EFI booting and troubleshooting

Understanding EFI

Unlike the legacy boot mechanism, EFI does not rely on MBR and partitions marked as bootable. It stores boot entries for each OS on an ESP (EFI System Partition, see "Booting") which the EFI firmware of the motherboard can access and load directly. EFI boot entries are also stored in the EFI firmware itself in case of non-removable media (HDDs, SSDs). This is important to remember (see "Troubleshooting").

Legacy boot is being phased out and starting from 2018, new motherboards will no longer support it. EFI booting allows the OS better control over the hardware and takes less time, so we should strive to use it whenever possible.

Booting

The conditions that enable EFI booting are:

- A storage device formatted in GPT (which all storage devices should be formatted in nowadays anyhow)
- An ESP - essentially, just a FAT32 partition where boot entries are stored (recommended size: 100-200 MB, mount point: /boot/efi, Linux installer knows it)
- A 64-bit OS

To install Linux in EFI mode, do not use any "boot media creation" tools. Simply download a 64-bit ISO and unzip its contents to a FAT32 formatted USB. Then make sure that UEFI booting is enabled in the BIOS settings and boot from your USB. Install as usual but ensure an ESP is created.

ESP RAID 1

ESP **can** be put in RAID 1 for redundancy but while creating the RAID array, you have to use *–metadata=1.0* flag since newer metadata versions store superblock at the beginning of the partition, preventing EFI firmware from correctly detecting such an ESP. You can't use this flag during installation since RAID configuration options given are limited, so it's recommended to pre-format the disk prior to installing.

Alternatively, you can create a normal ESP partition during installation and later convert it into RAID 1. Simply backup the contents of /boot/efi, delete the corresponding partition, create a RAID 1 array (with *--metadata=1.0*) in its place, move the EFI data back to it and alter /etc/fstab with the new UUID for /boot/efi. After the second drive is added to the array and it syncs, create the second EFI boot entry for it manually (see "Troubleshooting").

Not using RAID 1 for ESP puts the system at risk of being unbootable if the storage device containing the ESP becomes inaccessible (e.g. hardware failure).

Troubleshooting

Sometimes, EFI boot entries will disappear from the firmware (mostly on older motherboards with poor EFI implementation, or due to bugs in how Linux handles EFI boot entries), rendering the system unbootable. This is easy to fix. Boot a live OS **in EFI mode** (you cannot fix EFI booting if you are working from a legacy-booted live OS) and use the tool *efibootmgr* to create boot entries. The important things to know about efibootmgr are:

- How to view existing boot entries: *efibootmgr -v*
- How to delete an existing boot entry: *efibootmgr -b XXXX -B*, where XXXX is the 4-digit number of the boot entry you get from *efibootmgr -v*
- How to create a new boot entry: *efibootmgr --create --label "LABEL" --disk /dev/sdX --part Y --loader "\EFI\ubuntu\grubx64.efi"*, where LABEL is the name you want for this boot entry (e.g. "Ubuntu 16.04"), /dev/sdX is the drive containing the ESP (e.g. /dev/sda), Y is the number of the ESP partition on that drive (usually 1), and the loader path is easy to check by looking at the contents of your /boot/efi folder. Note that **reverse slashes** are used and the path is relative to the root of the ESP, **not** the Linux root. (EFI firmware does not know and does note care where your ESP is mounted in Linux. It only knows what's on the ESP.)

Examples

Here are some common cases of efibootmgr usage.

Output: viewing existing boot entries

```
root@server:/home/bl# efibootmgr -v
BootCurrent: 0000
Timeout: 1 seconds
BootOrder: 0000,0001,0003
Boot0000* Ubuntu 16.04    HD(1,GPT,ce194703-2d2d-4515-b0e9-55808c2311f8,0x800,0x2f000)/File(\EFI\ubuntu\grubx64.
efi)
Boot0001* Hard Drive    BBS(HD,,0x0)
Boot0003* UEFI: SanDisk    PciRoot(0x0)/Pci(0x1a,0x0)/USB(1,0)/USB(5,0)/HD(1,MBR,0x53,0x800,0x1cf7800)
```

The first boot entry is the actual one this machine boots from. It indicates the label ("Ubuntu 16.04"), the UUID of the ESP (in case of RAID 1, this will be the PARTUUID as retrieved from blkid, **not** the UUID of the underlying RAID partition) and the path to the GRUB EFI executable on the ESP. Not seeing a boot entry like this is a clear problem: your EFI firmware doesn't know what to boot. If the entry **is** there but the PC still won't boot, it's advisable to check whether the indicated UUID is actually present in *blkid* output and if so, which partition it refers to. Chances are it will not be there. In that case, delete boot entry and create one pointing to the correct ESP.

The second boot entry is for compatibility with legacy boot from MBR and will most likely not be there if you disable legacy boot in the BIOS. (Not a bad idea to avoid confusion.) The third boot entry is added automatically for a UEFI-bootable USB drive currently inserted into the PC.

Case: adding two boot entries for ESP RAID 1

Suppose a server has two SSDs, /dev/sda and /dev/sdb. You have successfully created a RAID 1 array with metadata 1.0 and placed your EFI data on it and made sure that it's mounted on /boot/efi in /etc/fstab. What else needs to be done? You need to create two boot entries in the EFI firmware, each one pointing to its own disk. This is needed since if one of the disks is gone, the boot entry pointing to its ESP will be invalid.

```
efibootmgr --create --label "Ubuntu 16.04" --disk /dev/sda --part 1 --loader "\EFI\ubuntu\grubx64.efi"
efibootmgr --create --label "Ubuntu 16.04" --disk /dev/sdb --part 1 --loader "\EFI\ubuntu\grubx64.efi"
```

As you can see, the only difference between the two entries is that they point to different drives.