01. ホスト型仮想化

ホストOS上で、サーバを仮想的に構築する。

（Provider例）VMware Workstation、Oracle VM VirtualBox

02. ハイパーバイザー型仮想化

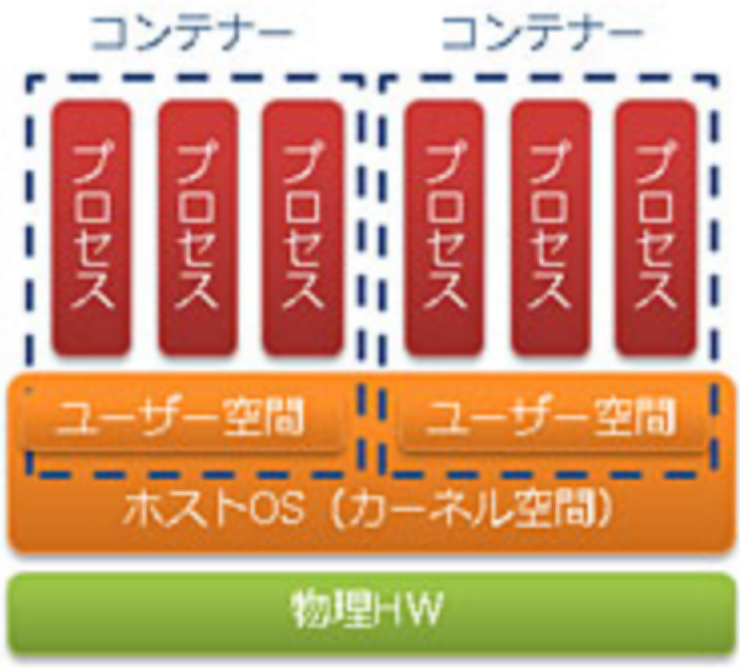
BIOSから起動したハイパーバイザー上で、サーバを仮想的に構築する（※ホストOSは用いない）。

（Provider例）VMware vSphere Hypervisor、Xen、KVM

03. コンテナ型仮想化

ホストOS上で、サーバではなく、コンテナを仮想的に構築する。カーネルのリソースを分割できるNamespace（PID namespace、Network namespace、UID namespace）とControl Groupsを用いて、単一のOS上に独立したコンテナを構築する。

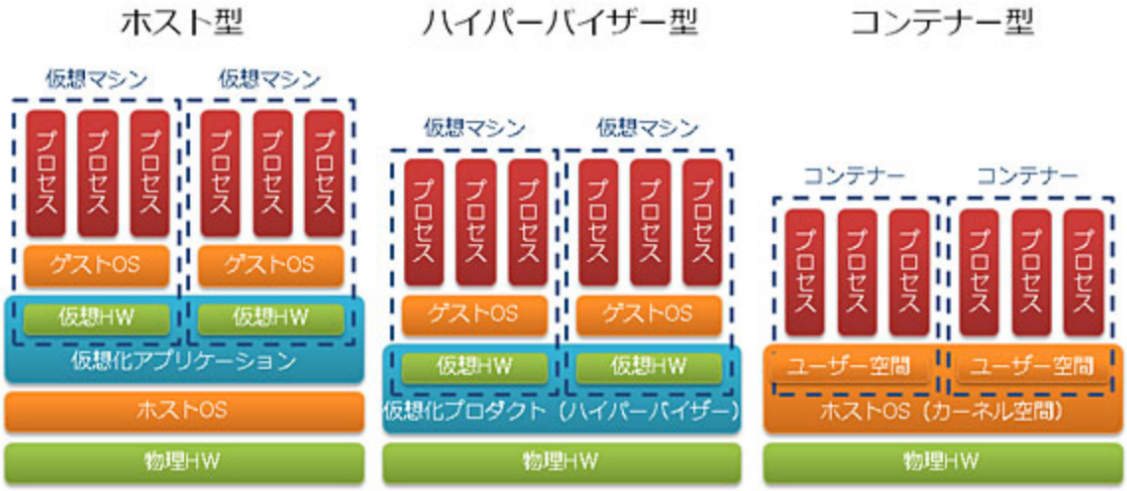
（Provider例）Docker、LXC、OpenVZ



04. 各仮想化の比較

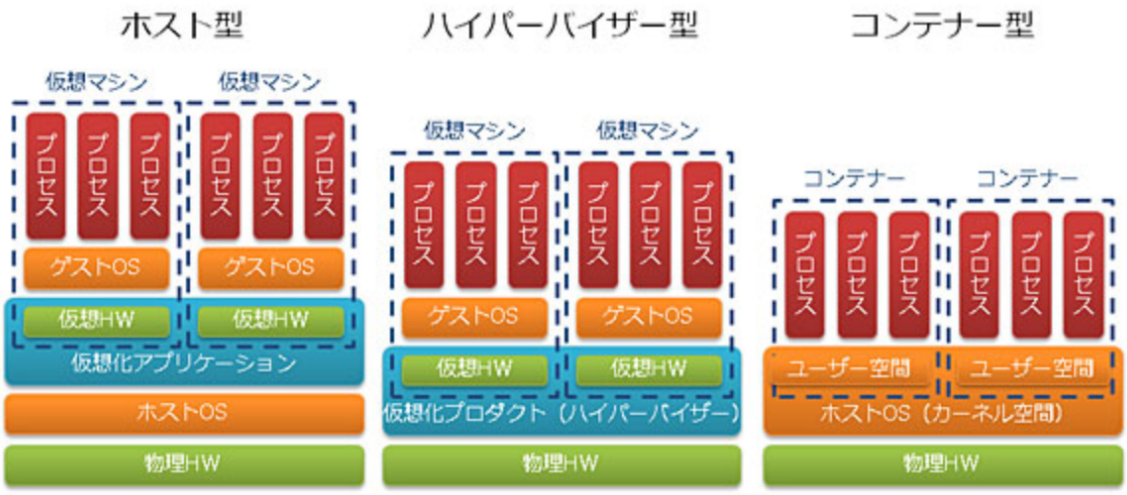
◇ 起動速度の違い

ホスト型とハイパーバイザ型では、ハードウェア（CPU、メモリ、ハードディスク）とゲストOSを仮想化することが必要である。一方で、コンテナ型では、ハードウェアとゲストOSの仮想化は行わず、namespaceを用いてコンテナを構成するため、その分起動が速い。



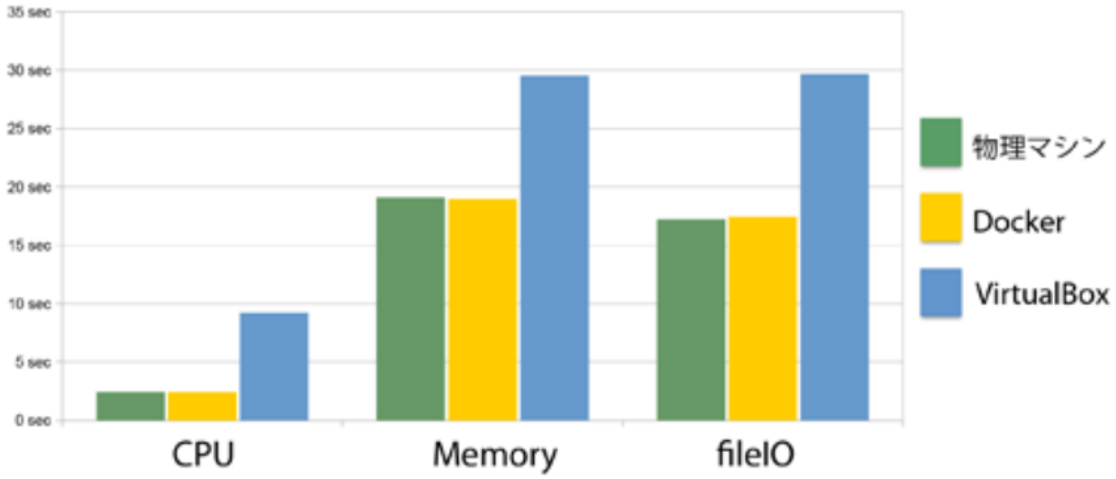
◇ 処理速度の違い

ゲストOS上のアプリを操作する場合、ホスト型とハイパーバイザ型では、ハードウェアやハイパーバイザーを経由する必要がある。この分だけ、時間（Overhead）を要する。一方で、コンテナ型では、各コンテナがホストOSとカーネルを共有するため、Overheadが小さい。

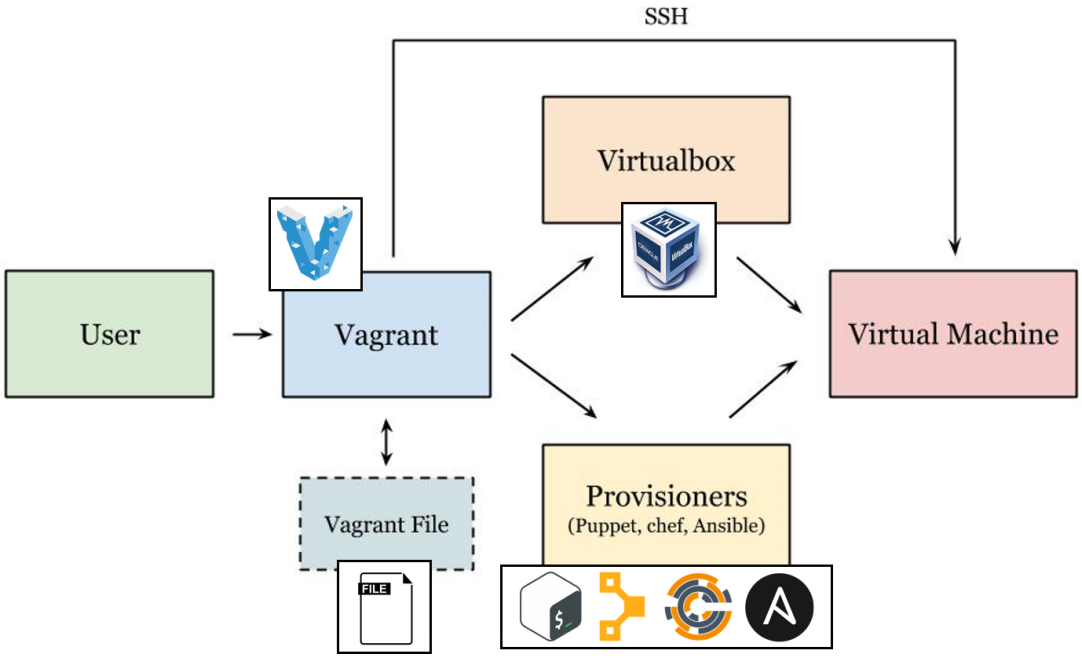


- **Overheadの比較**

sysbenchというベンチマークツールを用いて、CPU・メモリ・ファイルI/Oに着目し、物理マシン・コンテナ型仮想化（Docker）・ホスト型仮想化（VirtualBox）のパフォーマンスを比較。



05. Provider、Provisioner、Vagrantを用いた仮想環境の構築・環境設定・操作



- **Provider**

仮想サーバやコンテナを構築するためのアプリ。構築方法の違いによって、『ホスト型』、『ハイパーバイザ型』、『コンテナ型』に分類できる。

- **Provisioner**

仮想サーバやコンテナの環境を設定するためのアプリ。

- **Vagrant**

ProviderとProvisionerによる仮想サーバやコンテナの構築・環境設定・操作を自動化するプログラム。チームメンバーが別々に仮想サーバを構築する場合、ProviderとProvisionerの環境設定に違いが生じてしまう。Vagrantを使う場合、仮想サーバやコンテナの環境設定はVagrantfileに記述されている。また、仮想サーバやコンテナの構築・操作はvagrant経由で行える。これらのために、Vagrantを

用いれば、チームメンバーが同じ環境設定の下で、仮想サーバやコンテナを構築・操作することができる。