Zones/SNAP Design
8/25/2008

I. Overview

This specification describes how ZFS datasets will be used with zones for supporting software management operations with ipkg branded zones on OpenSolaris. Software management includes tasks such as a SNAP upgrade or installing/removing pkgs.

Issues are summarized in Part III.

One goal is that sw management behavior within the zone should be as similar as possible to the behavior in the global zone. That is, when the zone admin does sw management, the tools running within the zone should be able to take a snapshot of the zone root, clone it, and the zone admin should be able to roll-back if the sw installation was problematic. This implies that the zone root itself must be a delegated dataset. Up to now, zones does not support this, so we must extend zones to provide this capability.

(Note that these software management features within a zone are not a requirement for the 2008.11 release, however, this proposal lays the groundwork to enable that capability moving forward.)

There are some issues with delegating the zone root dataset to the zone. The way zones work, it is fundamental that the zone root be mounted in the global zone so that the system can set up the chrooted environment when the zone boots or when a process enters the zone. However, the ZFS mountpoint property cannot be interpreted from the global zone when a dataset is delegated, since this could lead to security problems.

(Note that the zone root is {zonepath}/root as seen in the global zone}

To address this, the zones code must be enhanced to explicitly manage the zone root mounts. This naturally falls out of the basic design described here.

There were two alternative proposal considered; a two-level zone layout and a single-level zone layout. After much discussion, the single-level approach was chosen. The two-level approach is not described further here.

II. Description

To support the management of boot environments (BEs) for zones within both the global zone context as well as the non-global zone context, a nested, two-level, BE dataset naming scheme is used so that there is a top-level global zone BE namespace, as well as a zone-level BE namespace for the zone's own use. Properties on the zone's datasets are used to manage which dataset is active and which datasets are associated with a specific global zone BE.

There are two properties on a zone's datasets that are used to manage the datasets; "org.opensolaris.libbe:parentbe" and "org.opensolaris.libbe:active".

The "org.opensolaris.libbe:parentbe" property is set to the UUID of the global zone BE that is active when the non-global zone BE is created. The "org.opensolaris.libbe:active" property is set to "on" or "off" to indicate that the dataset is the one that should be mounted.

We leave the "org.opensolaris.libbe:active" property set to "on" to correspond with older, inactive global zone BEs, and use the combination of this property, along with the matching "org.opensolaris.libbe:parentbe" to

determine which dataset to mount on the zone root. Thus, rolling back to an earlier global zone BE would cause the last active zone BE to be mounted for that global zone BE.

When a global zone BE is deleted, all corresponding zone-specific BEs with the matching "org.opensolaris.libbe:parentbe" property should also be deleted. When a non-global zone is deleted, all of zone-specific BEs with the matching "org.opensolaris.libbe:parentbe" property for the current global zone BE UUID should be deleted, however, any zone-specific BEs for other global zone BEs must be preserved.

The following example illustrates the dataset layout and management for zone z1 while the global zone is running BE1. The zones code must be enhanced to automatically create these datasets when the zone is installed.

(For clarity, the global zone "GBE" string is used instead of the global zone BE UUID in the following examples. Likewise, the "ZBE" string is used to for the non-global zone BE name.)

The zone has zonepath: /export/zones/z1

The relevant dataset that exists before the zone is installed:

dataset	mountpoint prop.	zoned prop.
rpool/export/zones	/export/zones	off

The datasets that must be automatically created during zone installation:

dataset	mountpoint prop.	zoned prop.
rpool/export/zones/z1/ROOT	legacy	on
rpool/export/zones/z1/ROOT/ZBE1	legacy	on

The equivalent commands to create these datasets:

```
# zfs create -o mountpoint=legacy -o zoned=on rpool/export/zones/z1/ROOT
# zfs create -o org.opensolaris.libbe:active=on \
  -o org.opensolaris.libbe:parentbe=GBE1 rpool/export/zones/z1/ROOT/ZBE1
```

During the zone installation and after the zone is installed, the zone's ZBE1 dataset is explicitly mounted by the global zone onto the zone root (note, the dataset is a ZFS legacy mount so zones infrastructure itself must manage the mounting. It uses the dataset properties to determine which dataset to mount, as described below.): e.g.

```
# mount -f zfs rpool/export/zones/z1/ROOT/ZBE1 /export/zones/z1/root
```

The rpool dataset (and by default, its child datasets) will be implicitly delegated to the zone. That is, the zonecfg for the zone does not need to explicitly mention this as a delegated dataset. The zones code must be enhanced to delegate this automatically:

```
rpool/export/zones/z1/ROOT
```

Once the zone is booted, running a sw management operation within the zone does the equivalent of the following sequence of commands:

1) Create the snapshot and clone

```
# zfs snapshot rpool/export/zones/z1/ROOT/ZBE1@mysnap
# zfs clone rpool/export/zones/z1/ROOT/ZBE1@mysnap \
  rpool/export/zones/z1/ROOT/ZBE2
```

2) Mount the clone and install sw into ZBE2

```
# mount -f zfs rpool/export/zones/z1/ROOT/ZBE2 /a
```

3) Install sw

4) Finish
    # unmount /a

Within the zone, the admin then makes the new BE active by the equivalent of the following sequence of commands:

    # zfs set org.opensolaris.libbe:active=off rpool/export/zones/z1/ROOT/ZBE1
    # zfs set org.opensolaris.libbe:active=on rpool/export/zones/z1/ROOT/ZBE2

Note that these commands will not need to be explicitly performed by the zone admin. Instead, a utility such as beadm does this work (see issue #2).

When the zone boots, the zones infrastructure code in the global zone will look for the zone's dataset that has the "org.opensolaris.libbe:active" property set to "on" and explicitly mount it on the zone root, as with the following commands to mount the new BE based on the sw management task just performed within the zone:

umount /export/zones/z1/root
mount -f zfs rpool/export/zones/z1/ROOT/ZBE2 /export/zones/z1/root

Note that the global zone is still running GBE1 but the non-global zone is now using its own ZBE2.

If there is more than one dataset with a matching "org.opensolaris.libbe:parentbe" property and the "org.opensolaris.libbe:active" property set to "on", the zone won't boot. Likewise, if none of the datasets have this property set.

When global zone sw management takes place, the following will happen.

Only the active zone BE will be cloned. This is the equivalent of the following commands:

    # zfs snapshot -r rpool/export/zones/z1/ZBE2@mysnap
    # zfs clone rpool/export/zones/z1/ROOT/ZBE2@mysnap rpool/export/zones/z1/ROOT/ZBE3

(Note that this is using the zone's ZBE2 dataset created in the previous example to create a zone ZBE3 dataset, even though the global zone is going from GBE1 to GBE2.)

When global zone BE is activated and the system reboots, the zone root must be explicitly mounted by the zones code:

    # mount -f zfs rpool/export/zones/z1/ROOT/ZBE3 /export/zones/z1/root

Note that the global zone and non-global zone BE names move along independently as sw management operations are performed in the global and non-global zone and the different BEs are activated, again by the global and non-global zone.

One concern with this design is that the zone has access to its datasets that correspond to a global zone BE which is not active. The zone admin could delete the zone's inactive BE datasets which are associated with a non-active global zone BE, causing the zone to be unusable if the global zone boots back to an earlier global BE.

One solution is for the global zone to turn off the "zoned" property on the datasets that correspond to a non-active global zone BE. However, there

seems to be a bug in ZFS, since these datasets can still be mounted within the zone. This is being looked at by the ZFS team. If necessary, we can work around this by using a combination of a mountpoint along with turning off the "canmount" property, although a ZFS fix is the preferred solution.

Another concern is that the zone must be able to promote one of its datasets that is associated with a non-active global zone BE. This can occur if the global zone boots back to one of its earlier BEs. This would then cause an earlier non-global zone BE to become the active BE for that zone. If the zone then wants to destroy one of its inactive zone BEs it needs to be able to promote any children of that dataset. We must make sure that any restrictions we use with the ZFS "zoned" attribute doesn't prevent this. This may require an enhancement in ZFS itself.

III. Issues and Tasks:
---

1) We will not allow zones in the ROOT dataset. When installing a zone, this will be validated and result in an error if the zonepath is under the ROOT dataset.

   If necessary, we can always relax this restriction later. This proposal does not discuss a namespace or layout for zones in the ROOT dataset.

2) We need some sort of tool to manage BE activation within the zone. This would automatically set the value for the ZFS "org.opensolaris.libbe:active" property for the different datasets. This most likely would be an extension to beadm to make it work inside of a zone. (Not required for 2008.11)

3) The sw management commands must be extended to be zone aware so that they can create the correct snapshot and clone when they are running inside the zone. (Not required for 2008.11)

4) Zones must always live in ZFS datasets. When installing a zone, this will be validated and result in an error if the zonepath is not under a dataset.

   If necessary, we can always relax this restriction later. This proposal does not discuss how to handle zones that do not have their own BE dataset.

5) We must either fix the zone dev to be part of the zone root and not a separate mount or we need to manage it as its own dataset that is cloned and mounted as part of the global zone BE management. Fixing this implies some devfs changes. This is assumed in the example above.

6) Global zone BEs need a UUID.

7) The zones code needs some way to determine the active global zone BE UUID for use in creating the zone datasets and for managing the explicit mounts.