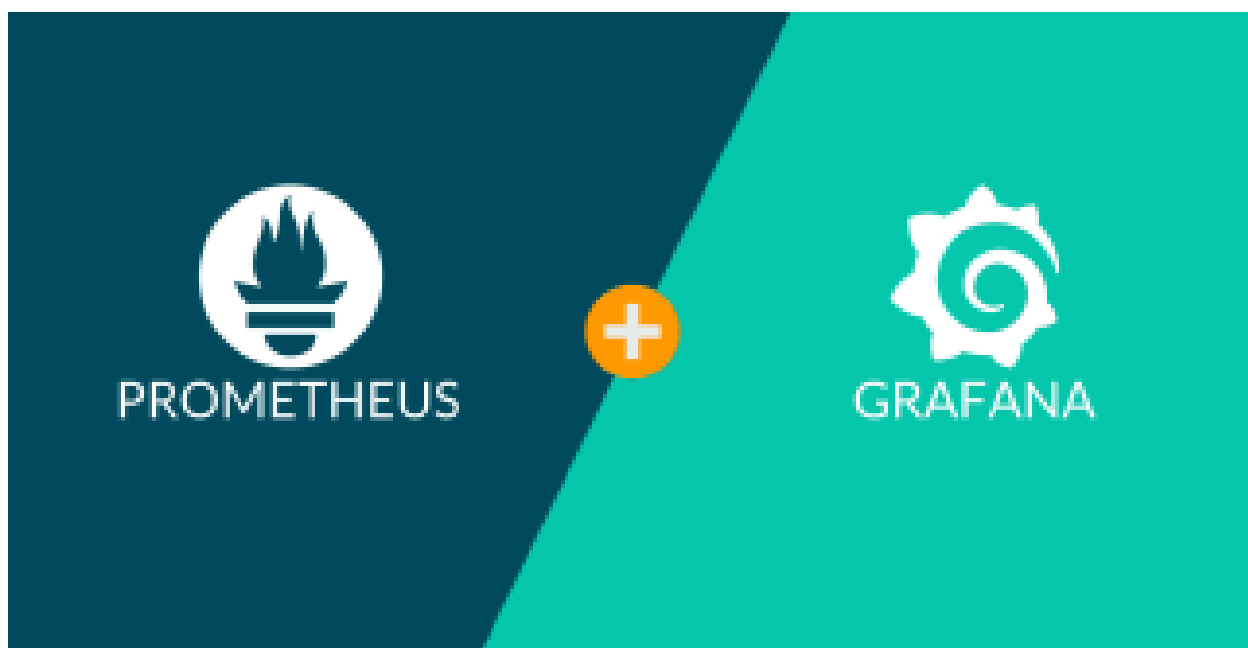




GESTION Y ADMINISTRACIÓN DE REDES

PROMETHEUS + GRAFANA



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Mario Serrano Romero

Índice

1.	Instalación de herramientas.....	3
1.1.	Prometheus.....	3
1.2.	Grafana.....	4
1.3.	Resto de herramientas.....	4
2.	Monitorización de un nodo desde otro.....	4
3.	Creación de dashboards.....	7
4.	Implementación de un nuevo nodo y su monitorización.....	10
5.	Monitorización de un servicio en el nodo.....	11
6.	Monitorización de las métricas extraídas de SNMP.....	13

Para la realización de esta práctica he tenido que realizar una serie de pasos, los primeros han sido instalarme todo lo que preveía que iba a ser necesario para la realización de esta y luego he comenzado a hacer cada uno de los hitos que se pedían. Aparte de lo que me instalé en un principio que fueron Prometheus y Grafana, he tenido que instalarme un par de cosas que han sido necesarias para terminar algunos de los apartados. Al ser dos programas que sirven para miles de usos he ido seleccionando las aplicaciones en lo que he querido entonces todas las instalaciones no son necesarias y algunas tienen otras alternativas. Como he dicho las principales son la de Prometheus y Grafana, las demás como Docker por ejemplo podría ser sustituida por una máquina virtual y se realizaría de manera diferente.

1. Instalación de herramientas.

1.1. Prometheus.

- Descargamos la última versión:

```
"wget https://github.com/prometheus/prometheus/releases/download/vX.X.X/prometheus-X.X.X.linux-amd64.tar.gz"
```

- Extraemos el archivo comprimido:

```
"tar xvfz prometheus-X.X.X.linux-amd64.tar.gz"
```

- Movemos los archivos del servidor de Prometheus a /usr/local/prometheus:

```
"sudo mv prometheus-X.X.X.linux-amd64 /usr/local/prometheus"
```

- Creamos un usuario para ejecutar el servidor de Prometheus:

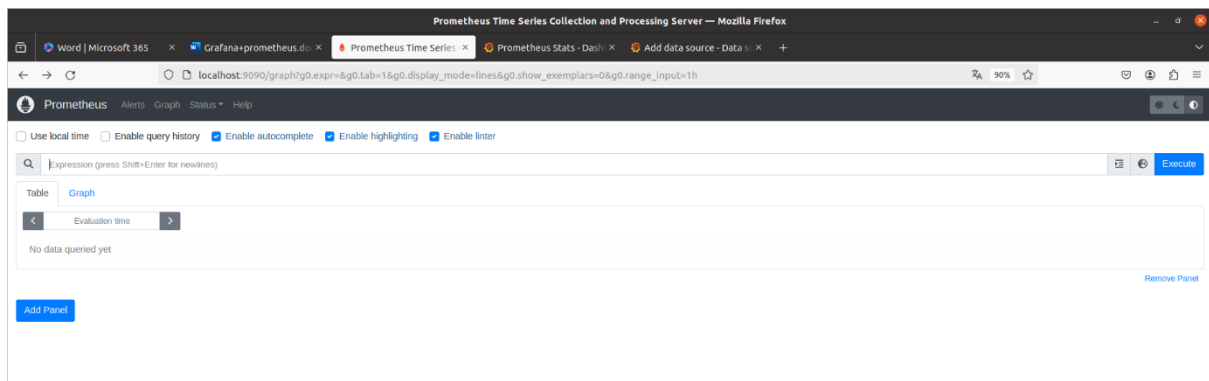
```
"sudo useradd --no-create-home --shell /bin/false prometheus"
```

- Establecemos los permisos necesarios:

```
"sudo mkdir /var/lib/prometheus"
```

```
"sudo chown prometheus:prometheus /var/lib/prometheus"
```

- Comprobamos que funciona ejecutando `“./prometheus”` dentro de la carpeta donde hayamos guardado prometheus que en este caso es /usr/local/prometheus/ y buscando en la dirección <http://localhost:9090>.



Nos debería salir algo así al acceder.

1.2. Grafana.

- Actualizamos el sistema:

“sudo apt upgrade”

“sudo apt update”

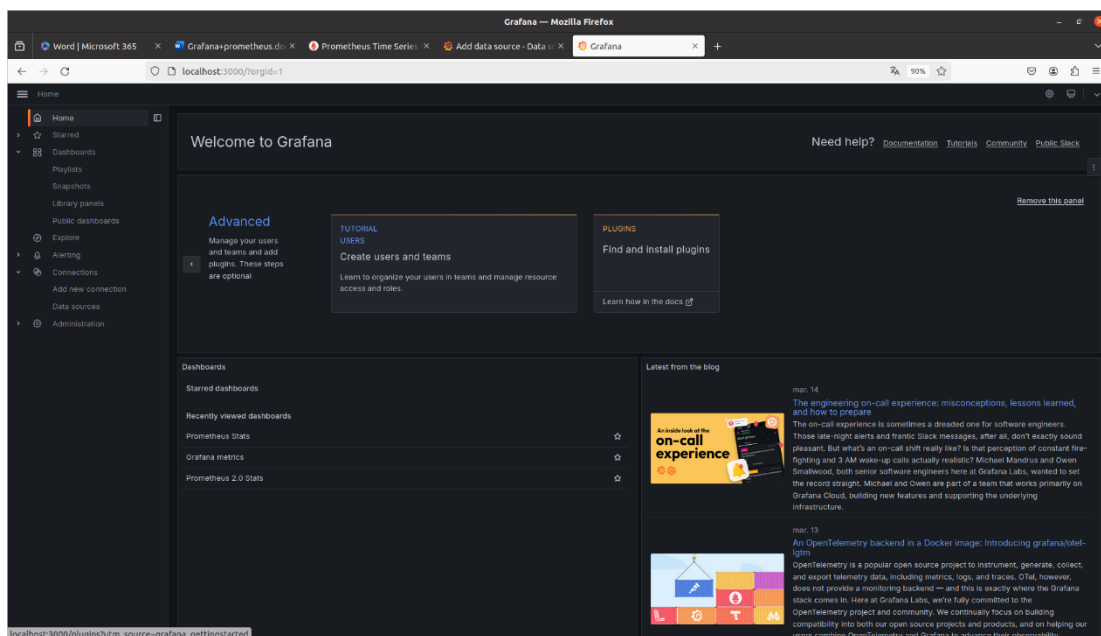
- Agregamos el repositorio:

```
“wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -echo "deb  
https://packages.grafana.com/oss/deb stable main" | sudo tee  
/etc/apt/sources.list.d/grafana.list”
```

- Instalamos:

“sudo apt install grafana”

- Iniciamos el servicio con “sudo systemctl start grafana-server” y podemos encontrarlo en la dirección que tiene por defecto <http://localhost:3000>.



Nos debería de salir algo así al acceder.

1.3. Resto de herramientas.

En mi caso además de estas 2 he tenido que instalar Docker y Apache, que posteriormente veremos para que las usamos. La manera de instalarlas nos aparece en internet, no voy a poner todos los comandos como he hecho con las 2 aplicaciones principales, ya que no son esenciales en este trabajo y podríamos usar muchas otras que tuvieran funciones similares.

2. Monitorización de un nodo desde otro.

Para la realización de esta parte vamos a seguir una serie de pasos ya que hay que configurar varias cosas además de instalarnos el “node-exporter”.

- Instalación node_exporter:

“wget

https://github.com/prometheus/node_exporter/releases/download/v1.2.2/node_exporter-1.2.2.linux-amd64.tar.gz”

- Extracción del archivo comprimido:

“tar xvfz node_exporter-1.2.2.linux-amd64.tar.gz”

- Movemos el binario a /usr/local/bin/:

“sudo mv node_exporter-1.2.2.linux-amd64/node_exporter /usr/local/bin/”

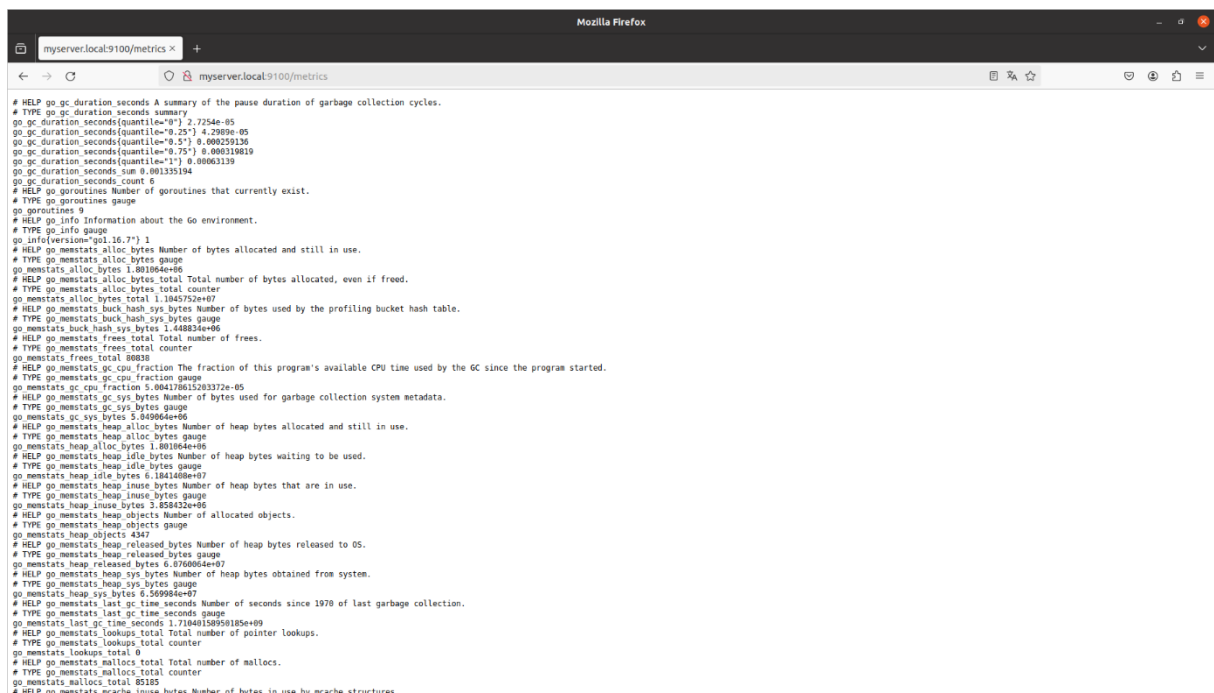
- Creamos un usuario para poder ejecutarlo:

“sudo useradd --no-create-home --shell /bin/false node_exporter”

- Establecemos los permisos necesarios:

“sudo chown node_exporter:node_exporter /usr/local/bin/node_exporter”

- Comprobamos que el servicio funciona correctamente situándonos dentro de /usr/local/bin/ con el comando “cd /usr/local/bin/” y ejecutando el comando “./node_exporter”.
- Una vez funcionando y comprobando que funciona en la dirección <http://localhost:9100/metrics> configuraremos la parte de Prometheus.



```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.7254e-05
go_gc_duration_seconds{quantile="0.25"} 4.2899e-05
go_gc_duration_seconds{quantile="0.5"} 0.000259236
go_gc_duration_seconds{quantile="0.75"} 0.000310819
go_gc_duration_seconds{quantile="1"} 0.00063139
go_gc_duration_seconds_sum 0.001353104
go_gc_duration_seconds_count 6
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.16.7"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.801064e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.1045752e+07
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.448834e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 80835
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 5.084178615283372e-05
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 5.849064e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.801064e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 6.1841408e+07
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.858432e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 4341
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 6.076086e+07
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 6.569986e+07
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 1.71040159595185e+09
# HELP go_memstats_lookups_total Total number of pointer lookups.
# TYPE go_memstats_lookups_total counter
go_memstats_lookups_total 0
# HELP go_memstats_mallocs_total Total number of mallocs.
# TYPE go_memstats_mallocs_total counter
go_memstats_mallocs_total 85185
# HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache structures.
```

Al acceder nos debería aparecer algo así.

- Editamos/creamos el archivo “prometheus.yml” con el siguiente comando y con el contenido de la foto:

“sudo nano /usr/local/prometheus/prometheus.yml”

```

GNU nano 4.8 prometheus.yml
# my global config
global:
  scrape_interval: 10s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 10s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# 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: 'node_exporter'
    static_configs:
      - targets: ['localhost:9090']

```

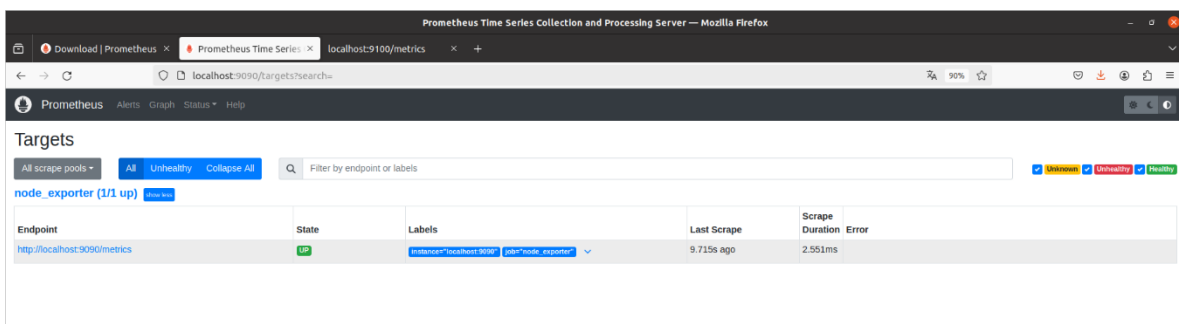
- Establecemos los permisos necesarios para este archivo:

“sudo chown prometheus:prometheus /usr/local/prometheus/prometheus.yml”

- Ahora simplemente ejecutamos prometheus con el siguiente comando:

“./prometheus --config.file=prometheus.yml”

- Una vez levantado el servicio tenemos que acceder a la dirección web de prometheus que he dicho antes y pulsar dentro en en Status > Targets, y una vez aquí, veremos los servicios que estamos monitoreando y que deberían estar activos (status = **UP**).

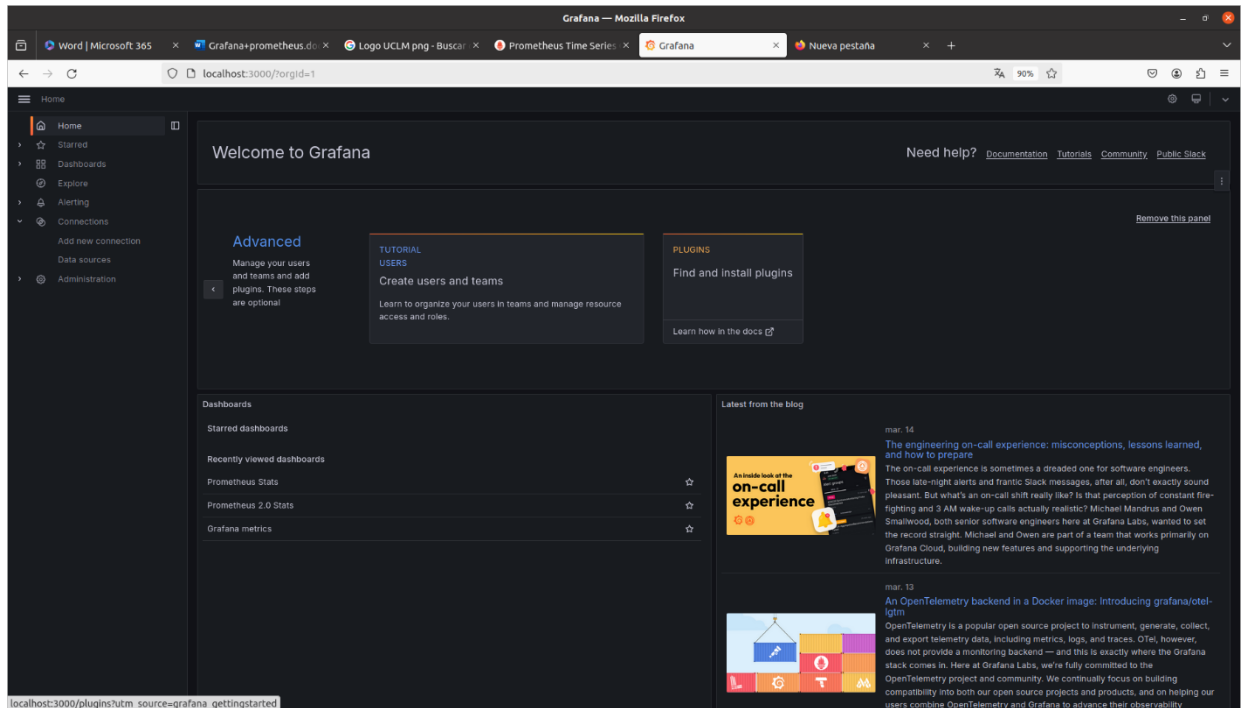


- Con esto realizado ya únicamente tendríamos que acceder en Grafana y donde quisiéramos ver la monitorización de las métricas del nodo, pondríamos como fuente de datos prometheus y nos aparecerán todas las métricas disponibles, no voy a entrar

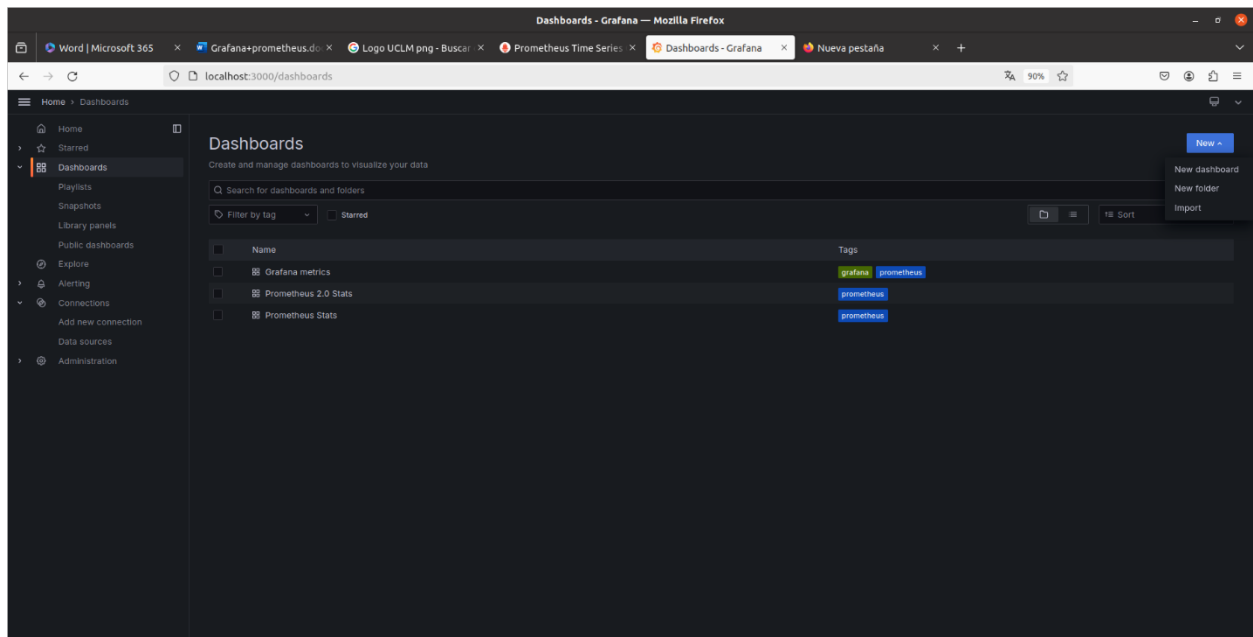
mucho en detalle ya que la siguiente parte consta de esta creación del dashboard donde podremos ver las métricas.

3. Creación de dashboards.

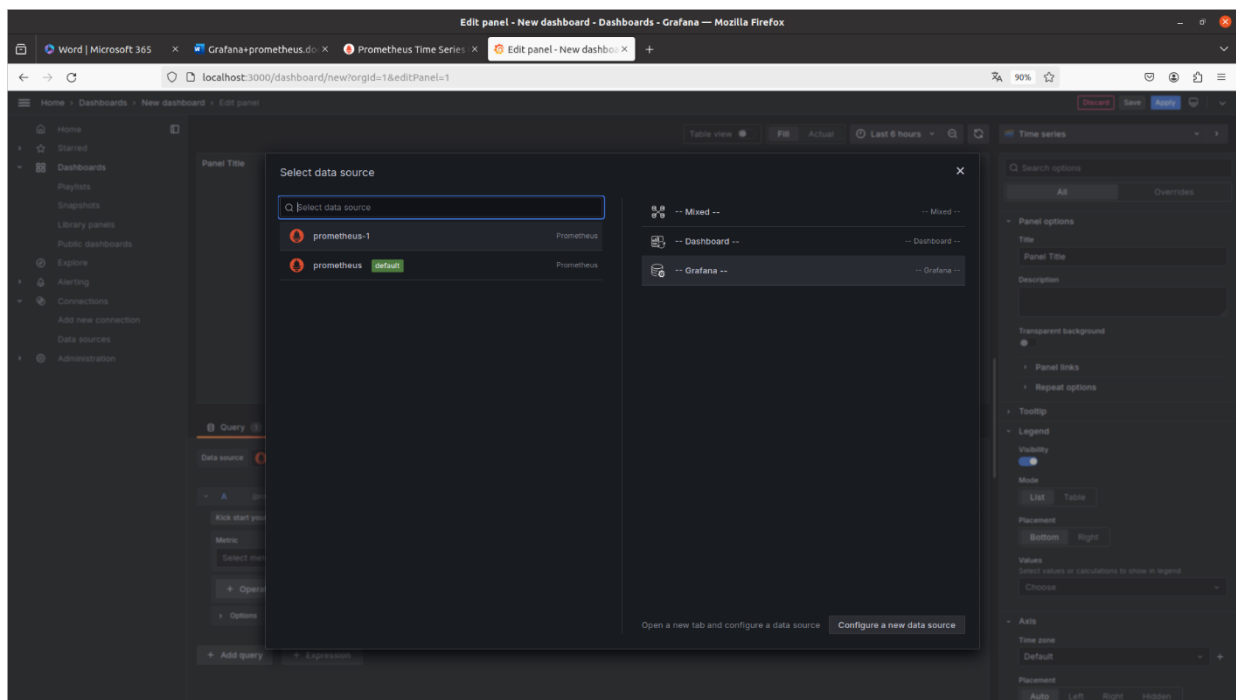
Una vez hayamos ejecutado grafana y accedamos su dirección nos saldrá una pantalla de inicio de sesión donde deberemos poner el usuario “admin” y contraseña “admin” y posteriormente nos dejará cambiar la contraseña. Seguidamente veremos esta pantalla que es la pantalla principal de grafana.



Una vez nos encontramos aquí tendremos que darle a dashboard en el menú que vemos en la izquierda. Ahora se nos abrirá esta pantalla:

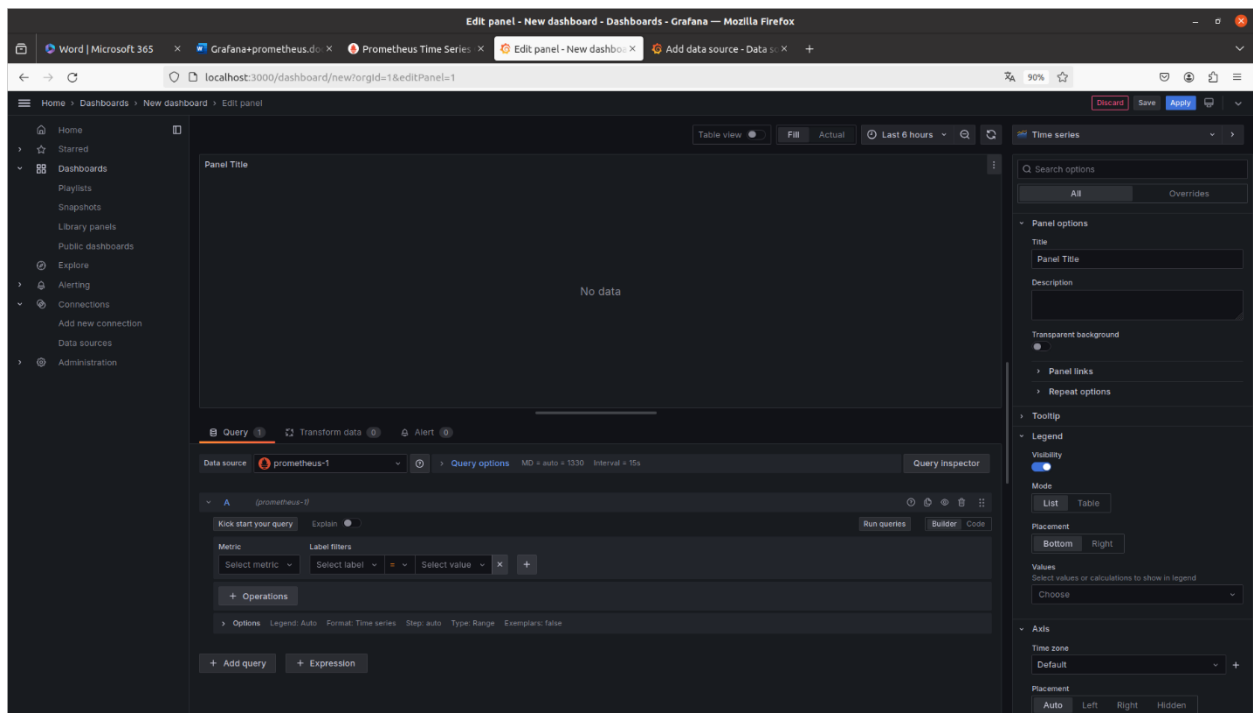


Una vez estemos aquí, nos aparecerá todo vacío ya que los dashboard que aparecen son los que he creado para la realización de esta práctica. Como podemos ver arriba a la derecha tenemos el botón "New" donde nos da varias opciones, pero vamos a centrarnos en la de crear un dashboard. Si clicamos ahí nos llevará a otra pantalla donde deberemos darle a "+add visualization" una vez clicado ahí se nos abrirá esta pantalla:

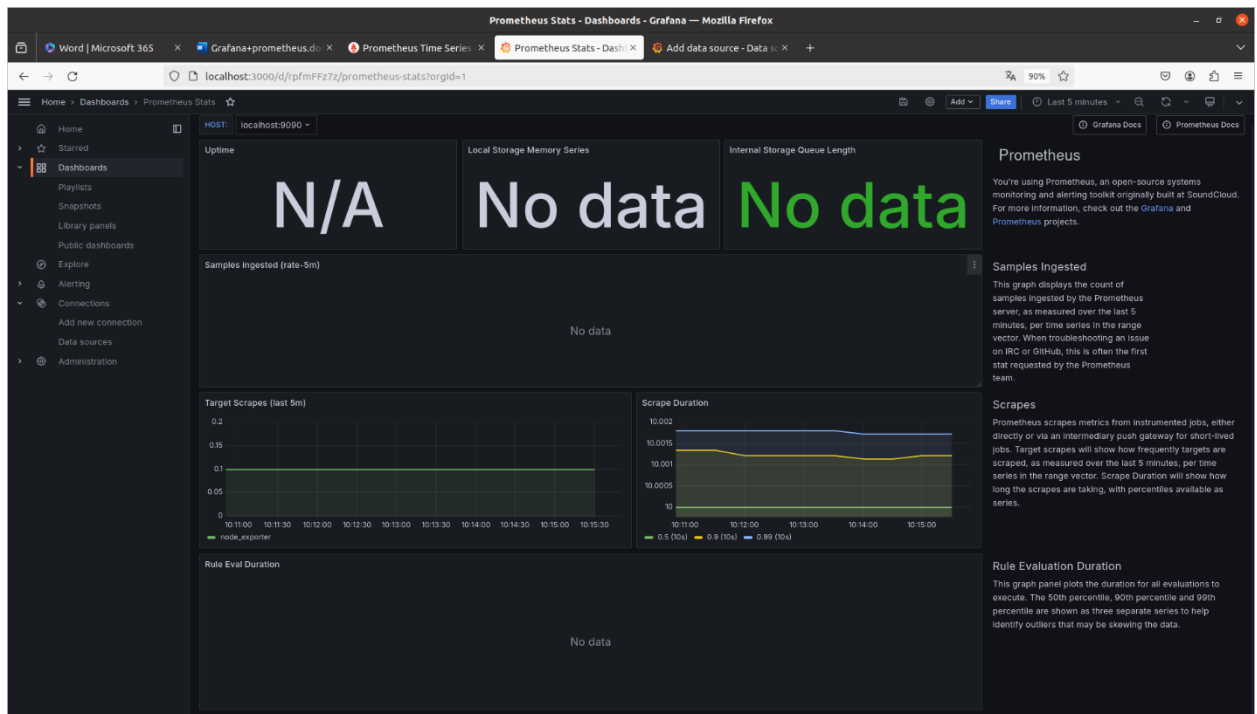


Esto también estaría vacío, sin las bases de datos de prometheus y tendremos que darle abajo a la derecha de la ventana que pone "Configure a new data source" y seleccionaremos la base de datos que necesitamos, añadiendo la URL donde se encuentre. En esta práctica utilizamos prometheus y por eso podemos ver que ya lo tengo configurado. Después de seleccionar la

base de datos se nos creara la primera visualización y se nos cerrara esta ventana, quedándose así:



Aquí ya viene la parte importante, donde empezamos a crear el dashboard. Voy a explicar de manera rápida y sencilla las cosas principales para crear un dashboard. Abajo como podemos ver tenemos la base de datos que hemos introducido anteriormente y podemos cambiarla en cualquier momento, más abajo podemos ver que tenemos para seleccionar el dato que queremos representar exactamente y se nos rellenara el panel con el mismo ya que ahora aparece no data porque no hemos cogido ningún dato aún. En la parte de la derecha se configuraría el propio panel y dependiendo del tipo que seleccionemos, tendrá unas características u otras. Para cambiar el tipo de panel, tenemos que clicar arriba a la derecha donde pone “Time series” y se nos desplegara un menú con todos los tipos de paneles que hay. Por último, más arriba tenemos 3 botones “Discard” que es para borrar el dashboard entero, “Save” que es para aplicar los cambios que hayamos realizado y guardar el dashboard y “Apply” que es para guardar los cambios y volver al dashboard donde ahí podremos añadir más paneles, cambiarlos de sitio... Voy a mostrar una imagen de ejemplo de un dashboard que viene ya creado para ver como quedaría y como se pueden mover y modificar los paneles de nuestro dashboard.



4. Implementación de un nuevo nodo y su monitorización.

Para esta parte había varias opciones de realizarlo, pero como en mi caso Docker ya lo he utilizado otras veces y lo tenía en mi ordenador ya, decidí implementar el nuevo nodo a través de Docker. Únicamente hay que usar el siguiente comando: “docker run -d --name=node_exporter2 -p 9100:9090 prom/node-exporter”. Con esto ya tendríamos nuestro segundo nodo creado en la dirección <http://localhost:9100metrics> y deberíamos acceder a la dirección para comprobar que funciona correctamente.

Una vez este el nodo funcionando solo tenemos que hacer un par de cambios dentro del archivo “prometheus.yml” que dejo la imagen a continuación y reiniciar el servicio de prometheus.

```
marlosr@marlosr-ROG-Strix-G531GT-G531GT: ~/Escritorio/UNI/GAR/prometheus
marlosr@marlosr-ROG-Strix-G531GT-G531GT: ~/Escritorio/UNI/GAR/prometh...
marlosr@marlosr-ROG-Strix-G531GT-G531GT: /usr/local/bin

GNU nano 4.8 prometheus.yml
# my global config
global:
  scrape_interval: 10s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 10s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# 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: 'node_exporter'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node_exporter2'
    static_configs:
      - targets: ['localhost:9100']
```

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
node_exporter (1/1 up)	UP	instance="localhost:9090" job="node_exporter"	3.539s ago	92.457ms	
node_exporter2 (1/1 up)	UP	instance="localhost:9100" job="node_exporter2"	1.986s ago	141.015ms	

5. Monitorización de un servicio en el nodo.

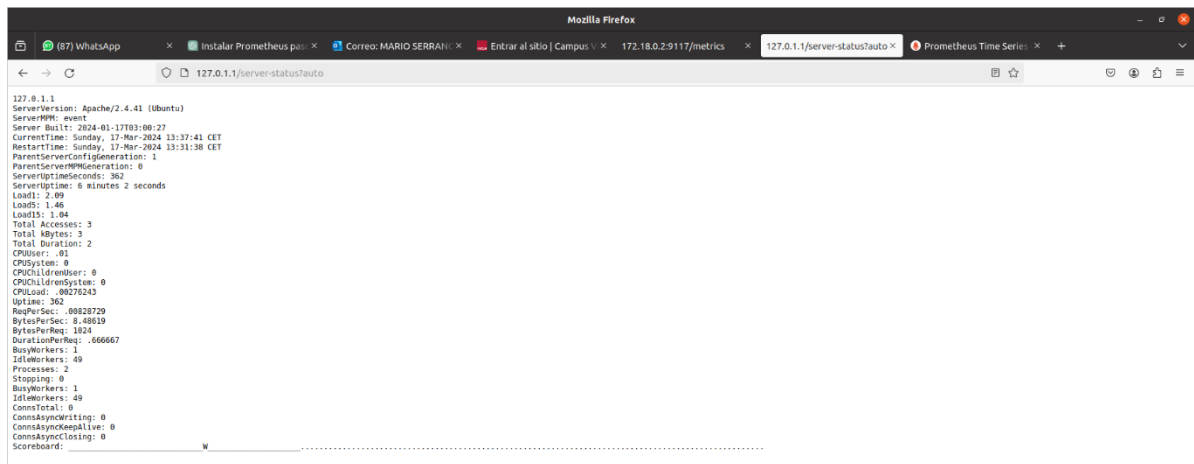
En esta parte tuve muchos problemas porque todas las herramientas complementarias para poder monitorizar un servicio en un nodo no están bien construidas o quizás sea un problema con mi ordenador. El caso es que en un principio iba a monitorizar el servicio dea pache, pero no conseguía levantar el Docker de apache_exporter, por lo que cambie de servicio y probe con mqtt y tampoco conseguía levantar el mqtt_exporter. Después de varias horas y varios intentos, encontré donde estaba el pequeño fallo del apache_exporter y volví a utilizarlo. Una vez estaba en funcionamiento, hice los cambios necesarios de prometheus y ya me aparecía, pero me aparece todo el rato en “Down” y cada vez que intento arreglarlo me da un error diferente. He probado de muchas maneras y finalmente pude sacarlo desinstalando apache y volviendolo a instalar, despues tuve que cambiar las direcciones ip y los puertos donde escucha el apache_exporter por defecto.

```

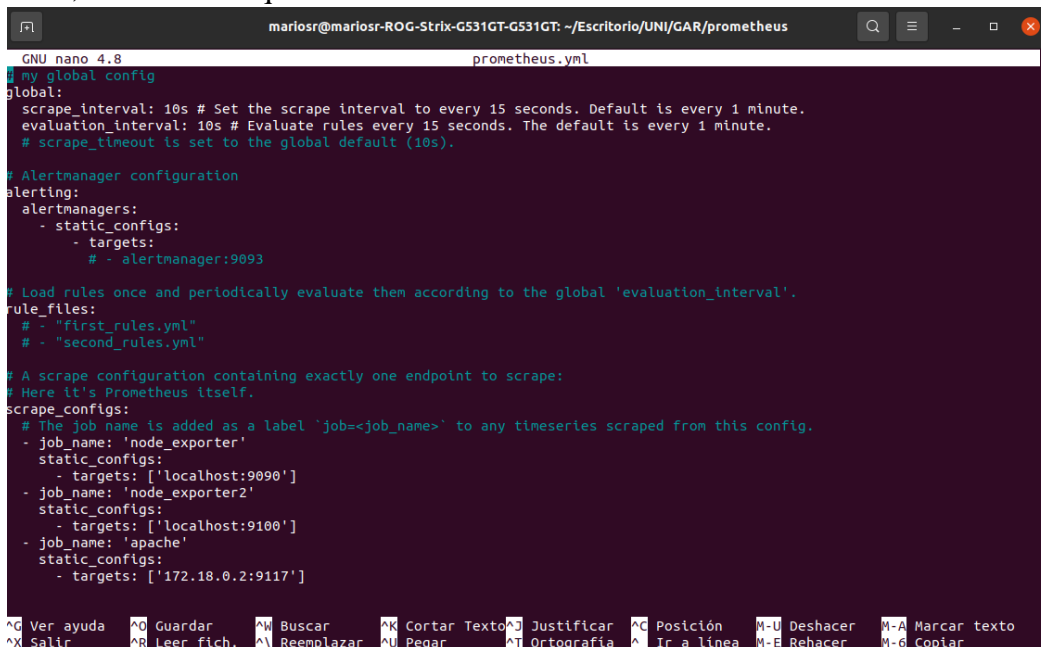
1 name: apache_exporter
2 services:
3   apache_exporter:
4     image: usioycon/apache-exporter
5     container_name: apache_exporter
6     privileged: true
7     ports:
8       - node: ingress
9         target: 9117
10        published: "9117"
11        protocol: tcp
12    restart: unless-stopped
13    extra_hosts:
14      - "localhost:127.0.0.1"
15    entrypoint: /bin/apache_exporter --scrape_uri="http://127.0.0.1:1/server-status?auto"

```

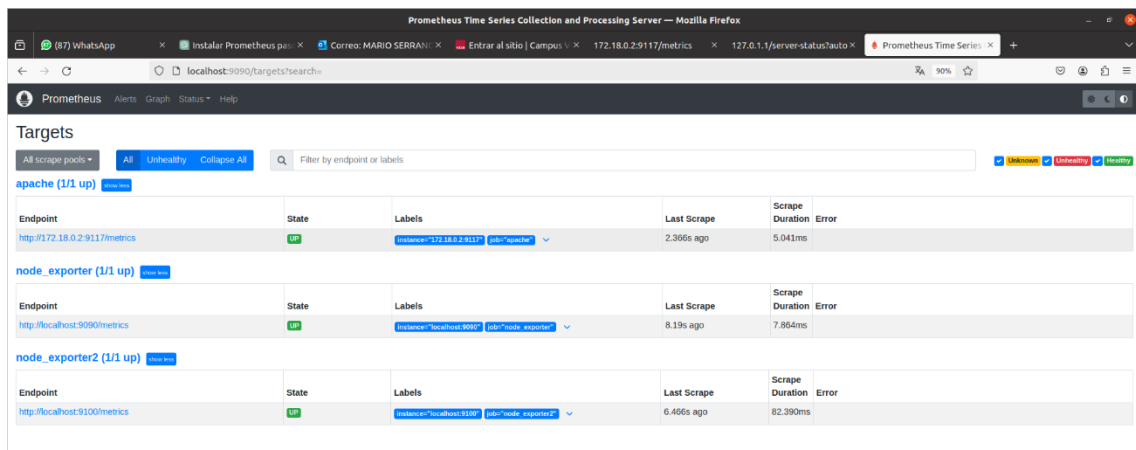
Una vez cambiado esto, el apache_exporter ya está recolectando métricas y guardándolas en su URL para que prometheus las pueda coger después. De la URL que podemos ver en el archivo de configuración de arriba podemos ver si la abrimos esto:



Luego tuve que cambiar el archivo de configuración de prometheus para que aparezca el nodo, el archivo se quedaría así:



Y como se quedaría la representación desde prometheus de los 3 nodos que llevamos hasta el momento. Luego para representarlos en grafana seria de la misma manera que hemos explicado antes.



6. Monitorización de las métricas extraídas de SNMP.

Esta parte debería ser parecida a la anterior, teniendo que configurar un par de archivos dentro de la carpeta que nos descargamos de github y posteriormente ejecutándolo y configurando el archivo de prometheus ya que una vez tengamos las cosas en prometheus será fácil. El problema es que el snmp_exporter no funciona y por más que he intentado no he conseguido arreglarlo. Aquí dejo una captura de los fallos:

```
mariosr@mariosr-ROG-Strix-G531GT-G531GT:~/Escritorio/UNI/GAR/snmp_exporter-main
go: gopkg.in/alecthomas/kingpin.v2@v2.4.0: parsing go.mod:
    module declares its path as: github.com/alecthomas/kingpin/v2
    but was required as: gopkg.in/alecthomas/kingpin.v2
mariosr@mariosr-ROG-Strix-G531GT-G531GT:~/Escritorio/UNI/GAR/snmp_exporter-main$ make
fatal: no es un repositorio git (ni ninguno de los directorios superiores): .git
>> checking code style
>> checking license header
>> running golanci-lint
go list -e -compiled -test=true -export=false -deps=true -find=false -tags= -- ./... > /dev/null
/home/mariosr/go/bin/golangci-lint run ./...
config/config.go:40:18:      err = yaml.UnmarshalStrict(content, cfg)
                        ^
                                (typecheck)
config/config.go:328:13: re.String undefined (type Regexp has no field or method String) (typecheck)
    return re.String(), nil
                ^
main.go:48:18: undefined: kingpin (typecheck)
    configFile = kingpin.Flag("config.file", "Path to configuration file.").Default("snmp.yml").Strings()
                ^
main.go:49:18: undefined: kingpin (typecheck)
    dryRun = kingpin.Flag("dry-run", "Only verify configuration is valid and exit.").Default("false").Bool()
                ^
main.go:50:18: undefined: kingpin (typecheck)
    concurrency = kingpin.Flag("snmp.module-concurrency", "The number of modules to fetch concurrently per scrape").Default("1").Int()
                ^
main.go:122:5: sc.Block undefined (type "SafeConfig" has no field or method Block) (typecheck)
    sc.Block()
    ^
main.go:125:6: sc.Runlock undefined (type "SafeConfig" has no field or method Runlock) (typecheck)
    sc.Runlock()
    ^
main.go:134:7: sc.Runlock undefined (type "SafeConfig" has no field or method Runlock) (typecheck)
    sc.Runlock()
    ^
main.go:141:5: sc.Runlock undefined (type "SafeConfig" has no field or method Runlock) (typecheck)
    sc.Runlock()
    ^
main.go:174:5: sc.Lock undefined (type "SafeConfig" has no field or method Lock) (typecheck)
    sc.Lock()
    ^
main.go:180:5: sc.Unlock undefined (type "SafeConfig" has no field or method Unlock) (typecheck)
    sc.Unlock()
    ^
main.go:337:6: sc.Lock undefined (type "SafeConfig" has no field or method Block) (typecheck)
    sc.Lock()
    ^
config_test.go:32:5: sc.Block undefined (type "SafeConfig" has no field or method Block) (typecheck)
    sc.Block()
    ^
collector/collector.go:337:24: module.WalkParams undefined (type "NamedModule" has no field or method WalkParams) (typecheck)
    g.Retries = "module.WalkParams.Retries
    ^
collector/collector.go:338:23: module.WalkParams undefined (type "NamedModule" has no field or method WalkParams) (typecheck)
    g.Timeout = module.WalkParams.Timeout
    ^
collector/collector.go:339:30: module.WalkParams undefined (type "NamedModule" has no field or method WalkParams) (typecheck)
```