



Arquitectura del Computador y Sistemas Operativos

Vigesimosexta Clase



Máquinas Virtuales (1/10)

Hasta hace un tiempo era común que las Empresas tuvieran un servidor para cada aplicación importante.

Al encarar el proyecto de incorporación de un determinado software a la Empresa, de la misma forma en que se consideraban los tiempos y costos de compra, provisión, instalación y capacitación se involucraba al área de IT para que considerara los costos y plazos para dimensionar, comprar e instalar el servidor en que correría ese software.

Si bien usar un servidor independiente no era “requisito” para correr el nuevo software dentro de una Empresa, muchas veces pasaba por simple dimensionamiento: Dado que el nuevo software no podía correrse en el servidor existente porque era viejo (en el caso de actualizaciones de una aplicación) o simplemente no había uno con los recursos necesarios, cada proyecto terminaba teniendo su propio servidor.

Para Empresas de servicio, sí era requisito muchas veces mantener servidores diferentes para Clientes diferentes, otro motivo para servidores independientes.



Máquinas Virtuales (2/10)

Esta práctica, para IT, tenía varios problemas:

- Los Datacenters, cuyo lugar es caro, terminaban llenos de muchos servidores de mediana potencia.
- Si se deseaba evitar horas sin sistema cuando un componente (por ejemplo un procesador) se rompía, había que tener servidores de repuesto. Arrancar uno no era simple y el proceso dependía de la aplicación que corría en él.
- Había mucho desaprovechamiento de recursos: Por ejemplo si una aplicación usaba mucha RAM a la noche para contabilizar los comprobantes cargados durante el día y otra usaba mucha durante el día porque atendía muchos usuarios concurrentes, ambos debían tener la RAM necesaria para el peor momento.



Máquinas Virtuales (3/10)

Pronto todos recordaron la IBM S/360, diseñada por Gene Amdahl y presentada al mercado en 1964. Si bien este mainframe estaba muy detrás en la historia (para dar una idea el sistema básico tenía 4K de RAM), la forma en que estaba implementada era radicalmente distinta, y resolvía muchos de los problemas que se enfrentaban.

En vez de correr un único programa (el Sistema Operativo) que le presentaba a cada aplicación dispositivos (teclado, mouse, monitor) virtualizados como vimos hasta ahora ...

Se usaba un software diferente que le presentaba a múltiples Sistemas Operativos chips de hardware virtualizados.

En otras palabras:

- Antes, un programa llamado SO, le virtualiza a cada aplicación la funcionalidad de los dispositivos.
- Ahora, otro programa llamado virtualizador, le virtualiza los chips a cada SO.



Máquinas Virtuales (4/10)

Cuando hablamos de virtualizar chips de hardware, se entiende:

- Los chips de RAM
- Los chips que llamamos microprocesadores
- Los chips que tiene el teclado, mouse, etc.
- Los chips de la placa de video y la RAM de video.
- Los chips que implementan los buses para acceder a discos, impresoras, etc.

En otras palabras cada SO piensa que tiene su propio hardware debajo, mientras que en realidad ve un subconjunto del hardware realmente disponible.

El programa que permite “crear” estas diferentes instancias de hardware y exponerlas como una máquina independiente se conoce como “Hipervisor”

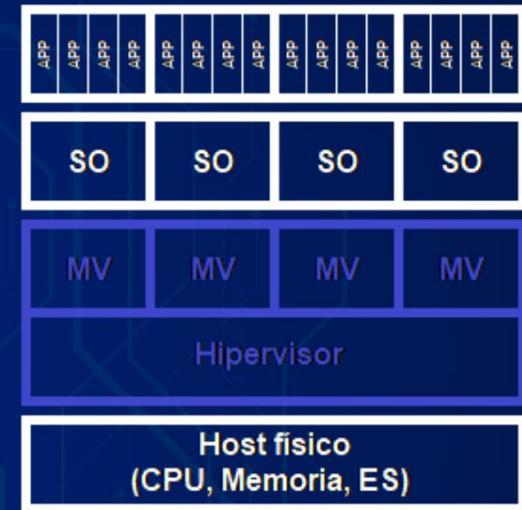


Máquinas Virtuales (5/10)

Arquitectura

Ahora el Hipervisor virtualiza el hardware en lo que llamamos máquinas virtuales (VMs). Cada una “ve” su propio procesador, memoria y dispositivos de E/S.

En cada una se puede cargar un Sistema Operativo, que no tiene que ser el mismo que el que corre en otras VMs. A su vez, en cada SO se corren aplicaciones.



De la misma forma que cada tarea dentro de un SO ve sus propios recursos y no puede interferir con otras tareas, los diferentes SOs de cada máquina virtual ven su propio hardware, el cual administran como si fuera real y propio.

Al igual que el SO necesitaba apoyo del HW para implementar eficientemente la multitarea, lo mismo sucede con los hipervisores.



Máquinas Virtuales (6/10)

Soporte del Hardware (1/2)

Como los Hipervisores virtualizan el hardware, los SOs tratan de manejarlo tal como si el hipervisor no existiera. Por ejemplo cada SO trata de administrar el sistema de paginado y segmentado para darle memoria a sus tareas. En esta situación, cuando el hipervisor tiene que realizar un “machine switch” (pasar a ejecutar el código de otra máquina virtual), puede resultar costoso salvar el contexto dejado por la máquina que deja de ejecutarse y restaurar el de la máquina a ejecutar.

Los fabricantes de procesadores incluyen en el hardware del procesador nuevas instrucciones y funcionalidades para “ayudar” a los hipervisores a hacer más eficientemente su tarea.

En caso de los procesadores x86, los fabricantes agregaron:

- AMD: AMD-v + AMD Nested Page Tables + AMD-vi
- Intel: VT-x + Intel EPT + VT-d



Máquinas Virtuales (7/10)

Soporte del Hardware (2/2)

- AMD-v / VT-x
 - Se conocen como VMX (Virtual Machine Extensions) y agregan instrucciones como:
 - VMPLTRLD: Load Pointer to Virtual Machine Control Structure
 - VMPLTRST: Store Pointer to Virtual Machine Control Structure
 - VMLAUNCH: Virtual Machine Launch
 - VMRESUME: Virtual Machine Resume
 - VMCALL/VMFUNC: Permiten a aplicaciones invocar funciones del hipervisor.
- AMD-NPT / Intel EPT
 - Esta funcionalidad de agrega un nuevo nivel al sistema de paginado, uno desconocido por los SOs. De esta forma, cambiar todas las tablas de paginado es tan simple como ejecutar una sola instrucción. También se conoce como SLAT (Second Level Address Translation)
- AMD-vi / VT-d
 - Permite emular de forma eficiente los accesos a los puertos de E/S, donde se encuentra el hardware que el hipervisor debe emular. Además permite darle a un SO acceso directo a un dispositivo, manejar cómo las interrupciones se redistribuyen entre las VMs, permitirle a un dispositivo como DMA acceso a la memoria virtual de un SO, etc.



Máquinas Virtuales (8/10)

Paravirtualized Hardware

Si bien los hipervisores pueden emular el hardware y “mentirle” a los SOs que piensan que le mandan comandos a los chips cuando en realidad son interpretados por el hipervisor, hay una forma más eficiente...

Algunos hipervisores, además de virtualizar el hardware, le presentan hardware al SO que en realidad no existe, pero que soportan instrucciones de alto nivel. Un caso típico son los adaptadores de red.

Cada adaptador (de cada marca) tiene un chip distinto que soporta ciertos comandos. Cada adaptador requiere, obviamente, un driver. El driver, cuando el sistema operativo quiere enviar un paquete por la red, le carga el paquete al chip, le calcula el framing/checksum y luego lo transmite.

El adaptador paravirtualizado se presenta como un chip “de más alto nivel”. El driver del SO para ese chip no tiene que instruir al chip para que haga todas esas tareas, porque ese “chip” inteligente toma el paquete y hace todo él (en realidad lo hace el hipervisor).



Máquinas Virtuales (9/10)

Over Assignment (1/2)

De la misma forma que los SOs permitían entregarle a las tareas una cantidad de memoria mayor a la disponible mediante el mecanismo de memoria virtual, los hipervisores pueden hacer lo mismo. El proceso de dar más recursos que los disponibles se llama sobre-asignación (Over-Assignment)

Cuando se procede así, si uno suma la cantidad de memoria que tiene cada una de las VMs, puede superar la física. Pero al estar repartiendo la memoria entre SOs y no entre tareas, aparece un problema:

Los SOs tratan de usar toda la memoria disponible. A lo sumo, si le sobra para las tareas que tiene, lo usa como caché de disco. En estos casos, el over-assignment siempre resultaría en una pérdida de performance porque habría que estar siempre subiendo y bajando imágenes de disco.



Máquinas Virtuales (10/10)

Over Assignment (2/2)

Para evitar éso, se corre un driver dentro de cada VM, a veces llamado “ballon driver”. Este driver se comunica directamente con hipervisor y cuando éste necesita memoria analiza la memoria del Sistema Operativo de la máquina en la que corre. Si ve que hay memoria usada como caché, pide memoria, forzando al Sistema Operativo a liberar RAM de caché para satisfacer la demanda. Una vez que se la dan la libera y ahora queda RAM libre que el hipervisor la pueda usar.

El concepto de over-assignment también puede aplicarse a los cores (CPUs). Uno puede asignar más de lo que realmente tiene.

El Over-Assignment produce mejoras en la performance de máquinas que trabajan mucho en distintos horarios, dado que cuando llega el horario de uso intensivo de los recursos el hipervisor se los puede dar porque los tiene (hay otras máquinas que no los están usando).

El over-assignment, bien usado, optimiza los recursos y permite tener la misma performance que con servidores individuales a un menor costo.



Docker (1/2)

Docker no debe ser confundido con un hipervisor. No está diseñado para correr Sistemas Operativos dentro de un hardware virtualizado sino aplicaciones dentro de un Sistema Operativo.

Docker es un proyecto de código abierto que permite correr aplicaciones en lo que llama “*contenedores*”.

La principal ventaja de Docker frente a instalar la aplicación directamente sobre el SO es que toda aplicación requiere configuración. Muchas veces la instalación también tiene pre-requisitos, o sea otras aplicaciones, módulos y/o bibliotecas que deben instalarse antes para el correcto funcionamiento.

Docker permite distribuir en una única imagen todas estas cosas, o sea que uno recibe la aplicación no “para instalar” sino “instalada y configurada”.



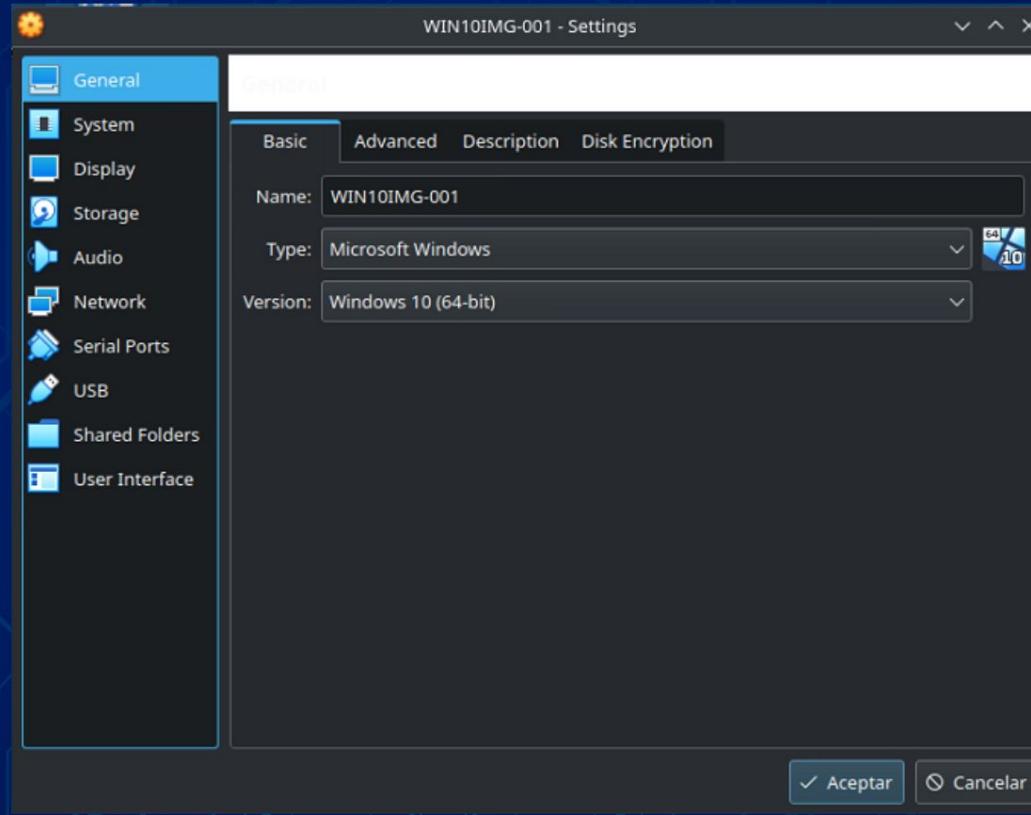
Docker (2/2)

La imagen contiene todo, inclusive el Sistema Operativo. Por este motivo, una aplicación distribuida con Docker puede ejecutarse en cualquier Sistema Operativo soportado sin cambios.

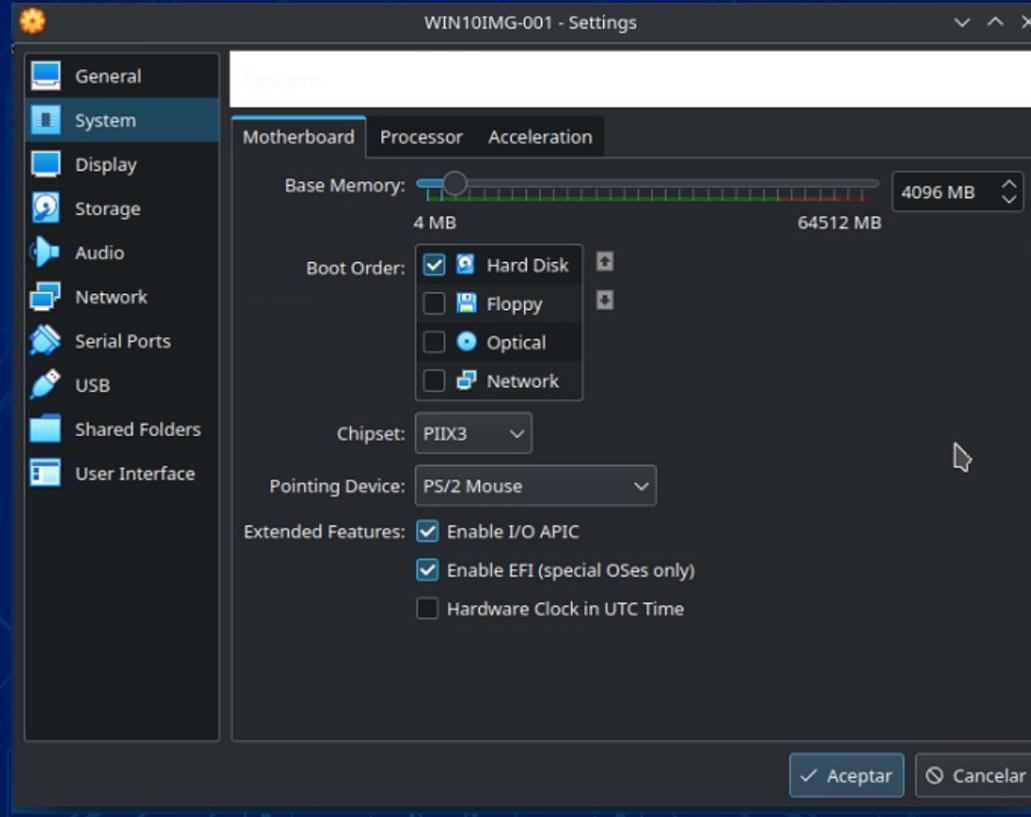
Una funcionalidad útil a los efectos de la seguridad es que se deben exponer explícitamente las funcionalidades que serán “visibles” desde fuera de la aplicación. Por ejemplo, si corremos un servidor HTTP dentro de la aplicación, y se desea que sea accesible desde otras aplicaciones que corren en el mismo servidor, se lo debe declarar expresamente.

La imagen no solo tiene la aplicación y el SO, también tiene el FileSystem, firewall, etc. De esta forma puede distribuirse todo en uno.

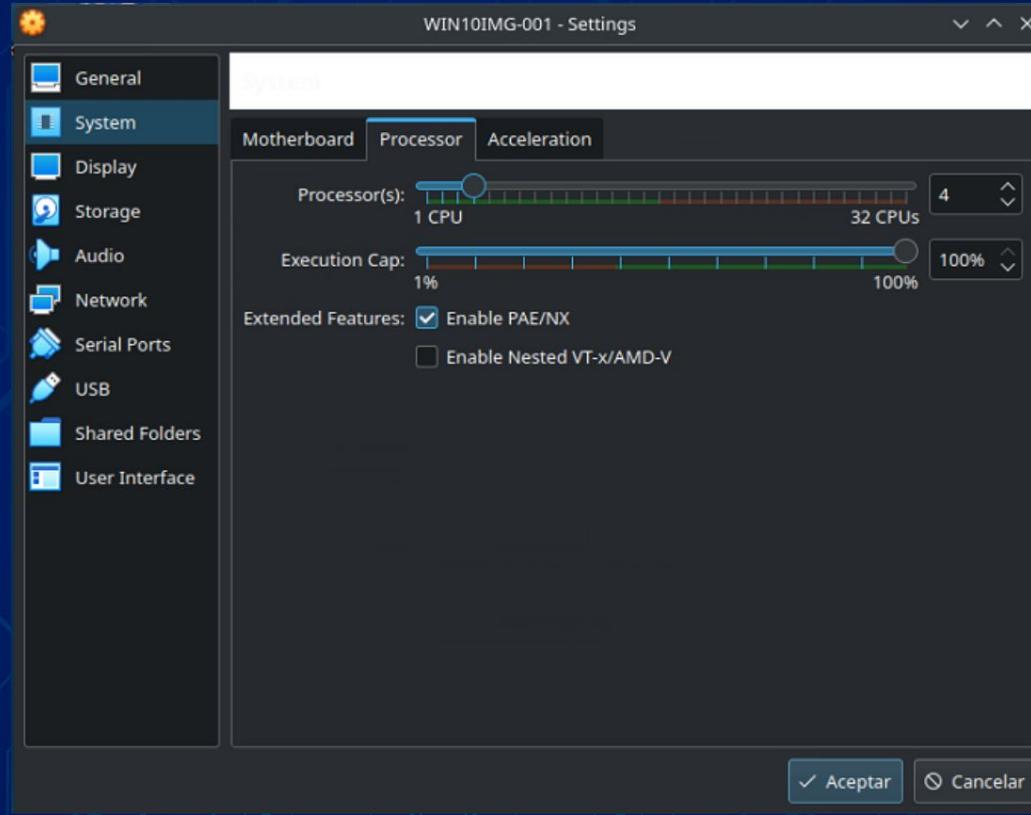
Ejemplo de configuración de VM



Ejemplo de configuración de VM

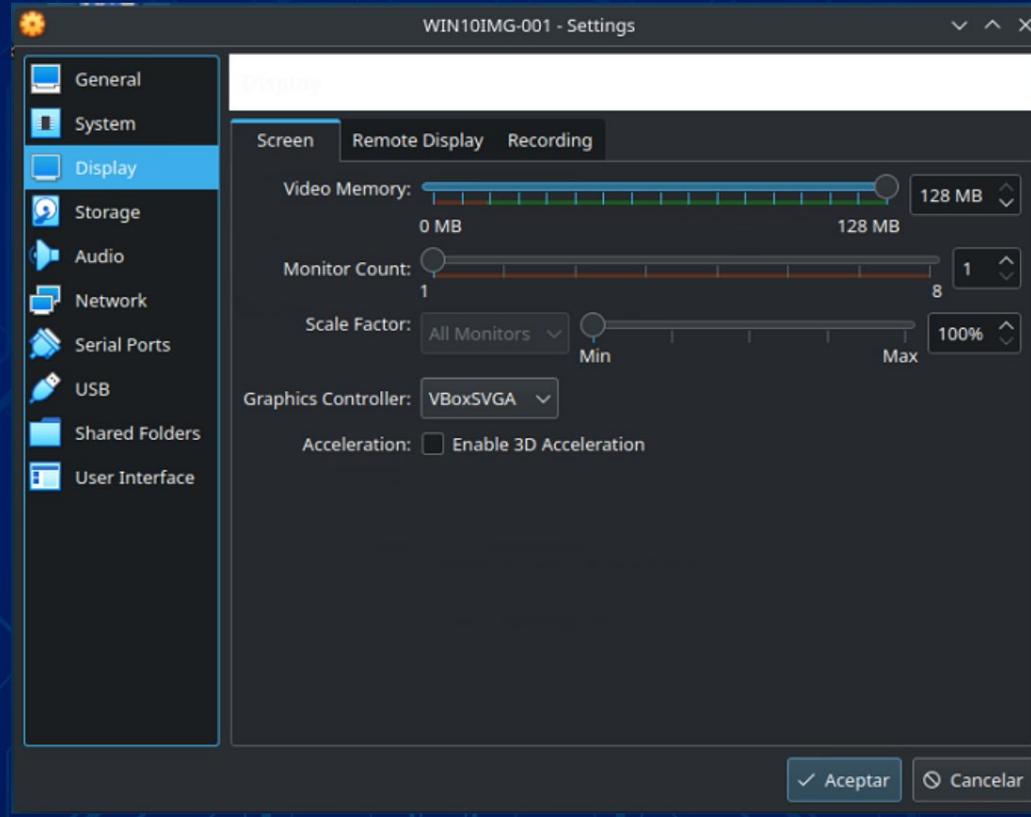


Ejemplo de configuración de VM

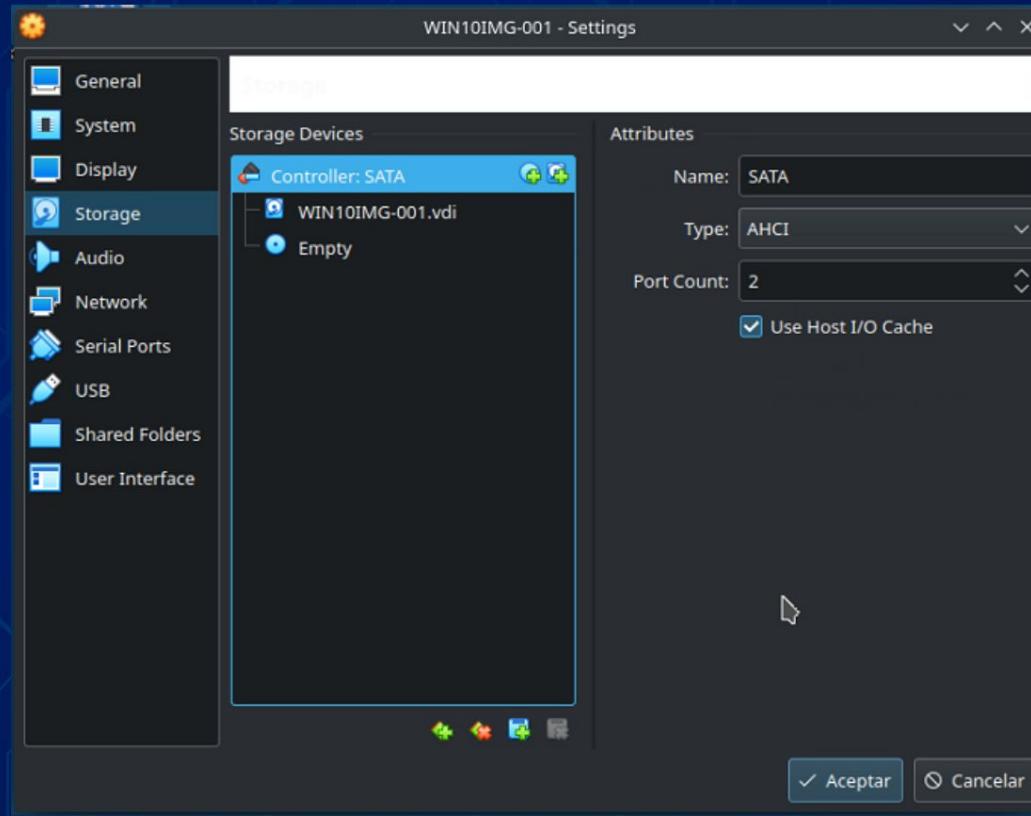




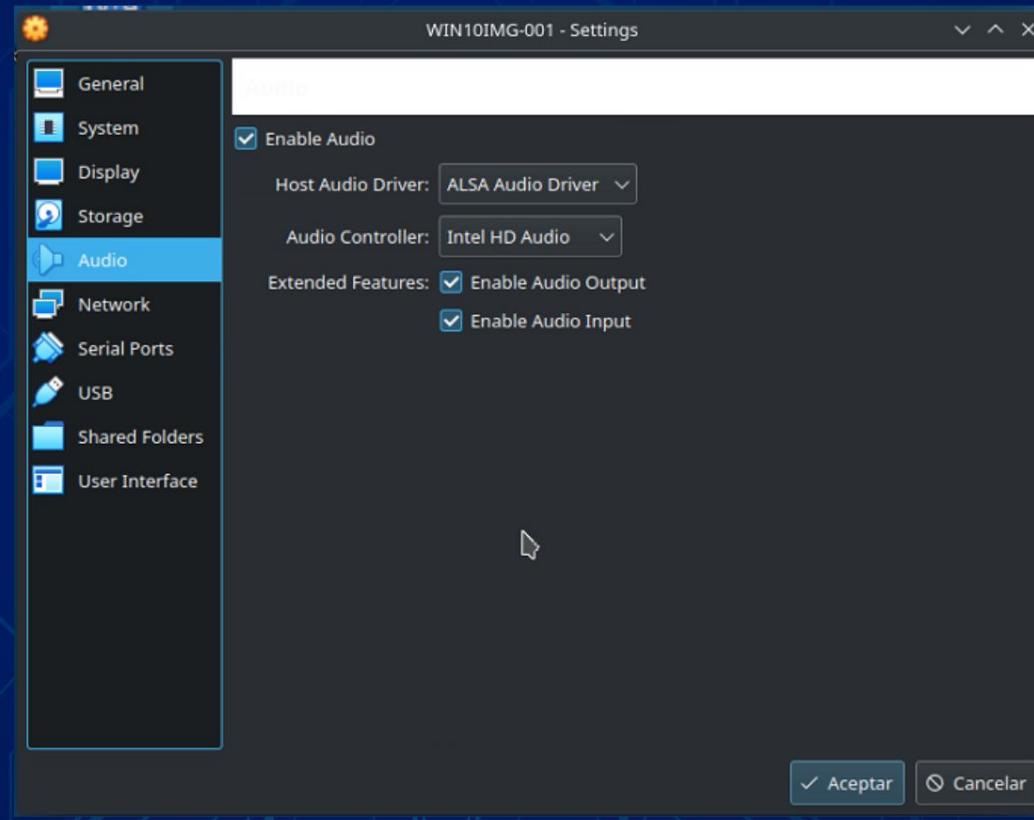
Ejemplo de configuración de VM



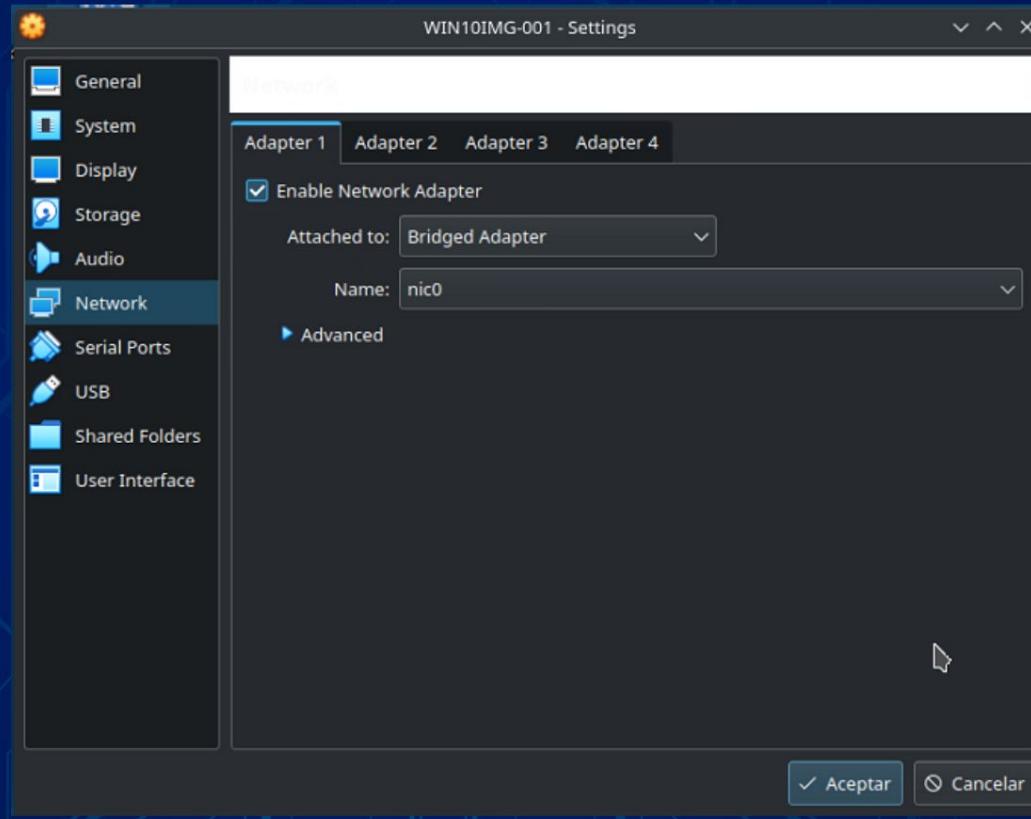
Ejemplo de configuración de VM



Ejemplo de configuración de VM

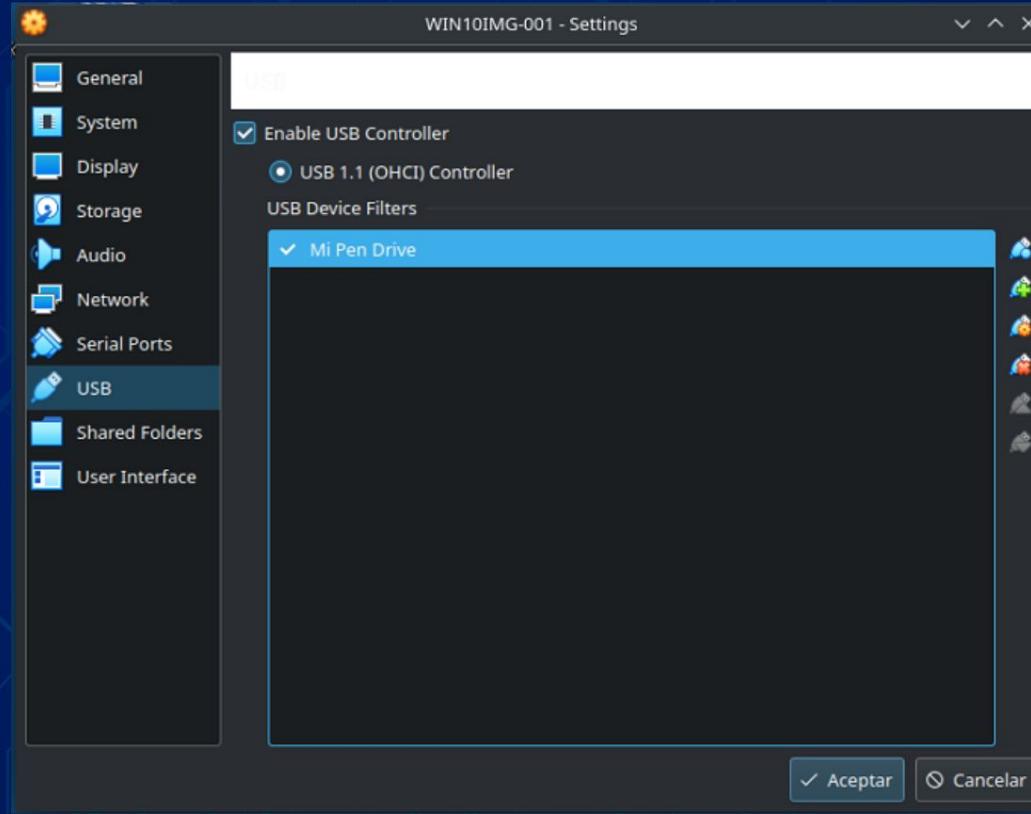


Ejemplo de configuración de VM





Ejemplo de configuración de VM





Fin
¿Preguntas?