

Proyecto 6: Un dashboard imprescindible en el SOC



Índice:

Introducción	3
Prometheus	3
Checkmk	7
Tabla comparativa	

11

Introducción

Para este proyecto se nos ha pedido realizar una comparativa entre 2 sistemas de monitorización, debemos recalcar al menos 6 características; obviamente, instalarlos, yo he elegido Prometheus y Checkmk; y por último debemos monitorizar 2 dispositivos en cada sistema.

En cuanto a la instalación, el Prometheus se ha completado en mi ordenador personal con Windows 10 y usando docker desktop, y el Chekmk lo hice en el portátil de clase con Ubuntu 20.04 y usando docker de terminal.

Procedemos con la instalación de los sistemas de monitorización.

Prometheus

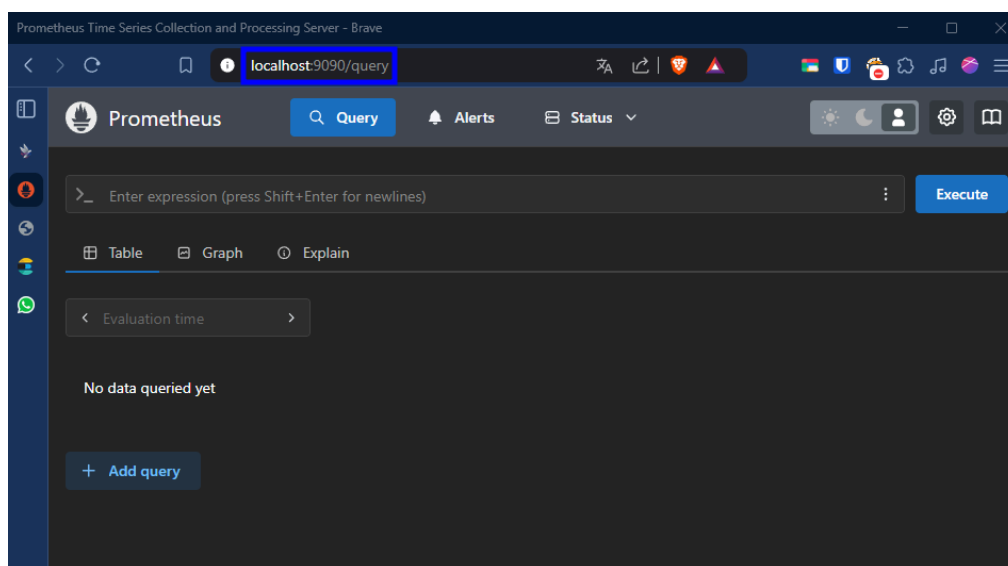
El proceso de instalación del Prometheus es bastante sencillo, simplemente debemos descargarnos la imagen de docker:

```
docker pull prom/prometheus
```

e iniciamos el contenedor con el siguiente comando:

```
docker run --name prometheus -d -p 127.0.0.1:9090:9090 prom/prometheus
```

ahora, si entramos en la siguiente url <http://localhost:9090/query>, podemos ver el siguiente dashboard:



Ahora procedemos a crear los clientes con la misma mecánica. Descargamos la imagen con:

```
docker pull prom/node-exporter
```

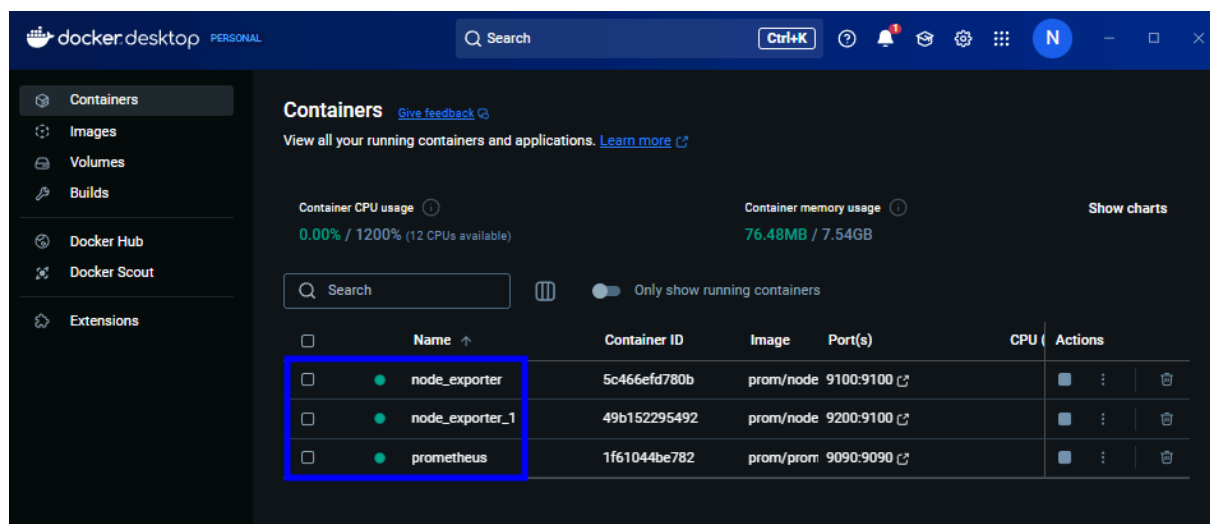
y lo iniciamos con el siguiente comando:

```
docker run -d --name node_exporter -p 9100:9100 prom/node-exporter
```

y como son 2 clientes, creamos otro contenedor pero le cambiamos el nombre y el puerto del host:

```
docker run -d --name node_exporter_2 -p 9200:9100 prom/node-exporter
```

Al finalizar, deberíamos tener los contenedores así:



Es importante recalcar que los puertos del host deben ser todos diferentes, dos dockers no pueden usar el mismo puerto, por eso uno tiene el **9100** y el otro el **9200**.

Ahora, para agregar los clientes debemos entrar al contenedor de Prometheus y acceder a su terminal. En docker desktop simplemente seleccionamos el contenedor y le damos a **Exec**. Dentro del contenedor debemos editar el siguiente archivo:

```
vi /etc/prometheus/prometheus.yml
```

Y abajo del todo, en la sección de *scrape_configs*, añadimos lo siguiente:

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: 'node_exporter_1'
    static_configs:
      - targets: ['172.17.0.2:9100']

  - job_name: 'node_exporter_2'
    static_configs:
      - targets: ['172.17.0.3:9100']
```

Eso son los targets del sistema de monitorización, ahora debemos reiniciar todos los contenedores, y ya nos podremos meter en la url <http://localhost:9090/query>

Prometheus

QueryAlertsStatus > Target health

Select scrape poolFilter by target healthFilter by endpoint or labels

node_exporter_1

1 / 1 up

Endpoint	Labels	Last scrape	State
http://172.17.0.2:9100/metrics	instance="172.17.0.2:9100"job="node_exporter_1"	18.62s ago10ms	UP

node_exporter_2

1 / 1 up

Endpoint	Labels	Last scrape	State
http://172.17.0.3:9100/metrics	instance="172.17.0.3:9100"job="node_exporter_2"	12.4s ago10ms	UP

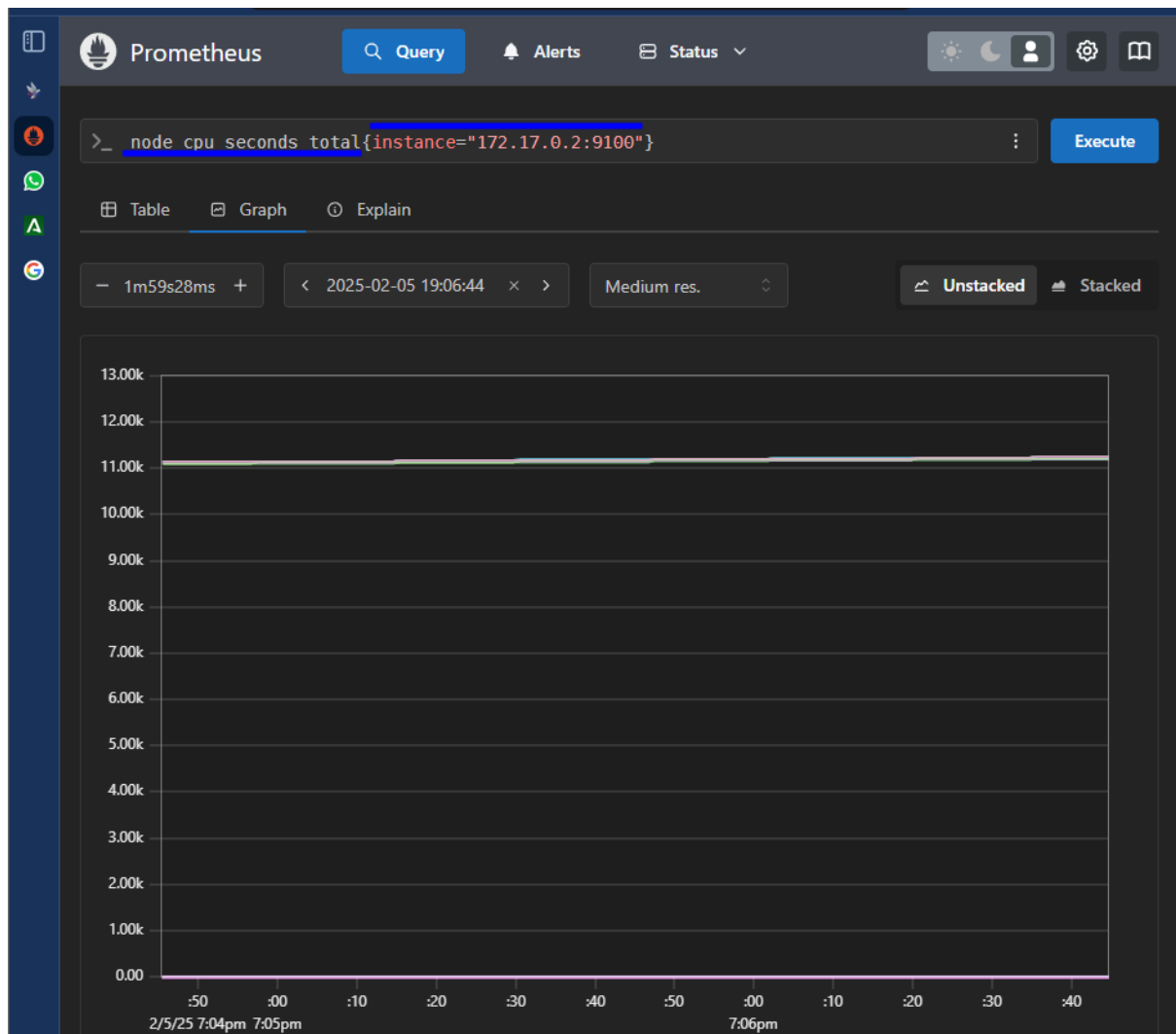
prometheus

1 / 1 up

Endpoint	Labels	Last scrape	State
http://localhost:9090/metrics	instance="localhost:9090"job="prometheus"	1.716s ago3ms	UP

Tenemos los 2 targets que hemos metido con sus respectivas métricas, además, a la derecha vemos que están activos. Volvemos a la url anterior y vamos a ver si se está monitoreando correctamente.

Podemos crear una **query** que mire el uso de la cpu total de un objetivo:



En la imagen podemos ver la query que consiste en una métrica **node_cpu_seconds_total** y un filtro **instance="172.17.0.2:9100"**

Checkmk

Veamos el proceso de instalación de Checkmk. Para esta instalación, me he basado en la documentación oficial de Checkmk. Para iniciar el contenedor ejecutamos el siguiente comando:

```
docker container run -dit -p 8080:5000 -p 8000:8000 --tmpfs
/opt/omd/sites/cmk/tmp:uid=1000,gid=1000 -v monitoring:/omd/sites --name monitoring -v
/etc/localtime:/etc/localtime:ro --restart always checkmk/check-mk-raw:2.3.0p26
```

Nos fijamos en los puertos 8080 y 8000. Si entramos en la url <http://localhost:8080>, veremos el login de la aplicación, cuyas credenciales las veremos en los logs del contenedor:

```
Executing post-create script /opt/omd/sites/cmk/tmp/post-create-script.sh
Skipping Apache restart.
Created new site cmk with version 2.3.0p26.cre.

The site can be started with omd start cmk.
The default web UI is available at http://fa20e0a44e40/cmk/

The admin user for the web applications is cmkadmin with password: hcGJ0UzngioL
For command line administration of the site, log in with 'omd su cmk'.
After logging in, you can change the password for cmkadmin with 'cmk-passwd cmkadmin'.

WARNING: You have to execute 'omd update-apache-config cmk' as root to update and apply the configurati
WARNING: You have to execute 'omd update-apache-config cmk' as root to update and apply the configurati
### STARTING XINETD
* Starting internet superserver xinetd [ OK ]
### STARTING SITE
Temporary filesystem already mounted
Starting agent-receiver...OK
Starting mkeventd...OK
Starting cmk-agent...OK
```

Y para los clientes debemos levantar 2 imágenes de ubuntu y dentro le descargamos el agente:

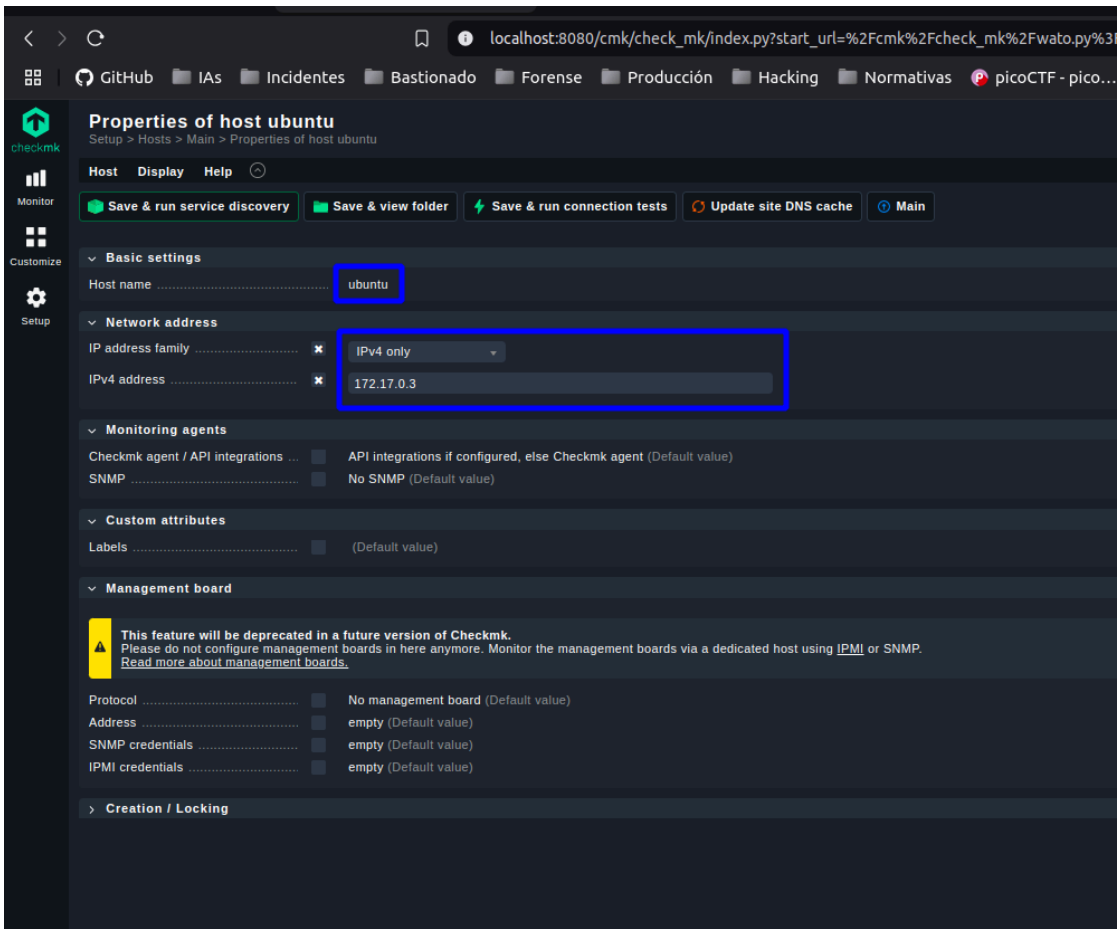
```
docker run -dit --name ubuntu ubuntu:latest
docker run -dit --name ubuntu_2 ubuntu:latest
```

Una vez levantados, tendremos que hacer lo mismo en los 2 contenedores:

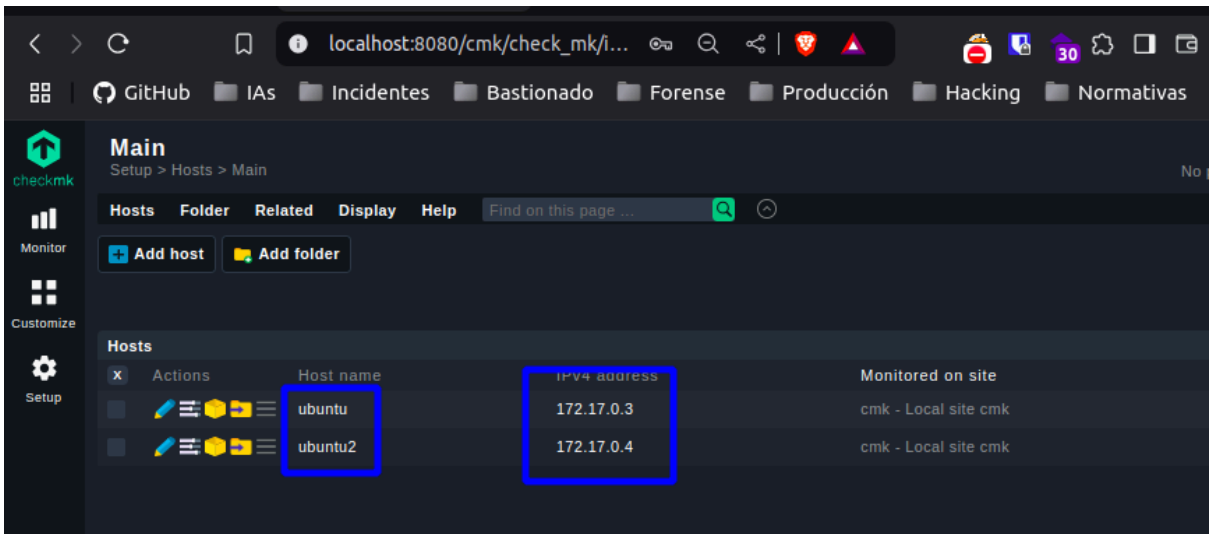
```
apt update && apt install wget xinetd
wget
http://ip\_del\_servidor:5000/cmk/check\_mk/agents/check-mk-agent\_2.3.0p26-1\_all.deb
# Esta dirección la sacamos yendo a setup/ agente/linux
dpkg -i check-mk-agent\_2.3.0p26-1\_all.deb
check_mk_agent
```

Si la consola se nos llena de basura, significa que el agente se ha instalado correctamente.

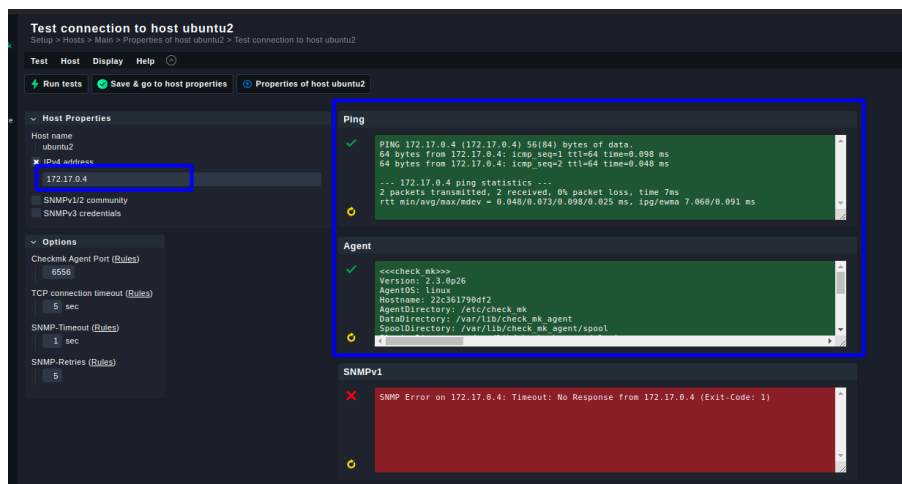
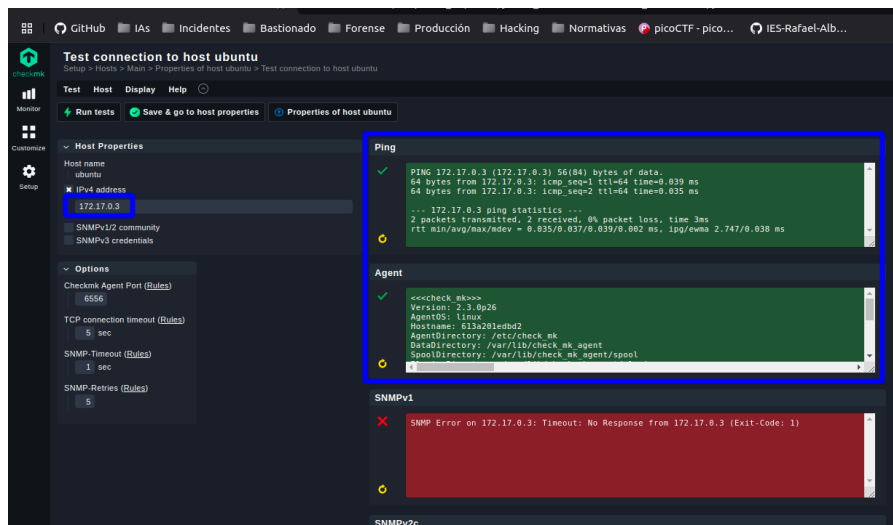
Es turno de crear los hosts, y para ello nos vamos a setup/host y le damos a add host:



Le ponemos un nombre identificativo, seleccionamos ipv4 únicamente y ponemos la ip del contenedor y repetimos con el que queda:

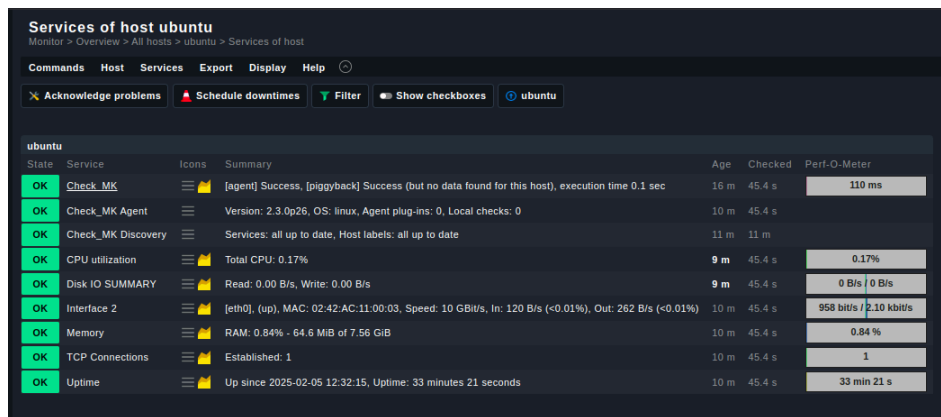


Para comprobar que todo ha salido bien, vamos a ejecutar los tests:



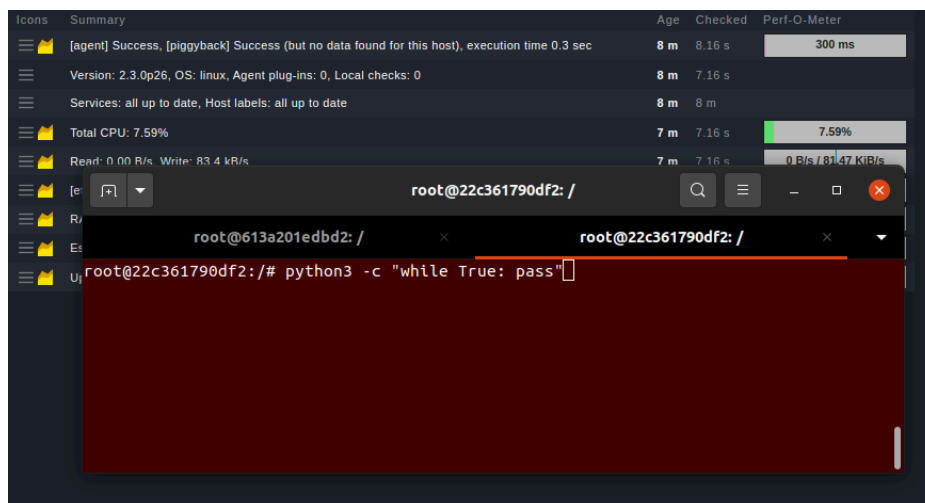
Los únicos que nos interesan ahora mismo son los 2 primeros.

Las reglas de monitorización se crean automáticamente, y tenemos las siguientes:

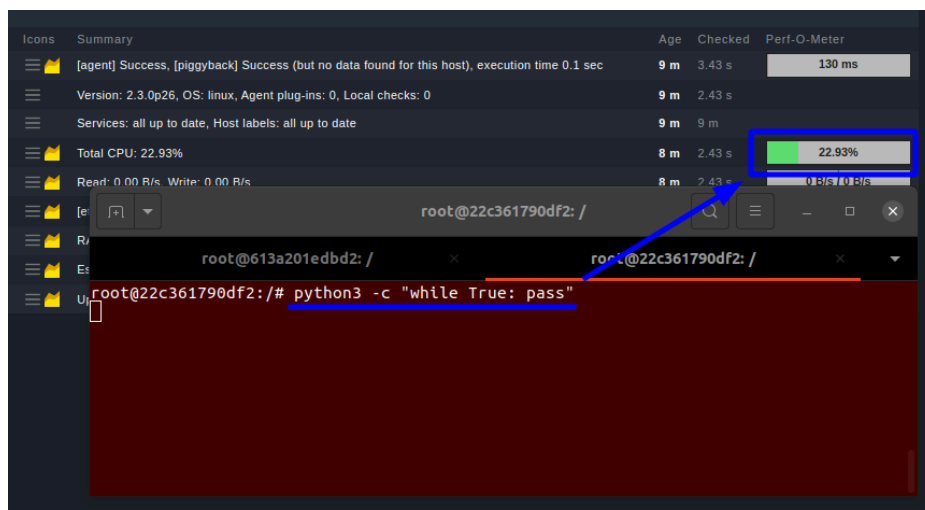


State	Service	Icons	Summary	Age	Checked	Perf-O-Meter
OK	Check_MK	📁	[agent] Success, [piggyback] Success (but no data found for this host), execution time 0.1 sec	16 m	45.4 s	110 ms
OK	Check_MK Agent	📁	Version: 2.3.0p26, OS: linux, Agent plug-ins: 0, Local checks: 0	10 m	45.4 s	
OK	Check_MK Discovery	📁	Services: all up to date, Host labels: all up to date	11 m	11 m	
OK	CPU utilization	📁	Total CPU: 0.17%	9 m	45.4 s	0.17%
OK	Disk IO SUMMARY	📁	Read: 0.00 B/s, Write: 0.00 B/s	9 m	45.4 s	0 B/s / 0 B/s
OK	Interface 2	📁	[eth0], (up), MAC: 02:42:AC:11:00:03, Speed: 10 Gbit/s, In: 120 B/s (<0.01%), Out: 262 B/s (<0.01%)	10 m	45.4 s	958 bits/s / 2.10 kbit/s
OK	Memory	📁	RAM: 0.84% - 64.6 MiB of 7.56 GiB	10 m	45.4 s	0.84%
OK	TCP Connections	📁	Established: 1	10 m	45.4 s	1
OK	Uptime	📁	Up since 2025-02-05 12:32:15, Uptime: 33 minutes 21 seconds	10 m	45.4 s	33 min 21 s

Vamos a ver que funcionan realizando un bucle infinito en la máquina host:



The screenshot shows the monitoring interface with a terminal window open. The terminal shows the command `python3 -c "while True: pass"` being executed on the host. The CPU utilization is shown as 7.59%.



The screenshot shows the monitoring interface with a terminal window open. The terminal shows the command `python3 -c "while True: pass"` being executed on the host. The CPU utilization is shown as 22.93%.

Y ya estaría, el servidor monitorea correctamente los 2 agentes.

Tabla comparativa

Característica	Prometheus	Checkmk
Complejidad de instalación	Baja - Instalación sencilla con Docker	Media - La instalación requiere muchos parámetros
Configuración de clientes	Simple - Uso de node-exporter	Más compleja - Requiere instalación de agentes y varios paquetes en cada cliente
Interfaz de usuario	Básica - Dashboard simple en http://localhost:9090/query	Más elaborada - Interfaz web completa con login
Visualización de datos	Gráficos básicos, enfocado en queries	Dashboard más completo con múltiples vistas y gráficos
Escalabilidad	Alta - Fácil adición de nuevos targets	Media - Requiere configuración individual de cada host
Flexibilidad en consultas	Alta - Permite crear queries personalizadas	Media - Reglas de monitorización predefinidas
Integración con Docker	Nativa - Diseñado para funcionar con contenedores	Buena - Funciona con contenedores, pero requiere más configuración