
root@container$ echo 'mistify/demo' > /proc/self/ns/dataset
root@container$ zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
mistify                     127M  9.66G  10.1M  /mistify
mistify/demo                 19K  9.66G    19K  /mistify/demo
root@container$ echo '-mistify/demo' > /proc/self/ns/dataset
root@container$ zfs list
no datasets available
```

Remaining problems

Security is still broken

- */dev/zfs* ioctls do not check privileges/capabilities and non-root users have full access
- ZFS feature of delegation of datasets to users/groups is not supported
- User namespace support to allow containers with remapped UID/GID ranges (including UID 0) also needs to be implemented

Future directions

In additional to changes to ZFS and the SPL, the current implementation requires patching the Linux kernel to add the namespace and some supporting code for list management, due to the way namespaces are implemented.

- Ways to reduce changes to the Linux kernel?

Now that dataset management is available inside a container, we should look into making it better available to applications

- libzfs_core (we're working on Go bindings)

We're also looking at improving the overall resiliency of ZFS on Linux to hardware failures - stay tuned.

Thank you!

http://mistify.io

trisk@omniti.com