UNIVERSITY of York
Europe Campus
CITY College

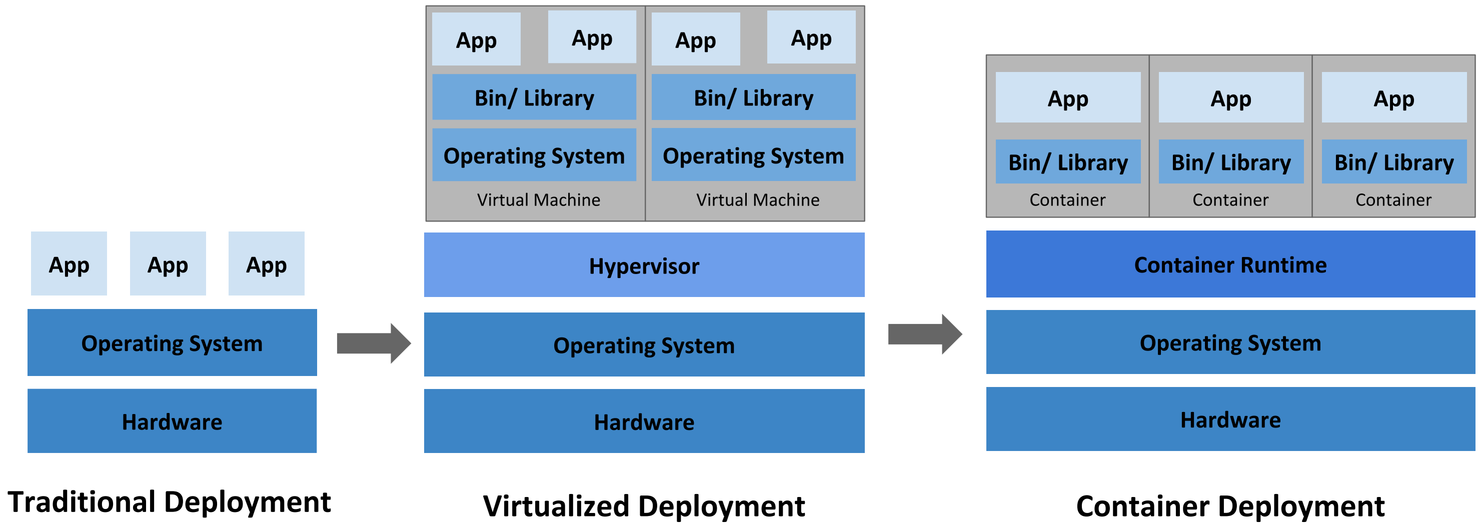
docker & Linux Containers

CCS3341 Cloud Computing

Dr S. Veloudis

Overview

- Containerized deployment



- No isolation between apps
- No resource limiting

- Isolation & resource limiting
- Heavyweight VMs

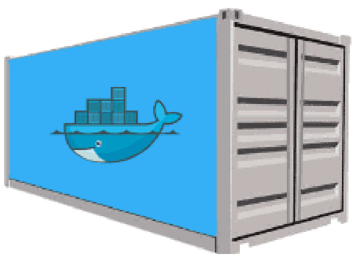
- OS sharing → relaxed isolation properties
- Lightweight containers

OS-level isolation through Linux Containers (LXC)

Docker Containers

Docker is the most widely used containerization platform!

Docker containers



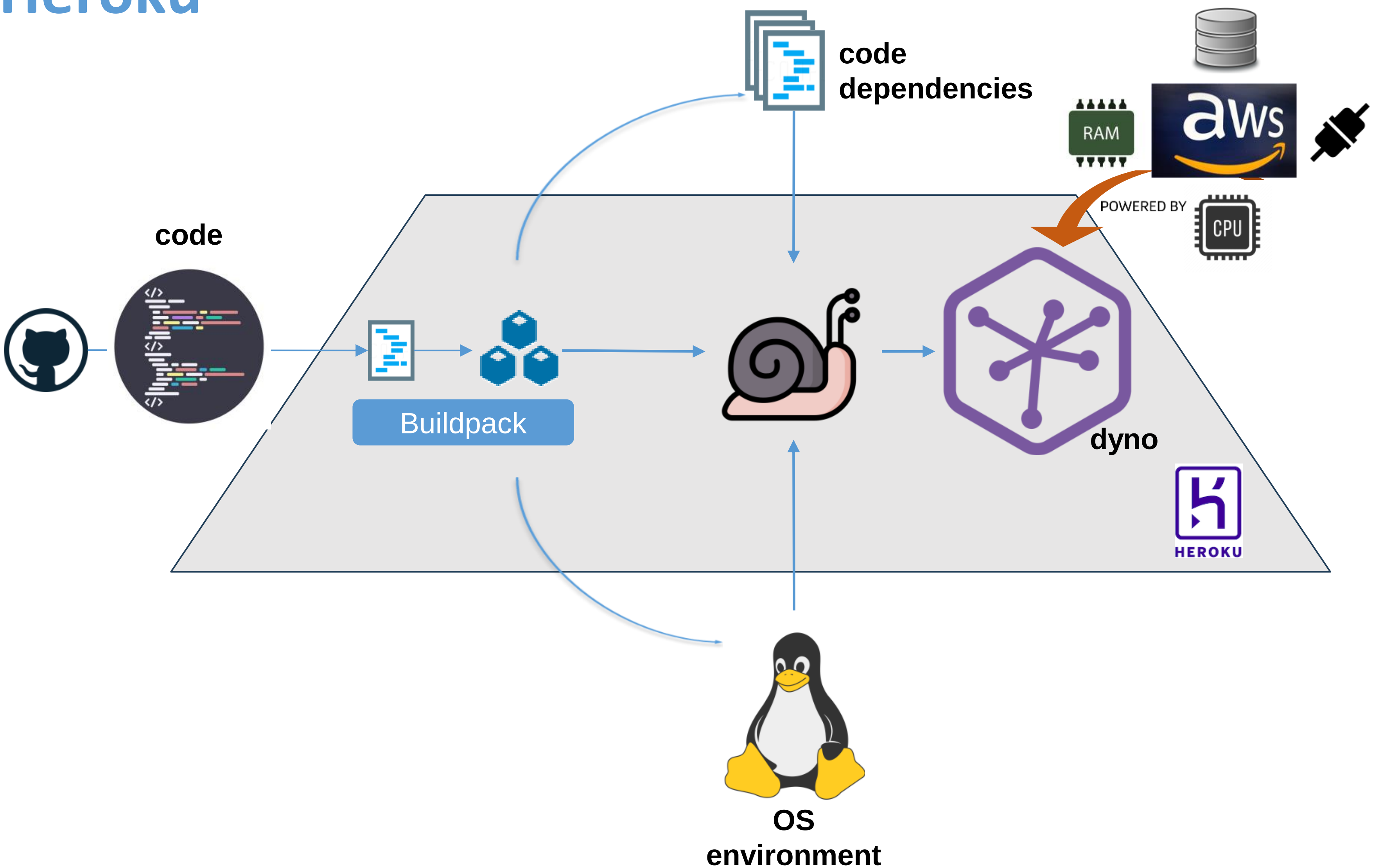
Lightweight, standalone, and executable software packages that include everything needed to run an application, including the code, runtime, libraries, system tools, and settings

But does this ring any bells?

Heroku dynos serve a similar purpose!

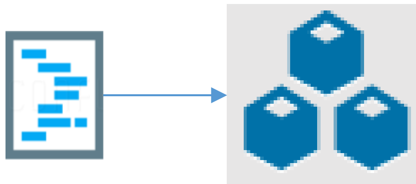
But what are the differences?

Heroku



code dependencies
RAM
aws
POWERED BY
CPU
code
Buildpack
dyno
HEROKU
OS
environment

Heroku: Buildpacks



Buildpack

- **Building** an app
  - Linking
    - Combining multiple object files and libraries into a single executable
  - Compiling
    - Transforming a program P written in a language L to a semantically-equivalent program P' expressed in some object code language L'

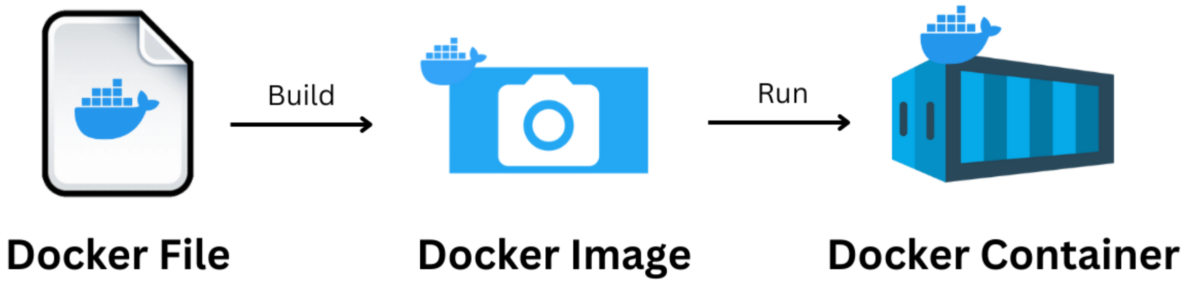
- **Runtime** configuration
  - Exception handling
  - Implements **upper layers** of TCP/IP stack
  - Lower stack layers are handled by the **Heroku router**

Heroku Router

Ingress traffic
- Dynamic request routing to appropriate dyno
- Load balancing
- SSL/TLS termination (decrypting incoming traffic)

Egress traffic
- Routing based on IP tables
- Firewall

Docker Containers



Dockerfile

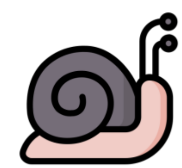
≈ **≈ Buildpack**

A script containing instructions for building a Docker image

```
# Comment
INSTRUCTION arguments
```

Docker image

≈



Lightweight, standalone, and executable package that contains all the necessary components and dependencies to run a piece of software, including the application code, runtime, and libraries

Docker Containers

Docker image

- Its main difference with a slug is that a slug is optimised to run on the Heroku platform
  - Slugs are tightly integrated with the Heroku platform and can only run on the Heroku platform
  - Containers are more generic and **portable**

Docker containers encapsulate the entire TCP/IP stack (Application, Transport, Network, and Data Link layers). In contrast, slugs only cover the Application layer.

- Docker containers have their own **isolated** network stacks
- In Heroku, network isolation across dynos is achieved by the Heroku router

Custom ports

- Docker enables application port numbers to be explicitly specified and exposed via host ports
- In Heroku ports are assigned by the Heroku router (accessible from the `PORT` environment variable)

Routing

- Docker containers maintain their own iptables and therefore their routing rules
- In Heroku routing is handled by the Heroku router for all dynos

Docker Containers

- A Docker image comprises read-only layers each one of which is created by an instruction in the Dockerfile
- Layers are stacked and each layer is a delta of the changes from the previous layer

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

<u>Note</u>: Instructions are given in a **Domain Specific Language**

Docker daemon (dockerd)

- Engine that creates images by executing Dockerfile instructions

`docker build` command

- Engine that creates (and runs) containers from Docker images

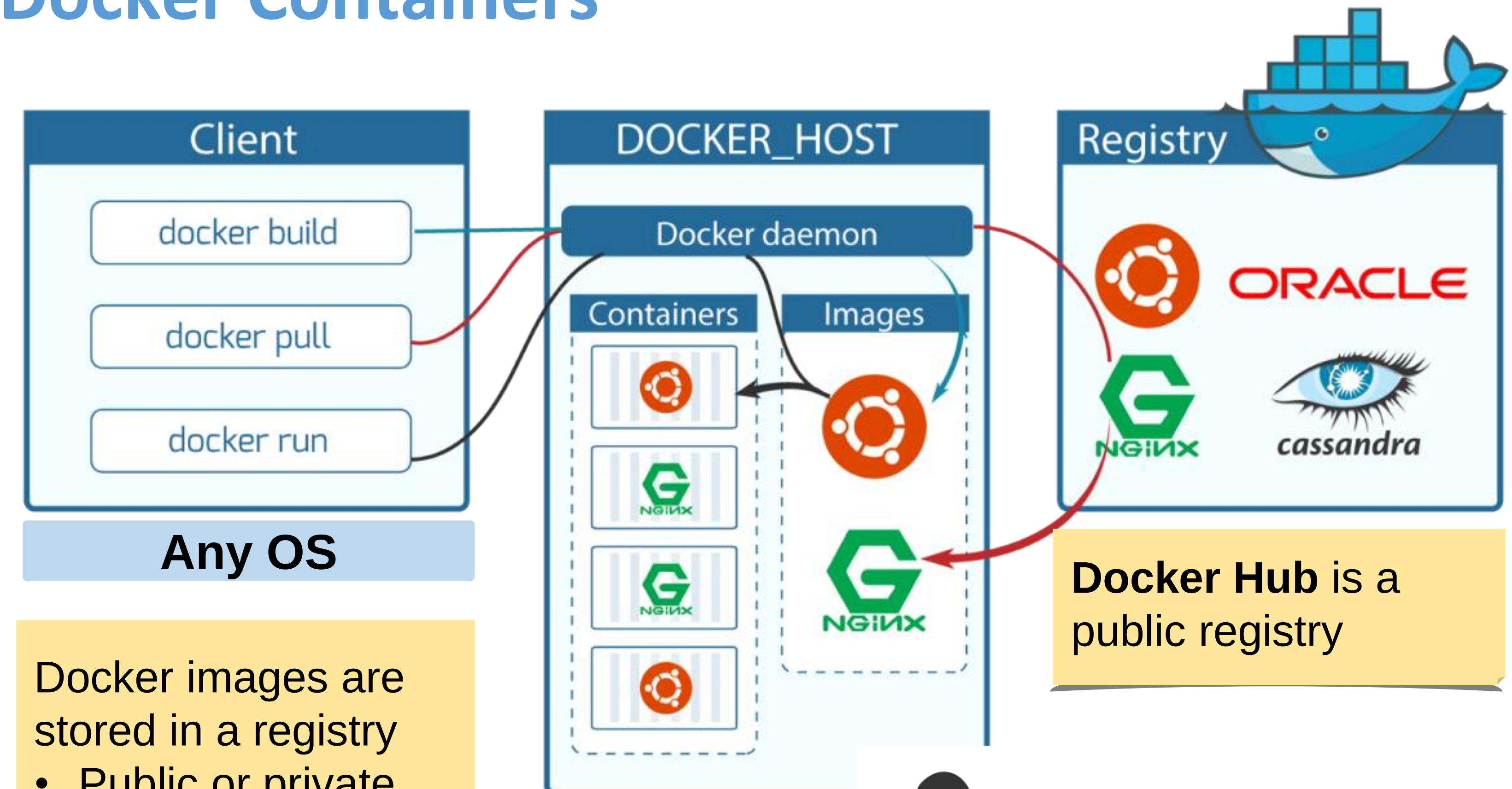
`docker run` command

- File system setup
- Runtime isolation
- Start application

See later!

University of York Europe Campus
CITY College

Docker Containers

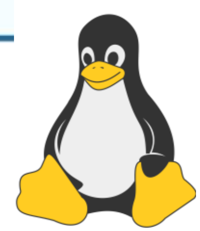


Any OS

Docker images are stored in a registry

- Public or private
- Versioning

Exposes a **RESTful API**



Docker Hub is a public registry

LXC (Linux Containers)

- Docker is based on **LXC**
- LXC uses Linux's **cgroups** (control groups)
  - A Linux kernel feature that limits and isolates resource usage (CPU, memory, disk I/O, network access) for one or more processes
- A cgroup is a collection of processes that are bound by the same characteristics
- cgroups provide isolation through the use of **namespaces**

LXC allows creation and running of multiple virtual environments (VEs) at the OS level

- Resource limiting
  cgroups can be set not to exceed a preconfigured memory limit
- Prioritisation
  some groups may get larger share of CPU utilization
- Accounting
  measures a group's resource usage (e.g. for billing or benchmarking purposes)
- Control
  groups may be paused (frozen), checkpointed, and restarted

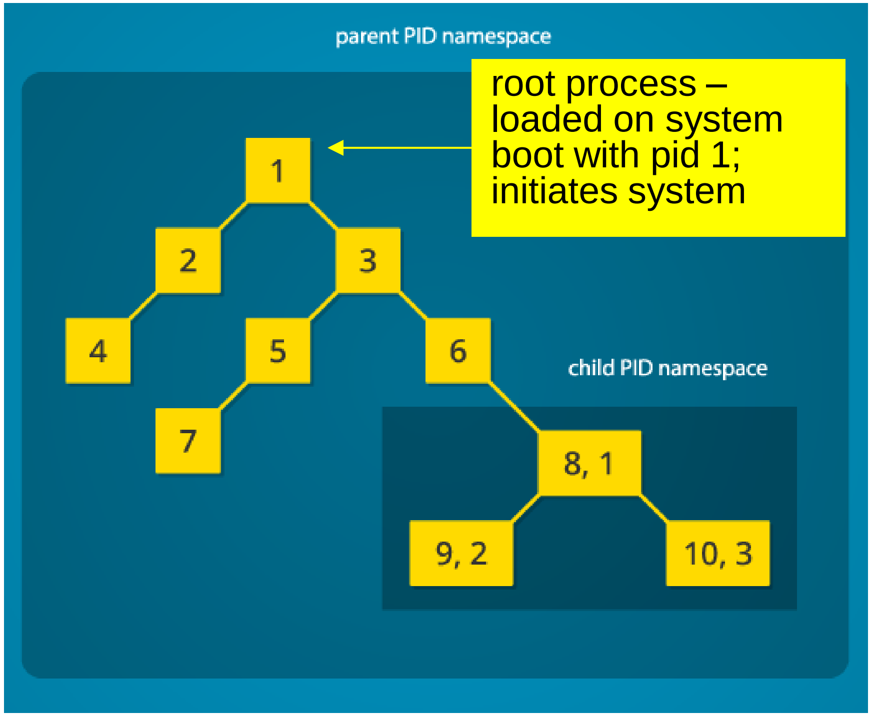
Namespaces

- Each process is associated with a set of namespaces
- Linux offers different namespace 'subsystems'

```
pid
mnt
net
ipc
user
uts
```

- A process can access (read/write) a resource if it is associated with the same namespace as the resource

pid namespace



- Processes in child pid namespace are not aware of processes in parent namespace - **ISOLATION**
- Processes in parent namespace 'see' processes in child namespace (but with original pids)

University of York Europe Campus CITY College

Namespaces

pid namespace

<u>NOTE</u>:

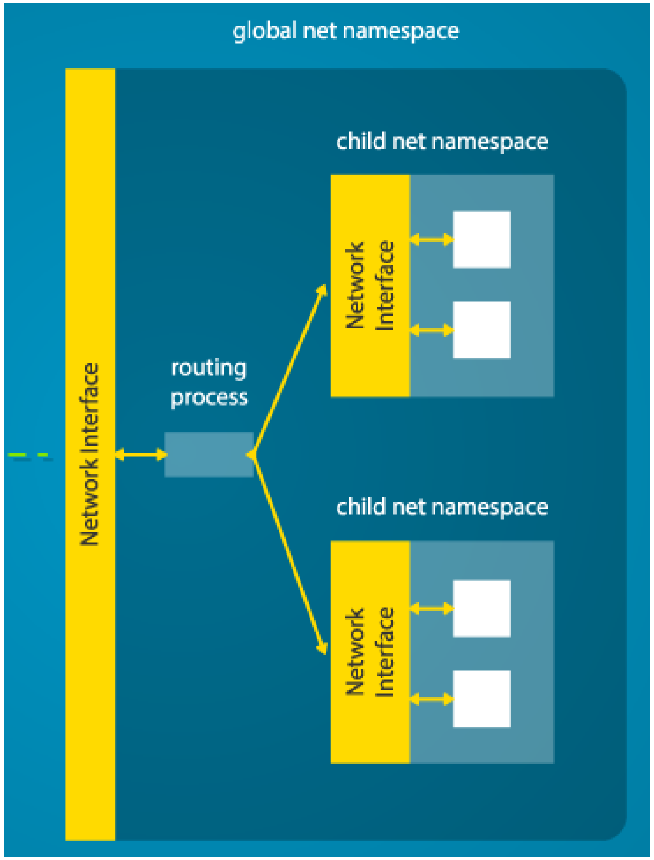
- To create a new pid namespace, one must call the clone() system call with a special flag CLONE_NEWPID. This may be done either programmatically, or from the CLI through the **unshare** command
- The **unshare** command runs its argument process in a new namespace with pid 1

A process may have multiple pids – one per each namespace with which it is associated

A process may influence processes in other namespaces through shared resources

net namespace

- Virtualisation of networking interfaces



Namespaces

net namespace

- Each net namespace has its own virtualised set of interfaces
- Each net namespace has its own set of iptables
  - Different message routing and forwarding rules per namespace
  - Different security settings per namespace
- net namespaces are associated with pid namespaces

- Virtualization of network interfaces is implemented using **virtual ethernet devices** (**veth**)
- A veth consists of a pair of virtual network interfaces, one of which is attached to the container's network namespace and the other is attached to the host system's network namespace
- This allows the container to communicate with the outside world through its own virtual network interface, which is isolated from other containers and the host system

Namespaces

net namespace

- The command

`ip netns add <ns_name>`

creates a new namespace

- A net namespace can only be assigned virtual network interfaces e.g.

`ip link net veth1 netns <ns_name>`

To configure different iptables rules for traffic passing through each of the virtual network interfaces (veth1 and veth2) corresponding to separate network namespaces, you would typically perform the following steps:

1. **Identify the Network Namespaces:** Determine which network namespaces veth1 and veth2 are associated with. You can do this by inspecting the output of `ip netns list` or by checking the `netns` symlink in the `/proc/<pid>/ns` directory for the processes associated with each network namespace.
2. **Enter the Network Namespace:** Use the `ip netns exec` command to execute commands within the context of each network namespace. For example:

```
ip netns exec <namespace_name> <command>
```

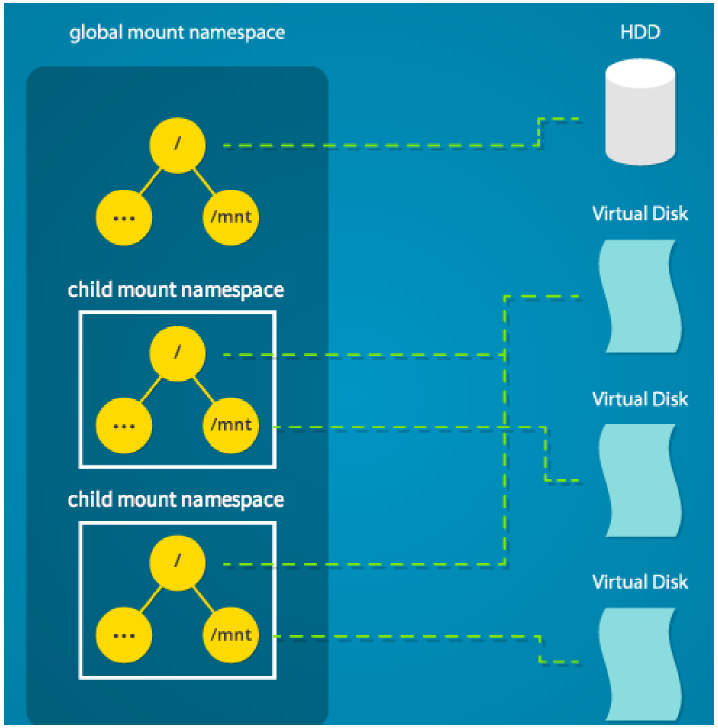
3. **Configure iptables Rules:** Within the context of each network namespace, configure iptables rules as desired using the `iptables` command. You can add rules to the `INPUT`, `OUTPUT`, and `FORWARD` chains to control traffic for each interface separately. For example:

```
ip netns exec <namespace_name> iptables -A INPUT -i veth1 -j ACCEPT
ip netns exec <namespace_name> iptables -A INPUT -i veth2 -j DROP
```

Namespaces

mnt namespace

- Linux maintains a data structure for each mountpoint in the system
  - Includes data on disk partitions mounted
- This data structure is cloned per mnt namespace
  - This way, processes under different namespaces can change the mountpoints without affecting each other



A new mnt namespace is created programmatically through the `clone()` system call with the flag `CLONE_NEWNS`, or from the CLI via the `unshare` command