ALL REFERENCES USED IN THIS DOCUMENT

DOCKER

Click the URL to CREATE A DOCKER FILE

Dockerfile Commands

To check the commands used in a Docker File click here

```
$ sudo docker pull <image-name>     # FOR PULLING THE IMAGES FROM THE DOCKER
REPOSITORY
$ docker images                     # DISPLAYS THE LIST OF IMAGES
$ sudo docker container ls          # SHOWS THE RUNNING CONTAINERS
$ sudo docker container ls -a       # SHOWS ALL THE PULLED IMAGES
$ sudo docker start <container_id>  # TO START THE CONTAINER
$ sudo docker stop <container_id>   # TO STOP THE CONTAINER
$ sudo docker run -it <image_id>    # To run the container with interactive shell
```

systemctl command

```
Use systemctl for starting a service (similar to service command used in sudo
service start <service_name>)
```

Shell

```
A shell is a interface to communicate with the shell
```

RUNNING A APACHE SERVER

Fedora Image file in docker

To run the apache server on the host machine follow this URL

```
$ sudo apt install apache2      # To install apache server on the host machine
$ docker pull httpd             # To install container of apache2 from the docker
(IMAGE FILE IS AVAILABLE IN ABOVE LINK)
```

To run the apache server in Docker follow the below steps or follow the URL

```
Step 1: Create a folder for compose file
Step 2: Enter into the folder and create a docker-compose.yml file
Step 3: Create a folder for the html files to map these files into the contaners
apache working directory
Step 4: Execute the cmd "$ docker compose up" to start the created compose file
```

Sample yaml file

```
docker-compose.yml  :- FILE NAME
------------------
version: '3'

services:
  apache:
    image: httpd:latest
    container_name: my-apache-container
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/local/apache2/htdocs
    restart: unless-stopped
```

COMPOSE SPEC REFERENCE

==========

ABOUT DISK

ALL ABOUT DISK

ABBREVATIONS

- *SATA* :- Serial AT Attachement (Advance Technology)
    - AT is a protocol
- *PATA* :- Parallel AT Attachement
- *SAS* :- Serially Attached SCSI (SCSI :- SMALL COMPUTER SYSTEM INTERFACE)
- *NVME* :- Non-Volatile Memory Access
- *UEFI* :- Unified Extensible Firmware Interface
- *FAT* :- File Allocation Table
- *NTFS* :- New technology file system
- *SMP* :-

SATA is used for Hard Drives NVME is used for PCI

We are Having 3 informations stored in the disk

- Cylinder Number
- Head or Platter
- Sector Number

In short we usually say ***CHS*** value.

A disk has different number of cylindrical parts concentric to each other. So, when writing data to a disk the MBR table stores the info of *Cylinder Number, Head or Platter and Sector Number*. **Head or Platter** means a movable hand is present which is used to write or read into the disk.

Physical *CHS* is commonly used for smaller drives. For the larger drive we use ***Logical CHS*** which maps to physical CHS. In this we use **LBS**(Logical Block Addressing)

The process happened after pressing the Power Button

BIOS	MBR	ACTIVE PARTITION	
FAT32	WINDOWS	GRUB	LBA(THIS IS USED WHEN MORE OS's are there)

1. Partition is a logical seperation of disk
2. File system is the way to organize the data
3. Partition is done to make use of it very efficiently
4. MBR - Master Boot Record :- It is a type of Partition Table
5. Partition Table tells the Start CHS and End CHS
6. Now we are not using MBR and using GPT
7. MBR the first sector that is fetched that contains info about the partitions
8. Only 4 partitions are allowed

Parition table tells you at which particular section it starts

GPT - GUID Partition Table

GUID :- Globally Unique Identifier

```
ls blkid # Gives the GUID of disk
```

Boot ROM Code It only boots from a disk with particular GUID

- BIOS search for a *MBR*
- BIOS was invented by *IBM*
- EFI started by Intel
- Intel to overcome BIOS which is having a limitation of having only 4 partitions

Limitations of BIOS

- It thinks like 8086 microprocessor which is 16 bit microprocessor
- BIOS acts as a 16 bit microcontroller even it supports 64 bit which wastes the memory
- BIOS fetches only 1Mb of memory
- 512*1Mb can be used during BIOS

- 20 bit address Bus :- $512*2^{20}$
- EDK-II is a modern, feature-rich, cross-platform firmware development environment for the UEFI and UEFI Platform Initialization (PI) specifications.
- EFI search for GPT
- EFI is a implementation and UEFI is a specification
- SRAM is connected to a small chip(memory) connected with some lines using SPI protocol this is the implementation done in *INTEL*
- EFI applications
    - EDK-2
    - u-boot
    - GPT

FAT

- FAT 8/16/32 : The number represents the file storage it can occupy on a single file
- FAT32 ➖ 2^32 => is the max file size it can support i.e. 4GB
- FAT was started by *Microsoft* and they made it open license
- Linux kernel sees the file system in **inode** format
- Inode structure is the combination of *HEADER* | *DATA* | *ERROR CHECK*
- FAT dont't support inode
- EXT4, BTRFS, APFS are different file system formats
- exFAT is the other format which don't have the limitation of 4gb

BTRFS

BTRFS is a modern copy on write (COW) filesystem for Linux aimed at implementing advanced features while also focusing on fault tolerance, repair and easy administration. You can read more about the features in the introduction or choose from the pages below. Documentation for for command line tools btrfs(8), mkfs

Containers

- Control Group :- CPU, RAM, Disk
- Namespaces :- Address Space, PID, Networking.Here we can get control over the child processes

MOCK - Building Tool For Containers

To know how to make disk partitions of a disk CLICK HERE

Creating a Dummy Disk

```
dd if=/dev/zero of=/home/surya/image.img bs=1024 count=10 :- To create a 1Gb dummy
disk
```

kpartx -a <file_name>

- This maps the file which have different partitions under device files ==> /dev/mappers/

mount /dev/s /home/surya/

- To mount a device to particulare folder this is the way to do. After this we can see all the files that are present in the image file or ISO file.

```
## PARTING AND MOUNTING

$ sudo parted <image_names>

(parted) print  :- This displays the partition made in the image

Sudo kaprtx -av image_name      :- To get /dev/mapper/loopxx file

sudo mount /dev/mapper/loop17p1 /mnt/partition1
sudo mount /dev/mapper/loop17p2 /mnt/partition2
sudo mount /dev/mapper/loop17p3 /mnt/partition3
```

BOOTING

ZSBL => FSBL => RUNTIME => SSBL => KERNEL

BOOT FLOW:-

```
How Boot is auto-mounted ?
 - All the instructions are written in the /etc/fstab file, how it should mount.
```

1. ZSBL / BROM Code (BROM:- Boot ROM code) :- *Zero State Boot Loader*
    - It is a OTP(One Time Programmable) memory
    - It can initialize CPU and some H/W
    - It will check from where to **BOOT**
        - eMMC
        - sdcard
        - UART
    - This runs from ROM
2. FSBL (First Stage Boot Loader)
    - FSBL is used to load SPL into the SRAM
    - DDR is a controller which talks with the **DRAM**
    - Runs in *SRAM*
    - Used to Initialize the DDR
    - Trains RAM
        - Frequency
        - Clock Timings
        - Voltage

    - Example of *FSBL* is U-Boot-spl, Core Boot
    - It can also load *SSBL*

3. RUNTIME

    - Loads into DDR
    - It **PERSIST** in Memory
    - OpenSBI is a *Implementation* and SBI is a *Protocol*
    - kernel communicates with run time when necessary

```
- When you are using EDK2 as runtime it is linked as a Static Library
- EDK2 can be at _RUNTIME_ or _SSBL_
- Memory takes lots of space in the SOC - SRAM
```

4. SSBL
    - U-Boot proper is used
    - Grub is also an example of BootLoader
    - **GRUB** :- Grand Unified Bootloader
    - Types of Boot
        - Network Boot
        - TFTP Boot
        - USB Boot
    - It can provide a shell
5. KERNEL

ROLE OF ROM CODE

- CPU INIT
- POWER GIVEN TO CPU
- MLO is used & it is TI specific => Texas Instruments

SRAM is having very minimal amount of memory 64K - 128K

FIRST STAGE BOOTLOADER

- FSBL helps to load the U-Boot (SPL) into the SRAM
- Uboot-SPL
- DDR init
- DDR controller talks to RAM
- MLO is transferred to SSBL

SECOND STAGE BOOTLOADER

- Uboot - proper
- Grub
- Network - boot
- File System
- Shell

RUNTIME

- Kernel makes the call to the RUNTIME to maintain power management CPU frequency scaling
- ATF - Arm Trusted Firmware on ***ARM***
- SBI - Supervisory Binary Interface on ***RISC-V***
- EDK2
- TFA Github
- Uboot handles three types of firmware
- Uboot
- TFA
- Kernel
- For RISC-V it is openSBI - *Open Source Supervisory Binary Interface*
- OpenSBI is a *Implementation* and SBI is a *Protocol*
- kernel communicates with run time

A Tool Chain Contains - Gcc -compiler - Assembler, linker - Libraries

ATF

ARM Trusted Firmware (ATF) is an open source framework that uses Trustzone to boot a Secure payload and a Non trusted firmware. ATF is used as the initial start code on ARMv8-A cores for all K3 platforms.

ATF provides a reference to secure software for ARMv8-A architecture. It implements the firmware for EL3, including the standard and layered implementation ARM Cortex Application processors ARMv7-A and ARMv8-A.

KOJI

```
- The central server, it receives commands from client, assigning/ passing tasks
to other components
- Koji Client requests Koji Hub in this way
- Koji build fc32-testing openssh-9.0.fc38.riscv64
- Koji Hub stores information about build tasks, status of builds, authorised
users
- Koji Builder is the one which builds the package. It continuously poll the
server for any new builds to be done or not.
```

KOJI BUILDER

```
- Koji Builder should be a native compiler
- Kojid  is the daemon running in the koji builder
- The Kojid invokes the mock to build the packages
- There is a Repository which contains the actual rpm packages
- The Koji Hub gives the actual rpm packages from the repository to the koji
builder for building process
- After building the packages the builder again sent back to the hub and hub
resends the new package to the repository
```

KOJIRA

```
- It is used to create repository and perform maintainance
```

KOJI WEB

```
- It provides a web interface for people to view on-going builds and progress
```

RISC-V BOOT

hexdump -C <.o file> | less

TARGET TRIPLET

`machine-vendor-operating system`

`x86_64-unknown-freebsd`

.itb file

itb: Image Tree Blob

- uboot-proper
- openSBI
- *dtb* :- Device Tree Binary
    - dtb contains information about all the peripherals contained in the board to which it was connected (**HOW THE PCB BOARD WAS CONFIGURED**)

PRIVILEGE LEVELS

- It decides which control registers should access which parameters
- Privilege levels are used to restrict some things that affect the system
- u-boot runs at privilege level and user space have no privilege level

MACHINE MODE IS THE MOST PRIVILEGED MODE AMONG ALL OF THEM

1. Machine mode
    - BROM code
    - u-boot SPL
    - openSBI
2. Supervisor mode
    - u-boot

        - if any interaction need with the H/W it requests the RUNTIME i.e., openSBI
    - kernel
        - After kernel starts it leave us in the user mode
3. User Level mode

ASSEMBLY CODE FROM A FILE

- objdump -d .o file
- Objdump -d a.out
- This gives the assembly instruction of the code
- ld –verbose This gives the linker script rules

```
Loader is always invoked while executing elf on linux
```

REFERENCES

1. STEPS TO CREATE A DOCKER FILE
2. COMMANDS USED IN DOCKER FILE AND HOW TO USE
3. RUNNING APACHE SERVER ON HOST MACHINE
4. RUNNING APACHE SERVER ON DOCKER
5. DOCKER-COMPOSE FILE SPEC REFERENCE
6. DETAIL INFORMATION ABOUT DISK PARTITION AND MBR
7. MANUAL DISK PARTITION USING COMMAND LINE
8. ATF REFERENCE LINK
9. U-BOOT SOURCE CODE
10. RISC-V GNU-TOOLCHAIN SOURCE CODE
11. STAR-FIVE BUILD ROOT SOURCE CODE
12. BUILD ROOT SOURCE CODE
13. ARTICLE ON openSBI
14. FLATTENED uImage TREE (FIT) IMAGES OVERVIEW
15. FLATTENED uImage TREE (FIT) IMAGES