le cnam

Software supply chain security for the cloud

Cloud virtualization stack

Agathe Blaise (Thales), Jacopo Bufalino (CNAM)

Overview

Overview

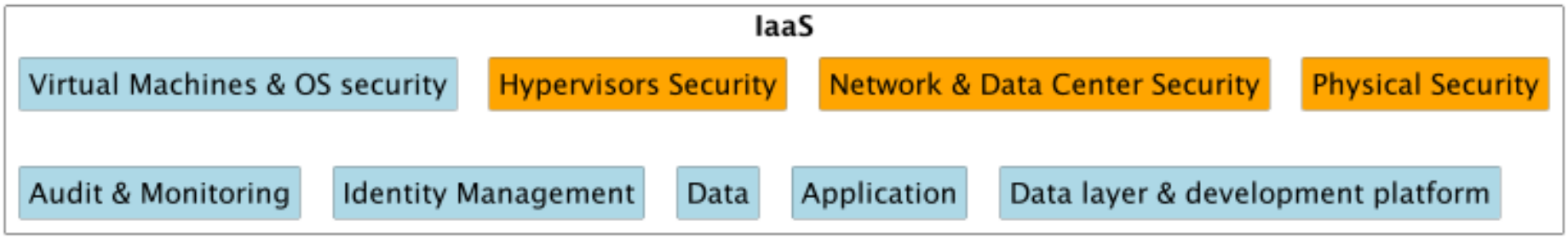
This lecture covers:

- ➜ Linux as the primary OS for cloud environments
- ➜ Virtual Machines
- ➜ Containers and their communication

Containers will be the topics of the first lab

Overview

Recall shared responsibility model

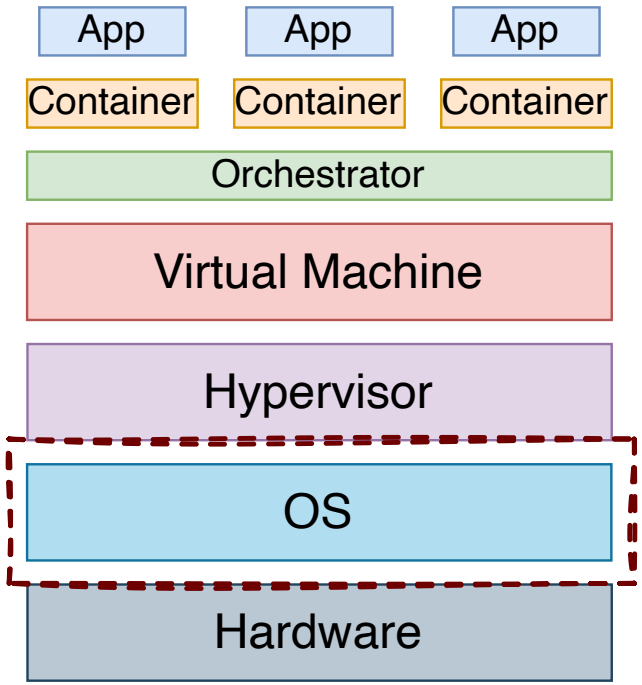


OS, Virtual Machines and Development Platforms (containers) are user's responsibility.

Linux

Linux - a primer

- Linux is an open-source operating system first released in 1991 by Linus Torvalds.
- It is the most popular OS for cloud computing and containerization. That is for two main reasons:
  - Customization: Easy to install and configure software and resources (FOSS - Free and Open Source Software)
  - Security / Isolation: Cornerstone for multi-tenant environments like the cloud



Linux Architecture

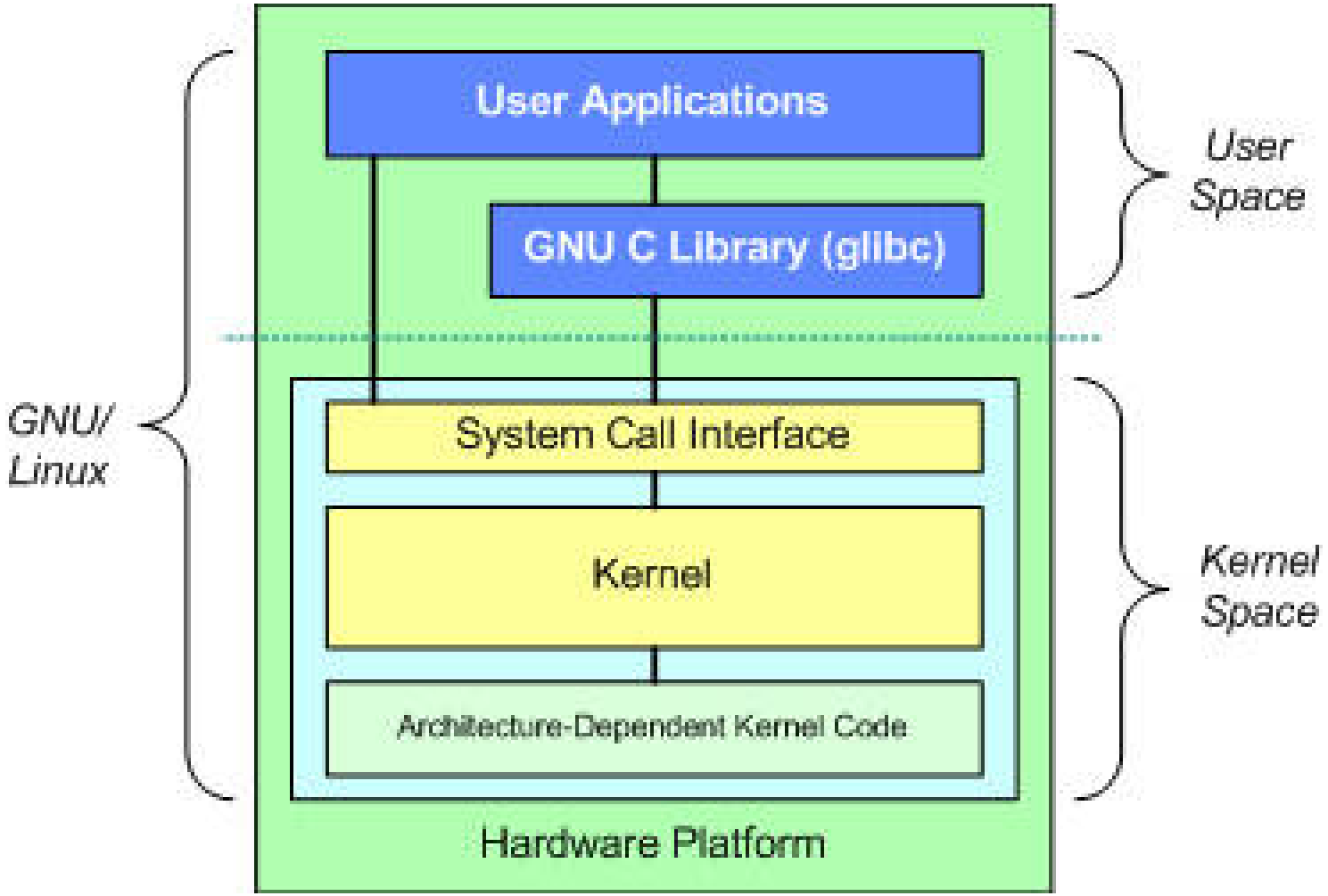
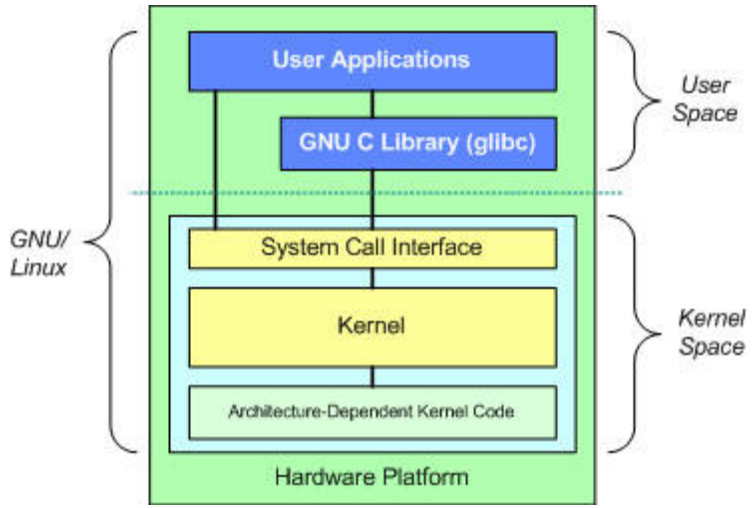


Figure: OS Architecture[1]

[1] https://developer.ibm.com/articles/l-linux-kernel/

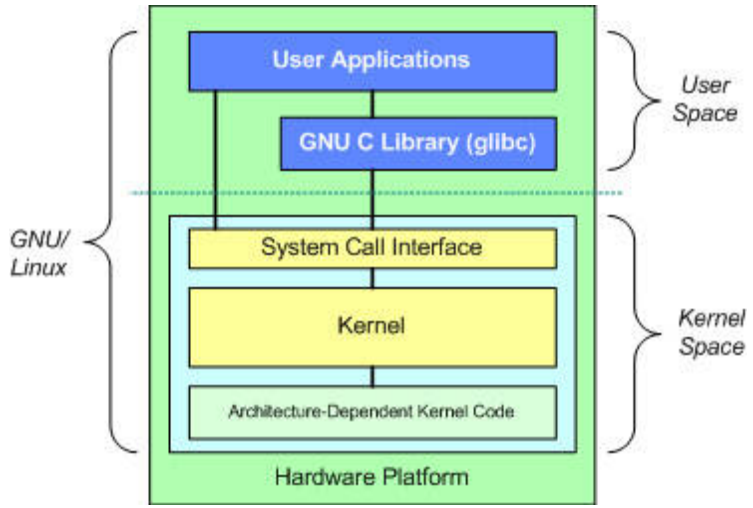
Linux Architecture

- The Linux kernel is at the core of the operating system.
- Allows software to interact with hardware.
- The kernel provides essential services such as process management, memory management, and device management.



Linux Architecture

- There is a translation layer between Kernel and User space. Translation takes time.
- Kernel modules are addons to the kernel.



Overview of Linux networking

Routing in Linux

- ➜ Kernel maintains routing tables.
- ➜ User-space tools to interact with tables (e.g., ip route command).

Firewall

- ➜ iptables: user-space firewall.
- ➜ eBPF: kernel VM and firewall.

Packet flow

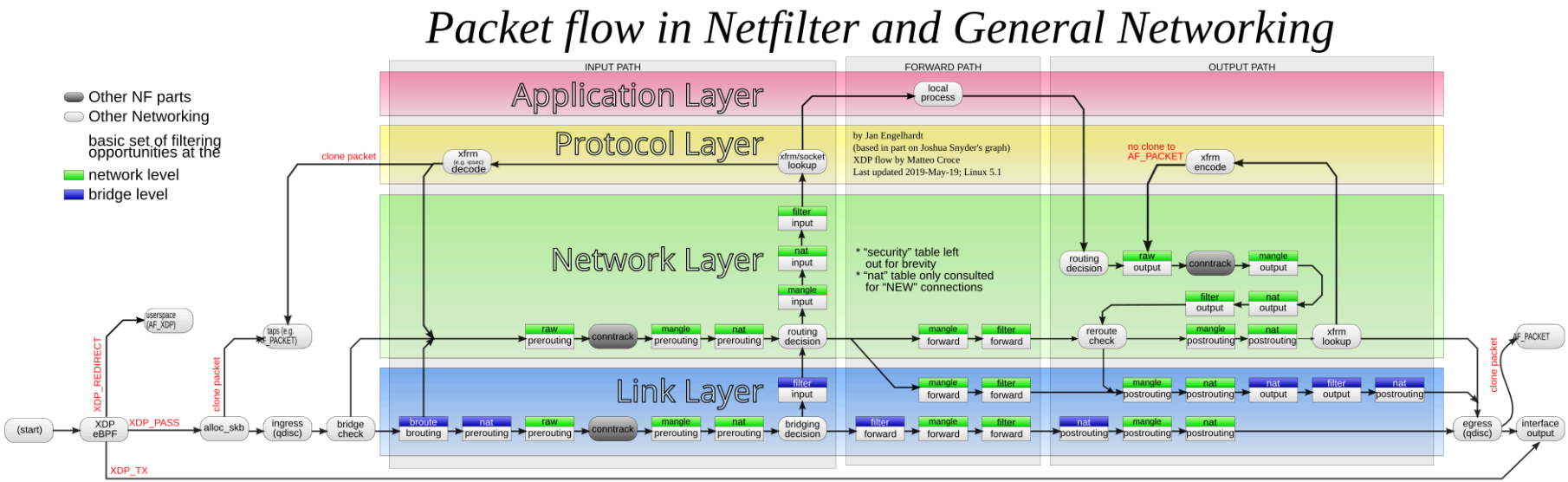


Figure: Linux packet flow (Credits to Jan Engelhardt)

iptables

`iptables` is a user-space utility program that allows a system administrator to configure IP packet filtering and routing.

```
# Allow SSH
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Drop all other incoming traffic
iptables -A INPUT -j DROP
```

eBPF

- eBPF is a Linux kernel VM: custom programs that run in the kernel.
- Two programs: one for the kernel and one for the userspace
  - Kernel app written in C
  - User SDKs for many languages
- Originates from BPF which is used e.g., in Wireshark.

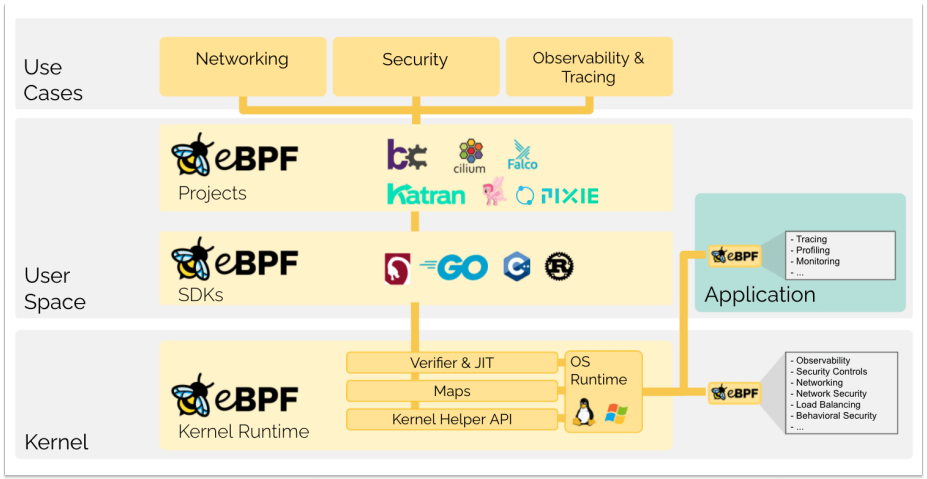


Figure: eBPF Architecture[2]

[2] https://ebpf.io/what-is-ebpf/

eBPF

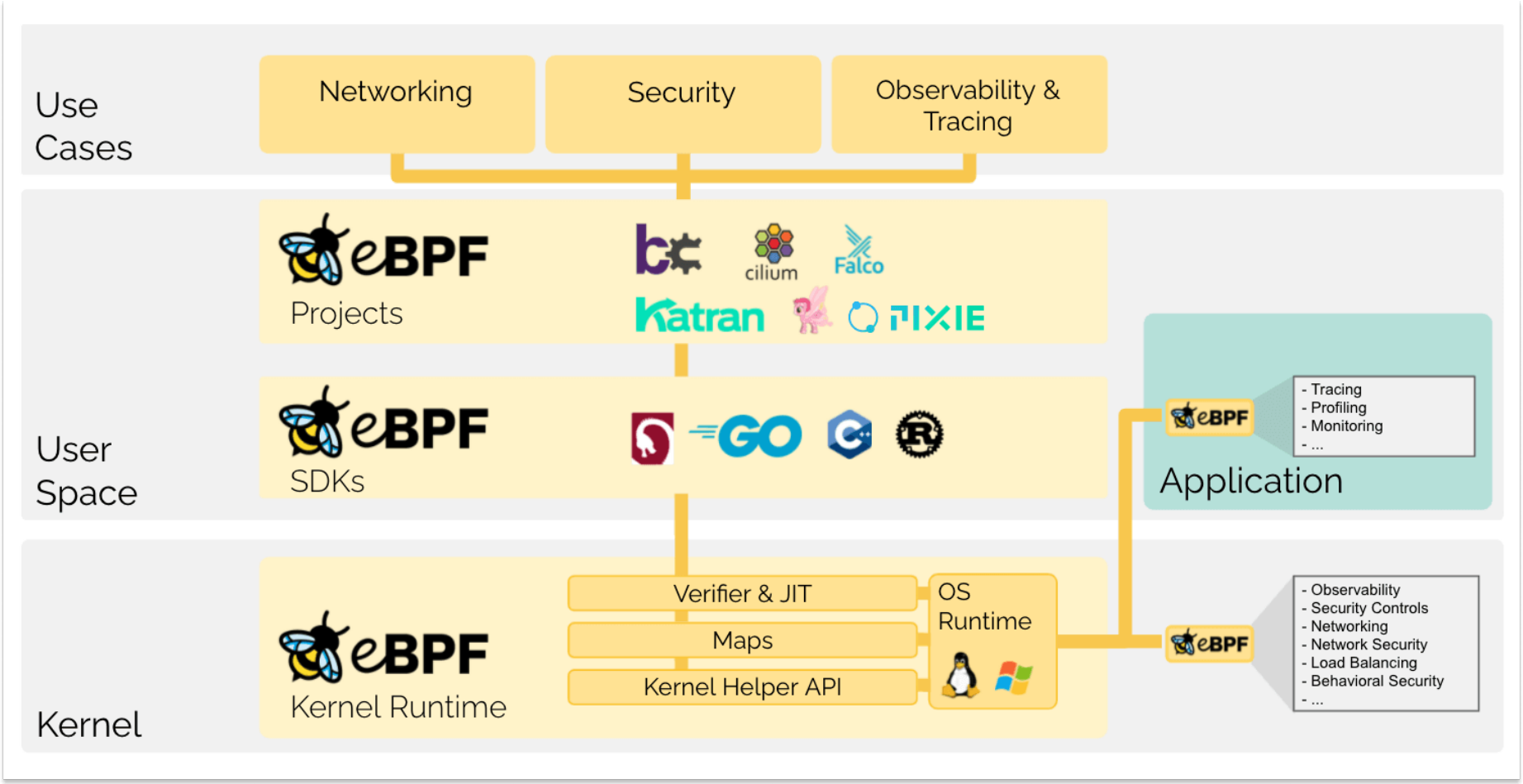


Figure: eBPF Architecture[3]

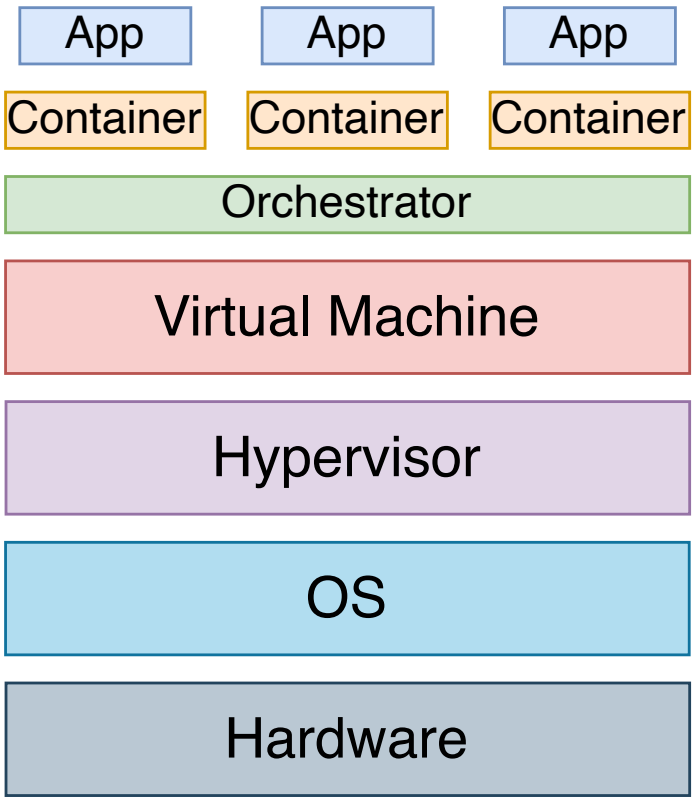
[3] https://ebpf.io/what-is-ebpf/

Virtualization

Why virtualization

- Hardware is expensive
- One single machine may be used by multiple users
- Cost savings
- Less management
- Easier replication

Hypervisors



➜ Typically a Cloud provider responsibility

- There are (few) cases of nested virtualization

NIST (National Institute of Standards and Technology)[4] created a list of baseline functions and requirements.

[4] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-125Ar1.pdf

Hypervisor baseline functions (1/2)

- **VM Process Isolation** Scheduling of VMs and context switching between various processor states during the execution of applications in VMs.
- **Devices Mediation and Access Control** Provides devices to VMs and controls their access (e.g., Network Interface Card (NIC)).
- **Direct Execution of Commands from Guest VMs** Some commands need to be executed bypassing the OS (in some cases).

Hypervisor baseline functions (2/2)

- **VM Lifecycle Management** Includes all functions such as the creation and management of VM images, control of VM states (Start, Pause, Stop), VM migration, making snapshots, VM monitoring, and policy enforcement.
- **Management** Involves updates and patching of the software modules.

Hypervisor security

- Secure Hypervisor Functions: Ensure resource mediation and VM isolation are secure. Apply strict access controls and conduct regular security assessments.
- Enforce VM Isolation: Prevent VMs from interfering with each other. Configure the hypervisor to enforce strict boundaries.
- Secure Virtual Networks: Use encryption for data in transit. Segment networks and apply consistent security policies.

.

Virtualization

Hypervisor

Two types:

- ➜ Bare-metal hypervisors. Directly communicates to the hardware via the Kernel (e.g., KVM, ProxMox, Incus).
- ➜ Hosted hypervisor. They communicate with the OS to access the hardware (e.g., VirtualBox).

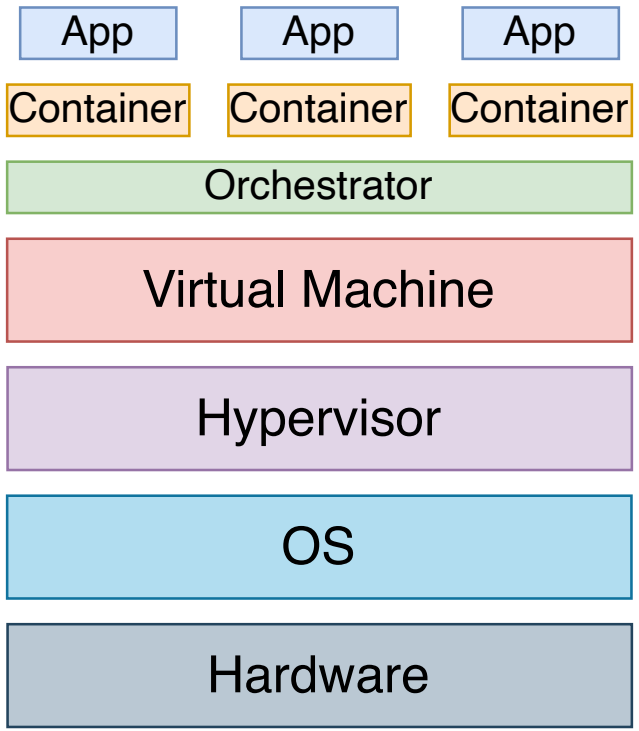


Figure: Virtualization Stack

Why bare-metal is better

➜ Modern CPUs have virtualization instructions that simplify management of VMs.
➜ These allow, for instance, to directly access parts of the memory or to connect to devices (PCI passthrough).
➜ Instructions run directly on the CPU.

Virtual Machine (VM)

- Full OS virtualization
- Individual kernel
- May have different architecture than the host

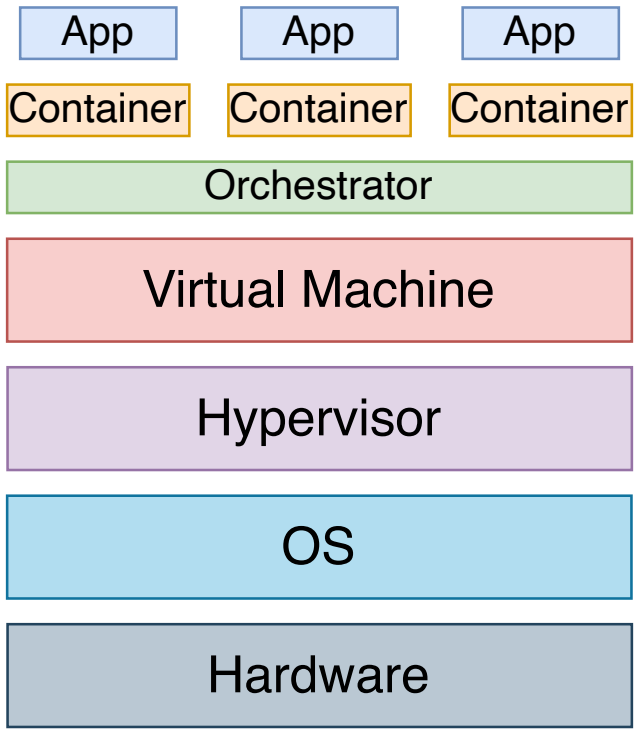


Figure: Virtualization Stack

Virtual networking

Linux has rich virtual networking capabilities that are used as basis for hosting VMs and containers, as well as cloud environments. They are useful for e.g.:

- ➜ Allow VMs and containers to communicate with each other
- ➜ Separate traffic
- ➜ Bridge containers
- ➜ Create tunnels

From VMs to containers

VMs Limitations

1. Storage (a VM typically occupies at least 30GB of space)
   - Kernel data is the same if you always use Linux
   - Many times the same data is stored (e.g., Libraries)
2. Startup time: in the order of minutes
3. Mutability: when VM run, they store their state and rollbacks are difficult.

Complex transition towards containers

Lightweight virtualization

There was always the need for lightweight virtualization in terms of isolation in multi-tenant systems (multiple processes running on the same host).

Problems

- ➜ Lack of features in the kernel to properly isolate processes.
- ➜ Difficult to *orchestrate* all such features effectively.

Key innovations in Linux kernel

- Process permissions
- Resource management
- Linux Namespaces

User permissions

- Traditionally, users and groups have permissions over files, folders and executables
  - `chmod g+rw file.txt`
- Either sudo or not
- Very complex to have fine-grained permissions

Problem: by default a process has the same permissions as the user

Process permissions: Linux Capabilities

Traditionally process could run as root or not. With Linux Capabilities we have a more fine-grained control over what a process can do (good for security: limit the impact of a compromised process).

Examples of capabilities

- **CAP_NET_ADMIN**: Allows network-related operations such as configuring interfaces and routing tables.
- **CAP_SYS_ADMIN**: Process is system administrator.
- **CAP_SYS_BOOT**: Process can reboot the system.

Capabilities also help in reducing the risk of privilege escalation attacks.

Resource management

Cgroups (control groups) are a Linux Kernel feature that allows the system to allocate its resources to processes.

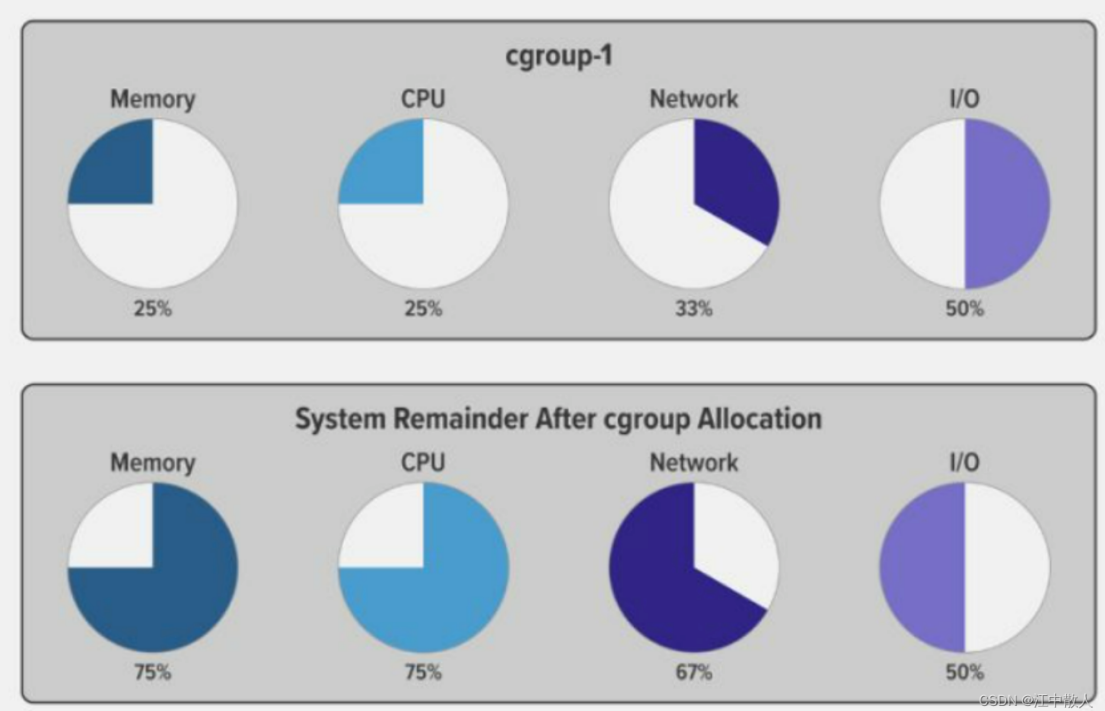


Figure: Cgroups

Linux Namespaces

It is important to isolate processes for multi-tenancy purposes and to protect against malicious actors.
Linux uses the concept of namespace to isolate the different elements of a process such as:

- Process IDs
- Network interfaces
- Users
- Mount points
- Cgroup

Linux namespaces – Isolation of processes

```
1     struct nsproxy {
2     ...
3     struct uts_namespace     *uts_ns; # User
4     struct ipc_namespace     *ipc_ns; # Process
5     struct mnt_namespace     *mnt_ns; # Mount
6     struct pid_namespace     *pid_ns; # PID
7     struct net               *net_ns; # Network
8     struct time_namespace    *time_ns; # Time
9     struct cgroup_namespace *cgroup_ns; # Resources
0     };
```

Listing 1: Snippet from the Linux Kernel (`nsproxy.h`) (Lines 32-42)

Containers

What is a Container?

- A container is a lightweight, process that includes a Filesystem (Image) and an Execution Environment.
- The container filesystem includes everything that is needed for an application to run:
  - Software
  - Dependencies
  - Libraries
  - Filesystem
  - Environment

Containers share the host's kernel

Linux containers: features

- ➜ Immutable: do not change across runs
- ➜ Portable: easy to download and share
- ➜ Scalable: can run multiple instances
- ➜ Efficient: fast startup time and low resource consumption

Linux containers: solving the VM bottlnecks

- Isolation and Efficiency
  - Linux namespaces provide isolation for processes, network, and file systems.
  - Control groups (cgroups) to limit and prioritize resource usage (CPU, memory, disk I/O, etc.).
- Portablity and Scalability
  - Layered filesystem allows for data caching
  - CoW (Copy on Write) makes it easy to deploy multiple containers at the same time
- Immutability
  - Each container has writable layer on top of read-only image layers. When container exists that layer is deleted.

Container Runtime

A container runtime is a software that helps deploy and manage containers in their entire lifecycle.
Container runtimes fostered the adoption of containers

- ➜ Image management
- ➜ Networking
- ➜ Container creation and deletion
- ➜ Log collection
- ➜ Environment setup

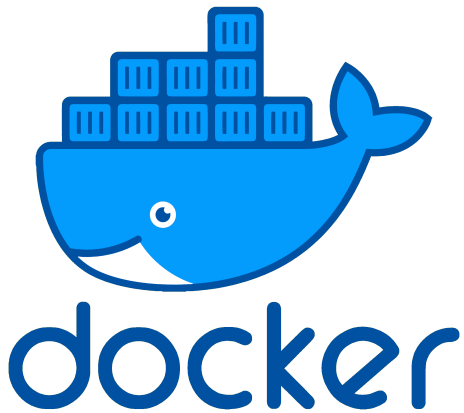


Figure: Docker is a popular container runtime

Container Image format (OCI)

In principle, you only need a filesystem or a file to run containers. OCI (Open Container Initiative) is an open governance structure that defines how to package and distribute container images:

- ➜ Container images are divided in layers
- ➜ OCI images are composed of a manifest, a configuration, and a set of layers.
- ➜ The manifest lists the layers and provides metadata about the image.
- ➜ The configuration includes details about the container runtime environment.
- ➜ Layers are tar archives that contain the filesystem changes made by the image.

Containerfile

Example Containerfile / Dockerfile

```
FROM alpine
COPY hello.sh /
ENTRYPOINT [ "/hello.sh" ]
```

Use of Containerfile

- ➜ Set of instructions to automate the creation of an image
- ➜ (Almost) Each instruction generates a new layer in the container image.

Full list of instructions:
`https://docs.docker.com/reference/dockerfile/`

Container networks

Containers can be connected to each other and to the Internet using different types of networks. The most common types are:

- Bridge: Connects containers to the host
- Host: Connects containers to the host's network
- Overlay: Connects containers across different hosts