Pivotal

Application Container Security in PCF

Madhav Sathe
Platform Architect
Oct 2018

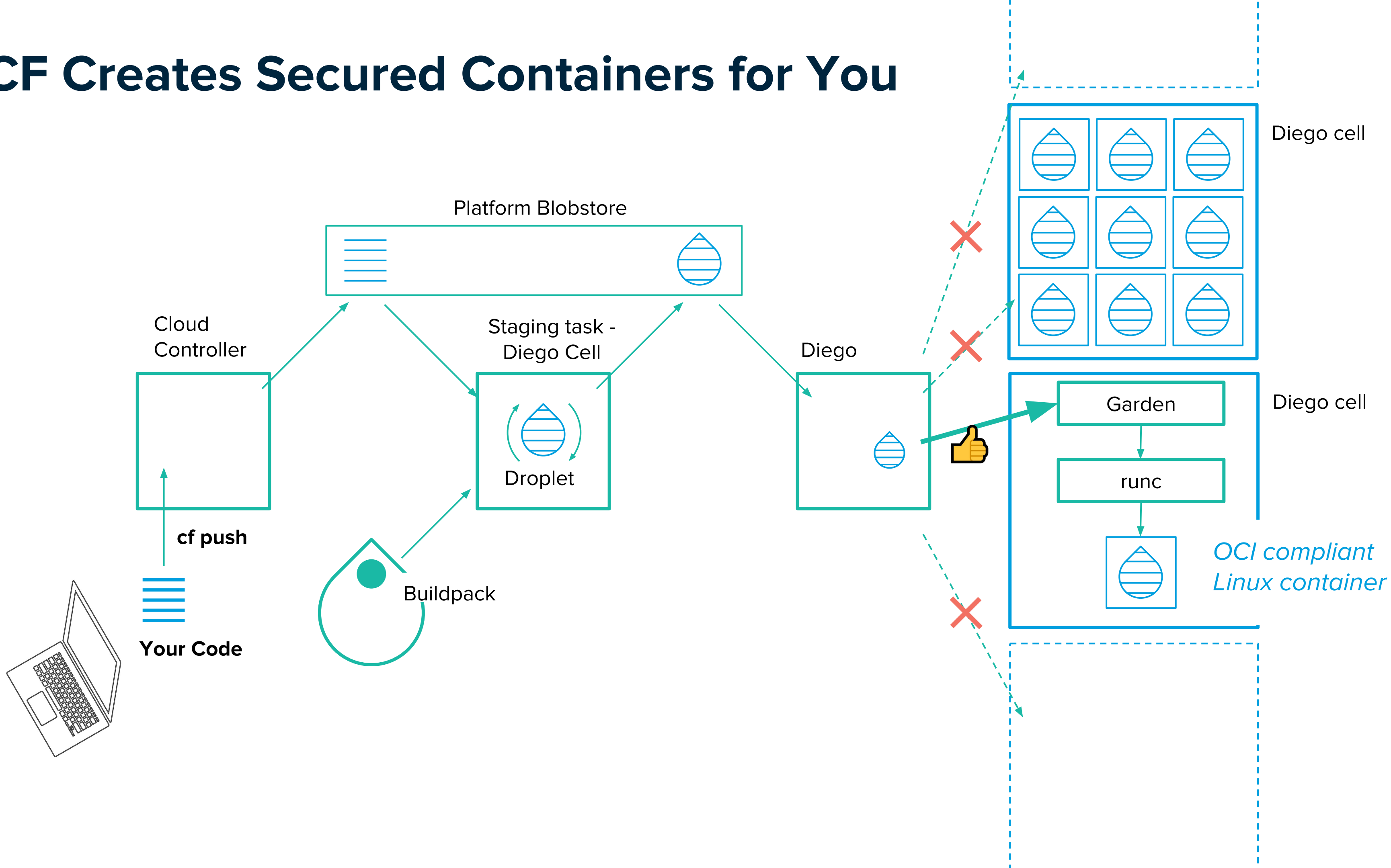
Container?

Containers

Bedrock of cloud applications

Every application instance runs in its own Container

PCF Creates Secured Containers for You



OCI

Run under the auspices of the Linux Foundation

Open Container Initiative

Image Spec

How image looks on disk

Runtime Spec

Any guess?

But what is Container? And how does it help?

Containers

Bedrock of cloud applications

Linux primitives leveraged to provide higher density, speed & agility

Containers

Bedrock of cloud applications

Linux primitives leveraged to provide higher density, speed & agility

Namespaces - what a process sees

Control Groups (cgroups) - what a process can do

Containers

Container has everything an application needs to run

Advantage of Containers
shared OS kernel and resources

Containers

Advantage of Containers
shared OS kernel and resources

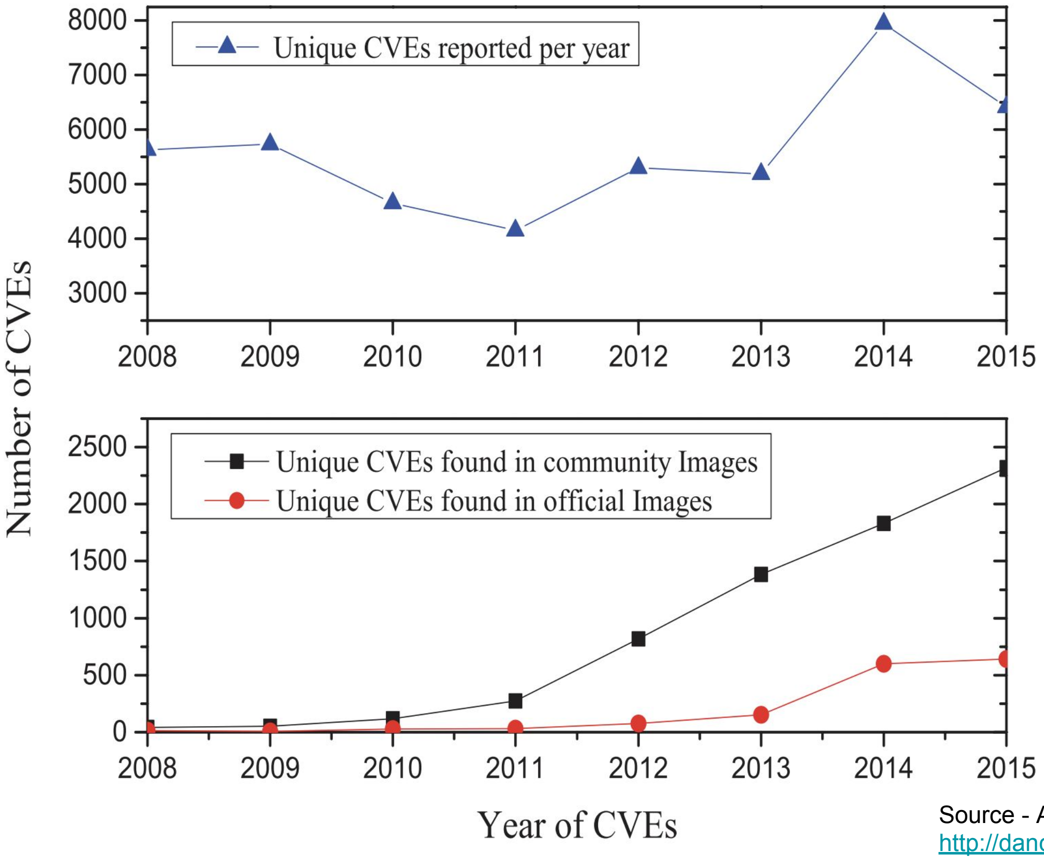
Risk of Containers
shared OS kernel and resources

Risks

- **Unpatched vulnerabilities**
  - Application code
  - Kernel
  - File system
  - Application runtime
  - Dependencies
  - Container runtime
  - Container scheduler
  - Container base images
  - Container registry
- Unnecessary executables
- Access to unnecessary resources
- Snowflake configurations
- Lack of audit logs
- **Lack of communication across teams**

Unpatched CVEs

Most common & critical risk



Source - A Study of Security Vulnerabilities on Docker Hub
http://dance.csc.ncsu.edu/papers/codaspy17.pdf

Threats

- ➢ Container breakout
- ➢ Privilege escalation
- ➢ Malware
- ➢ Advanced persistent threats
- ➢ Denial of service to other containers
- ➢ Denial of service to host
- ➢ Spoofing, man-in-the-middle attacks
- ➢ Kernel modification
- ➢ File system modification
- ➢ Lack of accountability
- ➢ …

Most organizations secure the perimeter
But what about threats that are already inside?

Security 101

Pivotal

Namespaces

What a process can see

- Provides isolation for processes running on the same host, sharing the same kernel
- Global resources are isolated to give process a restricted ‘view’ of the system
- First line of defense from one container disrupting another container or host for that matter

Namespaces

Isolation

Namespace	Isolates
IPC	System V IPC, POSIX message queues
Network	Network devices, stacks, ports, etc.
Mount	Mount points
PID	Process IDs
User	User and group IDs
UTS	Hostname and NIS domain name

Namespaces

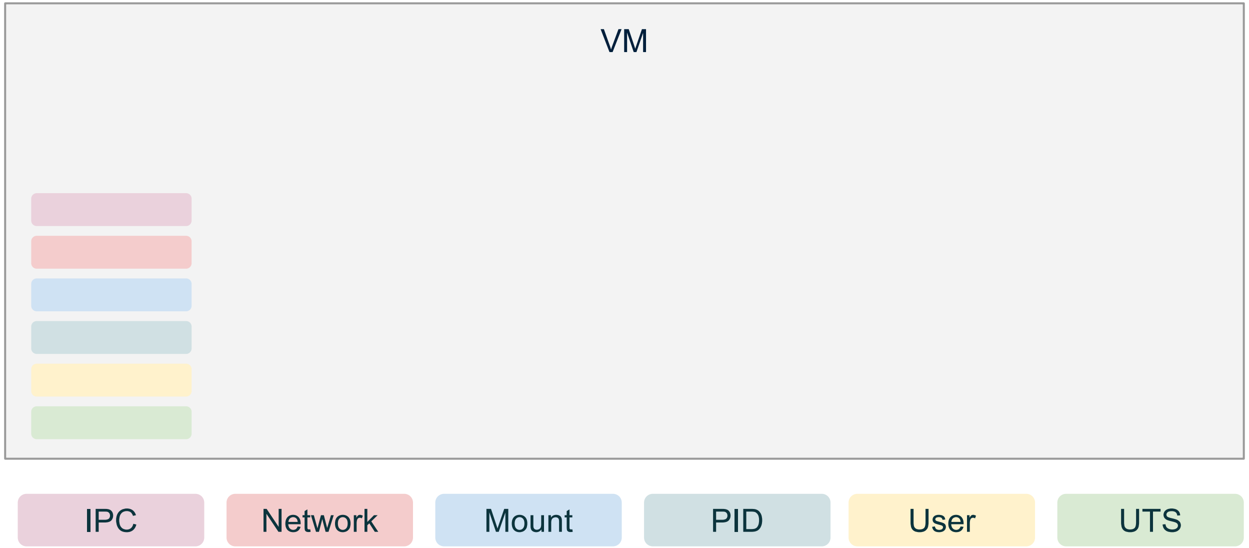
Foundation of secured containers

However, there are still some resources that are not namespace-aware e.g. devices

Namespaces

Foundation of secured containers

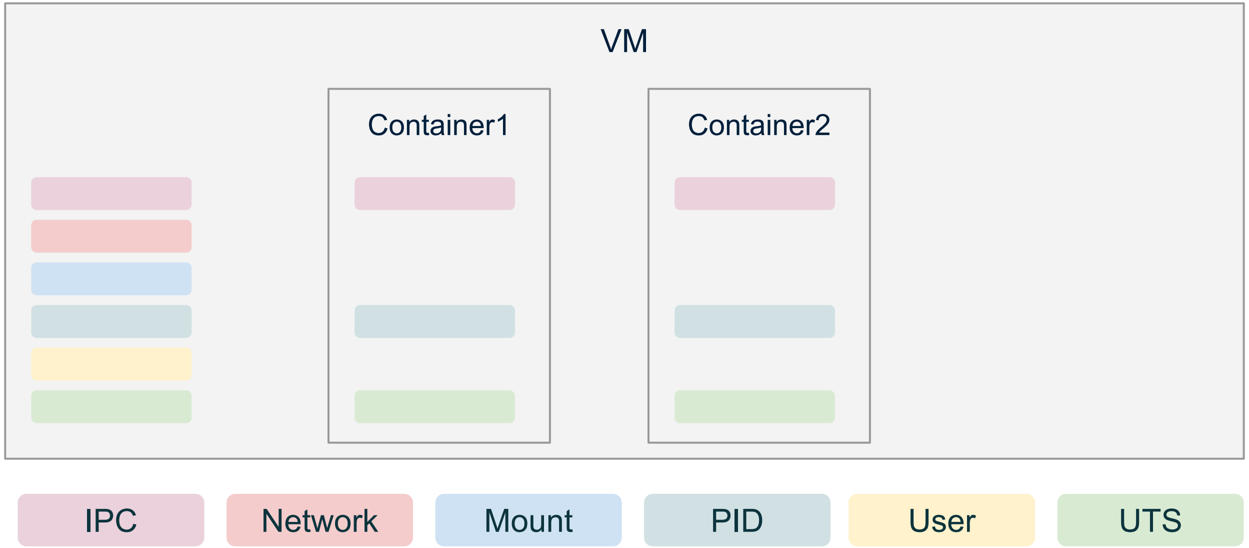
Host/VM has its own namespaces which represent the global resources



Namespaces

Foundation of secured containers

A new container will typically get its own [some or all] namespaces



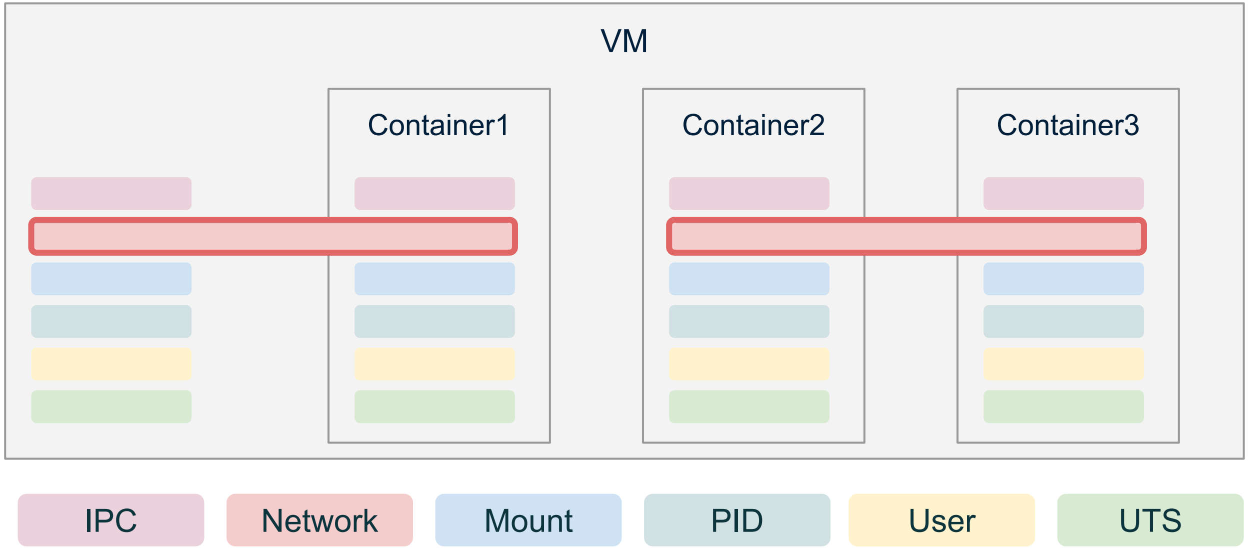
Namespaces

Foundation of secured containers

Container can share a namespace, with the host or even with other containers.

For instance, if a container shares Network namespace it will get same same network interfaces, IP, etc

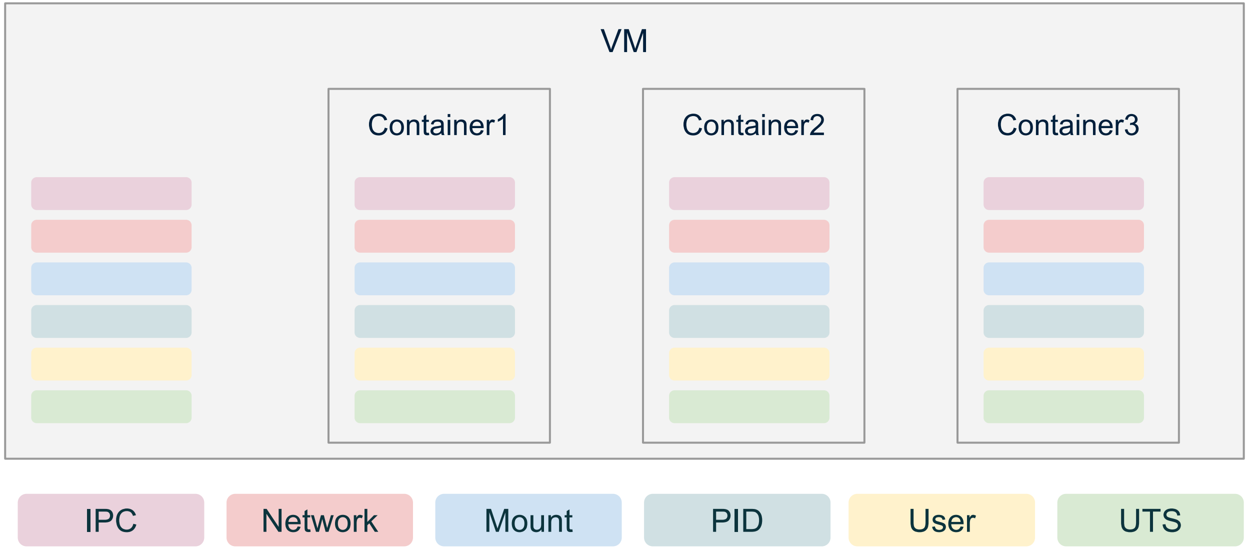
E.g. in K8s containers in the same POD share network namespace



Namespaces

PCF guarantees complete isolation

PAS guarantees complete isolation of application instances by leveraging all Linux namespaces for each container ➔ container can't see anything on other container, it can't communicate with other container ➔ secured multi-tenancy



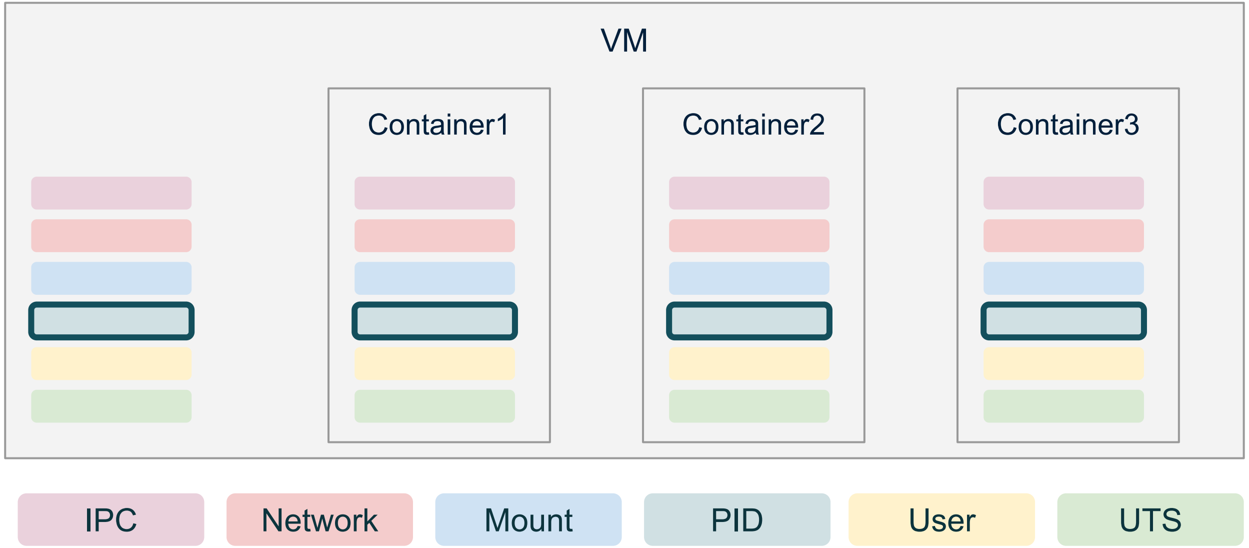
Process Isolation

Foundation of secured containers

PID Namespace

PID namespace provides isolation of PIDs

PID namespace controls ability of processes to see and interact with each other



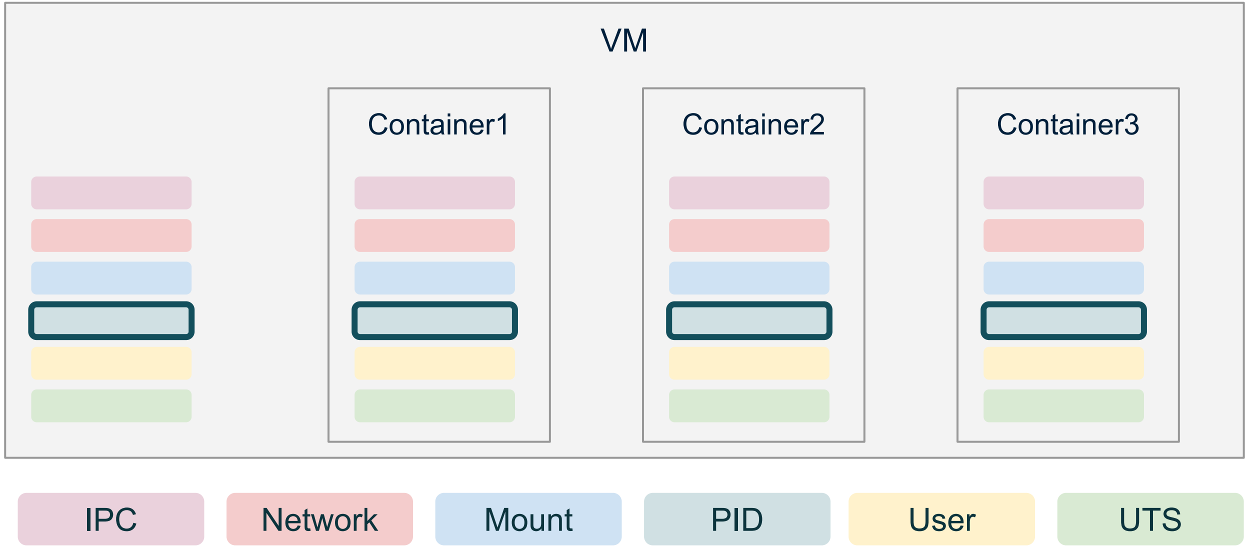
Process Isolation

Foundation of secured containers

PID Namespace

It also provides PID virtualization i.e. processes in different PID namespaces can have same PID in their respective containers but they are mapped a different PID on host

First process on in container carries PID 1



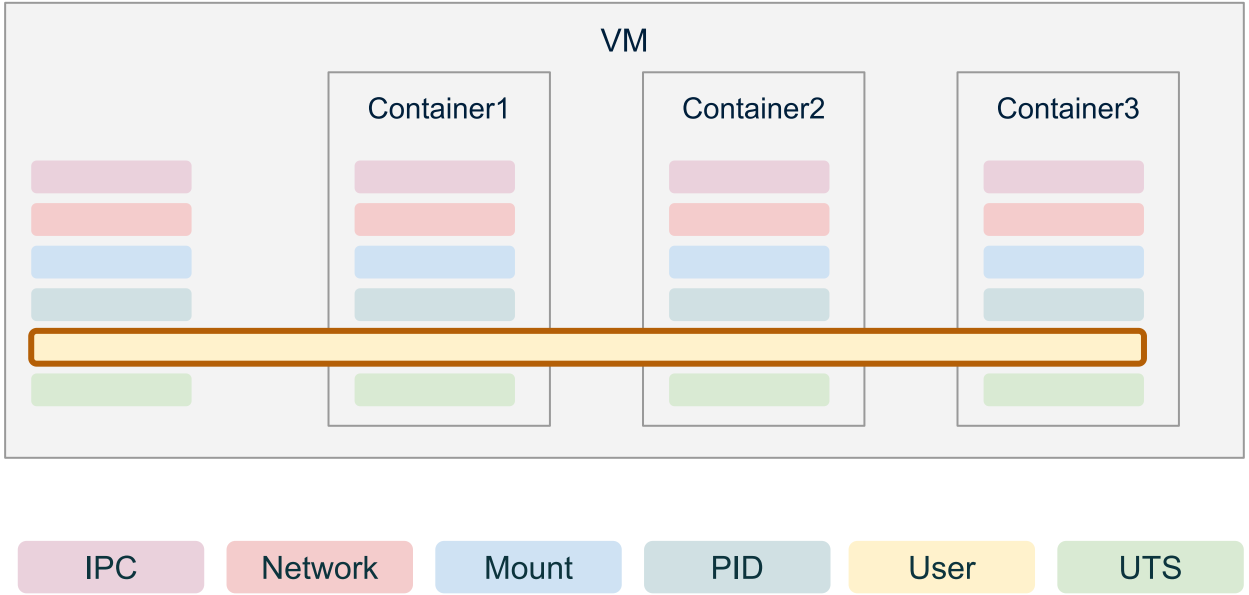
User Isolation

Foundation of secured containers

User Namespace

Relatively new addition to Namespaces

Earlier (when there was no User namespace) a Root user on Container maps to Root on Host. This is called a privileged container



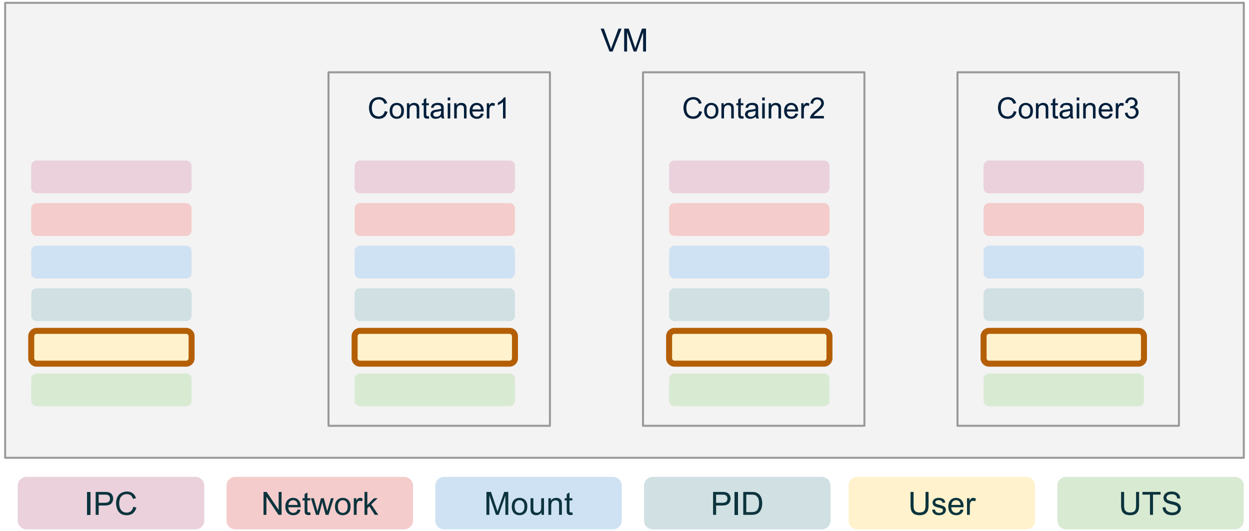
User Isolation

Foundation of secured containers

User Namespace

User namespace enables unprivileged containers

CF maps UID/GID 0 (root) inside the container user namespace to a different UID/GID on the host ➔ prevent an app from inheriting UID/GID 0 on the host if it breaks out of the container ➔ container Root does not grant Host Root permissions



Filesystem Isolation

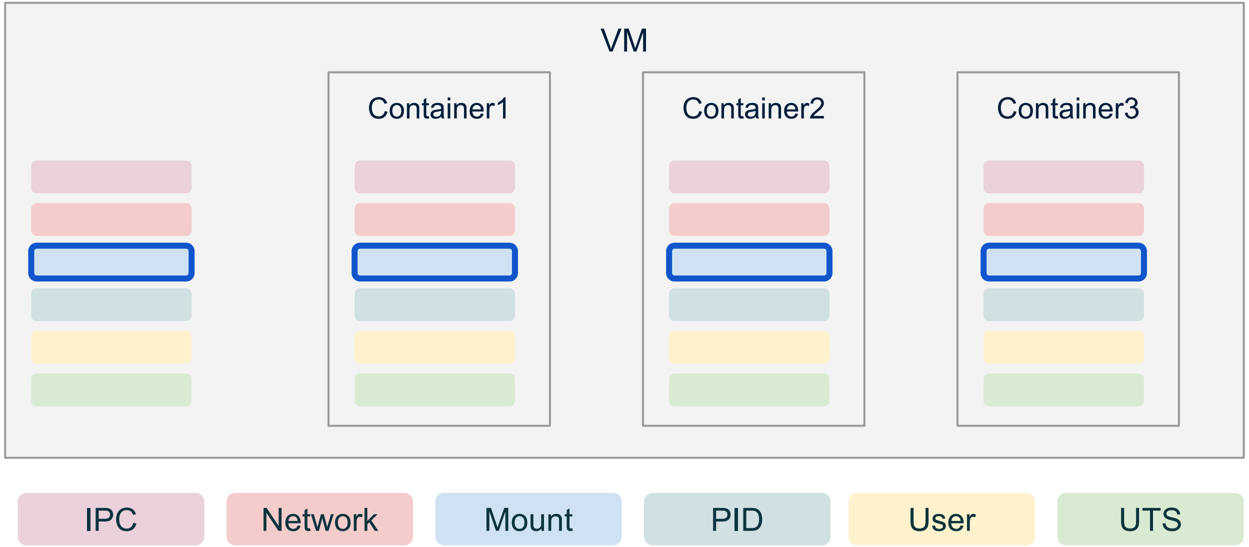
Foundation of secured containers

Mount Namespace

Access to mount points is controlled by Mount namespace

Containers can issue mount/unmount calls on those mount points

However, container inherits the view of filesystem mounts from parent ➔ container can access all parts of the filesystem



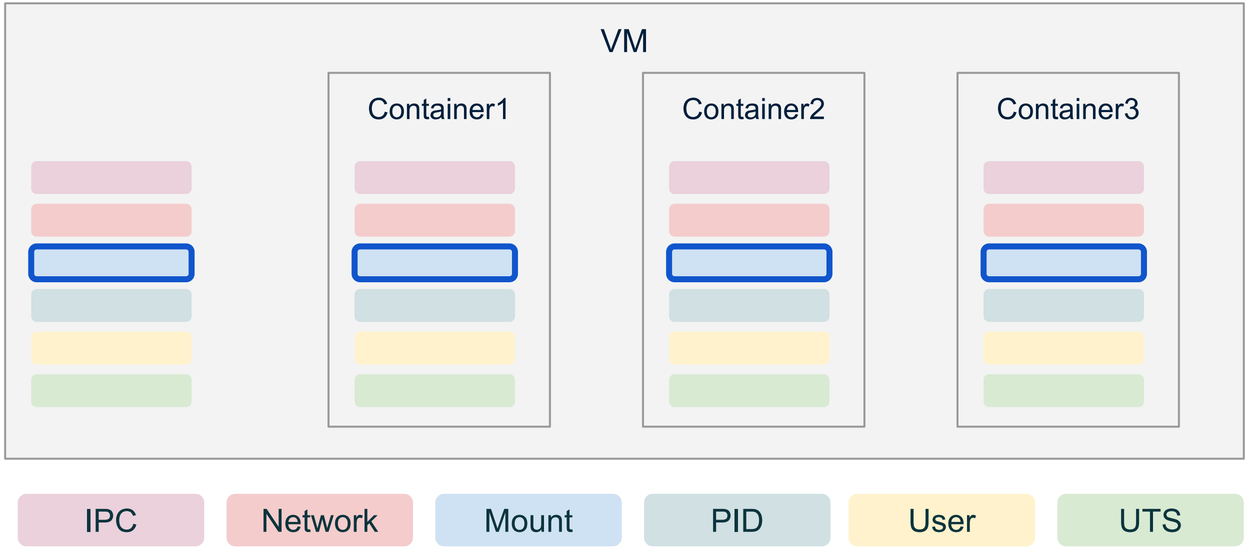
Filesystem Isolation

Foundation of secured containers

Filesystem access containment is provided by using **chroot** ➔ changes the root directory of container

However, privileged process (with CAP_SYS_CHROOT) can escape chroot (jail)

pivot_root call is used to actually change the root file system for each container



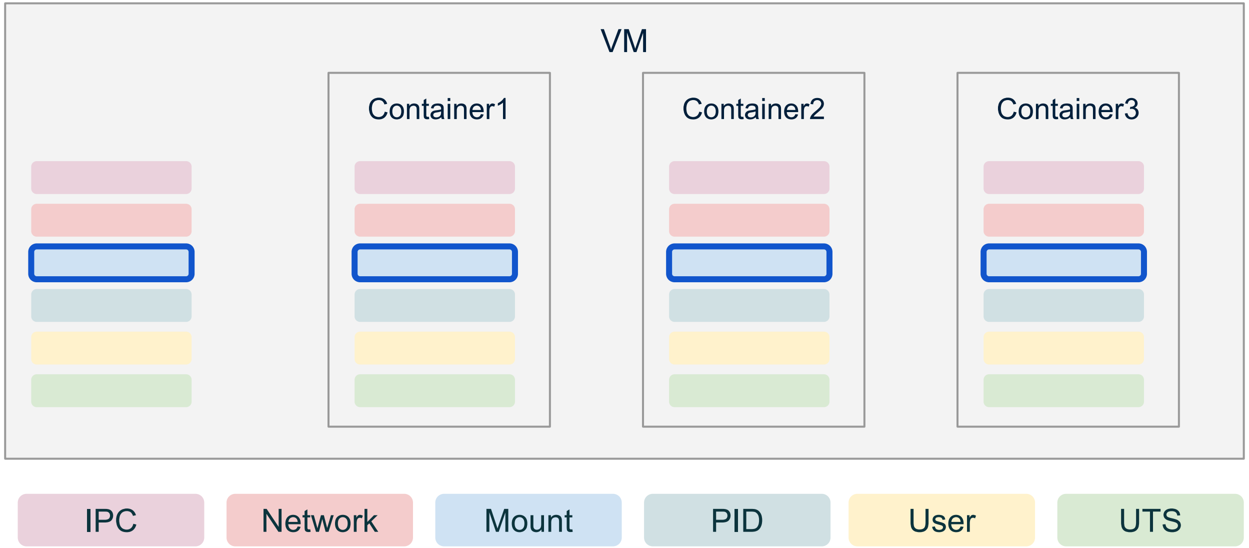
Filesystem Isolation

Foundation of secured containers

Mount Namespace

pivot_root call actually changes in **'/' mount point**

Without using Mount namespace this would mean causing change to '/' mount point of the host as well



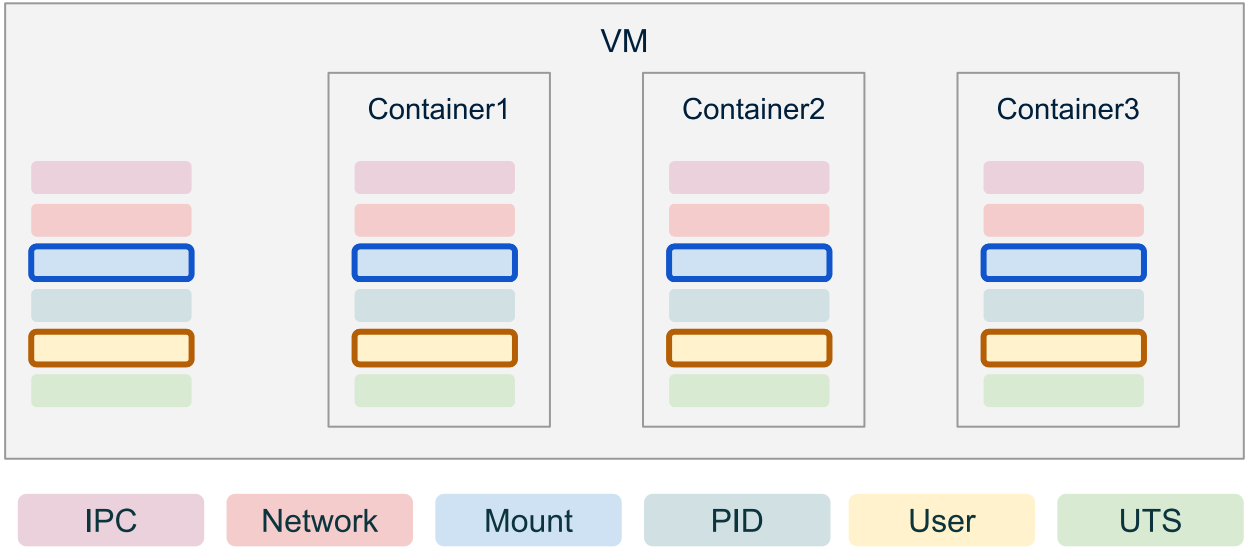
Filesystem Isolation

Foundation of secured containers

Mount Namespace

Mount namespace + pivot_root provide solid file system isolation

This is further augmented by restricting user scope using User namespace



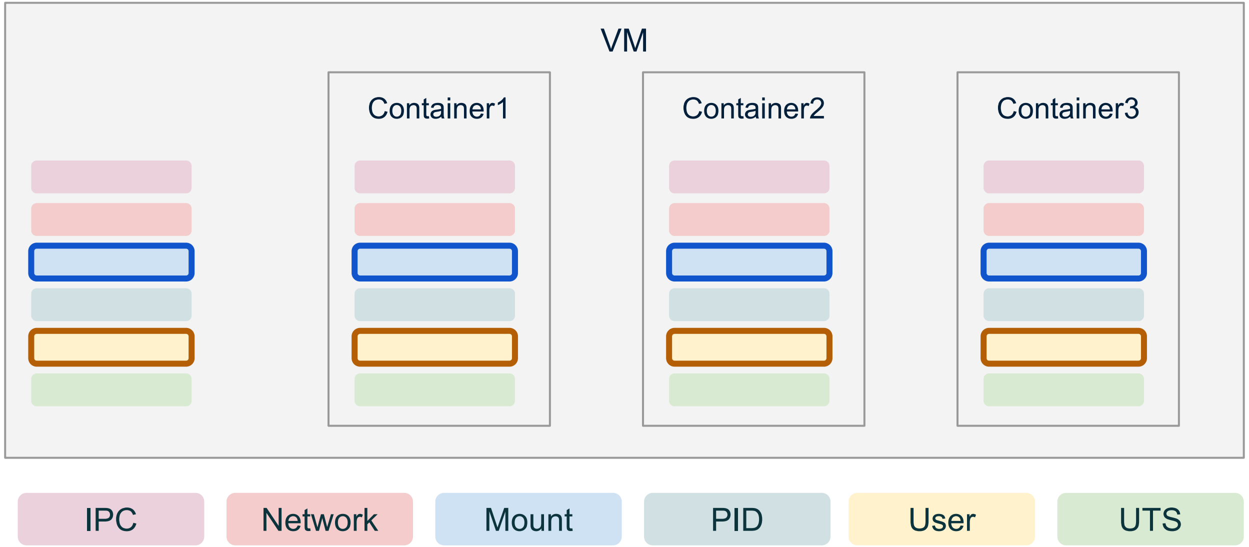
Vetted Filesystems

Foundation of secured containers

Enables containers to have own root file system that is separate from that of the host

This gives a powerful choice for PCF to pick the most suitable file system for each layer

PCF uses hardened Ubuntu stemcell for VM and hardened cflinuxfs3 RootFS for container
More on OS hardening in Part2..



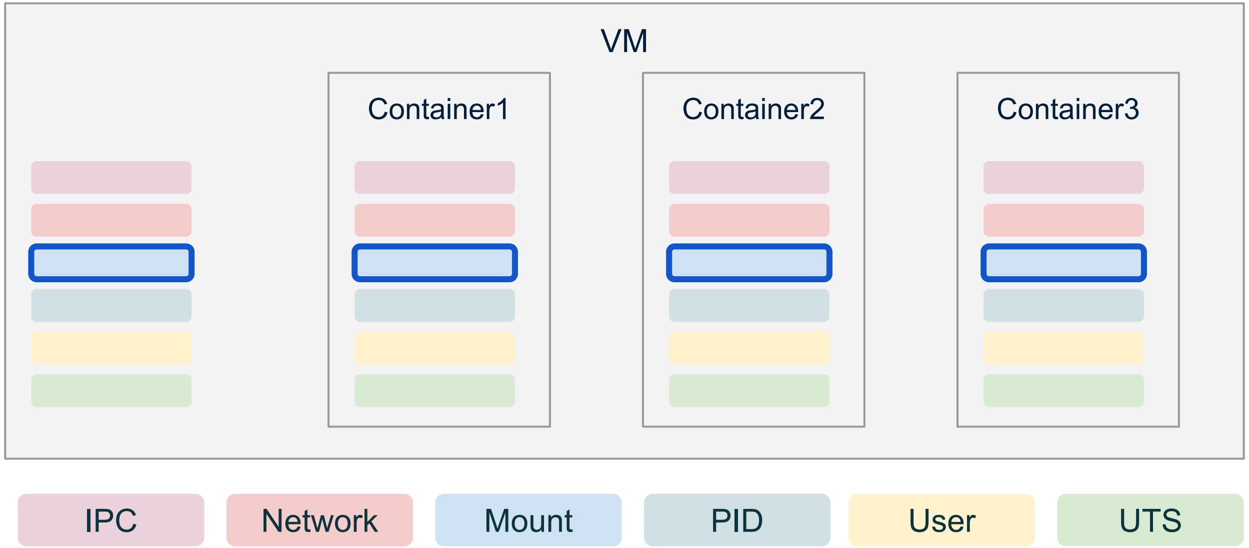
Vetted Filesystems

Foundation of secured containers

PCF uses combination of OverlayFS and XFS as a filesystem for containers

The read-only layer in all containers is the RootFS

The application binaries are in the read-write layer which is very small layer of the file system



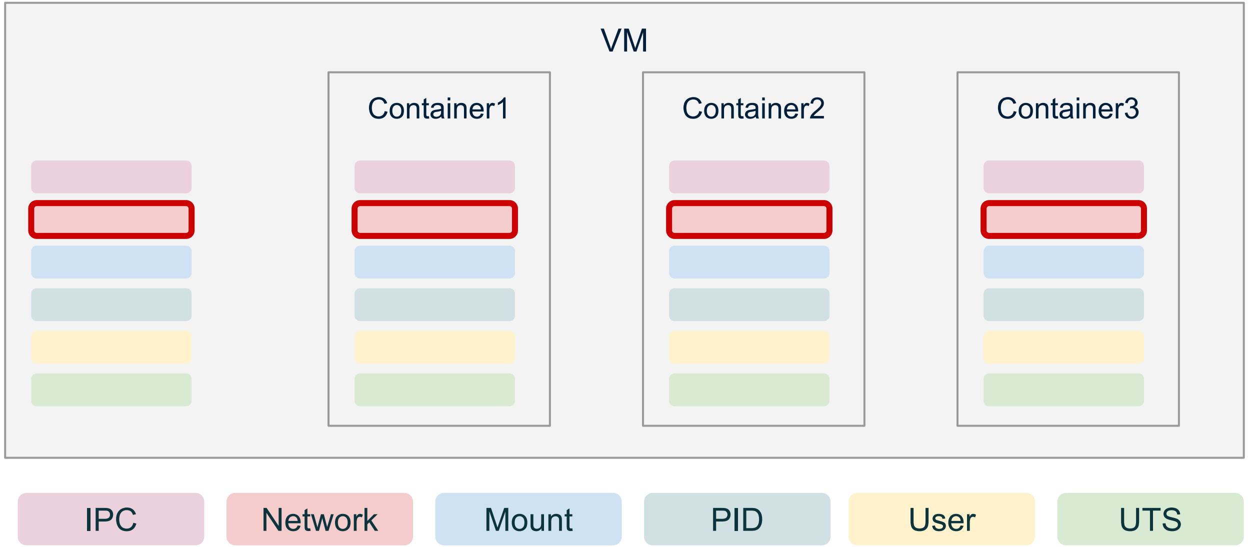
Network Isolation

Foundation of secured containers

Network isolation is achieved by starting a container in a separate network namespace

Network namespace enables ports virtualization ➔ different containers running on the same VM can run on port 8080

Each container gets its own IP address which is not directly accessible from outside of the VM



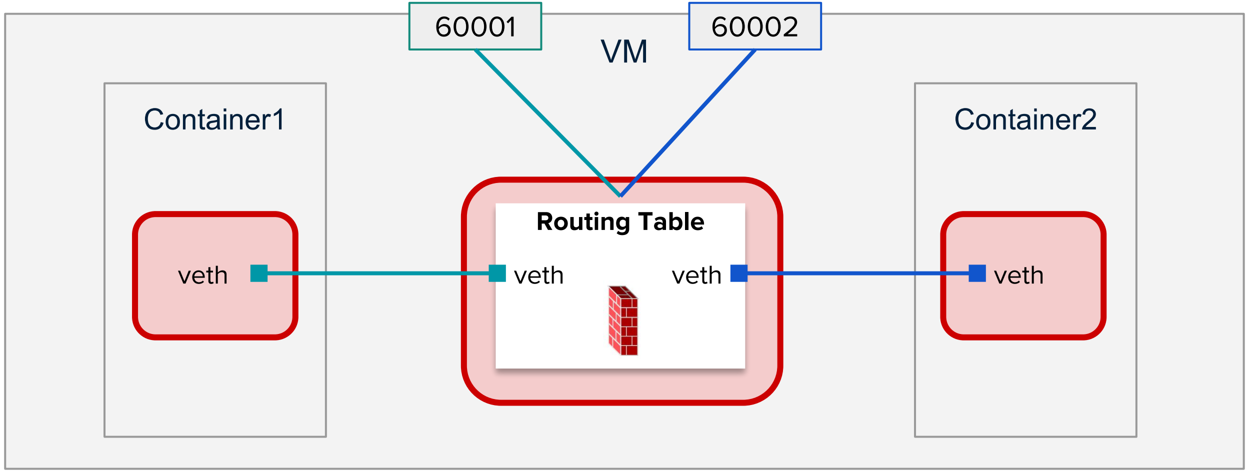
Network Isolation

Foundation of secured containers

Container runtimes use a pair of virtually linked Ethernet (veth) interfaces for networking

One of the interfaces is assigned to same network namespace as the container and other on the VM namespace.

A virtual link is established between two Veth interfaces ➔ connecting container to VM network .....more on this in Part 2



Network

Resource Control

Resource Limiting

What a process can do

Cgroups or Control Groups

- ➢ Control what a process can do with system resources
- ➢ Resource limiting, prioritization, accounting

cgroups

What a process can do

- One application instance can’t hog all the resources while other application instances starve
- Fair share resource allocation
- First line of defense against Denial of Service attack

cgroups

How much a process can do

- Memory control group to allocate memory quota for application container
- Application container is killed if the memory usage exceeds quota

cgroups

How much a process can do

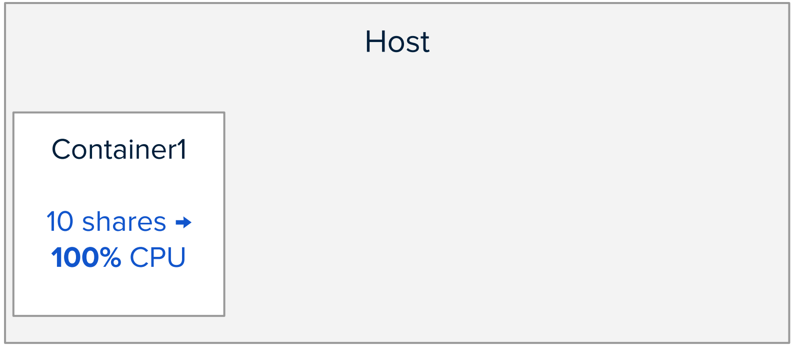
cpu.shares control group to allocate CPU for application container

Application Instance gets CPU based on the ratio of shares

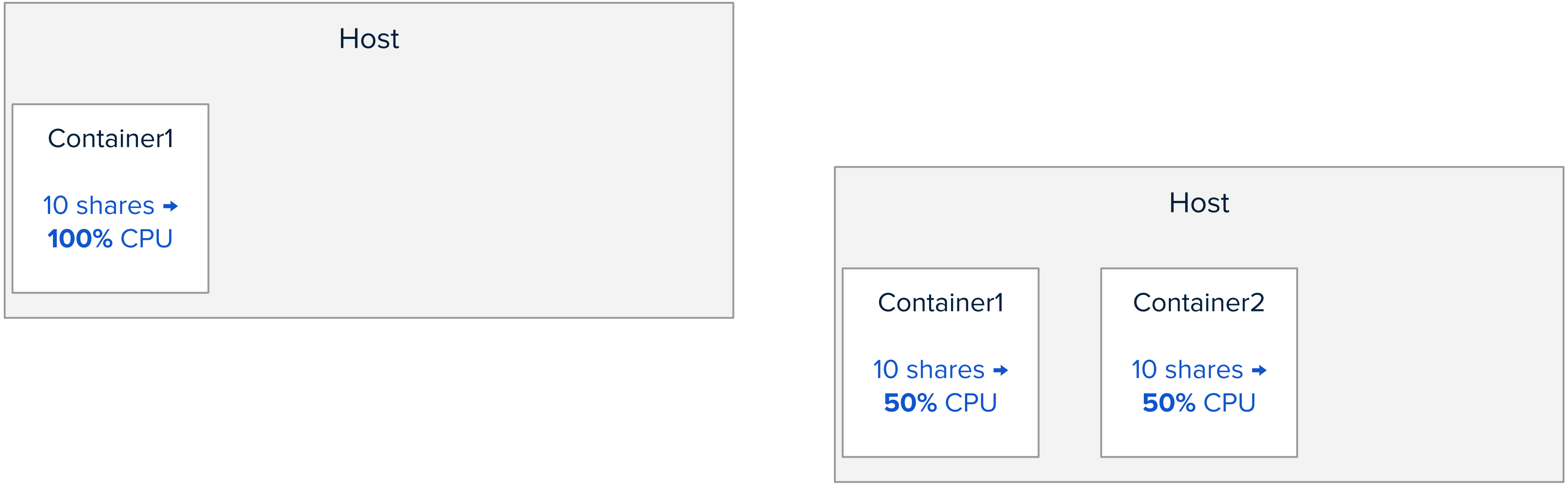
E.g.

- ➢ AppInstance-1 has 10 shares, and is the only AI on the Cell → total shares = 10 → AppInstance-1 gets 100% of the available CPU cycles
- ➢ AppInstance-2 is added on Cell and has 20 shares → total shares = 30 → AppInstance-1 gets ⅓ and AppInstance-2 gets ⅔ of the CPU cycles

cgroups

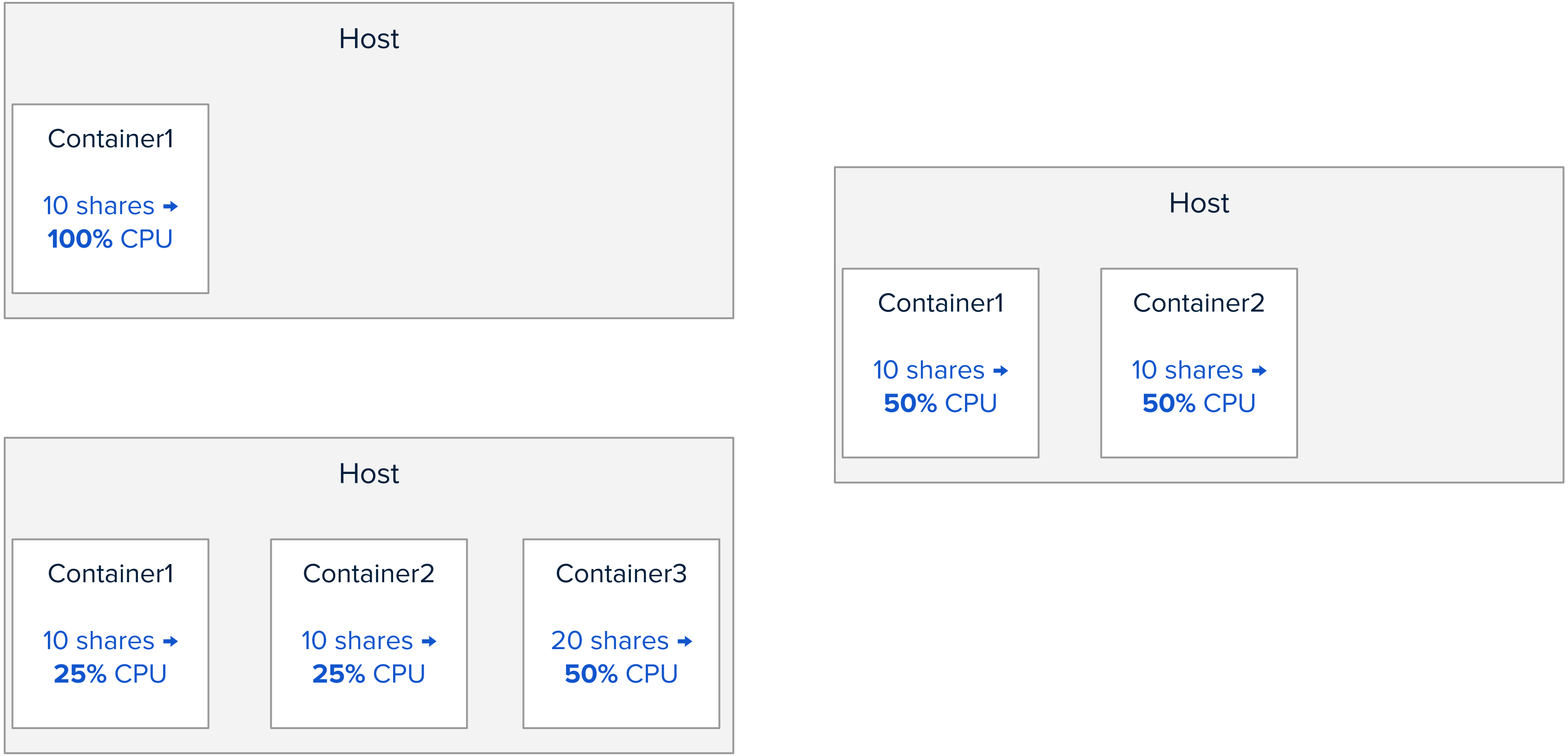


cgroups



Host
Container1
10 shares → 100% CPU
Host
Container1
10 shares → 50% CPU
Container2
10 shares → 50% CPU

cgroups


Host
Container1
10 shares → 100% CPU
Host
Container1
10 shares → 50% CPU
Container2
10 shares → 50% CPU
Host
Container1
10 shares → 25% CPU
Container2
10 shares → 25% CPU
Container3
20 shares → 50% CPU

cgroups

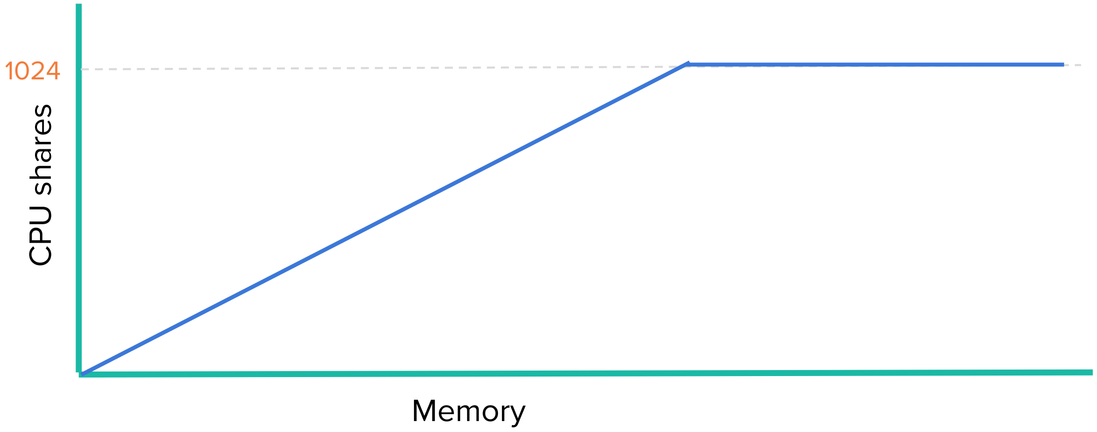
- PCF grants CPU shares to Application Instance linear to memory allocation
- Shares capped at 1024
- process_cpu.shares = max((application_memory / 8), 1024)

cgroups

PCF grants CPU shares to Application Instance linear to memory allocation

Shares capped at 1024

process_cpu.shares = max((application_memory / 8), 1024)



cgroups

Access control

PCF leveragers cgroups to provide a whitelist of devices that container can access

- /dev/full
- /dev/fuse
- /dev/null
- /dev/ptmx
- /dev/pts/*
- /dev/random
- /dev/tty
- /dev/tty0
- /dev/tty1
- /dev/urandom
- /dev/zero
- /dev/tap
- /dev/tun

Disk Quota

- It is provided by XFS file system
- Each container gets its disk quota on the RW layer

More on this in Part 2

Defense in Depth

Philosophy

- ➢ There are several dangerous places and paths in Kernel, once a malicious code gets in there it can get really nasty
- ➢ Vulnerability in any one layer shouldn’t give a free pass to the malicious code
- ➢ Overlapping layers of security are essential

Capabilities

- ➢ Relatively new addition, introduced in Kernel 2.2
- ➢ Prior to Capabilities, either you are a root or a non-root
- ➢ Privileges of the root are partitioned into Capabilities, each Capability can be independently enabled or disabled

Capabilities

- PCF reduces attack surface area of container by dropping all capabilities that are not necessary for the container
- However, the container runtime needed to be a root use user
  - E.g. to use network namespace or cgroups
- Starting Linux 3.8 a non-root use can create User namespace and become a root user in container
  - This has provided a unique opportunity to explore rootless containers

Mandatory Access Control

- ➢ Linux Security Modules (LSMs) or Loadable Kernel Module (LKM) provide security hooks for Mandatory Access Control (MAC) systems.
- ➢ AppArmor is a LKM or LSM that gets loaded into the Kernel

AppArmor

- ➢ Provides defense in depth to augment security provided by Namespaces, cgroups, and capabilities
- ➢ AppArmor uses a profile for each process
- ➢ Untrusted process can be confined by an AppArmor profile

AppArmor

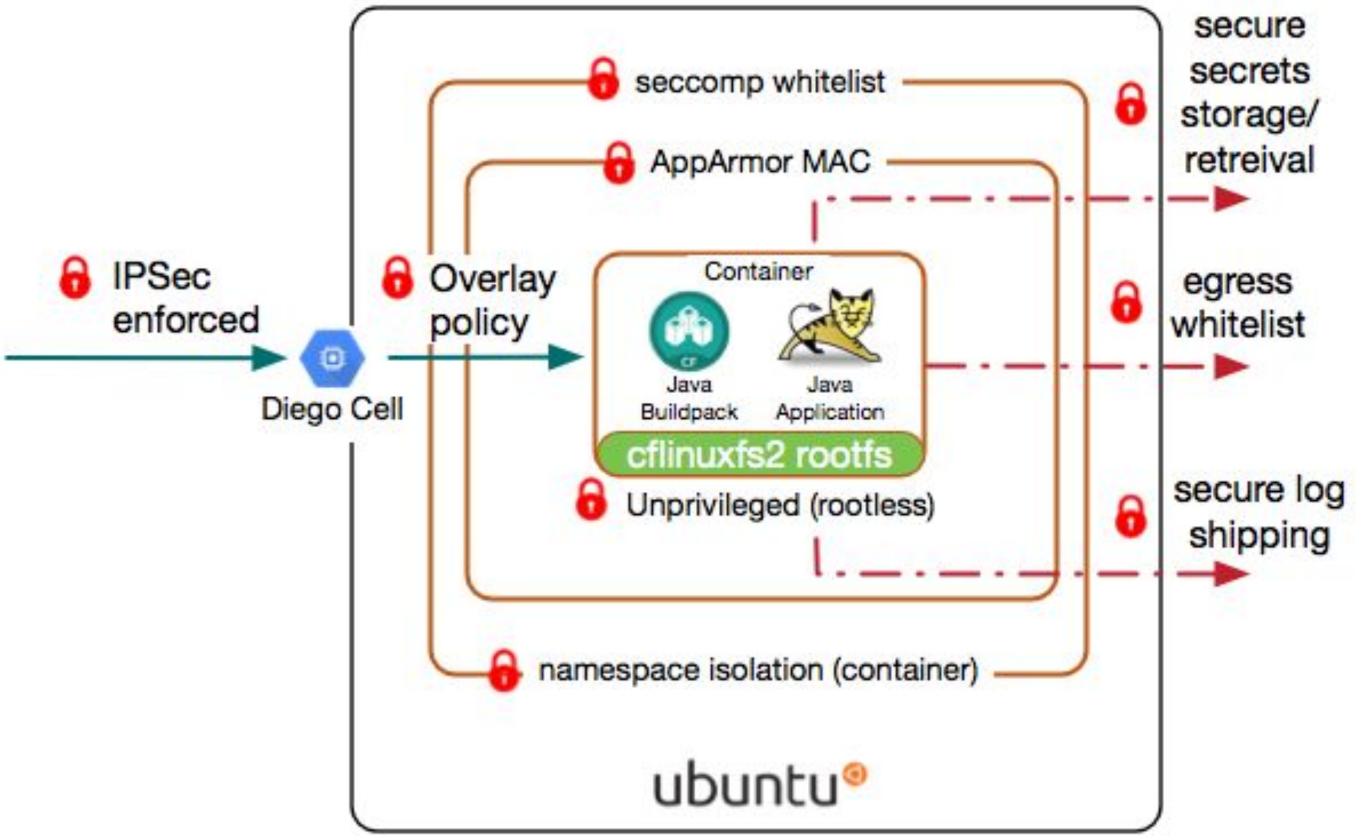
- AppArmor profiles can greatly limit the actions that a given program can take, as well as take complex actions on process-start (such as performing pivot_root()'s, and otherwise manipulating the mount namespace)

- With PCF AppArmor profile is configured and enforced by default for all unprivileged containers

Seccomp

- ➢ It is yet another Kernel Loadable Module, provides Defense in Depth
- ➢ Seccomp (Secure Computing Mode) is a mechanism for system call filtering
- ➢ Seccomp whitelisting restricts the set of system calls a container can access, reducing the risk of container breakout
- ➢ PCF by default enables Seccomp for containers

PCF Locks Down Application Instances



Pivotal

Summary

Containers are not safe by default, DIY platforms can leave behind gaps which can cost your enterprise a lot

PCF’s overlapping layers of container security provides defense in depth

- ✓ Complete isolation for containers using all namespaces + pivot_root
- ✓ Unprivileged containers by default
- ✓ Cgroups to restrict resource usage and access control
- ✓ Enforced disk quota via XFS
- ✓ Dropped capabilities to reduce attack surface
- ✓ AppArmor as Mandatory Access Control layer
- ✓ Seccomp filtering to block harmful system calls
- ✓ Hardened OS for VMs to reduce attack surface
- ✓ Hardened RootFS for containers to reduce attack surface
- ✓ **ZERO developer/operations overhead for all of the above**

Pivotal®

Transforming How The World Builds & Runs Software