Primeiramente, vamos gerar dados para que possamos trabalhar os conjuntos.

Vamos usar libs do Python, para carregar dados de máquina de uma maneira mais profissional. Mas também vamos usar uma API aberta, em python que contém uma estrutura já automatizada usando a lib psutil e a lib cure. Para termos uma referência dos dados que precisamos carregar em nosso banco de dados. Mas ela não será usada no projeto.

Precisamos gerar dados de uma maneira automatizada, mas vamos recordar como é fácil utilizar a lib psutil com comandos de python para obter dados de máquina.

Você já sabe instalar e importar uma lib, foi feito o procedimento com a psutil. Para que serve essa lib, mesmo?
Como estamos usando Microsoft em nosso teste então vamos recordar alguns comandos:

Instalação da biblioteca psutil(**documentação:** https://psutil.readthedocs.io/en/latest/#psutil.WindowsService)

pip install psutil

Qual versão da biblioteca está instalada? É a mais recente?

pip list

```
>>>import psutil
>>>psutil.cpu_times()                                    métrica: segundos
scputimes(user=25922.71875, system=12890.109375, idle=136599.328125, interrupt=816.21875,
dpc=1259.0625)

>>> psutil.cpu_times(percpu=True)
[scputimes(user=6825.828125, system=4662.843749999993, idle=32675.796875,
interrupt=659.3125, dpc=841.28125), scputimes(user=5740.109374999999, system=2679.5,
idle=35744.578125, interrupt=71.640625, dpc=195.5), scputimes(user=7230.484375,
system=2943.2031250000073, idle=33990.49999999999, interrupt=51.21875, dpc=136.078125),
scputimes(user=6485.953125, system=2765.53125, idle=34912.703125, interrupt=40.78125,
dpc=113.09375)]

>>> psutil.cpu_times(False)
scputimes(user=26296.4375, system=13057.51562500003, idle=137353.57812499997, interrupt =
823.3125, dpc=1286.625)

>>> psutil.cpu_percent(interval=1, percpu=True)          métrica: %
[3.4, 0.0, 0.0 0.0,0.0, 0.0, 0.0, 0.0]

>>> psutil.cpu_count()
8
```

Retorna os núcleos lógicos. Significa o número de núcleos físicos multiplicado pelo número de threads que podem ser executados em cada núcleo (isso é conhecido como Hyper Threading).

```
>>> psutil.cpu_count(logical=False)
4

>>> psutil.cpu_count(False)
4

>>> psutil.cpu_count(True)
8

>>>  psutil.cpu_freq()
```
Métrica: frequências atual , mínima e máxima expressas em Mhz(Mega Hertz).
```
scpufreq(current=991.0, min=0.0, max=1190.0)

>>> psutil.cpu_freq(percpu=True)
[scpufreq(current=991.0, min=0.0, max=1190.0)]

>>> psutil.virtual_memory()
```
Métricas: total e disponível em Bytes
Métricas (percent): (total - available) / total * 100
```
svmem(total= 8344649728, available= 2162302976, percent=74.1, used= 6182346752, free= 2162302976)

```
>>> psutil.swap_memory()
```
Métricas: total e disponível em Bytes
Métricas (percent): (total - available) / total * 100
```
sswap(total= 2897141760, used= 158707712, free= 2738434048, percent=5.5, sin=0, sout=0)
```

Como podemos conferir essa medida? Via PowerShell

```
PS C:\Users\Eduardo Verri> systeminfo | select-string "Memória"

Memória física total:                      7.958 MB
Memória física disponível:                 1.781 MB
Memória Virtual: Tamanho Máximo:           10.806 MB
Memória Virtual: Disponível:               2.084 MB
Memória Virtual: Em Uso:                   8.722 MB
```

```
>>> psutil.disk_partitions()
```
Métricas: sistemas de arquivos do fs.
```
[sdiskpart(device='C:\\', mountpoint='C:\\', fstype='NTFS', opts='rw,fixed', maxfile=255, maxpath=260)]

>>>psutil.disk_usage('/')
```
Métrica: obrigatório ter o path ‘/’- expressa em Bytes
```
sdiskusage(total= 255270580224, used= 96789012480, free= 158481567744, percent=37.9)
```
```

```
>>>psutil.disk_io_counters()
```
Métricas: Count é expresso em números, R &W em bytes,time em milisegundos
```
sdiskio(read_count= 966008, write_count= 1504237, read_bytes= 26951261696, write_bytes=
31336591360, read_time= 895, write_time= 890)

>>>psutil.disk_io_counters(perdisk=True)
```
Métricas por Device:qte, Bytes, milissegundos
```
{'PhysicalDrive0': sdiskio(read_count= 966017, write_count= 1505326, read_bytes= 26951511552,
write_bytes= 31347515392, read_time= 896, write_time= 890)}
```

Importante: psutil.disk_io_counters(perdisk=True) retorna um dicionário de dados, trabalhar como se fosse um JSON

```
>>>raw = psutil.disk_io_counters(perdisk=True)
>>>raw
{'PhysicalDrive0': sdiskio(read_count=966371, write_count=1507562, read_bytes=26958090752,
write_bytes=31372621312, read_time=896, write_time=891)}

>>>type(raw)
<class 'dict'>

>>raw['PhysicalDrive0']
sdiskio(read_count=966371, write_count=1507562, read_bytes=26958090752,
write_bytes=31372621312, read_time=896, write_time=891)

>>raw['PhysicalDrive0'][0]
966371
```

```
>>>psutil.net_io_counters()
```
Métricas: Bytes, qte pacotes, qte erros, pacotes perdidos
```
snetio(bytes_sent=2179553230, bytes_recv=2746763686, packets_sent=3298168,
packets_recv=3141570, errin=0, errout=0, dropin=0, dropout=0)

>>>psutil.net_io_counters(pernic=True)
```
Métricas: Bytes, qte pacotes, qte erros, pacotes perdidos
```
{'Ethernet': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0,
dropin=0, dropout=0), 'Conexão Local* 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0,
packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Conexão Local* 2': snetio(bytes_sent=0,
bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Wi-Fi':
snetio(bytes_sent=49489141, bytes_recv=276031653, packets_sent=114666, packets_recv=268716,
errin=0, errout=0, dropin=0, dropout=0), 'Conexão de Rede Bluetooth': snetio(bytes_sent=0,
bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Loopback
Pseudo-Interface 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0,
errout=0, dropin=0, dropout=0)}
```

```
>>> psutil.net_connections()
```
Métrica: rastreia todas as conexões
```
{'Ethernet': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0,
dropin=0, dropout=0), 'Conexão Local* 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0,
packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Conexão Local* 2': snetio(bytes_sent=0,
bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Wi-Fi':
snetio(bytes_sent=49489141, bytes_recv=276031653, packets_sent=114666, packets_recv=268716,
errin=0, errout=0, dropin=0, dropout=0), 'Conexão de Rede Bluetooth': snetio(bytes_sent=0,
bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Loopback
Pseudo-Interface 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0,
errout=0, dropin=0, dropout=0)}
psutil.net_connections() ……
```

```
>>> psutil.users()
[suser(name='Eduardo Verri', terminal=None, host=None, started=1692998509.0981452, pid=None)]
```

```
>>psutil.sensors_battery()
sbattery(percent=100, secsleft=<BatteryTime.POWER_TIME_UNLIMITED: -2>, power_plugged=True)
```

```
>>>def secs2hours(secs):
        mm, ss = divmod(secs, 60)
        hh, mm = divmod(mm, 60)
        return "%d:%02d:%02d" % (hh, mm, ss)
>>>bateria = psutil.sensors_battery()
>>>print("carga = {:.02f}%, tempo restante = {}".format(bateria[0], secs2hours(bateria[1])))
carga = 99.00%, tempo restante = 5:50:14
>>>print("carga = %s%%, tempo restante = %s" % (bateria[0], secs2hours(bateria[1])))
carga = 99%, tempo restante = 5:50:14
```

```
>>>psuitl.boot_time()
```
Métrica: Retorna o tempo de boot do sistema em segundos desde Unix Epoch

A época Unix (ou Unix time ou POSIX time ou Unix timestamp) é o número de segundos decorridos desde 1 de janeiro de 1970 (meia-noite UTC/GMT), sem contar os segundos bissextos (na ISO 8601: 1970-01-01T00:00:00Z). Literalmente falando, a época é Unix time 0 (meia-noite 1/1/1970), mas 'epoch' é frequentemente usado como sinônimo de tempo Unix. Alguns sistemas armazenam datas de época como um inteiro de 32 bits assinado, o que pode causar problemas em 19 de janeiro de 2038 (conhecido como o problema do Ano 2038 ou Y2038).

```
1692809986.9052093
```

```
>>>import datetime
>>>psutil.boot_time()
1692809986.9052093
>>>datetime.datetime.fromtimestamp(psutil.boot_time()).strftime("%Y-%m-%d %H:%M:%S")
'2023-08-23 13:59:46'
```

Informação de processos

```
>>> for proc in psutil.process_iter(['pid', 'name', 'username']):
...         print(proc.info)
...
{'pid': 0, 'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System Idle Process'}
{'pid': 4, 'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System'}
{'pid': 8, 'username': None, 'name': 'services.exe'}
{'pid': 140, 'username': None, 'name': 'Registry'}
{'pid': 568, 'username': None, 'name': 'smss.exe'}
{'pid': 640, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedgewebview2.exe'}
{'pid': 656, 'username': None, 'name': 'svchost.exe'}
...
{'pid': 3876, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedge.exe'}
{'pid': 3916, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'StartMenuExperienceHost.exe'}
...
{'pid': 12700, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'WidgetService.exe'}
{'pid': 12720, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedgewebview2.exe'}
{'pid': 12880, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'WindowsTerminal.exe'}
{'pid': 12964, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedge.exe'}
{'pid': 12992, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedge.exe'}
{'pid': 13264, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'SecurityHealthSystray.exe'}
{'pid': 13596, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'svchost.exe'}
{'pid': 13692, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'msedge.exe'}
{'pid': 13700, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'OUTLOOK.EXE'}
{'pid': 13708, 'username': None, 'name': 'dwm.exe'}
{'pid': 13956, 'username': None, 'name': 'csrss.exe'}
{'pid': 14148, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'explorer.exe'}
{'pid': 14656, 'username': 'SPTECH02608\\Eduardo Verri', 'name': 'LockApp.exe'}
{'pid': 14860, 'username': None, 'name': 'svchost.exe'}

>>> procs = {p.pid: p.info for p in psutil.process_iter(['name', 'username'])}
>>> procs
{0: {'name': 'System Idle Process', 'username': 'NT AUTHORITY\\SYSTEM'}, 4: {'name': 'System',
'username': 'NT AUTHORITY\\SYSTEM'}, 8: {'name': 'services.exe', 'username': None}, 140: {'name':
'Registry', 'username': None}, 568: {'name': 'smss.exe', 'username': None}, 640: {'name':
'msedgewebview2.exe', 'username': 'SPTECH02608\\Eduardo Verri'}, 656: {'name': 'svchost.exe',
'username': None}, 860: {'name': 'csrss.exe', 'username': None}, 940: {'name': 'svchost.exe',
'username': None}, 948: {'name': 'wininit.exe', 'username': None}, 960: {'name': 'lsass.exe',
'username': None}, 1132: {'name': 'ctfmon.exe', 'username': 'SPTECH02608\\Eduardo Verri'}, 1156:
{'name': 'svchost.exe', 'username': None}, 1184: {'name': 'fontdrvhost.exe', 'username': None}, 1196:
{'name': 'msedge.exe', 'username': 'SPTECH02608\\Eduardo Verri'}, 1232: {'name': 'svchost.exe',
'username': None}, 1260: {'name': 'msedgewebview2.exe', 'username': 'SPTECH02608\\Eduardo
Verri'}, 1264: {'name': 'WUDFHost.exe', 'username': None}, 1280: {'name': 'msedge.exe', 'username':
'SPTECH02608\\Eduardo Verri'}, 1328: {'name': 'msedge.exe', 'username': 'SPTECH02608\\Eduardo
Verri'}, 1332: {'name': 'msedge.exe', 'username': 'SPTECH02608\\Eduardo Verri'}, 1336: {'name':
'svchost.exe', 'username': None}, 1380: {'name': 'svchost.exe', 'username': None}, 1432: {'name':
'svchost.exe', 'username': None}, 1568: {'name': 'svchost.exe', 'username': None}, 1580: {'name':
'svchost.exe', 'username': None}, 1588: {'name': 'svchost.exe', 'username': None}, ……
```

```
>>> p = psutil.Process()
>>> with p.oneshot():
...         p.name()
...         p.cpu_times()
...         p.cpu_percent()
...         p.create_time()
...         p.ppid()
...         p.status()
...
'pythonw.exe'
pcputimes(user=3.9375, system=1.265625, children_user=0.0, children_system=0.0)
0.0
1693227192.247298
8944
'running'
```

Observação: para obter com eficiência mais de uma informação sobre o processo ao mesmo tempo, certifique-se de usar o gerenciador de contexto oneshot() ou o método de utilitário as_dict().

Sobre o with no python: What's New in Python 2.6 — Python 3.11.5 documentation

```
>>> psutil.Process().exe()
'C:\\Users\\Eduardo Verri\\AppData\\Local\\Programs\\Python\\Python311\\pythonw.exe'
```

```
>>> psutil.Process().cmdline()
['C:\\Users\\Eduardo Verri\\AppData\\Local\\Programs\\Python\\Python311\\pythonw.exe', '-c',
"__import__('idlelib.run').run.main(True)", '58063']
```

```
>>>  psutil.Process().environ()
```

Linux	Windows
cpu_num()	cpu_percent()
cpu_percent()	cpu_times()
cpu_times()	io_counters()
create_time()	memory_info()
name()	memory_maps()
ppid()	num_ctx_switches()
status()	num_handles()
terminal()	num_threads()
	username()
gids()	
num_ctx_switches()	exe()
num_threads()	name()
uids()	
username()	
memory_full_info()	
memory_maps()	

Vamos ver como uma API em Python funciona para capturar os dados da máquina.
Projeto Glances (requer Psutil)

Docs:

curses — Gerenciador de terminal para visualizadores de células de caracteres. — documentação Python 3.13.0a0
Installation - pip documentation v23.2.1 (pypa.io)
Glances - An Eye on your system (nicolargo.github.io)

Windows

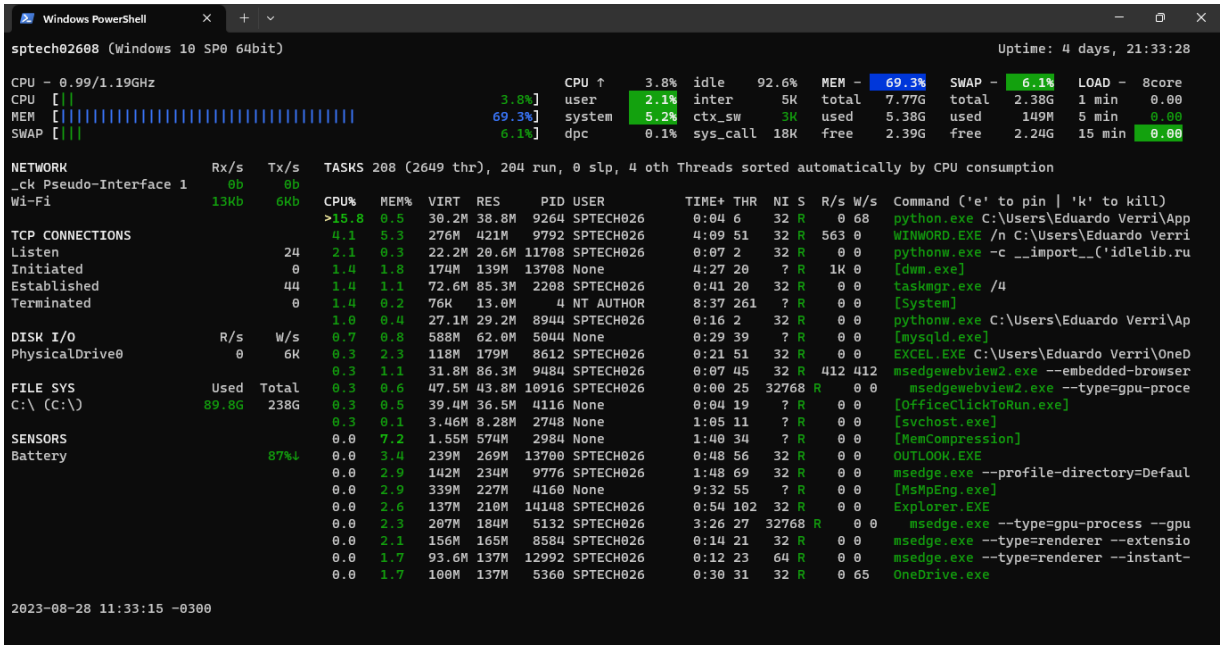
Verifique a versão do pip

```
PS C:\Users\Eduardo Verri> pip --version
pip 23.2.1 from C:\Users\Eduardo Verri\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)
```

PS C:\>pip install glances
PS C:\>pip install windows-curses
PS C:\>glances

Agora a tela da API aparece com a captura em tempo real dos dados de máquina:

Esta tela é de um computador Windows que tem pequenas diferenças no computador com SO Linux.



Linux (WSL)

Vamos instalar no Linux usando container WSL
Entre com o WSL Ubuntu (material de Sistemas Operacionais)
No usuário:

$sudo apt update && sudo apt upgrade -y
$sudo apt install python3
$python3 --version
Python 3.10.12

$sudo apt install python3-pip
$pip –version
pip 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10)

$sudo apt install glances
$glances