📂 Top-level Linux Directories and Their Purpose

- **`/`**
  - The **root directory** — everything starts here.
  - All other files and directories exist under `/`.

- **`/bin`**
  - **Essential user binaries (commands)** required for all users.
  - Examples: `ls`, `cp`, `mv`, `cat`, `bash`.
  - Needed even if no other filesystems are mounted.

- **`/sbin`**
  - **System binaries** (mostly for system administration).
  - Examples: `ifconfig`, `mount`, `shutdown`.
  - Only root/admin usually runs these.

- **`/etc`**
  - **System configuration files** (text-based).
  - Example: `/etc/passwd` (user accounts), `/etc/ssh/sshd_config` (SSH config), `/etc/fstab` (mount info).
  - Think of `/etc` as the **settings folder** of the OS.

🔹 **`/var`**

- **Variable data** that changes frequently.
- Examples:
  - `/var/log` → system & app logs
  - `/var/spool` → print/mail queues
  - `/var/tmp` → temporary files that survive reboot

🔹 **`/lib` and `/lib64`**

- **Shared libraries** needed by programs in `/bin` and `/sbin`.
- Similar to DLLs on Windows.
- Example: `/lib/x86_64-linux-gnu/libc.so.6`.

🔹 **`/usr`**

- **User applications and utilities** (not essential for boot, but for normal usage).
- Contains programs, libraries, docs.
- Inside `/usr`:
  - `/usr/bin` → extra commands for users (`vim`, `python`, `git`)
  - `/usr/sbin` → extra system admin commands (`apache2`, `nginx`)
  - `/usr/lib` → libraries for above binaries
  - `/usr/share` → shared files (icons, docs, man pages)

- **`/opt`**
  - **Optional software** (usually third-party apps).
  - Example: if you install Google Chrome manually, it may go in `/opt/google/chrome`.

- **`/home`**
  - **User home directories**.
  - Example: `/home/vikram` for your files, downloads, configs.
  - Like "C:\Users" in Windows.

- **`/root`**
  - The **home directory of the root user**.
  - Do not confuse with `/`.

- **`/tmp`**
  - **Temporary files**, deleted after reboot.
  - All users and apps use it for scratch space.

- **`/boot`**
  - Files needed for booting Linux.
  - Example: Kernel (`vmlinuz`), initramfs, bootloader (GRUB config).

🔹 **`/dev`**

- **Device files** (special files that represent hardware).
- Example: `/dev/sda` (disk), `/dev/tty` (terminal), `/dev/null`.

🔹 **`/mnt`**

- **Temporary mount point** for mounting filesystems (manual usage).
- Example: If you mount a USB drive manually, you might use `/mnt/usb`.

🔹 **`/media`**

- **Automatic mount point** for removable devices.
- Example: Plugging in a USB drive → `/media/vikram/USB`.

🔹 **`/proc`**

- **Virtual filesystem** that provides info about running processes and kernel.
- Example: `/proc/cpuinfo`, `/proc/meminfo`, `/proc/1234/` (process with PID 1234).
- It's not real files, just runtime info.

🔹 **`/sys`**

- Another **virtual filesystem** to interact with the kernel and devices.

- Example: `/sys/class/net/` shows network interfaces.

🔹 **`/srv`**

- **Service data** served by the system.
- Example: web server files may be stored in `/srv/www`.

🔹 **`/run`**

- Stores **runtime process data** since last boot.
- Example: PID files (`/run/nginx.pid`).
- Cleared on reboot.

📌 Summary (Easy Mapping)

- `/bin` → Basic commands
- `/sbin` → System commands
- `/etc` → Configs
- `/var` → Logs, caches, variable files
- `/lib` → Libraries
- `/usr` → Extra software
- `/opt` → Third-party apps
- `/home` → User files

- `/root` → Root user's home
- `/tmp` → Temporary files
- `/boot` → Boot loader & kernel
- `/dev` → Devices
- `/mnt`, `/media` → Mount points
- `/proc`, `/sys` → Kernel & process info
- `/srv` → Service data
- `/run` → Runtime info

👉 Now, in companies (MNCs), when they deploy apps, they usually:

- Put **config** files in `/etc/appname/`
- Put **binaries** in `/usr/bin` or `/opt/appname/`
- Put **logs** in `/var/log/appname/`
- Keep **data** in `/var/lib/appname/`

Would you like me to also make you a **visual tree diagram** (like `/etc → config, /var → logs`) so you can memorize it quickly?

Perfect 👍 now we're getting to the **practical installation workflow**.

Since you are installing **3rd party software (Airflow, MLflow, Kafka, RabbitMQ, etc.) using Docker Compose**, the “installation” is really just **running containers** + **deciding where on the host filesystem configs, logs, and data should live**.

I’ll break it down step by step so you can use it for **any software** you install with Compose.

🔹 Step 1: Decide a Base Directory

On your server, pick one directory where you’ll keep all your Docker projects.
Most admins use:

- `/opt` (for apps)
- Example:

```
None
/opt/airflow/
/opt/mlflow/
/opt/kafka/
/opt/rabbitmq/
```

Inside each folder, you’ll keep:

- `docker-compose.yml`
- configs (mapped from `/etc/...`)
- volume mappings

🔹 Step 2: Follow the Linux FHS Best Practice for Volumes

When you mount Docker volumes, map them like this:

- **Configs** → `/etc/<appname>/`
- **Logs** → `/var/log/<appname>/`
- **Data/state** → `/var/lib/<appname>/`

So that even if the container crashes, **your configs, logs, and data stay safe**.

🔹 Step 3: Practical Example — Installing Airflow

1. Create folders:

```shell
sudo mkdir -p /etc/airflow
sudo mkdir -p /var/log/airflow
sudo mkdir -p /var/lib/airflow/dags
sudo mkdir -p /opt/airflow
cd /opt/airflow
```

2. Create a **docker-compose.yml** inside `/opt/airflow`:

```none
version: "3.9"

services:
  airflow:
    image: apache/airflow:2.10.2
    container_name: airflow
    restart: always
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
```

```
    ports:
      - "8080:8080"
    volumes:
      - /etc/airflow:/opt/airflow/config       # configs
      - /var/lib/airflow/dags:/opt/airflow/dags  # DAGs
      - /var/log/airflow:/opt/airflow/logs     # logs
```

3. Run it:

```shell
cd /opt/airflow
docker compose up -d
```

Now you'll have:

- Config → `/etc/airflow`
- Logs → `/var/log/airflow`
- DAGs/data → `/var/lib/airflow/dags`
- Compose file → `/opt/airflow/docker-compose.yml`

🔹 Step 4: Practical Example — Installing MLflow

1. Create folders:

```shell
sudo mkdir -p /etc/mlflow
```

```
sudo mkdir -p /var/log/mlflow
sudo mkdir -p /var/lib/mlflow
sudo mkdir -p /opt/mlflow
cd /opt/mlflow
```

2. Create **docker-compose.yml** inside `/opt/mlflow`:

```
None
version: "3.9"

services:
  mlflow:
    image: ghcr.io/mlflow/mlflow:latest
    container_name: mlflow
    restart: always
    command: >
      mlflow server
      --backend-store-uri sqlite:///mlflow.db
      --default-artifact-root /mlflow/artifacts
      --host 0.0.0.0
      --port 5000
    ports:
      - "5000:5000"
    volumes:
      - /etc/mlflow:/mlflow/config         # configs
      - /var/log/mlflow:/mlflow/logs       # logs
      - /var/lib/mlflow:/mlflow/artifacts  # data
```

3. Run it:

```shell
cd /opt/mlflow
docker compose up -d
```

🔹 General Rule of Thumb for Any Dockerized App

Whenever you install a new app with Docker Compose:

- Keep the **compose file** in `/opt/<appname>/docker-compose.yml`
- Mount:
    - `/etc/<appname>` → container config dir
    - `/var/log/<appname>` → container logs dir
    - `/var/lib/<appname>` → container persistent data dir

That way, **all apps look the same** to you and any sysadmin.

👉 Question for you: do you want me to give you a **ready-made directory + compose template** that you can reuse for *any new app* (you just change the image + paths)? That way you won't need to rethink every time.