調研報告

ITEM

虛擬化技術

在計算機技术中，虛擬化（技術）或虛擬技術（英语：Virtualization）是一種資源管理技術，是将计算机的各種實體资源（CPU、内存、磁盘空间、網路適配器等），予以抽象、转换后呈现出来並可供分割、組合為一個或多個電腦組態環境。[1]

按照抽象程度的不同，虛擬化分為五個層次：指令及架構等級的虛擬化，硬件抽象層等級的虛擬化，操作系統等級的虛擬化，程序語言等級的虛擬化，函數庫等級的虛擬化。計算機用戶經常接觸到的虛擬機，通常即是基於硬件抽象層虛擬化和操作系統層虛擬化技術。

基於波佩克與戈德堡虛擬化需求，虛擬機被定義為*有效的*、*真實機器的副本*。[2]它被允許向用戶或特定應用程序提供由軟件模擬的、具有完整硬件功能的、運行在一個完全隔離環境中的計算機系統，使得用戶或特定應用程序看上去像獨佔了這個計算機系統。用戶因此被允許快速克隆、部署、掛起、回滚虛擬機鏡像，並阻止可能引起系統崩潰、信息洩露或其他敏感後果的指令到達分級保護域[3]底層，獲得安全性。

容器化技術

作業系統層虛擬化（英语：Operating system–level virtualization），亦称容器化（英语：Containerization），是一種虛擬化技術，這種技術將作業系統內核虛擬化，可以允許使用者空間軟體實體（instances）被分割成幾個獨立的單元，在內核中運行，而不是只有一個單一實體運行。[4]

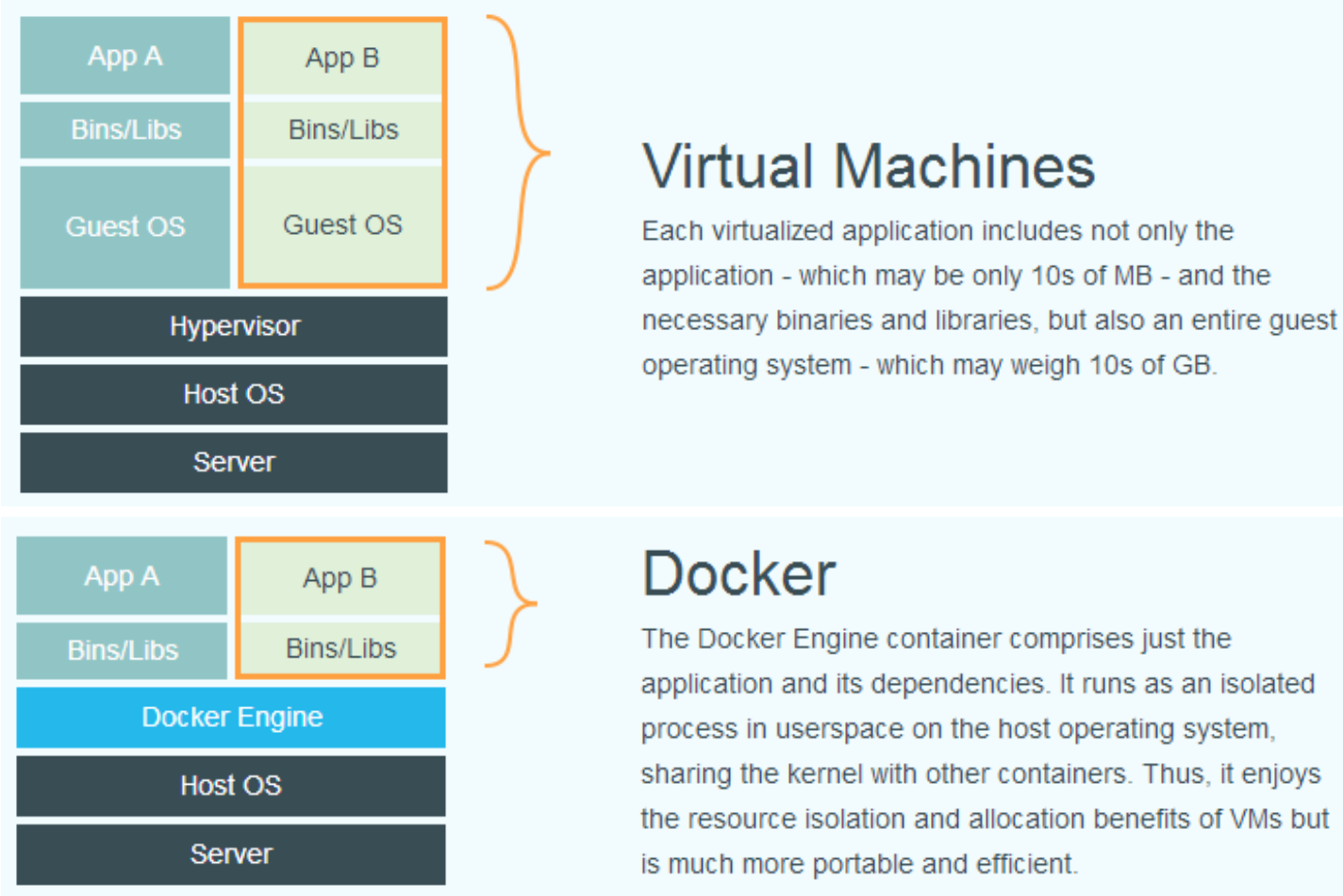
在類 Unix 作業系統中，這個技術最早起源於標準的 chroot 機制，再進一步演化而成。除了將軟件獨立化的機制之外，內核通常也提供資源管理功能，使得單一容器在運作時，對於其他容器的造成的交互影響最小化。

容器化技術的代表以及現在最常用的容器方案是 Docker [5]。Docker 在容器的基礎上進行了進一步封裝，從文件系統、網絡互聯到進程隔離等等，極大簡化了容器的創建與維護，使得 Docker 技術比傳統的虛擬機技

術更為輕便、快捷。

容器與虛擬機

在一些方面，容器被視為虛擬機的替代品。傳統虛擬機技術通常需要先虛擬出一套硬件、在其上運行一個完整的操作系統，再在系統上運行所需的應用程序。而容器內的應用程序直接運行於宿主機的內核，容器內沒有自己的內核，也不進行硬件模擬。因此，在用戶只是希望部署特定應用程序的時候，容器比虛擬機更有效率。



由於容器不需要支付這些額外開銷，無論是應用運行速度、內村損耗或者文件存儲速度，都要比傳統虛擬機技術更為高效。因此，相比使用虛擬機技術，一個相同配置的主機可以借助容器化技術部署更多的應用。除此之外，容器應用直接運行於宿主機內核，不需要啟動完整的操作系統，

容器安全

相比虛擬機，容器的不足之處在於其安全性。虛擬機在主機系統與客機系統間提供了一個虛擬層，處理和模擬那些敏感指令，並使客機內的應用程序認為自己處於真實的機器中，天然地為用戶將其用作沙箱時提供了安全性。但容器內的應用程序直接運行在宿主內核上，惡意代碼可以直接威脅到宿主機系統。

Docker 為此設計了一套容器安全方案。

namespace

Docker 最初基於 LXC [6]而設計，因此核心安全特性也差不多。它為每個容器提供了一個獨立的命名空間，為容器中運行的程序提供了網絡、PID、IPC、UTS、用戶等資源的隔離。這使得容器內的進程對主機進程不可見，容器間的進程也不能相互通信。

cgroup

cgroup（控制組）是 LXC 機制的另一個關鍵組件，負責資源的審計和限制，以組為單位對進程使用的各種資源進行控制。這使得容器公平地分享主機提供的資源，並且不會因為容器內的資源壓力而影響主機。

Capability

Capability（能力機制）是 Linux 內核自 2.2 版本起出現的一個特性。它將權限劃分為更細粒度的操作能力。這使得用戶被允許嚴格而細緻地限制容器可使用的內核的一部分能力。在這種情形下，即使攻擊者在容器內取得了 root 權限，也不能取得主機的較高權限，能造成的破壞受到限制。

Unikernel

儘管 Docker 已經設計了一套容器安全方案，它仍然不適合被作為沙箱使用。為了提供運行環境一致性的特性，Docker 將應用所需的環境依賴打包到容器內，其中難免有可能存在安全漏洞的關鍵組件。這極大擴張了 Docker 的攻擊面。並且，由於容器直接運行於宿主內核上，在一個容器中發生的 kernel panic 會間接影響到同一內核上的其他容器，導致一台主機上的服務大面積宕機。

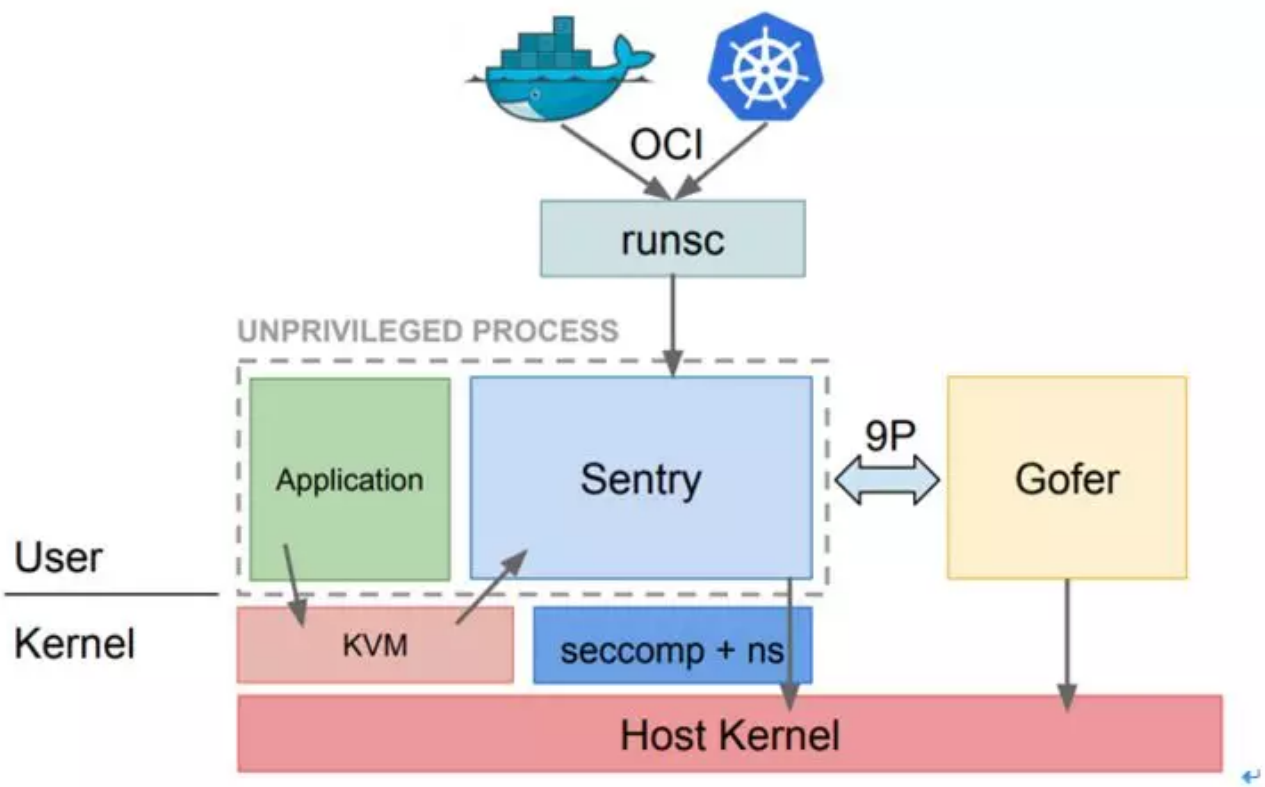
Unikernel [7]提供了一套獨特的虛擬化方案，間接解決了這個問題。它是一個高度定製化的虛擬機鏡像，與容器類似，只為運行特定應用組而存在。它是一個標準的虛擬機，因此具有像虛擬機一樣的安全性能，並且可以直接運行於裸金屬上，與此同時，它拋棄了運行的那些特定應用不需要的模塊，極大精簡了內核，具有接近容器的性能和啟動速度。

gVisor

「可以更快嗎？」當談及將 Unikernel 作為 Docker 的替代，我們會問這個問題。畢竟 Unikernel 運行於裸金屬上專業性過高，不適合日常使用，而運行於宿主機上就相當於一個虛擬機，無論怎樣優化性能，容器化就是它的漸近上界。

Docker 畢竟不是專為作為沙箱使用而設計的。我們很容易想到基於 Docker 改造出一套沙箱解決方案。不需要打包內核，而是直接進一步包裝容器，實現宿主機內核與容器內應用的分離，像虛擬機那樣阻止敏感指令不受限制地到達分級保護域底層。

gVisor，一款新型沙箱解決方案，解決了這個問題。它能够为容器提供安全的隔离措施，同时继续保持远优于虚拟机的轻量化特性。[8]



gVisor 使用了一種被稱作進程虛擬化的技術。它在內核之外實現了一個虛擬內核進程，Sentry，為容器提供了大部分系統調用。容器的那些系統調用實質上轉化為對 Sentry 的訪問。

gVisor 存在兩種劫持模式，基於 **Ptrace** 的和基於 **KVM** 的。

Ptrace

Ptrace 是 Linux 提供的一個系統調用藉口，通過 Ptrace，可以在兩個進程之間建立 Tracer 和 Tracee 之間的關係。Tracer 可以控制 Tracee，例如當 Tracee 收到信號時主動進入 stopped 狀態，此時 Tracer 可以選擇是否對 Tracee 做一些操作（比如設置 Tracee 的寄存器上下文或者內存內容等），在操作執行後，Tracer 可以選擇是否讓 Tracee 繼續執行。在 gVisor 中，Sentry 可以通過 Ptrace 來控制應用程序。

KVM

Ptrace 模式的性能不及 KVM。因為應用的每個系統調用都要經過 Ptrace 訪問 Sentry。同樣的，它的安全性也不及 KVM 模式[來源請求]。在 KVM 模式下，gVisor 能夠截獲應用程序的每個系統調用，並將其轉交給 Sentry 進行處理。相比較 VM，我們看不到 Qemu，也看不到 Guest Kernel，Sentry 包攬了所有必要的操作，正是我們希望的更高層、輕量和專門化的虛擬化實現。

不足

系統調用

Linux Kernel 為 x86_64 提供了 318 個系統調用（4.16），而 Sentry 只實現了兩百多個。當應用請求未被實

現的系統調用是，Sentry 會直接報錯返回。除此之外，在 gVisor 已經支持的系統調用中，還有相當一部分依賴 Host Kernel。在處理這些調用時，Sentry 仍然會陷入內核態。

內存管理

容器內應用的所有代碼均由 Runsc 映射和代理。gVisor 為每個應用程序都維護了一個 Guest 頁表， Runsc 進程自身也有一個 Guest 頁表。在這種架構下，應用程序每次觸發系統調用，都會伴隨一次 Guest 頁表的切換。

網絡

gVisor 的協議棧未被很好地優化，其在 memcached 測試中的性能僅為 KATA 的 20%。

IO

gVisor 的根文件系統通過 9p 協議訪問 Host 文件，其在 mysql 測試中的時間開銷為 aliuk 的 1300%。

總而言之，gVisor 現在在性能上還不盡如人意，某些非外部資源訪問的內核問題對 Host Kernel 的依賴也增大了攻擊面。

[1]：虛擬化 - 維基百科
[2]：波佩克與戈德堡虛擬化需求 - 維基百科
[3]：分級保護域 - 維基百科
[4]：作業系統層虛擬化 - 維基百科
[5]：Docker - 維基百科
[6]：LXC - 維基百科
[7]：Unikernel - Wikipedia [8]：gVisor - Github