Network Fundamentals for Cloud

BITS Pilani
Pilani Campus

Nishit Narang
WILPD-CSIS

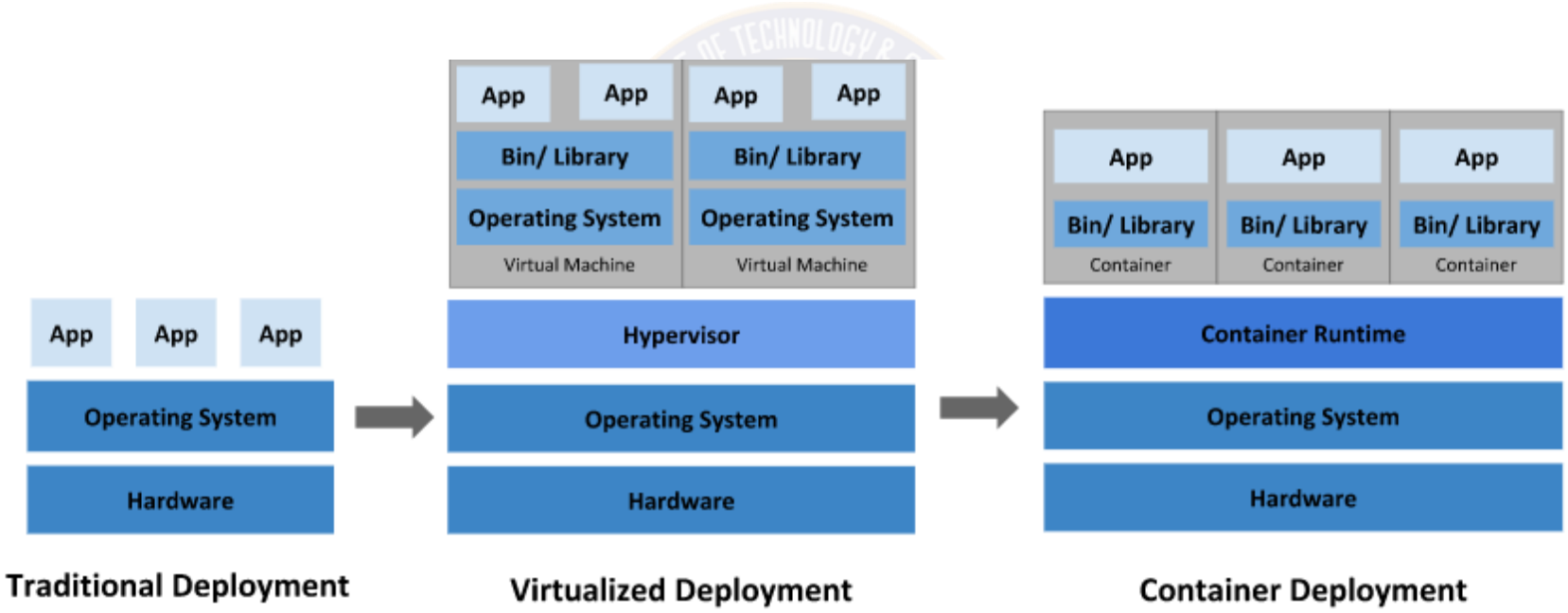
innovate achieve lead

BITS Pilani
Pilani Campus

CC ZG503: Network Fundamentals for Cloud

Lecture No. 13: VM, VNF and Container Networking

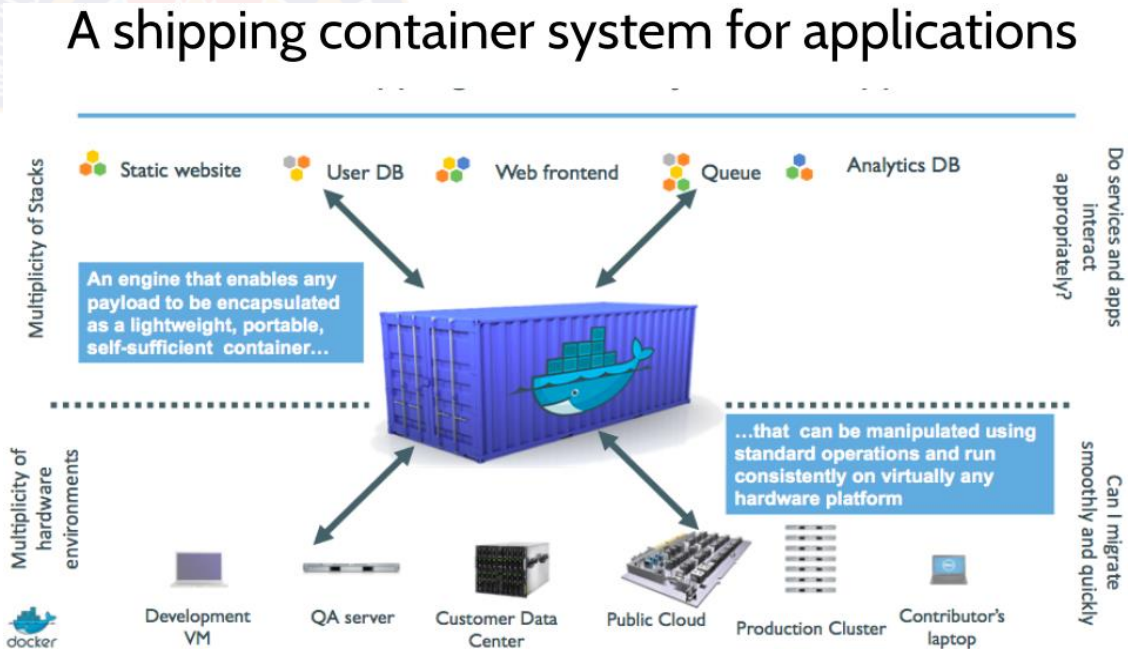
Going back in time



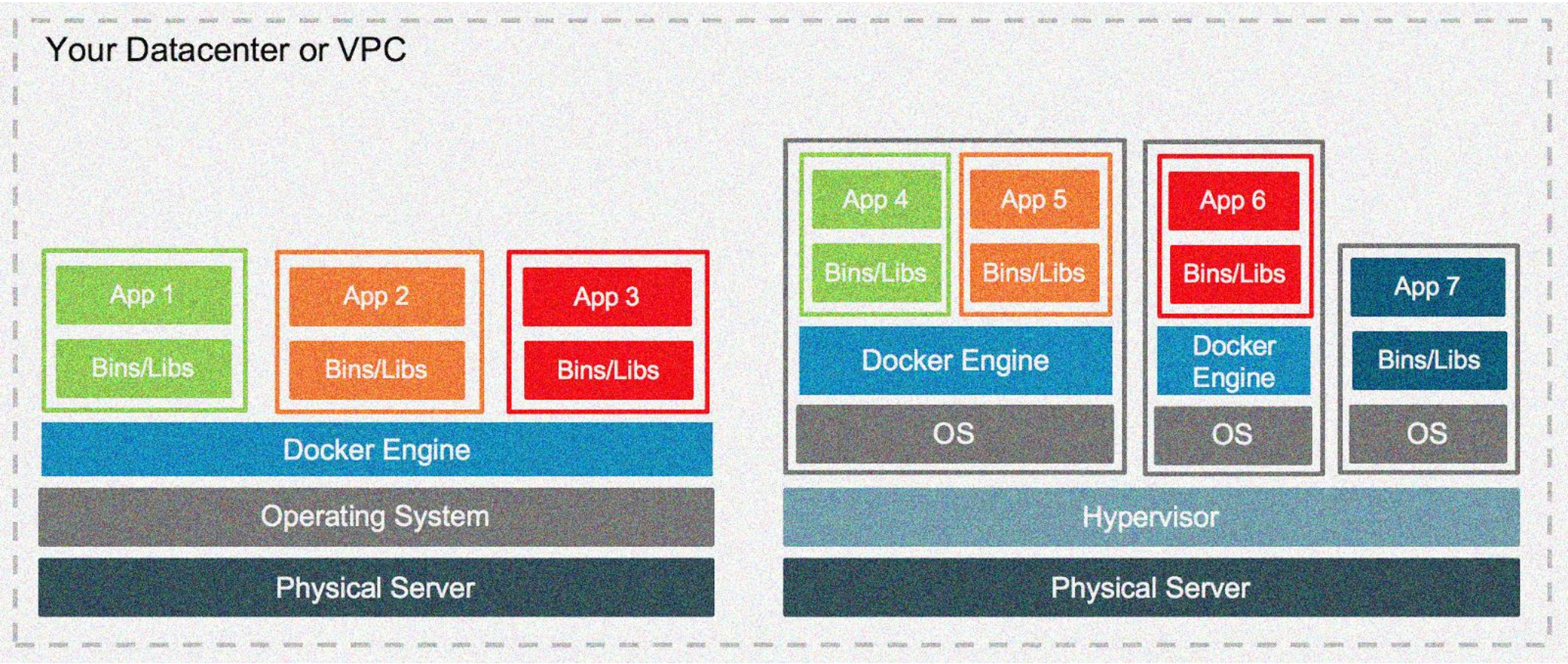
Linux is the de facto leader and host OS in the modern data center

What are Containers?

- A software container is a standardized package of software.
- Everything needed for the software to run is inside the container.
- The software code, runtime, system tools, system libraries, and settings are all inside a single container

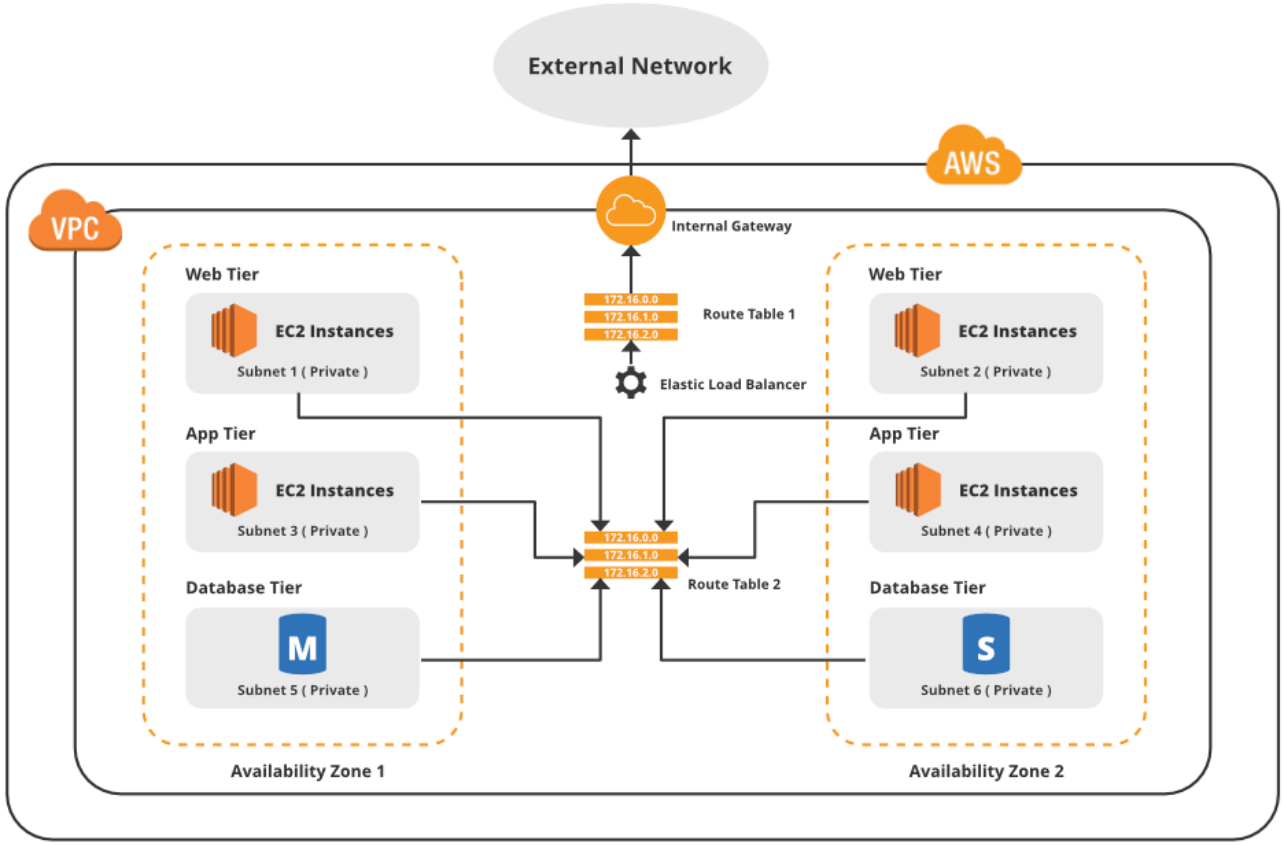


Containers and Virtual Machines



Overlay Network Traffic

- Traffic exchange between VMs (instances)
- Traffic handling at VNFs (NAT Gateway, Internet Gateway, Router, Load Balancer…)
- Traffic exchange between Containerized applications



VM / VNF Networking

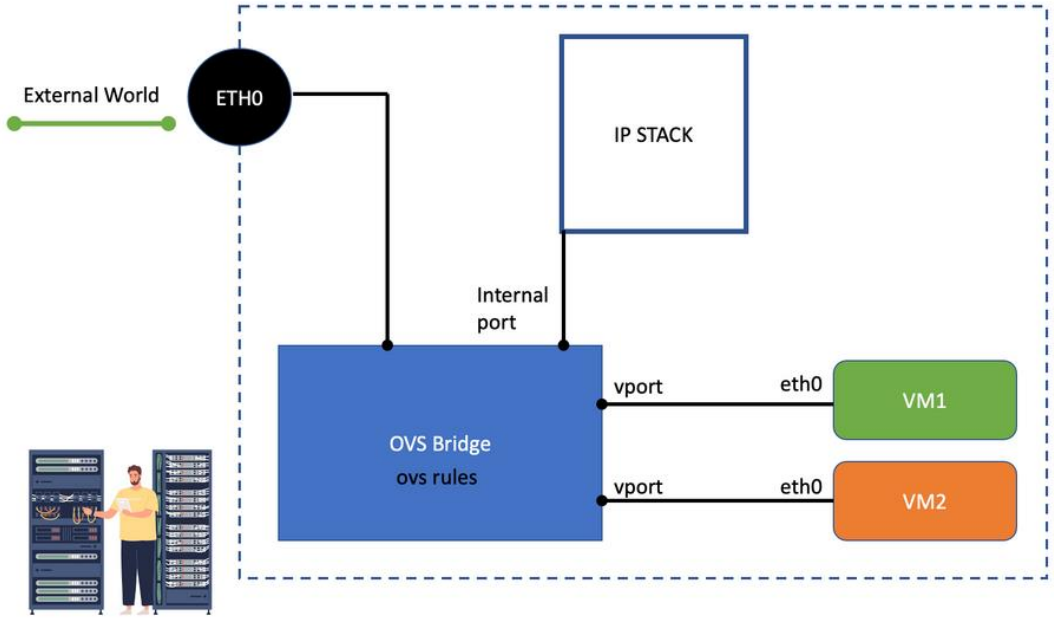
The vSwitch!!

- Virtual Switch:
  - Virtual switches are software-based switches that reside in the hypervisor kernel providing local network connectivity between virtual machines (and now containers)
  - They deliver functions such as MAC learning and features like link aggregation, etc, just like their physical switch companions have been doing for years.
- Open vSwitch (OVS):
  - The OVS is a multilayer virtual switch implemented in software.
  - It uses virtual network bridges and flow rules to forward packets between hosts.
  - It behaves like a physical switch, only virtualized.
  - Like a traditional switch, OVS maintains information about connected devices, such as MAC addresses.
  - In addition, it enhances the monolithic Linux Bridge plugin and includes overlay networking (GRE & VXLAN), providing multi-tenancy in cloud environments.

Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net)

OVS

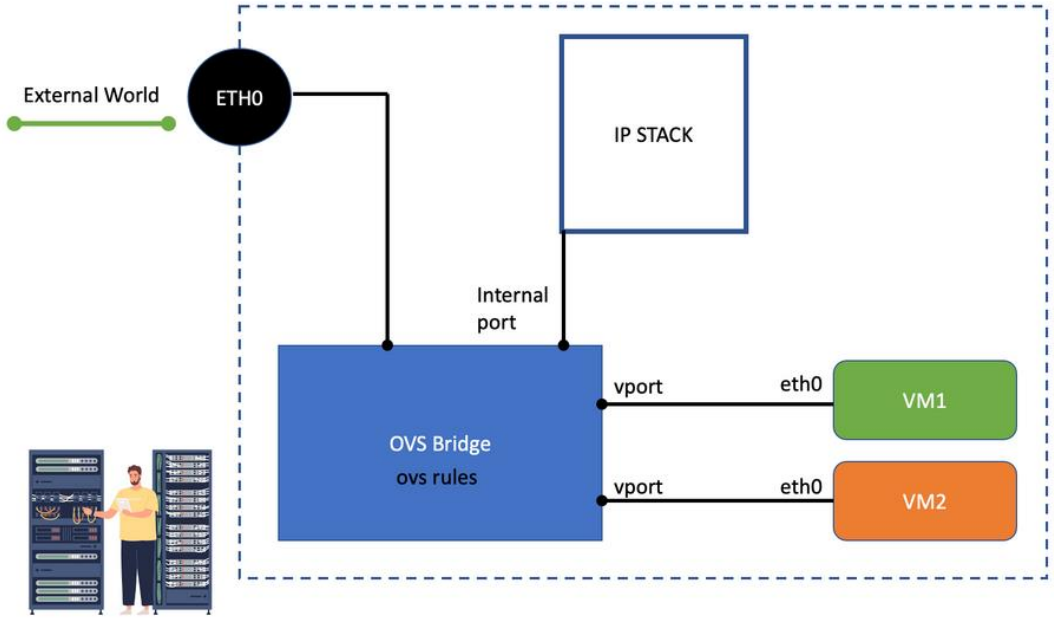
- OVS (Open vSwitch) is a fundamental component of modern and open data center SDNs, where it aggregates all the virtual machines at the server hypervisor layer.
- It represents the ingress point for all the traffic exiting VMs, and can be used to forward traffic between multiple virtual network functions in the form of service chains.



Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

OVS (Contd.)

- It is licensed under the open source Apache 2 license.
- Open vSwitch supports multiple Linux-based virtualisation technologies, including Xen/XenServer, KVM, and VirtualBox.



Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

OVS Supported Features

- High-performance forwarding using a Linux kernel module (*aka Fast Path*)
- Standard 802.1Q VLAN model with trunk and access ports
- OpenFlow protocols, NetFlow, sFlow(R), and mirroring for increased visibility
- QoS (Quality of Service) configuration, plus policing
- Multiple tunneling protocols like Geneve, GRE, VXLAN, STT, and LISP
- 802.1ag connectivity fault management
- With SDN and VNF being at the heart of modern data centre architectures, Open vSwitch, seen as a network function, can be an entity of an SDN estate driven directly by an SDN Controller or an OpenStack plug-in like Neutron.

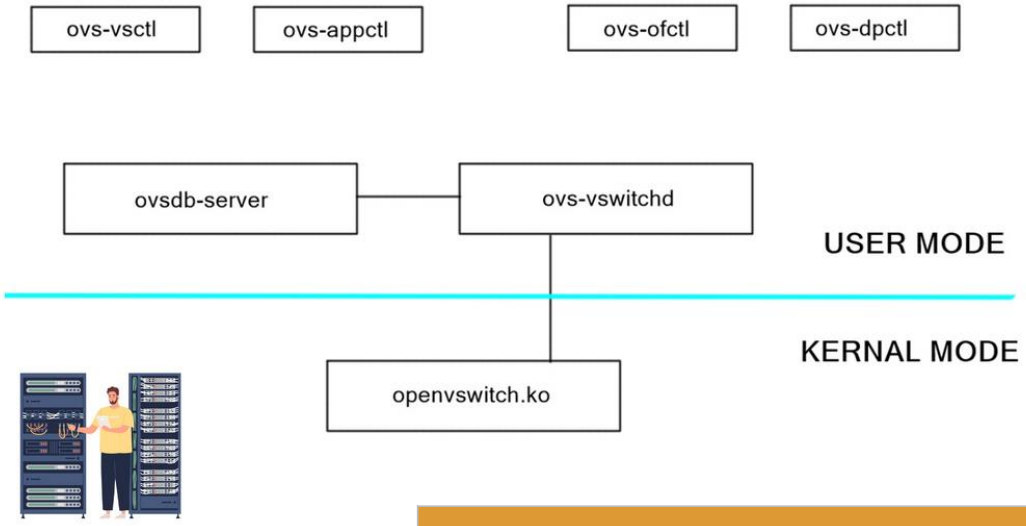
Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

OVS Component Architecture

- Main Components:
  - ovs-vswitchd, a daemon that implements and controls the switch on the local machine, **along with a companion Linux kernel module** for flow-based switching. This daemon performs a lookup for the configuration data from the database server to set up the data paths.
  - ovsdb-server, a lightweight database server that ovs-vswitchd queries to obtain its configuration, which includes the interface, the flow content, and the Vlans. It provides RPC interfaces to the vswitch databases.
  - ovs-dpctl, a tool for configuring the switch kernel module and controlling the forwarding rules.
  - ovs-vsctl, a utility for querying and updating the configuration of ovs-vswitchd. It updates the index in ovsdb-server.
- Supporting Tools:
  - ovs-ofctl, a utility for querying and controlling OpenFlow switches and controllers.
  - ovs-testcontroller, a simple OpenFlow controller that may be useful for testing
  - Ovsdbmonitor, a graphical utility to display the data stored in ovsdb-server.

Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

OVS Component Architecture (Contd.)

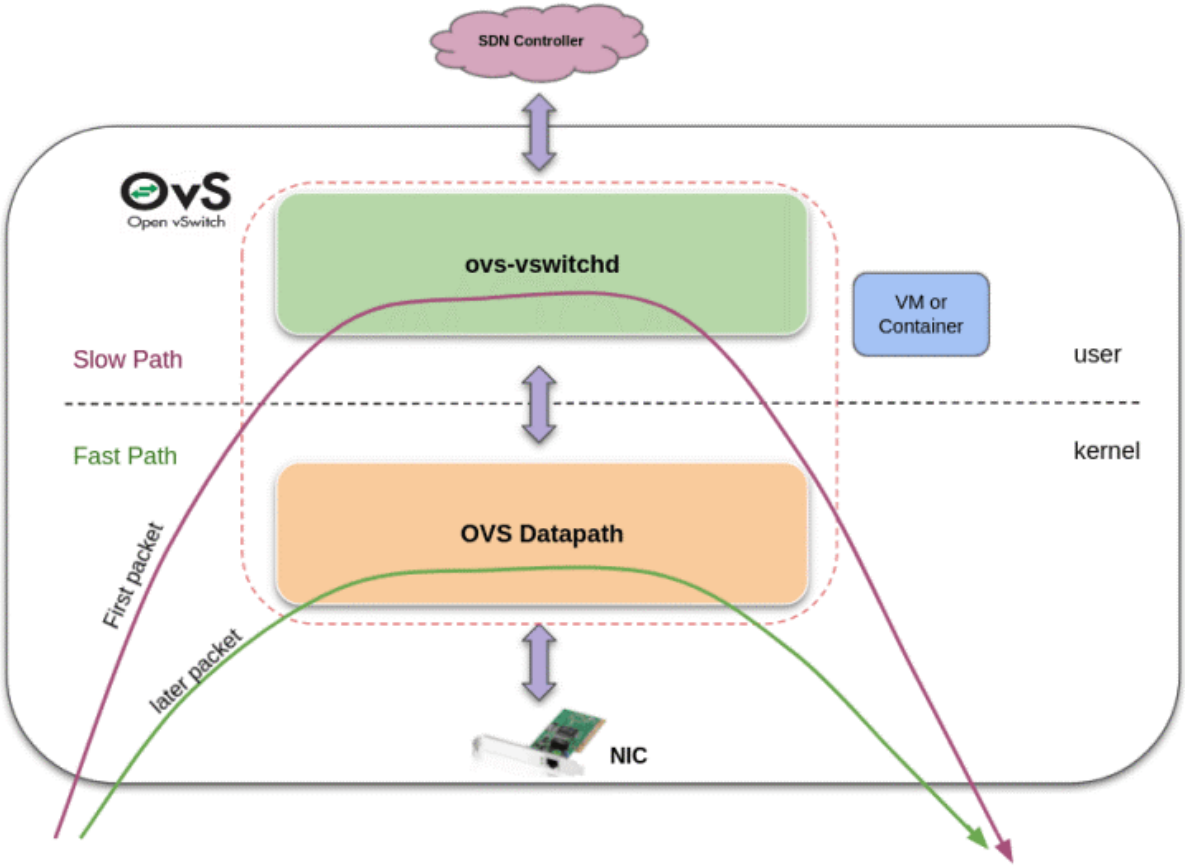


CLI Component	OVS Component
Ovs-vsctl manages the state	in the ovsdb-server
Ovs-appctl sends commands	to the ovs-vswitchd
Ovs-dpctl is the	Kernal module configuration
ovs-ofctl work with the	OpenFlow protocols

Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

OVS Flow Forwarding

- When building software-defined networks, an SDN controller is required to take care of the high level network policies to define things like how virtual networks should communicate with each other, and which flows should be permitted or blocked.
- OVS is the component that enforces these policies via OpenFlow.
- The classic OVS design and implementation make reference to two main components, which reside separately in the user and kernel spaces.
  - *OVS-VSWITCHD* resides within the operating system's user space, and is usually referred to as the "Slow path".
  - The other one, *OVS Datapath*, is the "Fast path" when traffic flows traverse OVS. (*openvswitch.ko*)



Source: OVS Bridge and Open vSwitch (OVS) Basics (network-insight.net) and Data centre networking: what is OVS? | Canonical

Performance Challenges with vSwitch

- In non-virtualized environments, the data traffic is received by the physical NIC (pNIC) and is sent to an application in the user space via the kernel space.
- However, in a virtual environment or VNF network, there are pNICs, virtual NICs (vNICs), a hypervisor, and a virtual switch in between.
- The hypervisor and the virtual switch take the data from the pNIC and then send it to the vNIC of the Virtual Machine or the Virtual Network Function (VNF), then to the application.
- The virtual layer causes virtualization overhead and additional packet processing that reduces I/O packet throughput and builds ups bottlenecks.

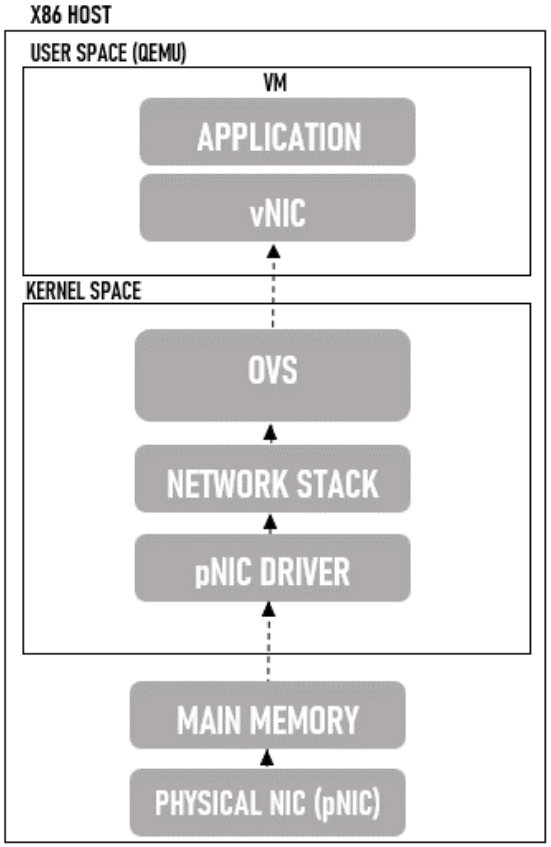
Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

Performance Bottlenecks in OVS

The packets received by the pNIC go through various steps before transmitting to the Virtual Machines' applications. These steps are as follows:

1. The pNIC receives the data traffic and places it in an Rx queue (ring buffers).
2. The pNIC forwards the packet to the main memory buffer via Direct Memory Access (DMA). The packet comes with a packet descriptor which includes the memory location and packet size.
3. The pNIC sends an Interrupt Request (IRQ) to the CPU.
4. The CPU passes the control to the pNIC driver that services the IRQ. The pNIC driver receives the packet and moves it into the network stack. Then, the packet gets into a socket and is placed into a socket receive buffer.
5. The packet is copied to the Open vSwitch (OVS) virtual switch.
6. The packet is processed by the OVS and is then sent to the VM. The packet switches between the kernel and user space, which consumes extensive CPU cycles.
7. The packet is received by the vNIC and is placed in an Rx queue.
8. The vNIC forwards the packet and its packet descriptor to the virtual memory buffer via DMA.
9. The vNIC sends an IRQ to the virtual CPU (vCPU).
10. The vCPU passes the control to the vNIC driver that services the IRQ. It then receives the packet and moves it into the network stack. Finally, the packet arrives in a socket and is placed into a socket receive buffer.
11. The packet data is then copied and sent to the VM application.

Each packet received must go through the same procedure, which demands the CPU to be interrupted frequently. Interrupts add a significant amount of overhead, and it affects the overall Virtualized Network Function performance.



Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

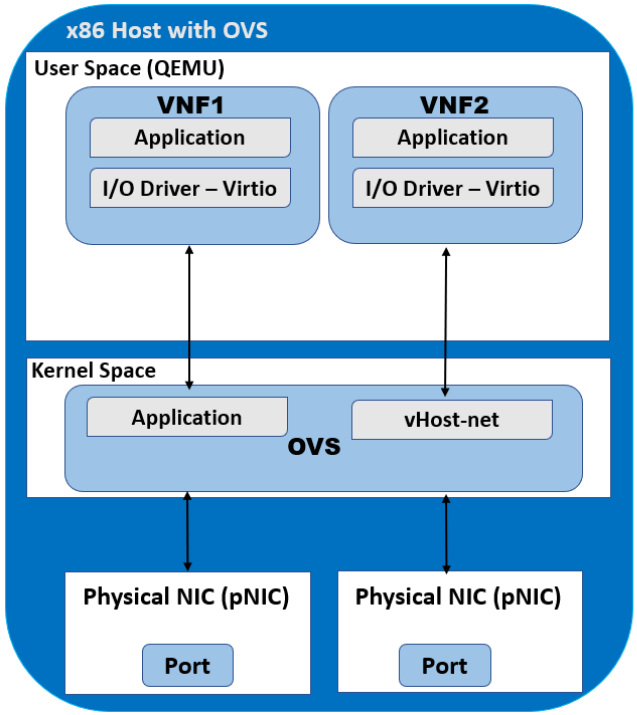
Overcoming OVS Bottlenecks

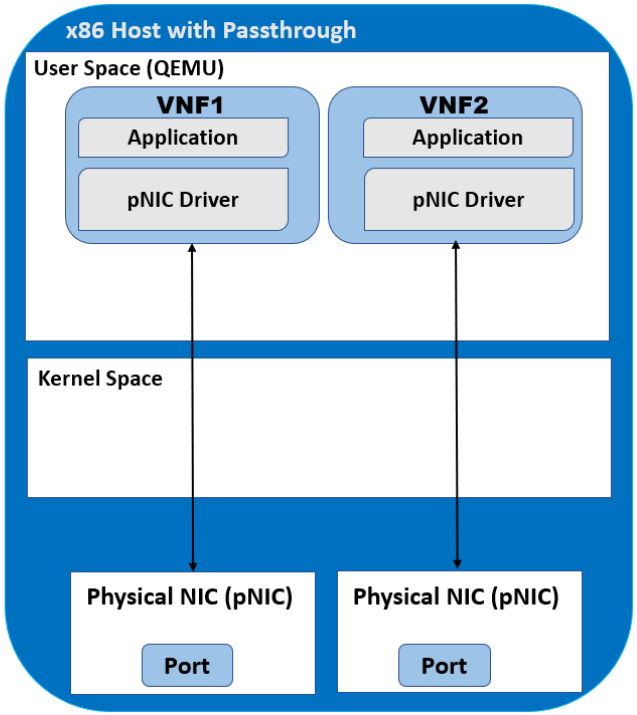
- Multiple I/O technologies are developed to optimize the VNF network by avoiding the virtualization overhead and increasing the packet throughput.
- However, these I/O technologies require some support on the physical Network Interface Cards (NICs) to be implemented.
- Few techniques used to improve VNF network I/O performance include:
  - PCI Passthrough
  - SR-IOV
  - OVS-DPDK

Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

PCI Passthrough

- Peripheral Component Interconnect (PCI) passthrough gives VNFs direct access to physical PCI devices that seem and behave as if they were physically connected to the VNF.
- PCI passthrough can be used to map a single pNIC to a single VNF, making the VNF appear to be directly connected to the pNIC.
- Using PCI passthrough provides performance advantages, such as:
  - One-to-One Mapping
  - Bypassed Hypervisor
  - Direct Access to I/O Resources
  - Increased I/O Throughput
  - Reduced CPU Utilization
  - Reduced System Latency
- However, using PCI passthrough dedicates an entire pNIC to a single VNF. It cannot be shared with other Virtual Network Functions (VNFs). Therefore, it limits the number of VNFs to the number of pNICs in the system.

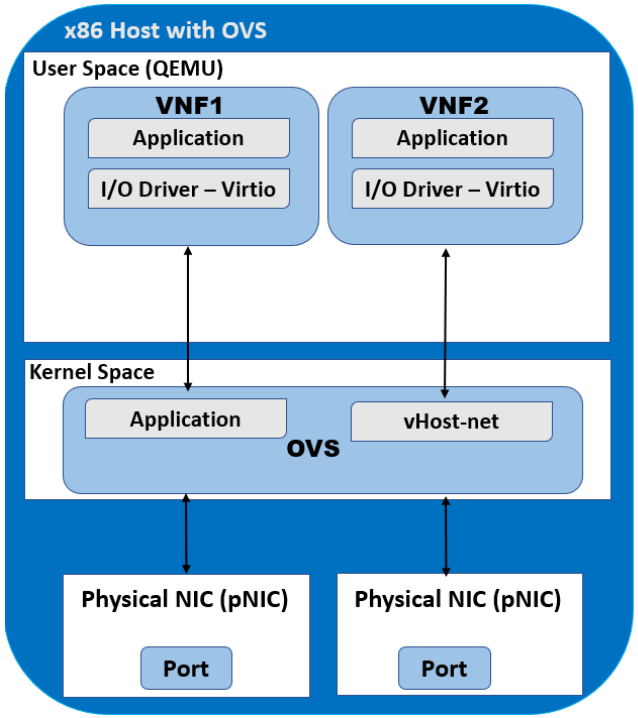


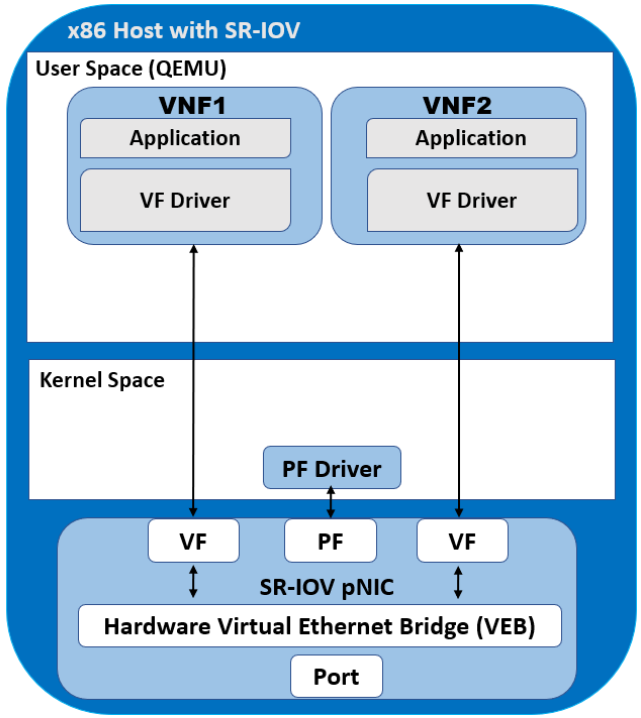


Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

Single-Root I/O Virtualization (SR-IOV)

- SR-IOV is a PCI passthrough feature that enables multiple VNFs to share the same pNIC. It emulates multiple Peripheral Component Interconnect Express (PCIe) devices, such as pNICs, on a single PCIe device.
- The emulated PCIe device is referred to as Virtual Function (VF), and the physical PCIe device is referred to as Physical Function (PF).
- Physical Functions have full PCIe features.
- Virtual Functions have simple PCIe functions that process I/O only.
- Using PCI passthrough technology, the VNFs have direct access to the VFs.

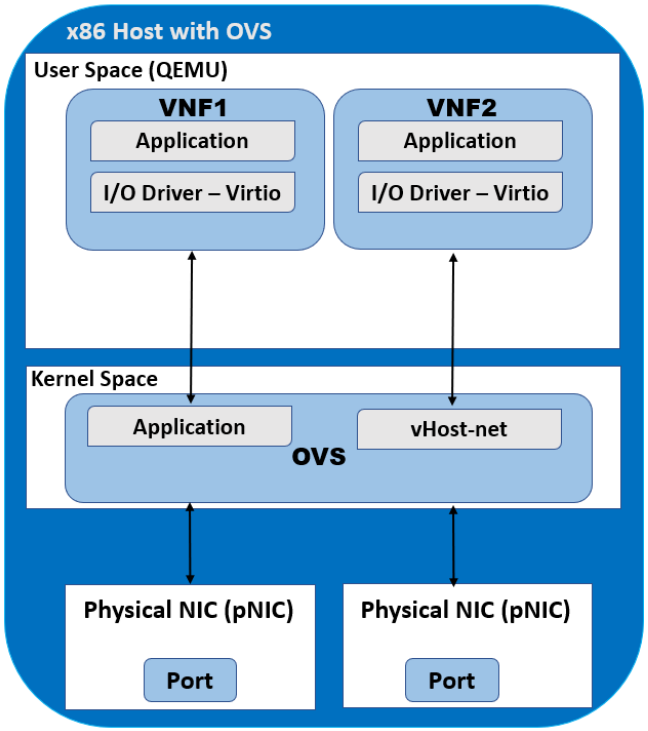


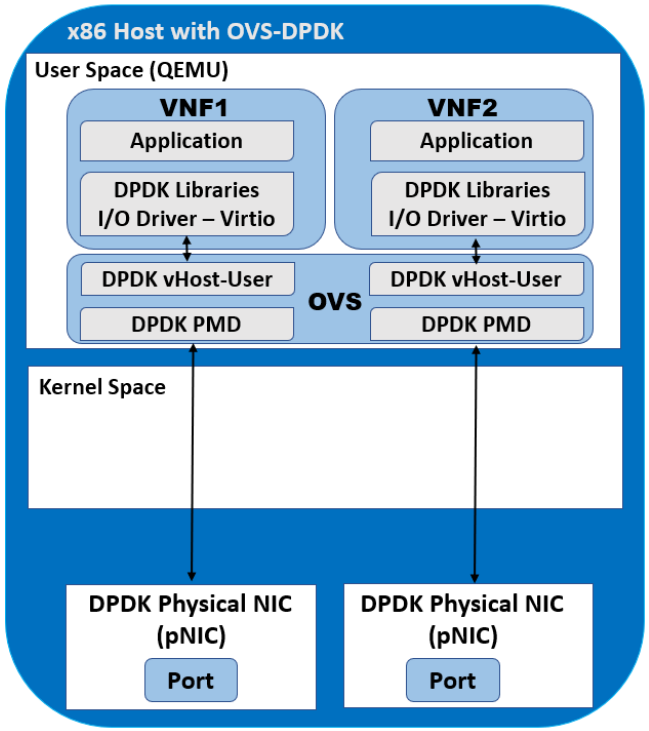


Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

OVS Data Plane Development Kit (OVS-DPDK)

- Open vSwitch (OVS) was upgraded with the Data Plane Development Kit (DPDK) libraries to address the performance impact of interruptions on packet throughput because of interrupts. OVS-DPDK operates in the user space.
- The OVS-DPDK Poll Mode Driver (PMD) polls and processes the data coming into the pNIC.
- It bypasses the whole kernel space since it wouldn't also need to send an IRQ to the CPU when a packet is received.
- DPDK PMD needs one or more CPU cores dedicated to polling and processing incoming data to accomplish this.
- When the packet enters the OVS, it is already in the user space. Therefore, it can be directly switched to the VNF and significantly improves performance.





Source: SR-IOV, PCI Passthrough, and OVS-DPDK - Study CCNP (study-ccnp.com)

OVS Vs Other VNF I/O Techniques

- Use of OVS vs one of the other I/O Techniques is a trade-off between:
  - Performance (PCI-passthrough, SR-IOV, OVS-DPDK)
  - Flexibility (OVS)
- OVS is like a standard switch, only software-based
  - Does not require anything special from the underlying hardware, esp. pNIC
  - Supports openflow protocol and can be controlled via SDN Controller
  - Therefore, VM migration is easy and flexible to perform
- The other I/O techniques provide I/O-performance for VNFs, but
  - Require specialized underlying hardware (SR-IOV pNIC, DPDK pNIC)
  - Can only be used on server machines where such hardware exists

Container Networking

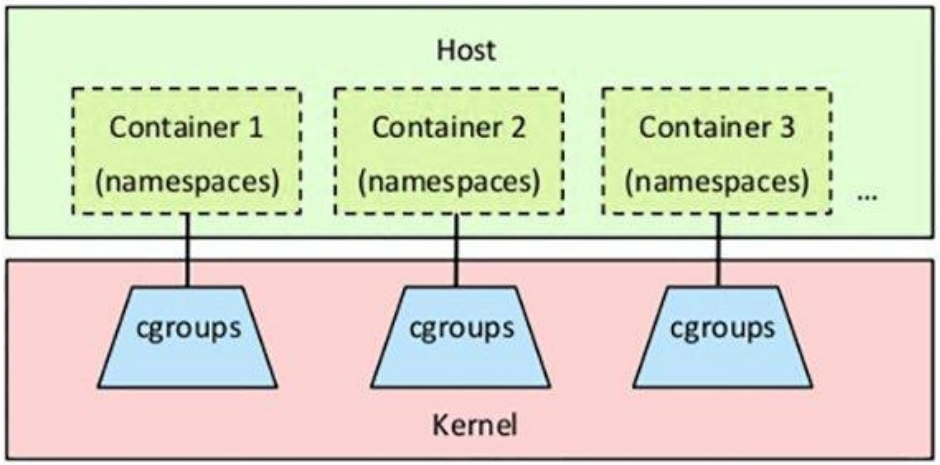
Containers

- Containers are defined a couple of different ways, one focused on packaging and the other on the execution environment
- As a packaging model, a container bundles the application code together with its dependencies, configuration, and data such that it can run without requiring that these things be provided by the environment in which it runs.
- As an execution model, a container is an isolation unit that enforces limits on much of a resource—CPU, memory, and so on—that it uses.
- Thus, a container allows the sharing of resources on a server running a common OS. A single container can run multiple processes.
- A container is more heavyweight than a process but less heavyweight than a VM.
  - Like a process and unlike a VM, the container shares the OS with other containers running on the same host.
  - But unlike a process, and akin to a VM, each container has its own private view of the network, including hostname and domain name, processes, users, filesystems, and inter-process communication

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

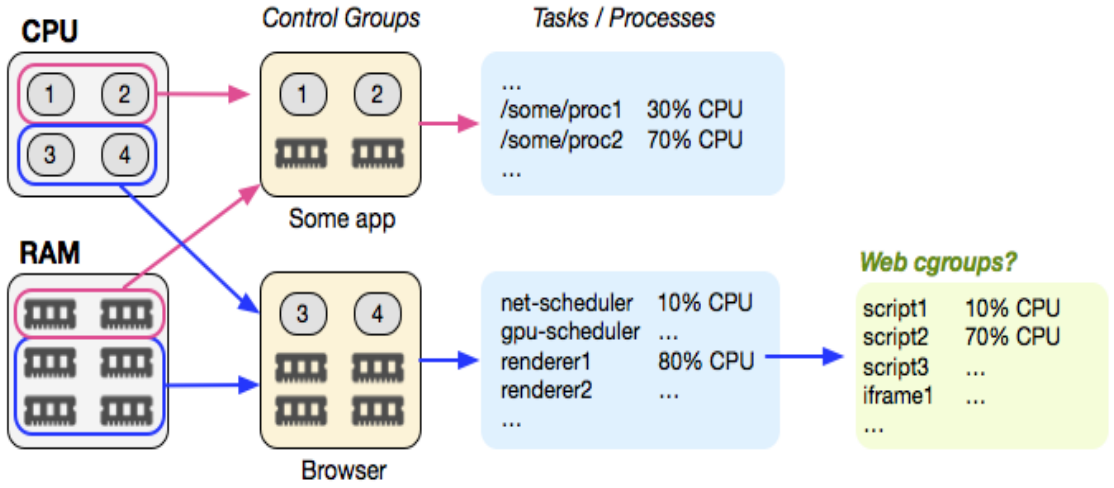
Containers - History

- Linux Containers are powered by two underlying Linux Kernel technologies:
  - cgroups
  - namespaces



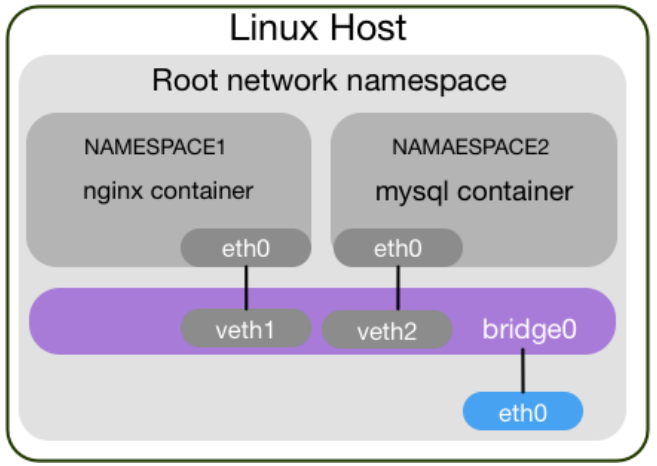
cgroups

- cgroups – Control groups
  - A kernel mechanism for limiting and measuring the total resources used by a group of processes running on a system
  - Processes can be applied with CPU, memory, network or IO quotas
- Cgroup merged into Linux 2.6.24
- Cgroups provides:
  - **Resource limiting**
  - **Prioritization**
  - **Accounting**
  - **Control**



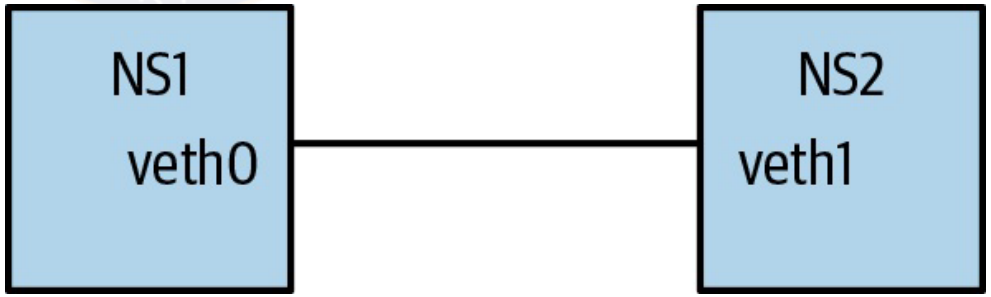
Namespaces

- Namespaces - kernel mechanism for limiting the visibility that a group of processes has of the rest of a system
- *Namespaces* are a Linux kernel virtualization construct akin to network and server virtualization.
- Namespaces virtualize specific resources managed by the kernel, thereby allowing multiple isolated instances of a virtual resource.
- A process is associated with one such virtual instance of the resource.
- Multiple processes can belong to a common virtual instance of the resource. From the processes' perspective, they appear to fully own the resource.
- Limit visibility
  - Certain process trees
  - Network interfaces
  - User IDs
  - Filesystem mounts
- Namespaces are fully usable as a virtualization construct since kernel version 3.8.



Network Namespaces (aka netns)

- The concept of a *network namespace* (aka *netns*) is akin to network virtualization, but a more heavyweight virtualization construct, because it encompasses the entire network stack all the way up to and including the transport layer of the network stack.
- You can have two separate containers each running a server serving port 80 traffic, for example.
- The common network virtualization constructs virtualize only a single layer, (L2) or Layer 3 (L3). Multiple such constructs need to be deployed concurrently to provide virtualization of multiple layers.
- An interface in one netns must use an external connection to send or receive a packet from an interface in a different netns on the same physical system.
- Linux provides a virtual interface construct called *veth*, for virtual Ethernet, to allow communication outside of a netns



Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Agenda

- **Docker Overview**
  - ✓Docker Platform
  - ✓Docker Architecture
  - ✓Example for running a Docker container
  - ✓Container Networking
    - ✓Bridge
    - ✓Host
    - ✓Overlay
    - ✓none

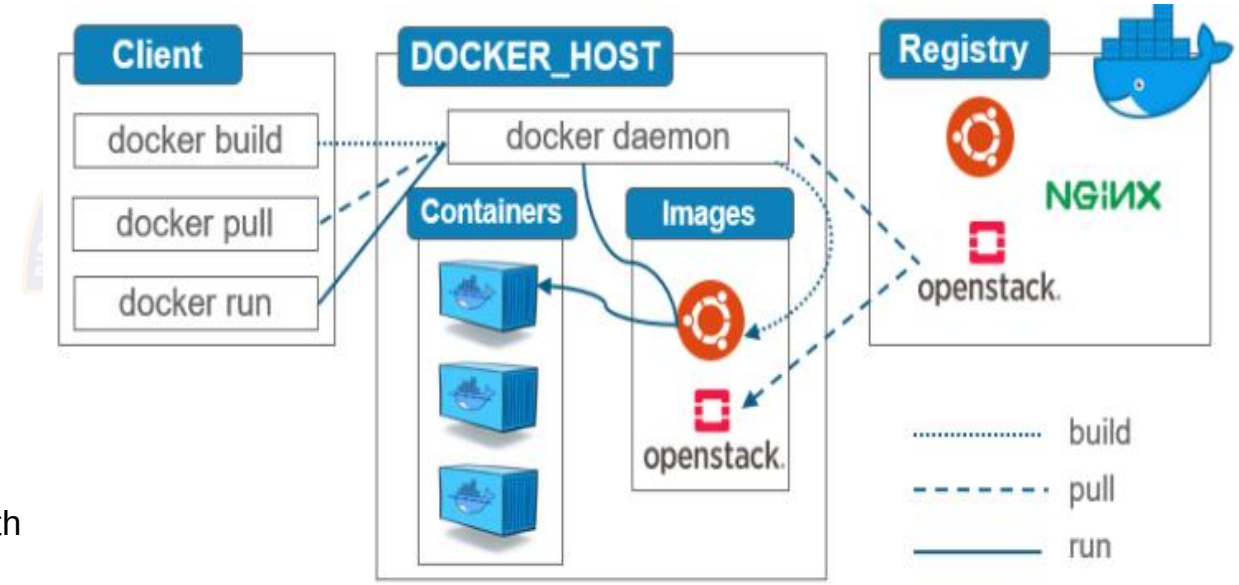
The Docker platform

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Develop application using containers.
  - Distributing and test application using containers.
  - Deploy application into production environment, as a container or an orchestrated service.

  .
- Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

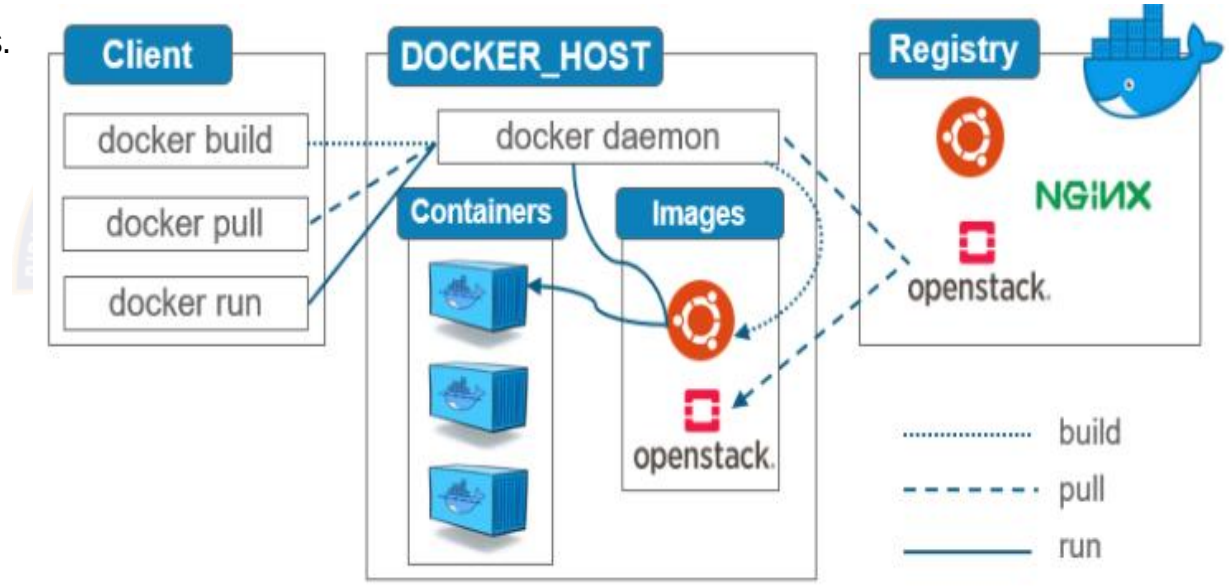
Docker architecture

- Docker uses a client-server architecture.
- The Docker daemon
  - The Docker daemon (dockerd) listens for Docker API requests
  - Manages Docker objects such as images, containers, networks, and volumes.
  - Builds, runs, and distributes containers
- The Docker client
  - The Docker *client* talks to the Docker *daemon*
  - The Docker client and daemon *can* run on the same system
  - The Docker client can communicate with more than one daemon..
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



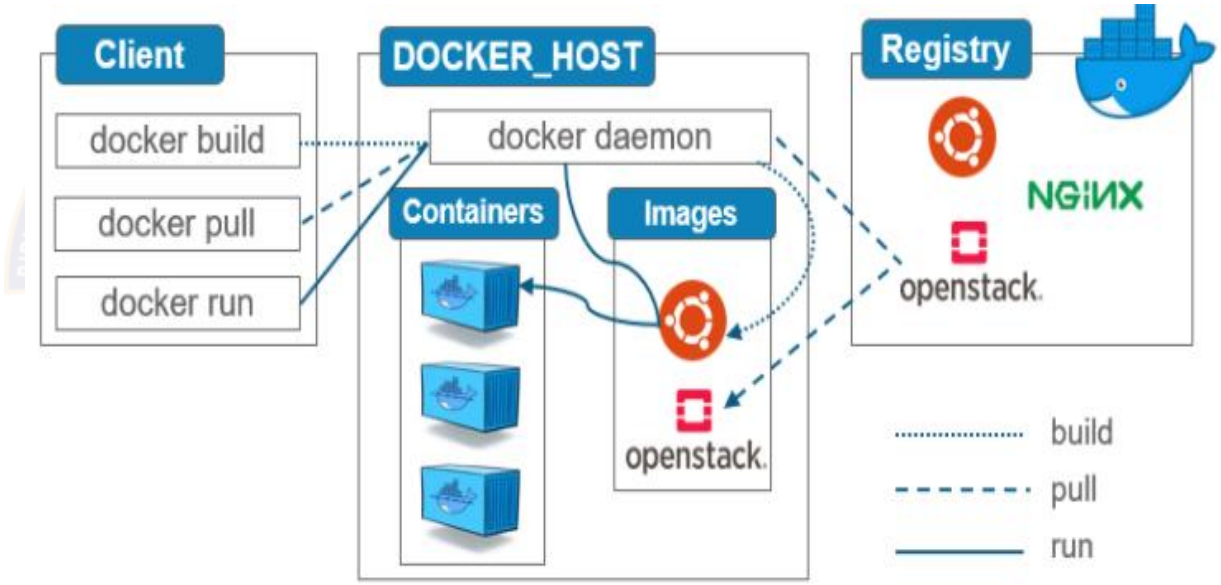
Docker architecture

- Docker registries
  - A Docker *registry* stores Docker images.
  - Docker Hub is a public registry that anyone can use.
  - Docker is configured to look for images on Docker Hub by default

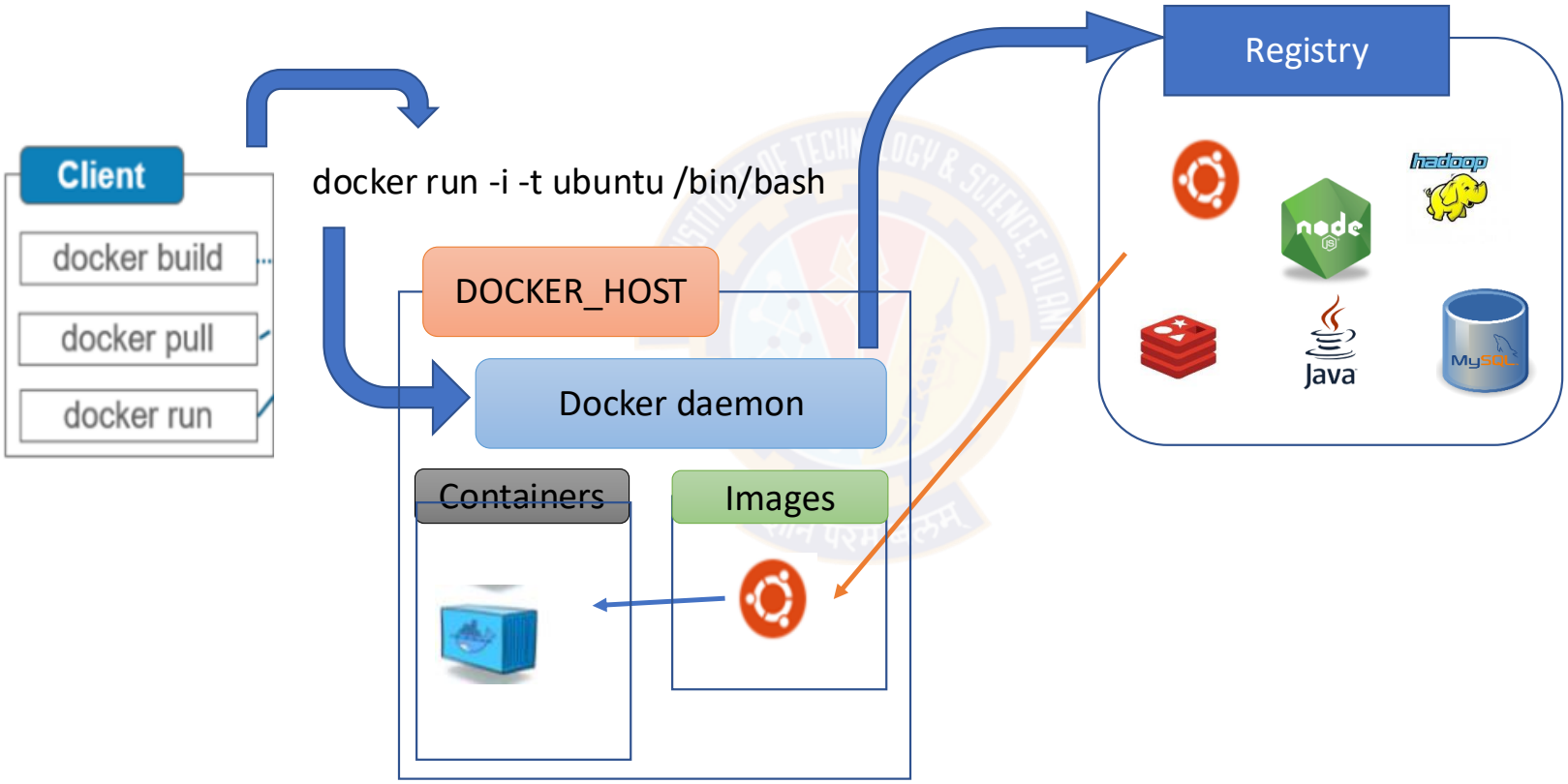


Docker architecture

- Docker objects
  - IMAGES: An *image* is a read-only template with instructions for creating a Docker container.
  - CONTAINERS: A container is a runnable instance of an image



Example for running a Docker container



Thank You!