

Trabajo Práctico Integrador

Virtualización con VirtualBox

Alumnos:

- Nicolas Bressan – nicolasbressan00@gmail.com
- Franco Bertotti – francob1997@gmail.com

Materia: Arquitectura y Sistemas Operativos

Profesor coordinador: David Roco

Fecha de Entrega: 05 de junio de 2025

Índice

1. Introducción
2. Marco Teórico
3. Caso Práctico
4. Metodología Utilizada
5. Resultados Obtenidos
6. Conclusiones
7. Bibliografía
8. Anexos

Introducción

La virtualización permite ejecutar múltiples sistemas operativos sobre una única máquina física, en los cuales existen dos tipos de virtualizaciones, en este caso incursionamos en las Máquinas Virtuales. En el siguiente Trabajo Práctico Integrador, exploraremos conceptos básicos de virtualización, comparación simple con otro sistema y cómo se realiza una instalación y configuración de una VM con Oracle VirtualBox.

Marco Teórico

Virtualización de hardware: permite correr máquinas virtuales completas sobre un host.

Hipervisores: tipo 1 (bare-metal) vs tipo 2 (hosted).

Imágenes ISO y snapshots.

Tipos de virtualización: VM y Docker

Caso Práctico

Se instaló y utilizó VirtualBox para crear una máquina virtual con Ubuntu Server 22.04.2. En donde se instaló Python, pip, Visual Studio Code y GIT, para crear y ejecutar un programa.

Pasos:

1. Crear nueva VM con 3 GB de RAM y 30 GB de disco.
2. Montar ISO y realizar instalación mínima de Ubuntu.
3. Instalar Python, pip, VSC y Git.
4. Crear código .py en VSC
5. Se ejecutó el programa
6. Se realizó un commit y push en GIT
7. Se subió al repositorio en GitHub
8. se prueba como se comparten los recursos del host con la VM

3)

```
bash
sudo apt upgrade && sudo apt install python3 -y
sudo apt install python3-pip -y
sudo snap install code --classic
sudo apt-get install git
```

6)

```
bash
git status
git add .
git commit
git push
```

Metodología Utilizada

- Instalación de VirtualBox en Windows.
- Descarga de imagen ISO oficial de Ubuntu.
- Configuración máquina virtual.
- Creación y ejecución del código.
- Visualización de gráficos de monitor de recursos.

Resultados Obtenidos

- El sistema operativo invitado (VM) se ejecutó correctamente.
- Se logró la instalación de todos los softwares necesarios.
- Se creó y ejecutó el código .py
- Se comprendió la interacción entre el sistema host y el invitado(VM).
- El consumo de los recursos compartidos se visualizaron en tiempo real.

Conclusiones

La virtualización ha transformado la forma en que se diseñan, implementan y gestionan entornos de prueba y aprendizaje, permitiendo a usuarios y organizaciones simular múltiples sistemas operativos y configuraciones en un solo equipo físico, sin necesidad de hardware adicional. Esta tecnología aporta una serie de ventajas claves:

- Ahorro de costos y recursos
- Flexibilidad y escalabilidad
- Mayor seguridad y aislamiento
- Entornos controlados para pruebas y aprendizaje

Si tenemos que nombrar cosas negativas de este tipo de virtualización, el tipo 2, podemos decir que lo más crítico es la gestión de los recursos frente al consumo de parte de la MV y el host, si bien es escalable a la necesidad, se llega rápidamente a un límite del consumo al ejecutar dos OS y su rendimiento es menor, como mostramos en el video, usando una máquina con recursos escasos.

VirtualBox es una herramienta accesible y potente para este propósito. Gracias a que es de código abierto, posee facilidad de uso y funciona con diferentes sistemas operativos. A su vez permite gestionar varias máquinas virtuales, compartir recursos entre ellas y configurar redes virtuales. Es útil tanto para principiantes como también para profesionales del sector tecnológico.

Bibliografía

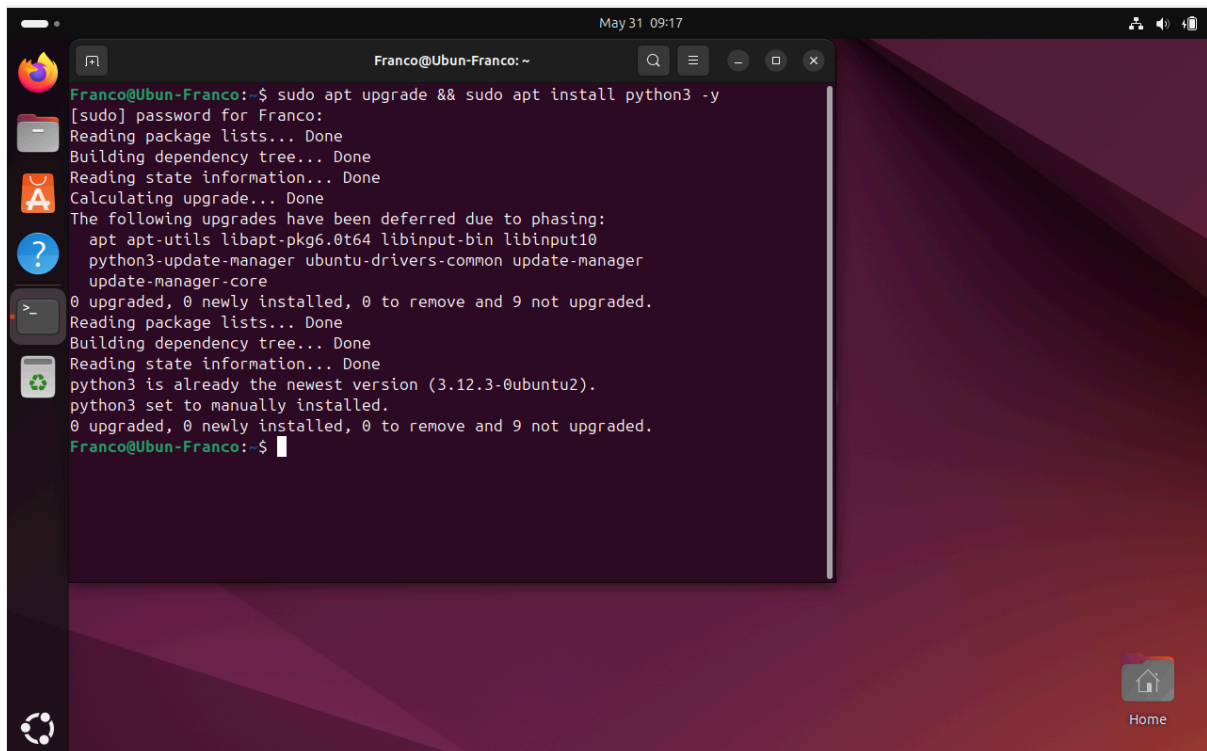
- Documentación de VirtualBox: <https://www.virtualbox.org/manual/>
- Ubuntu Server Guide: <https://ubuntu.com/server/docs>
- Visual Studio Code: <https://snapcraft.io/install/code/ubuntu>
- Git: <https://git-scm.com/downloads/linux>

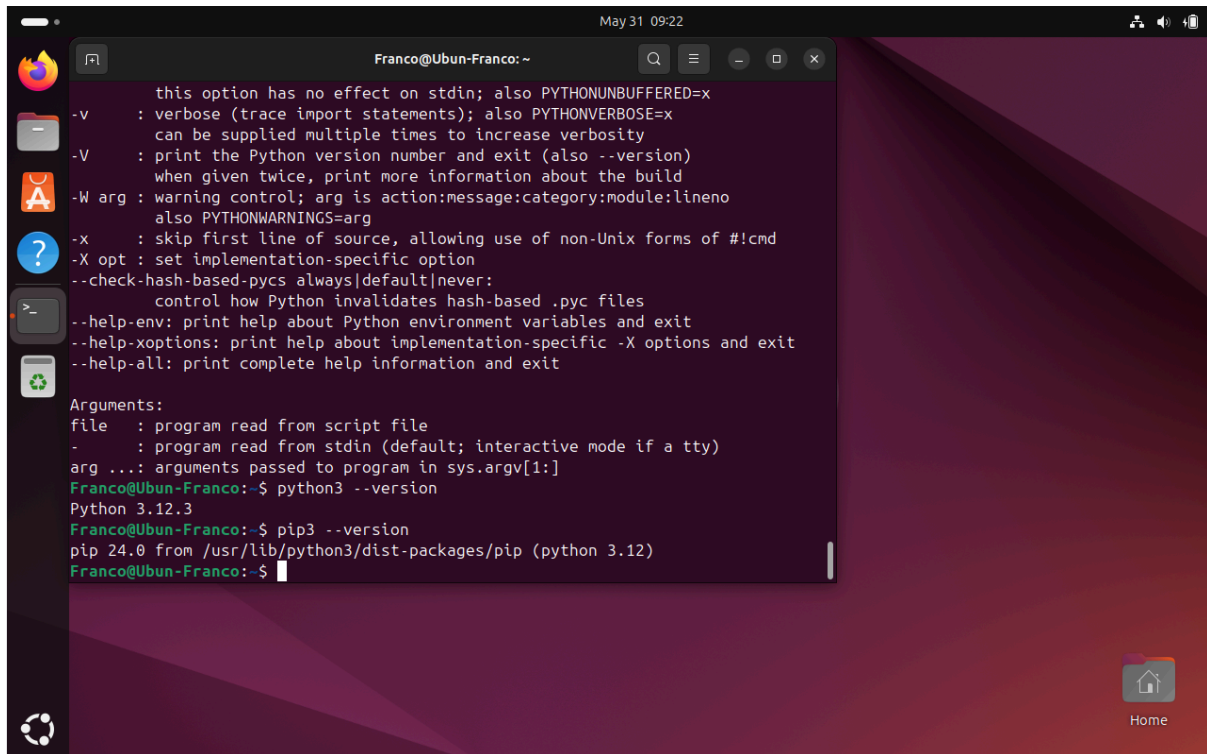
Anexos

Video explicativo: [link](#)

Repositorio: [link](#)

Imágenes:

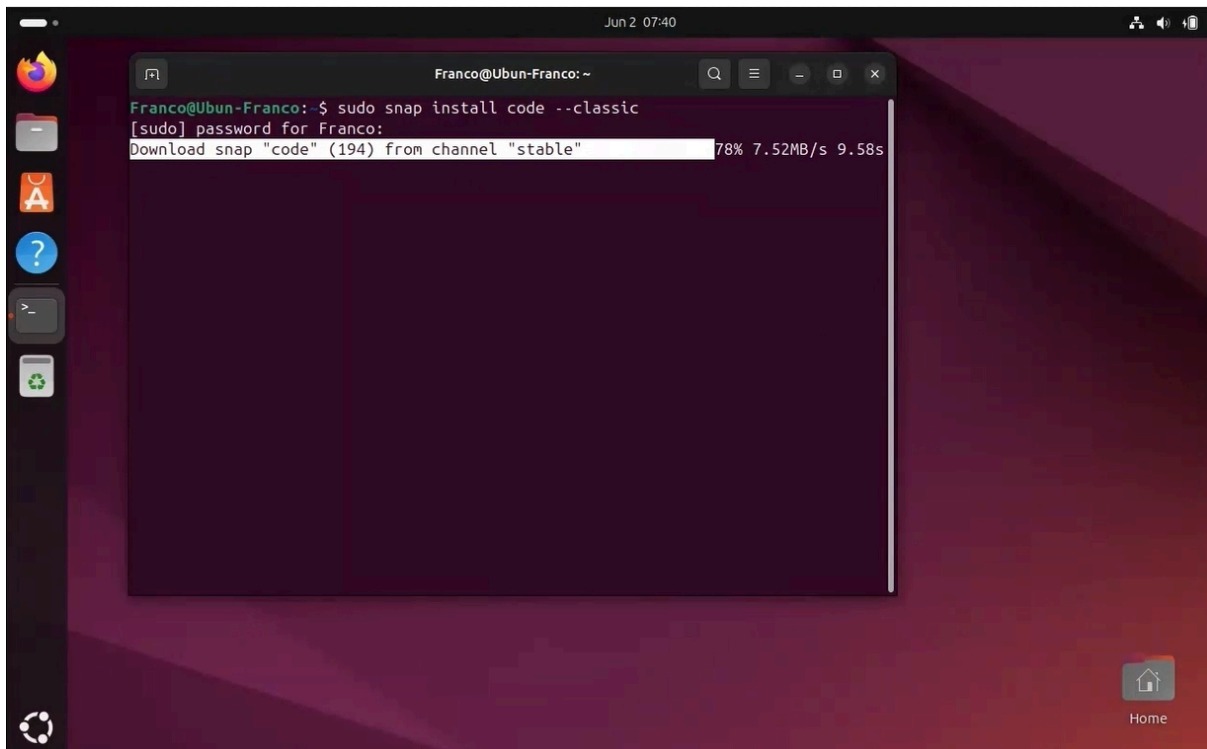




A terminal window titled 'Franco@Ubun-Franco: ~' showing the help output for the Python interpreter. The output lists various command-line options such as -v for verbose, -V for version, -W for warning control, and -x for skipping the first line of source. It also shows the arguments passed to the program. The user then runs 'python3 --version' and 'pip3 --version'.

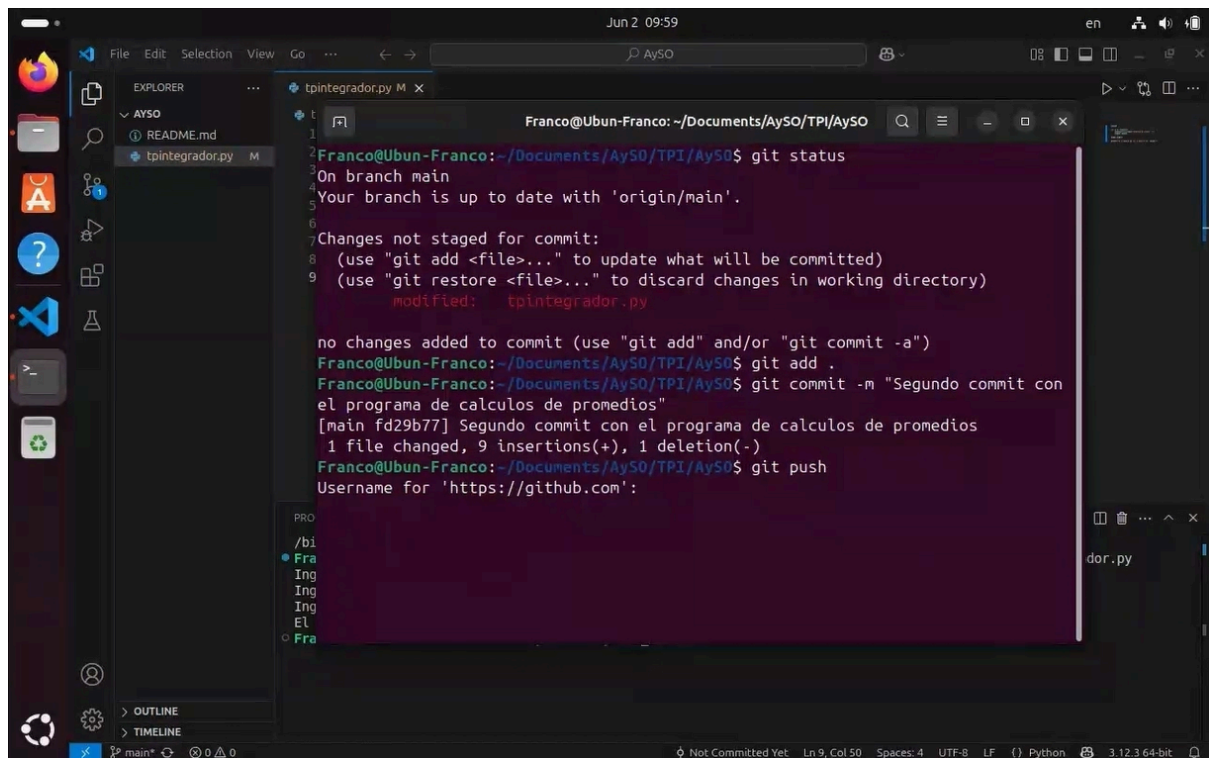
```
May 31 09:22
Franco@Ubun-Franco: ~
this option has no effect on stdin; also PYTHONUNBUFFERED=x
-v : verbose (trace import statements); also PYTHONVERBOSE=x
    can be supplied multiple times to increase verbosity
-V : print the Python version number and exit (also --version)
    when given twice, print more information about the build
-W arg : warning control; arg is action:message:category:module:lineno
    also PYTHONWARNINGS=arg
-x : skip first line of source, allowing use of non-Unix forms of #!cmd
-X opt : set implementation-specific option
--check-hash-based-pycs always|default|never:
    control how Python invalidates hash-based .pyc files
--help-env: print help about Python environment variables and exit
--help-xoptions: print help about implementation-specific -X options and exit
--help-all: print complete help information and exit

Arguments:
file : program read from script file
- : program read from stdin (default; interactive mode if a tty)
arg ...: arguments passed to program in sys.argv[1:]
Franco@Ubun-Franco:~$ python3 --version
Python 3.12.3
Franco@Ubun-Franco:~$ pip3 --version
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
Franco@Ubun-Franco:~$
```



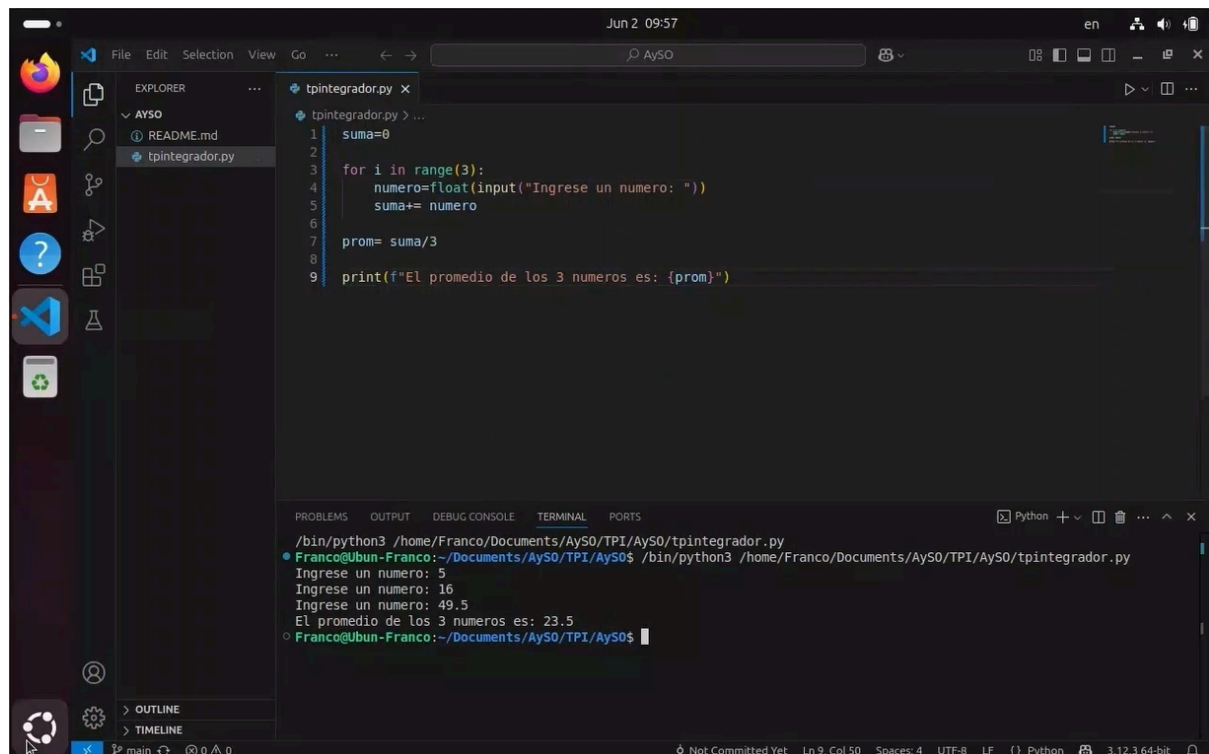
A terminal window titled 'Franco@Ubun-Franco: ~' showing the installation of the 'code' snap. The user runs 'sudo snap install code --classic'. The terminal prompts for a password and then shows the progress of downloading the snap from the 'stable' channel. The download is 78% complete at 7.52MB/s.

```
Jun 2 07:40
Franco@Ubun-Franco:~$ sudo snap install code --classic
[sudo] password for Franco:
Download snap "code" (194) from channel "stable" 78% 7.52MB/s 9.58s
```



This screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of a `git status` command, indicating that the branch is up to date with 'origin/main' and that there are changes not staged for commit. The user then stages the changes with `git add .` and commits them with `git commit -m "Segundo commit con el programa de calculos de promedios"`. The commit message is shown as `[main fd29b77] Segundo commit con el programa de calculos de promedios`. Finally, the user pushes the commit with `git push`, and the terminal prompts for a username for the GitHub repository.

```
1 Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$ git status
2 On branch main
3 Your branch is up to date with 'origin/main'.
4
5 Changes not staged for commit:
6   (use "git add <file>..." to update what will be committed)
7   (use "git restore <file>..." to discard changes in working directory)
8       modified:   tpintegrador.py
9
10 no changes added to commit (use "git add" and/or "git commit -a")
11 Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$ git add .
12 Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$ git commit -m "Segundo commit con
13 el programa de calculos de promedios"
14 [main fd29b77] Segundo commit con el programa de calculos de promedios
15   1 file changed, 9 insertions(+), 1 deletion(-)
16 Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$ git push
17 Username for 'https://github.com':
```



This screenshot shows the Visual Studio Code interface with the `tpintegrador.py` file open in the editor. The file contains a Python script that calculates the average of three numbers. The terminal window shows the execution of the script using `python3 tpintegrador.py`. The user is prompted to enter three numbers: 5, 16, and 49.5. The script then calculates the average and prints the result: "El promedio de los 3 numeros es: 23.5".

```
1 suma=0
2
3 for i in range(3):
4     numero=float(input("Ingrese un numero: "))
5     suma+= numero
6
7 prom= suma/3
8
9 print(f"El promedio de los 3 numeros es: {prom}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Python + Python 3.12.3 64-bit
/bin/python3 /home/Franco/Documents/AySO/TPI/AySO/tpintegrador.py
Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$ /bin/python3 /home/Franco/Documents/AySO/TPI/AySO/tpintegrador.py
Ingrese un numero: 5
Ingrese un numero: 16
Ingrese un numero: 49.5
El promedio de los 3 numeros es: 23.5
Franco@Ubun-Franco:~/Documents/AySO/TPI/AySO$
```

