Static Root FS UUID as GPT Partition UUID

We currently require an ad-hoc extension to a grub command, 'probe', to be able to boot an IMGA or IMGB partition (rootfs). This forces us to be dependent on GRUB as the first bootloader in any rootfs.

However, we might want to move towards simpler bootloaders (e.g., xen.efi or even acrn.efi) and not want to depend on ad-hoc extensions to GRUB for booting.

This proposal aims at simplifying the rootfs boot sequence, removing our grub extension, and allowing a more generic way to configure bootloaders.

Background and Motivation

An EVE system has two partitions, IMGA and IMGB, one of which is the *active* root partition (rootfs) that is run on boot.

There are two steps in booting an EVE system. The first step selects the *active* partition (IMGA or IMGB) and executes an EFI executable at a fixed path in that partition. The second step is partition-dependent and its goal is to load the hypervisor and instructs the linux kernel to set the current partition (IMGA or IMGB) as its root filesystem. This proposal focuses on the second step.

Finding the root device for the linux kernel is not a simple problem in EVE. Any rootfs built by the EVE build system can be run either in IMGA or IMGB. The partition where the rootfs is installed is due to the upgrade history of the system. Furthermore, every system can have a different disk device name, so setting a fixed device path as a linux kernel root (e.g., */dev/sda2*) is unfeasible.

Most linux distributions use filesystem labels or UUIDs to have a reliable way of selecting partition at boot (via *root=UUID=<UUID>* or *root=LABEL=<LABEL*
), but this has two problems:

1. requires the addition of initrd, which we currently don't have;
2. requires to use a filesystem that has support for either labels or UUIDs.

SquashFS, our preferred filesystem for the rootfs, doesn't support neither labels nor UUIDs, making this option also unfeasible.

Our current solution relies on using *root=PARTUUID=<UUID>* as a kernel parameter. *PARTUUID* is the GPT partition UUID, which is available early in the linux kernel, doesn't require initrd and requires no filesystem support, as it resides in the GPT partition table.

The GPT partition UUID is currently randomly generated during *mkimage-raw-efi*, that creates the partition table for the device during install. When GRUB in the rootfs is run (the second step as described above), we use an ad-hoc extension to the grub command *probe* to retrieve the current GPT partition UUID and pass it to the linux kernel's *root* parameter.

This scheme forces us to be dependent on a modified GRUB to load the hypervisor and the linux kernel.

This proposal aims at solving the following problems:

- Removing the need to find at runtime the UUID of the current partition, thus allowing us to remove GRUB in the rootfs.
- Creating a generic way to configure bootloader configuration files

Proposal

We are currently generating GPT partitions UUIDs during install, and they stay constant during the lifetime of the device. It is rootfs GRUB's responsibility, at every boot, to find the UUID of its partition. Note that the GPT partition table is otherwise *not* constant during the life time of the device: at every upgrade of rootfs, we explicitly change GPT partition flags through *zboot.*

Also, on rootfs generation, we have a crude way of configuring the bootloader with install-specific information, currently only for *cmdline*.

Instead of generating a random UUID for the IMGA and IMGB GPT partition, we propose to set the GPT partition UUID to a rootfs specific UUID. This means that the UUID will be generated during rootfs creation. We can then extend the bootloader configurator during rootfs creation to set the *PARTUUID* parameter of the linux kernel to the UUID just generated.

As an extra step, we save the UUID generated in the rootfs, at a known path.

Finally we need to ensure, every time we install a rootfs image into a partition, to change the GPT partition UUID to the UUID specified in the rootfs. There are two scenarios when this needs to happen: one is during initial image installation, the other during EVE upgrade.

Changes to the installation process

The changes required are minimal: simply, on *mkimage-raw-efi*, we extract the rootfs UUID from the image (it is saved in the filesystem at a known path), and set the GPT partition UUID of IMGA to that UUID.

Changes to the upgrade process

This is also simple, but has some side effects.

Currently in *zboot.go* we have a function, WriteToPartition(), that calls directly the unix utility *dd*, to copy the rootfs image to the partition. This proposal require that after this there must be another step, that extracts the rootfs UUID from the partition and changes the GPT partition UUID accordingly.

What this unfortunately means is that it will not be possible to upgrade a new image using this scheme from a system running the old pkg/pillar software.

This can be solved by creating an interim period where we first only change pkg/pillar to support the new format, or through versioning that requries some 'frontier' release required to upgrade to the new version. This problem is more generic in nature and scope than this proposal and will not be described further.

Implementation details

A git branch implementing this is here: https://github.com/glguida/eve/tree/gianluca_static

The implementation of this proposal is currently fully working and open to comments. A few details worth mentioning are:

1. *zboot* is extended with two commands, *partuuid* and *set_partuuid* to get and set the GPT partition UUID of a particular label, e.g. *zboot partuuid IMGA*
2. A new base utility, *rootfs_util*, is created. It handles rootfs images. It has two subcommands:
    a. *uuid*: returns the current rootfs UUID of the image, e.g. *rootfs_util uuid <rootfs.img>*
    b. *install*: perform a full install of a rootfs image to a partition label, e.g. *rootfs_util install <rootfs.img> IMGB*
3. *zboot.go* in *pkg/pillar* is modified to use *rootfs_util* to perform installation rather than calling the UNIX command *dd* directly. This should help with portability and future expansions.