

# UD 8 - Monitor Apache Kafka

## 1. Introducción

En un entorno de producción, la monitorización de Kafka (y cualquier otro de Big Data) es crucial para asegurar que el sistema esté funcionando de manera eficiente y sin interrupciones. Kafka es una plataforma de mensajería distribuida muy robusta, pero requiere una supervisión constante para garantizar que todos los componentes, como los brokers, controllers, productores y consumidores, estén operando correctamente. Sin una monitorización adecuada, es difícil detectar problemas de rendimiento, cuellos de botella, y errores en tiempo real que pueden afectar la calidad del servicio.

La monitorización de Kafka permite:

- **Rendimiento:** Evaluar las métricas de throughput, latencia y el uso de recursos para garantizar un rendimiento óptimo.
- **Escalabilidad:** Detectar problemas antes de que afecten a la escalabilidad del sistema.
- **Detección de fallos:** Identificar problemas potenciales con los brokers, consumidores o productores que puedan estar interfiriendo con el flujo de datos.
- **Optimización:** Tomar decisiones informadas sobre la configuración y la infraestructura para mejorar la eficiencia del sistema.

En esta guía, exploraremos cómo monitorizar un clúster de Kafka utilizando **JMX (Java Management Extensions)** para exponer métricas internas de Kafka, **Prometheus** para recolectar estas métricas, y **Grafana** para visualizar las métricas en paneles interactivos.

## 2. ¿Qué es JMX?

**JMX (Java Management Extensions)** es una tecnología de Java que permite gestionar y monitorizar aplicaciones, dispositivos y servicios basados en Java. Kafka, al ser una aplicación basada en Java, expone muchas de sus métricas internas a través de JMX.

JMX permite obtener métricas valiosas como el uso de CPU, la memoria, el rendimiento de los brokers, el rendimiento de los consumidores y los productores, la latencia, el número de particiones y más. Estas métricas son fundamentales para la administración eficiente de un clúster de Kafka.

### 3. ¿Qué es Prometheus?

**Prometheus** es una herramienta de monitoreo y recopilación de métricas de código abierto, diseñada para almacenar series temporales de datos. Funciona extrayendo métricas de las aplicaciones a través de un formato estándar que puede ser consultado mediante su lenguaje de consulta, PromQL.

Prometheus es ampliamente utilizado para la monitorización de sistemas distribuidos como Kafka debido a su capacidad para recolectar y almacenar grandes volúmenes de métricas de manera eficiente y en tiempo real.

#### Características principales de Prometheus:

- **Recolección de métricas basada en scrapes:** Prometheus obtiene métricas mediante el mecanismo de scraping desde los endpoints HTTP de las aplicaciones.
- **Modelo de datos multidimensional:** Las métricas son etiquetadas, lo que permite una rica semántica de consulta y filtrado.
- **Escalabilidad:** Puede recolectar y almacenar grandes cantidades de datos sin comprometer el rendimiento.
- **Integración con Grafana** (*además de otros servicios*): Prometheus se integra perfectamente con Grafana para crear dashboards visuales.

### 4. ¿Qué es Grafana?

**Grafana** es una plataforma de código abierto para la visualización y análisis de métricas. Es compatible con diversas fuentes de datos, y se usa principalmente para la visualización de series temporales de datos.

Cuando se combina con Prometheus, Grafana se convierte en una herramienta poderosa para representar visualmente las métricas de sistemas distribuidos como Kafka (**y muchísimos otros**). Con Grafana, puedes crear paneles interactivos y gráficos que muestran el rendimiento de Kafka en tiempo real, facilitando la detección de anomalías o problemas.

#### Características principales de Grafana:

- **Visualización de datos en tiempo real:** Ofrece gráficos interactivos que se actualizan en tiempo real.
- **Paneles personalizables:** Los usuarios pueden crear paneles personalizados según sus necesidades.
- **Soporte para múltiples fuentes de datos:** Grafana puede conectarse a una amplia variedad de fuentes de datos, como Prometheus, Elasticsearch, InfluxDB, entre otras.

## 5. ¿Por qué usar JMX, Prometheus y Grafana para la monitorización de Kafka?

La combinación de **JMX**, **Prometheus** y **Grafana** es una solución altamente eficiente para la monitorización de Kafka por las siguientes razones:

1. **JMX** permite acceder directamente a las métricas internas de Kafka, proporcionando una visión profunda de su rendimiento y estado.
2. **Prometheus** es ideal para recolectar y almacenar estas métricas de manera eficiente, soportando grandes volúmenes de datos y permitiendo consultas de alto rendimiento con PromQL.
3. **Grafana** es una herramienta muy completa para visualizar las métricas recolectadas, permitiendo una monitorización en tiempo real con dashboards altamente personalizables y fáciles de usar.

Esta combinación ofrece una solución completa y escalable para monitorizar un clúster de Kafka, ayudando a los administradores a detectar problemas rápidamente, optimizar el rendimiento y garantizar la disponibilidad continua del sistema.

## 6 Funcionamiento JMX + Prometheus + Grafana

### 6.1 Conectando JMX y Prometheus

Sin entrar en mucho detalle, el flujo de monitorización con JMX, Prometheus y Grafana es el siguiente:

Kafka expone una variedad de métricas a través de JMX, las cuales están relacionadas con los brokers, productores, consumidores, etc.

Por otro lado, Prometheus es un ecosistema con dos componentes principales: - el componente del lado del servidor - la configuración del lado del cliente.

El componente del lado del servidor se encarga de almacenar todas las métricas y de realizar el scraping de todos los clientes. *Prometheus se diferencia de servicios como Elasticsearch y Splunk, que suelen utilizar un componente intermedio responsable de extraer los datos de los clientes y enviarlos a los servidores.* Dado que no hay ningún componente intermedio que extraiga las métricas de Prometheus, todas las configuraciones relacionadas con las encuestas están presentes en el propio servidor.

El proceso tiene este aspecto:

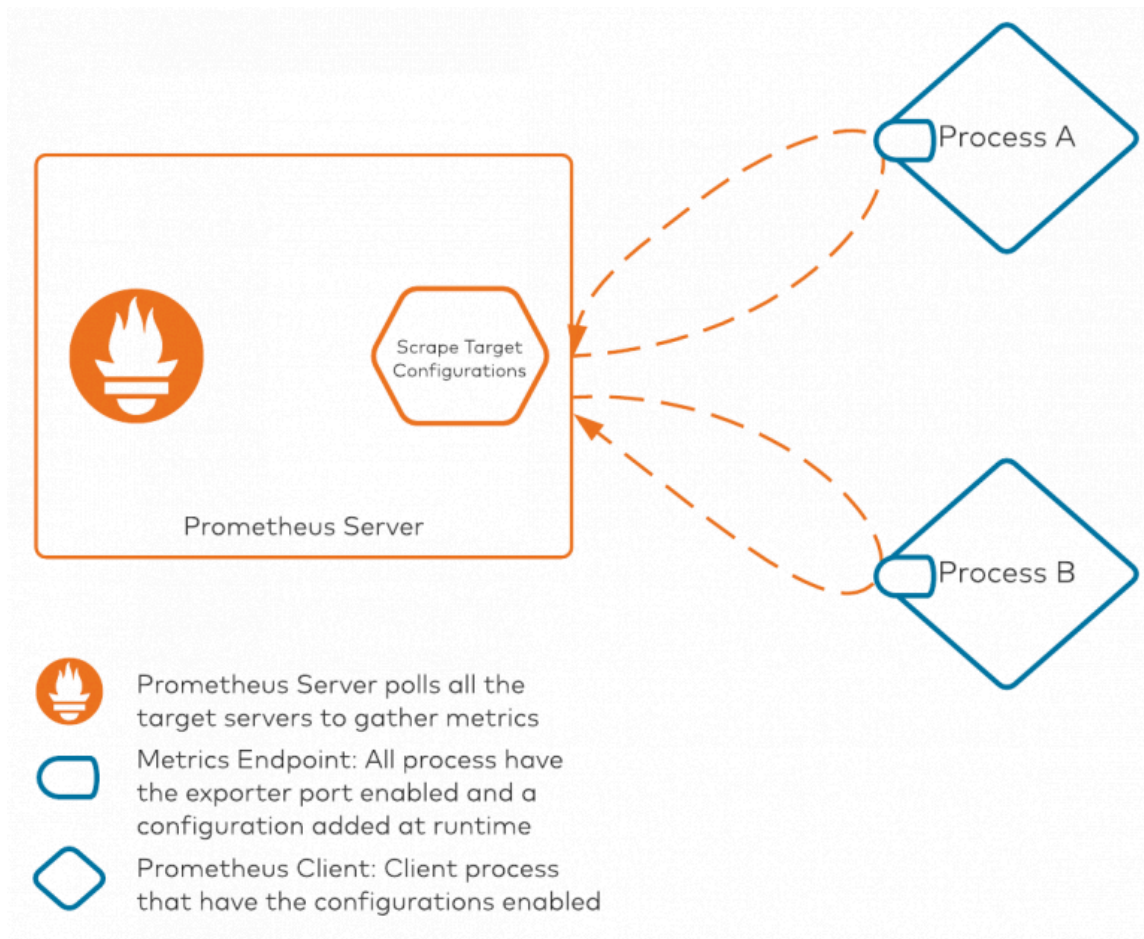


Figura 8.1\_Monitor\_Kafka: Prometheus (Fuente: confluent)

Hay dos piezas centrales en este diagrama:

- **Servidor Prometheus:** Este componente es responsable de sondear todos los procesos/clientes con sus métricas expuestas en un puerto específico. El servidor de Prometheus mantiene internamente un archivo de configuración que enumera todas las direcciones IP/nombres de host del servidor y los puertos en los que se exponen las métricas de Prometheus. La configuración de objetivos de scrape es el archivo que mantiene toda la asignación de objetivos dentro de Prometheus. Los objetivos de scrape son necesarios cuando estamos desplegando todo manualmente sin ningún tipo de automatización. Prometheus también admite módulos de descubrimiento de servicios, que puede aprovechar para descubrir cualquier servicio disponible que esté exponiendo métricas.
- **Procesos cliente:** Todos los clientes que quieran aprovechar Prometheus necesitarán dos piezas de configuración. En primer lugar, deben utilizar la biblioteca de clientes de Prometheus para exponer métricas en un formato compatible con Prometheus (OpenMetrics). En segundo lugar, deben utilizar un archivo de configuración YAML para extraer métricas JMX. Este archivo de configuración se utiliza para convertir, renombrar y filtrar algunos de los atributos para su consumo.

## 6.2 Connecting Grafana a Prometheus

Ahora que tenemos nuestros datos de métricas transmitiéndose al servidor Prometheus, podemos empezar a crear paneles de control de nuestras métricas. La herramienta elegida en nuestra pila es Grafana. Conceptualmente, este es el aspecto que tendrá el proceso una vez que hayamos conectado Grafana a Prometheus:

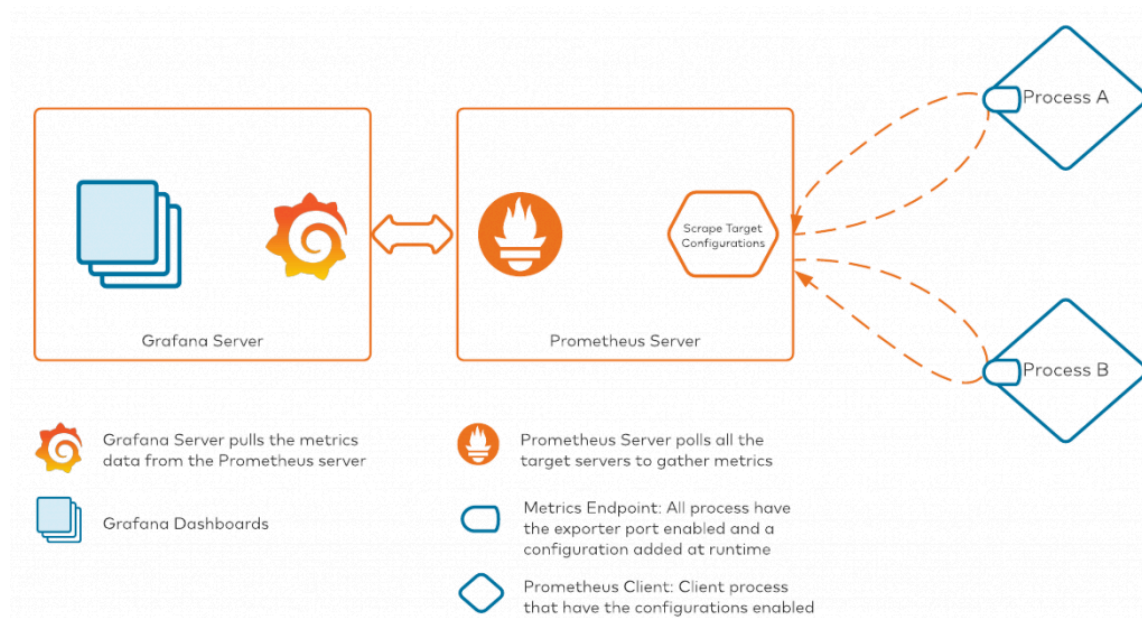


Figura 8.2\_Monitor\_Kafka: Connecting Prometheus y Grafana (Fuente: confluent)

Hay dos maneras de conectar Grafana con Prometheus: Podemos configurar la conexión desde la GUI de Grafana, o podemos añadir los detalles de la conexión a las configuraciones de Grafana antes del arranque. Hay artículos muy detallados disponibles en [la documentación de Grafana](#). Seguiremos esta documentación en el siguiente apartado donde explicamos la instalación y configuración.

## 7. Configuración de JMX + Prometheus

### 7.1 Habilitar JMX en Kafka

1. En primer lugar, necesitamos añadir la librería JMX Exporter. [Descarga](#) la última versión.

```
1  wget
   https://github.com/prometheus/jmx_exporter/releases/download/1.2.0/jmx_prometheus
   1.2.0.jar
```

2. Guardamos la librería en el directorio de librerías de Kafka

```
1 mv jmx_prometheus_javaagent-1.2.0.jar /opt/kafka_2.13-4.0.0/libs
```

3. Necesitamos configurar nuestro JMX Exporter para que sepa lo que va a extraer de Kafka. Para explicarlo brevemente, la configuración es una colección de regexps que nombra y filtra las métricas para Prometheus. Gracias a Prometheus, tenemos configuraciones de ejemplo su [repositorio de Github](#). Usaremos la configuración de ejemplo mas reciente ( `kafka-kraft-3_0_0.yml` ) en esta configuración.

```
1 wget
https://raw.githubusercontent.com/prometheus/jmx_exporter/refs/heads/main/example
kraft-3_0_0.yml
```

4. La movemos a la carpeta de configuración de kafka

```
1 mv kafka-kraft-3_0_0.yml /opt/kafka_2.13-4.0.0/config/jmx-exporter-
kafka.yml
```

5. Añadimos JMX a Kafka. Para ello, tenemos que indicarlo cada vez que usamos kafka. Por tanto, debemos abrir el script de inicio de los servidores de kafka ( `kafka-server-start.sh` ) y añadir JMX Exporter con su configuración correspondiente.

6. Primero hacemos una copia de respaldo y abrimos el script

```
1 cp /opt/kafka_2.13-4.0.0/bin/kafka-server-start.sh /opt/kafka_2.13-
4.0.0/bin/kafka-server-start.sh.bak
2 nano /opt/kafka_2.13-4.0.0/bin/kafka-server-start.sh
```

7. Añadimos al final la configuración de JMX Exporter. Debemos colocar esta línea antes de la linea de la ejecución de `exec $base_dir/kafka-run-class.sh $EXTRA_ARGS` `kafka.Kafka "$@"` . Por tanto, la colocamos, por ejemplo, al principio. *Hemos elegido el puerto **9091** para exponer los datos a Prometheus. Puedes usar el que quieras*

```
1 export KAFKA_OPTS="-javaagent:/opt/kafka_2.13-
4.0.0/libs/jmx_prometheus_javaagent-1.2.0.jar=9091:/opt/kafka_2.13-
4.0.0/config/jmx-exporter-kafka.yml"
```

8. En el caso de que tuviéramos kafka en ejecución, lo reiniciamos.

```
1 sudo systemctl daemon-reload
2 sudo systemctl restart kafka
```

## 7.2 Prometheus

1. [Descarga](#) la última versión LTS de Prometheus

```
1  wget
   https://github.com/prometheus/prometheus/releases/download/v2.53.4/prometheus-
   2.53.4.linux-amd64.tar.gz
2  tar -xzf prometheus-2.53.4.linux-amd64.tar.gz
```

2. Movemos el directorio a nuestro directorio `/opt` para una correcta organización

```
1  sudo mv prometheus-2.53.4.linux-amd64 /opt/prometheus-2.53.4
```

3. Accedemos al directorio de Prometheus

```
1  ls /opt/prometheus-2.53.4
```

4. Vemos que existe un fichero de configuración `prometheus.yml`. Hacemos una copia de respaldo

```
1  cp /opt/prometheus-2.53.4/prometheus.yml /opt/prometheus-
   2.53.4/prometheus.yml.bak
```

5. Abrimos el archivo de configuración

```
1  nano /opt/prometheus-2.53.4/prometheus.yml
```

6. Como vemos, ya existe una configuración de scraping del propio prometheus en `job_name`. Como puedes observar, para añadir diferentes *scrapings* (en este caso para nuestro JMX Exporter de Kafka), debemos añadir un configurar en un nuevo `job_name`. Añadimos la configuración indicando también el correspondiente endpoint en el puerto 9091.

7. Por tanto, añadimos debajo (recuerda respetar tabulaciones y formatos)

```
1  - job_name: "kafka"
2
3  # metrics_path defaults to '/metrics'
4  # scheme defaults to 'http'.
5
6  static_configs:
7    - targets: ["localhost:9091"]
```

8. El resultado sería.

```
1  # my global config
2  global:
3    scrape_interval: 15s # Set the scrape interval to every 15 seconds.
   Default is every 1 minute.
4    evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
   is every 1 minute.
5    # scrape_timeout is set to the global default (10s).
```

```

6
7 # Alertmanager configuration
8 alerting:
9   alertmanagers:
10     - static_configs:
11       - targets:
12         # - alertmanager:9093
13
14 # Load rules once and periodically evaluate them according to the global
15 # 'evaluation_interval'.
16 rule_files:
17   # - "first_rules.yml"
18   # - "second_rules.yml"
19
20 # A scrape configuration containing exactly one endpoint to scrape:
21 # Here it's Prometheus itself.
22 scrape_configs:
23   # The job name is added as a label `job=<job_name>` to any timeseries
24   # scraped from this config.
25   - job_name: "prometheus"
26
27     # metrics_path defaults to '/metrics'
28     # scheme defaults to 'http'.
29
30     static_configs:
31       - targets: ["localhost:9090"]
32
33   - job_name: "kafka"
34
35     # metrics_path defaults to '/metrics'
36     # scheme defaults to 'http'.
37
38     static_configs:
39       - targets: ["localhost:9091"]

```

9. En el caso de que tuviéramos prometheus en ejecución, lo reiniciamos.

```

1 sudo systemctl daemon-reload
2 sudo systemctl restart prometheus

```

## 8. Ejemplo 1. JMX + Prometheus

1. Antes de seguir con la configuración con Grafana, vamos a comprobar que nuestra configuración es correcta. Lanzamos un cluster de kafka en modo server y vemos si JMX Exporter da métricas correctamente y éstas son recogidas por prometheus.

2. Iniciamos Kafka

```

1 #Genera un cluster UUID
2 KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
3

```



```

4 #Si no te permite formatear (con el comando siguiente) es debido a que se
mantienen los logs del ejemplo anterior. debes borrarlos
5 #sudo rm -r /tmp/kraft-combined-logs/
6
7 #Formatea el directorio de log
8 bin/kafka-storage.sh format --standalone -t $KAFKA_CLUSTER_ID -c
config/server.properties
9
10 #Ejecuta el servidor Kafka
11 bin/kafka-server-start.sh config/server.properties

```

3. Comprobamos que se ha expuesto correctamente el endpoint en 9091 :

```
1 sudo ss -tunelp | grep 9091
```

```

1 tcp    LISTEN 0      3            *:9091      *:~ users:
(("java",pid=9692,fd=115)) uid:1000 ino:34770 sk:f cgroup:/user.slice/user-
1000.slice/session-1.scope v6only:0 <->

```

4. Y comprobamos que JMX Exporter está funcionando correctamente. Podemos hacerlo con curl o en el navegador en el puerto :9091/metrics

```
1 curl http://localhost:9091/metrics
```

```

1 # TYPE jmx_config_reload_failure_total counter
2 jmx_config_reload_failure_total 0.0
3 # HELP jmx_config_reload_success_total Number of times configuration have
successfully been reloaded.
4 # TYPE jmx_config_reload_success_total counter
5 jmx_config_reload_success_total 0.0
6 # HELP jmx_exporter_build_info JMX Exporter build information
7 # TYPE jmx_exporter_build_info gauge
8 jmx_exporter_build_info{name="jmx_prometheus_javaagent",version="1.2.0"}
1
9 # HELP jmx_scrape_cached_beans Number of beans with their matching rule
cached
10 # TYPE jmx_scrape_cached_beans gauge
11 jmx_scrape_cached_beans 0.0
12 # HELP jmx_scrape_duration_seconds Time this JMX scrape took, in seconds.
13 # TYPE jmx_scrape_duration_seconds gauge
14 jmx_scrape_duration_seconds 0.667689572
15 # HELP jmx_scrape_error Non-zero if this scrape failed.
16 # TYPE jmx_scrape_error gauge
17 jmx_scrape_error 0.0
18 # HELP jvm_buffer_pool_capacity_bytes Bytes capacity of a given JVM
buffer pool.
19 # TYPE jvm_buffer_pool_capacity_bytes gauge
20 jvm_buffer_pool_capacity_bytes{pool="direct"} 1574526.0
21 jvm_buffer_pool_capacity_bytes{pool="mapped"} 2.0971516E7
22 jvm_buffer_pool_capacity_bytes{pool="mapped - 'non-volatile memory'"} 0.0
23 # HELP jvm_buffer_pool_used_buffers Used buffers of a given JVM buffer
pool.
24 # TYPE jvm_buffer_pool_used_buffers gauge

```

```

25 jvm_buffer_pool_used_buffers{pool="direct"} 11.0
26 jvm_buffer_pool_used_buffers{pool="mapped"} 2.0
27 jvm_buffer_pool_used_buffers{pool="mapped - 'non-volatile memory'"} 0.0
28 # HELP jvm_buffer_pool_used_bytes Used bytes of a given JVM buffer pool.
29 # TYPE jvm_buffer_pool_used_bytes gauge
30 jvm_buffer_pool_used_bytes{pool="direct"} 1574526.0
31 jvm_buffer_pool_used_bytes{pool="mapped"} 2.0971516E7
32 jvm_buffer_pool_used_bytes{pool="mapped - 'non-volatile memory'"} 0.0
33 # HELP jvm_classes_currently_loaded The number of classes that are
    currently loaded in the JVM
34 # TYPE jvm_classes_currently_loaded gauge
35 jvm_classes_currently_loaded 9132.0

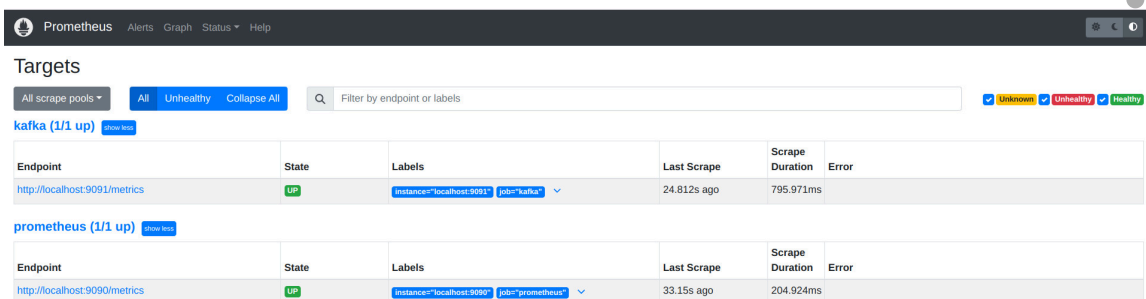
```

5. Iniciamos Prometheus. Vamos a su directorio `/opt/prometheus-2.53.4`. Levantamos prometheus

```
1 ./prometheus --config.file=prometheus.yml
```

6. Comprobamos que prometheus captura correctamente el endpoint

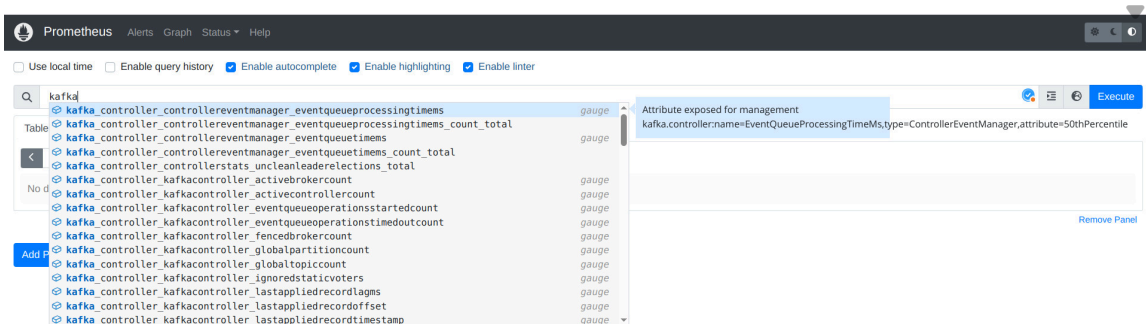
`http://192.168.56.10:9090/targets` (recuerda que accedemos desde nuestro host con esta configuración de red a nuestra máquina)



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<b>kafka (1/1 up)</b>					
http://localhost:9091/metrics	UP	instance="localhost:9091" job="kafka"	24.812s ago	795.971ms	
<b>prometheus (1/1 up)</b>					
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	33.15s ago	204.924ms	

Figura 8.3\_Monitor\_Kafka: Ejemplo1: WebUI Prometheus

7. También podemos comprobar que los datos de kafka se capturan correctamente desde Prometheus buscando `kafka`.



Query	Type
kafka_controller.controller.eventmanager.eventqueueprocessingtimes	gauge
kafka_controller.controller.eventmanager.eventqueueprocessingtimes_count_total	gauge
kafka_controller.controller.eventmanager.eventqueuetimes	gauge
kafka_controller.controller.eventmanager.eventqueuetimes_count_total	gauge
kafka_controller.controllerstats.uncleanleaderelections_total	gauge
kafka_controller.kafkacontroller.activebrokercount	gauge
kafka_controller.kafkacontroller.activecontrollercount	gauge
kafka_controller.kafkacontroller.eventqueueoperationsstartedcount	gauge
kafka_controller.kafkacontroller.eventqueueoperationstimedoutcount	gauge
kafka_controller.kafkacontroller.fencedbrokercount	gauge
kafka_controller.kafkacontroller.globalpartitioncount	gauge
kafka_controller.kafkacontroller.globaltopiccount	gauge
kafka_controller.kafkacontroller.ignoredstaticvoters	gauge
kafka_controller.kafkacontroller.lastappliedrecordlags	gauge
kafka_controller.kafkacontroller.lastappliedrecordoffset	gauge
kafka_controller.kafkacontroller.lastappliedrecordtimestamp	gauge

Figura 8.4\_Monitor\_Kafka: Ejemplo1: WebUI Prometheus 2

8. Puedes elegir cualquiera de las métricas

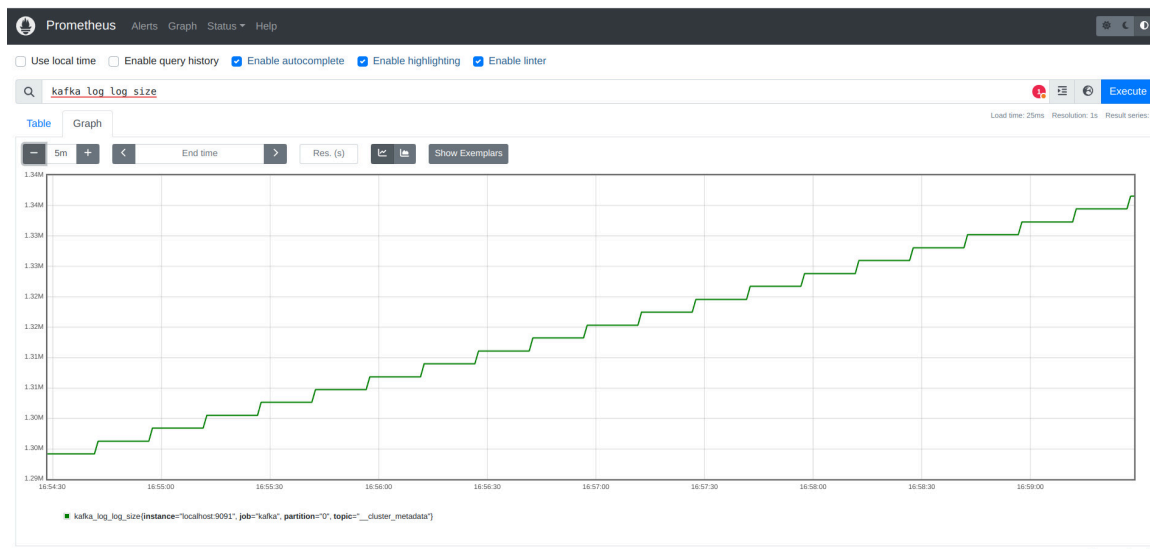


Figura 8.5\_Monitor\_Kafka: Ejemplo1: WebUI Prometheus 3

## 9. Configuración de Grafana

1. Toda la documentación de Grafana OSS la tenemos en su [página oficial](#)
2. [Descarga](#) e instala la última versión de Grafana.

```
1 sudo apt-get install -y adduser libfontconfig1 musl
2 wget https://dl.grafana.com/enterprise/release/grafana-
  enterprise_11.6.1_amd64.deb
3 sudo dpkg -i grafana-enterprise_11.6.1_amd64.deb
```

### 3. Inicia Grafana

```
1 systemctl start grafana-server
```

#### Grafana como servicio

Si quieres habilitar grafana para que arranque automáticamente usando `systemd`, ejecuta:

```
1 sudo systemctl daemon-reload
2 sudo systemctl enable grafana-server
```

4. Accedemos a su WebUI en el **puerto 3000** `http://localhost:3000`, en nuestro caso `http://192.168.56.10:3000/` desde el host
5. Entramos con las credenciales usuario `admin` y pass `admin`. Cámbiala a tu criterio.

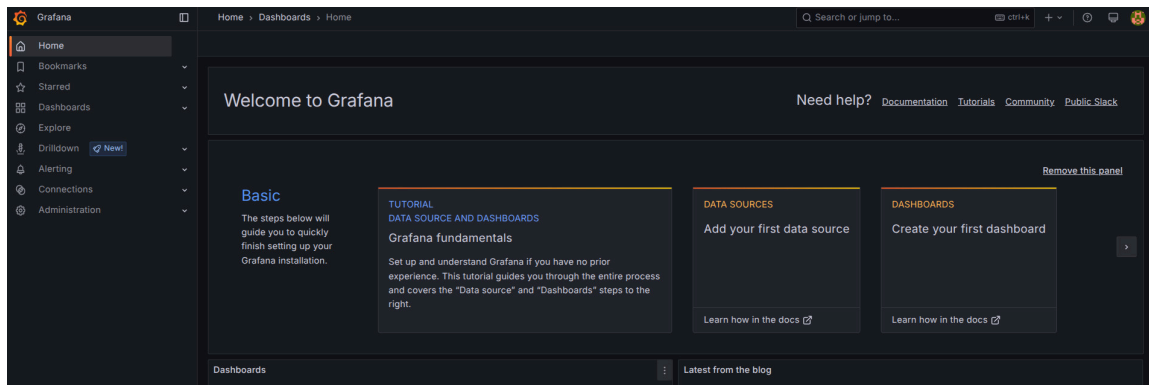


Figura 8.6\_Monitor\_Kafka: Grafana y Prometheus 1

1. Vamos a añadir los datos de monitorización de Kafka (Recuerda tener levantado kafka y prometheus)
2. Vamos a `Connections -> Data sources -> Add data source`

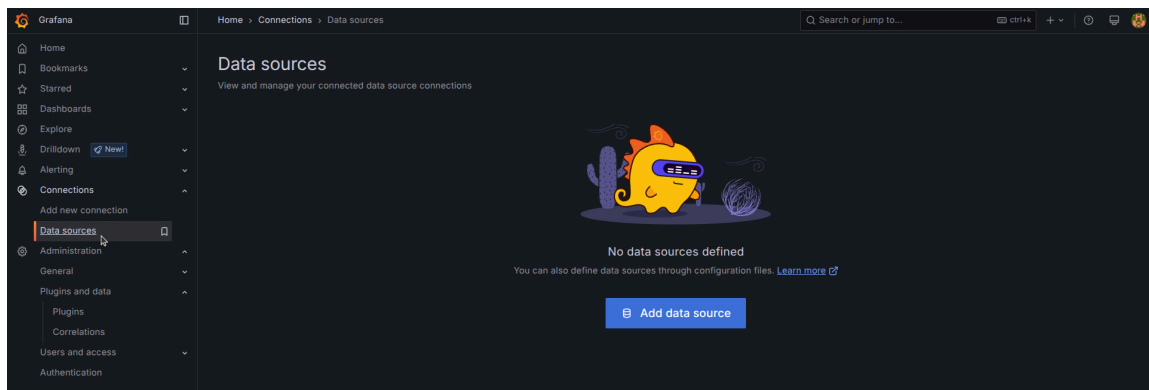


Figura 8.7\_Monitor\_Kafka: Grafana y Prometheus 2

8. Elegimos Prometheus y añadimos la configuración `http://localhost:9090`

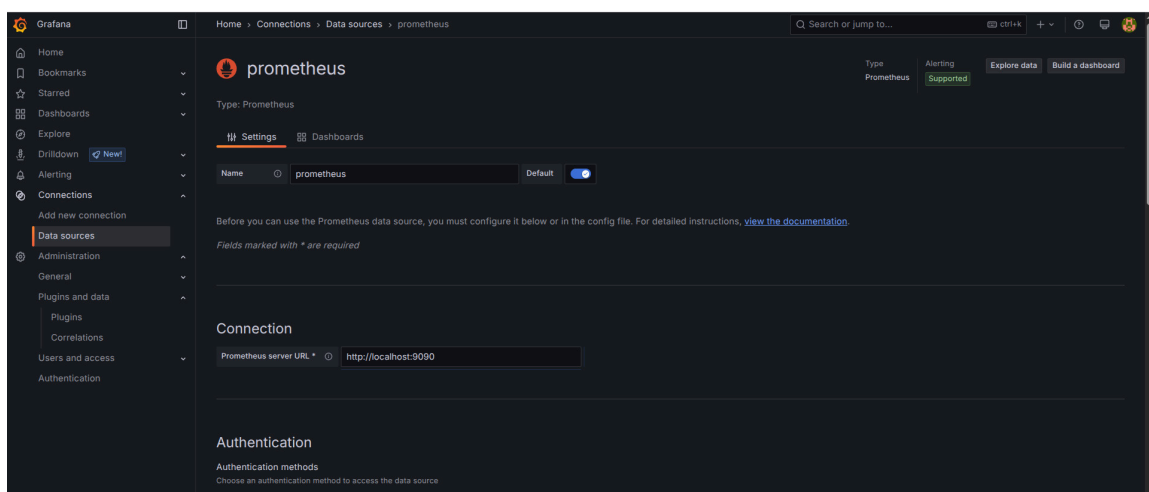


Figura 8.8\_Monitor\_Kafka: Grafana y Prometheus 3

## 9. Marcamos "Save & test"

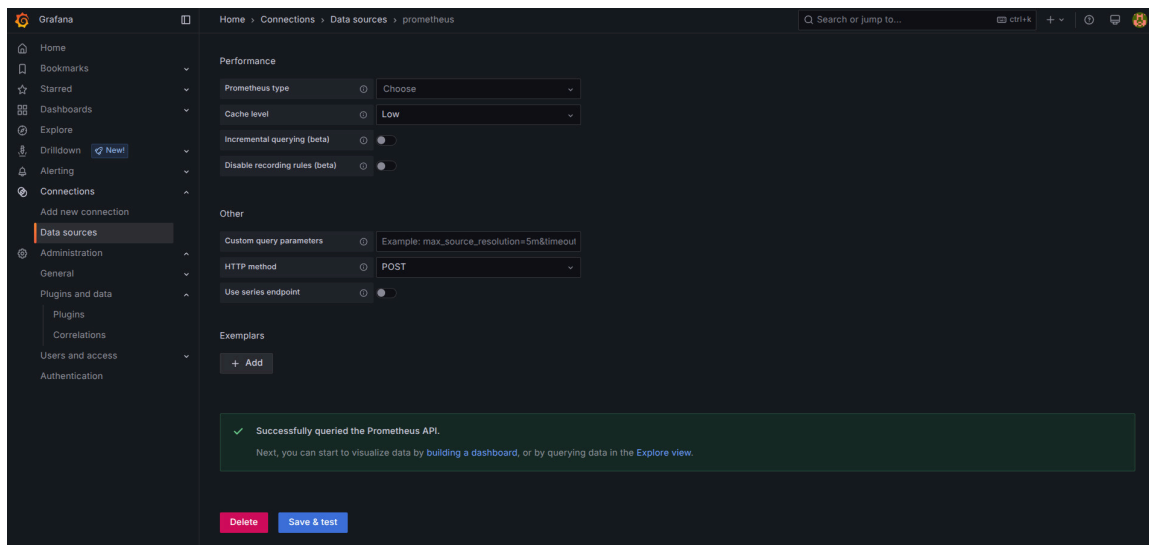


Figura 8.9\_Monitor\_Kafka: Grafana y Prometheus 4

10. Ahora necesitamos crear un dashboard para monitorizar visualmente los datos en tiempo real. Podemos crearlos nosotros manualmente, o usar algunas ya creados para importarlos. Puedes consultar dashboard en la [página oficial de dashboards de grafana](#). Elegimos el que más nos interese de [kafka](#)
11. Si queremos elegir uno sólo habría que añadir el código identificador. Para que veamos como sería cargar uno, usaremos como ejemplo [este](#) o [este](#)
12. Para ello nos vamos a Dashboards -> New -> Import

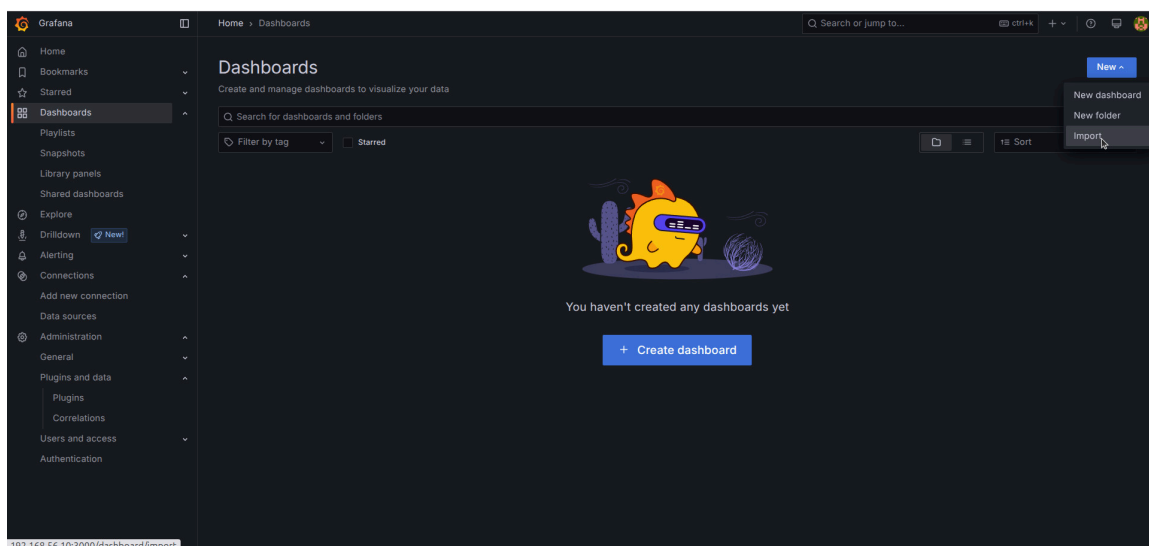


Figura 8.10\_Monitor\_Kafka: Grafana y Prometheus 5

13. Cargamos el dashboard indicando el id del dashboard. En nuestro caso el `11962` o `18276`

#### 14. Marcamos el identificador y **Load**

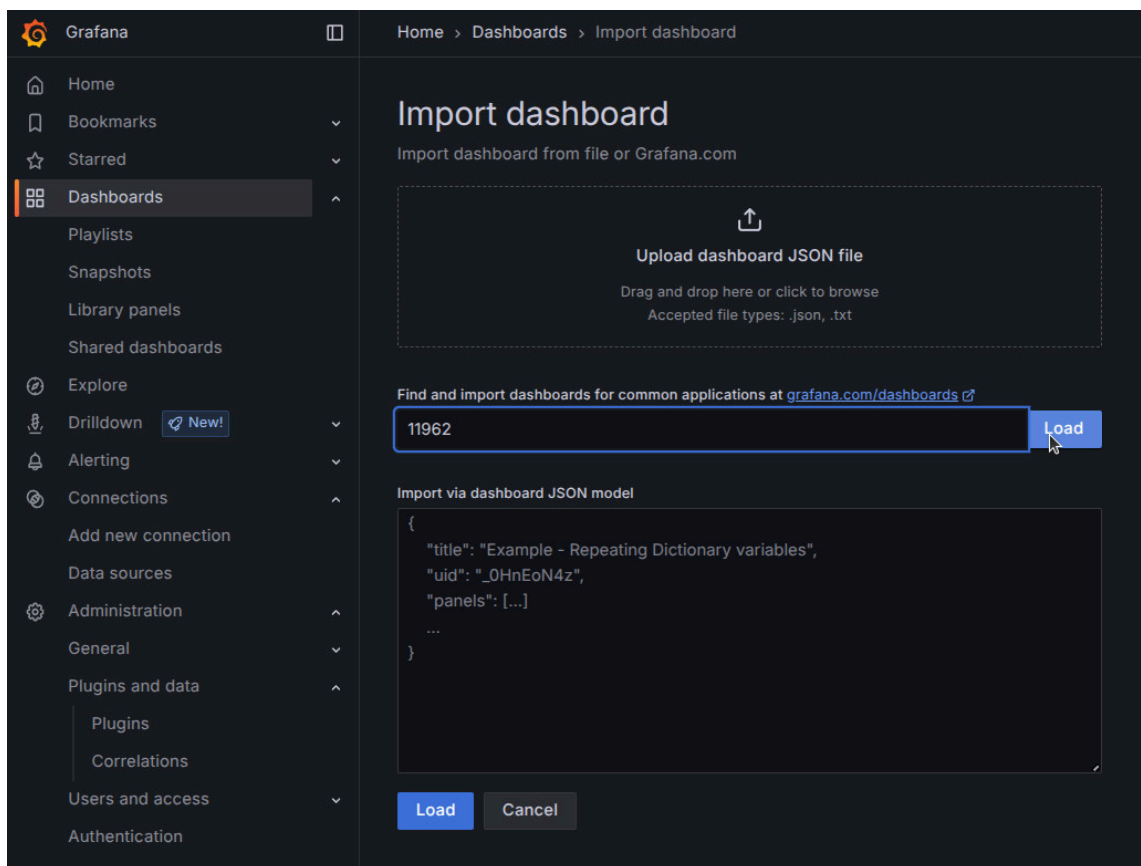


Figura 8.11\_Monitor\_Kafka: Grafana y Prometheus 5

#### 15. Configuramos el dashboard. Hay que indicarle nuestro data source prometheus.

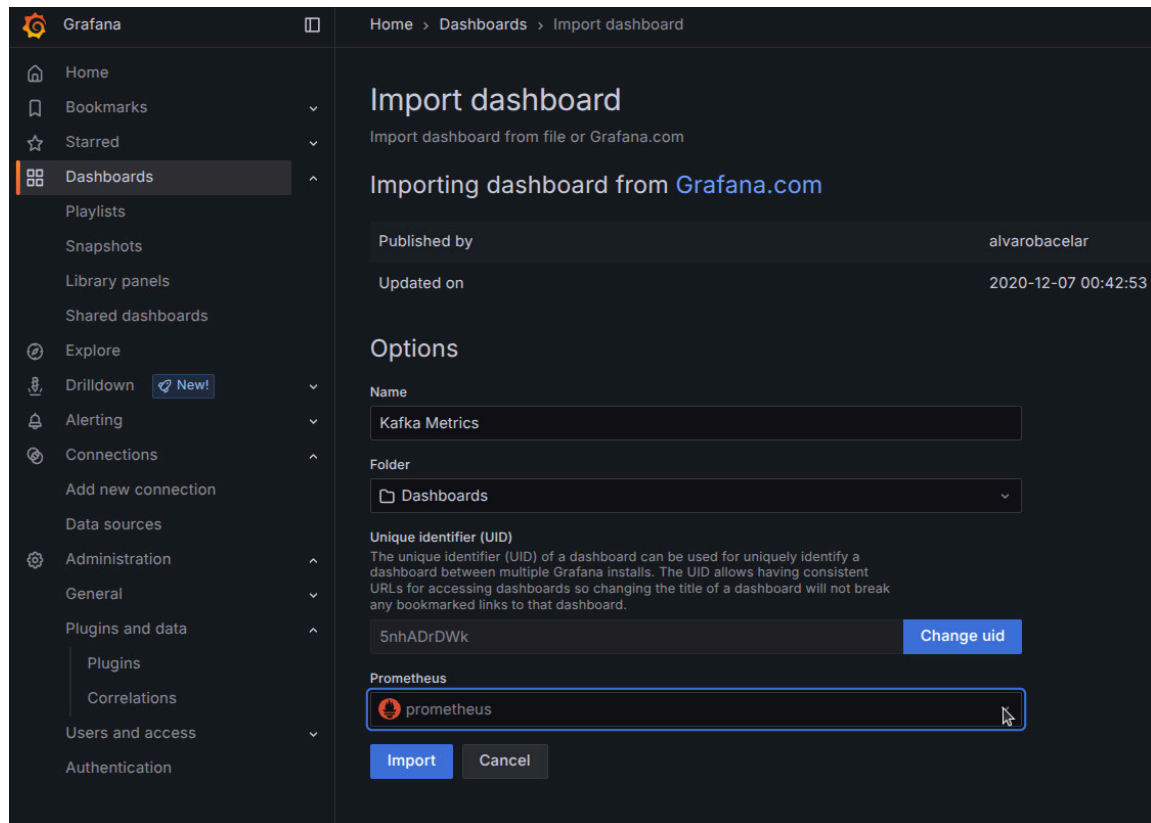


Figura 8.12\_Monitor\_Kafka: Grafana y Prometheus 7



### Versiones

Hay que tener en cuenta que cada dashboard está basado en unas versiones determinadas, tanto de la versión de kafka, como cual ha sido la configuración de la plantilla del scraping, si hemos usado JVM Exporter o Kafka Exporter,... ( `kafka-kraft-3_0_0.yml` en kafka 4.0 y JMX Exporter en nuestro caso). Por tanto, ninguno de estos dashboards nos funcionaría. Pero si es interesante saber como hacerlo para cuando haya alguno o queramos usarlo para otro tipo de servicio

16. Haz lo mismo con el otro dashboard o el que quieras usar según tus necesidades.
17. Nosotros vamos a desarrollar nuestro propio dashboard. Para ver las métricas que podemos añadir, podemos verlas directamente desde prometheus. Para ello nos vamos a Graph <http://192.168.56.10:9090/graph> y le damos al botón de `open metrics explorer` que está justo antes del botón "execute"

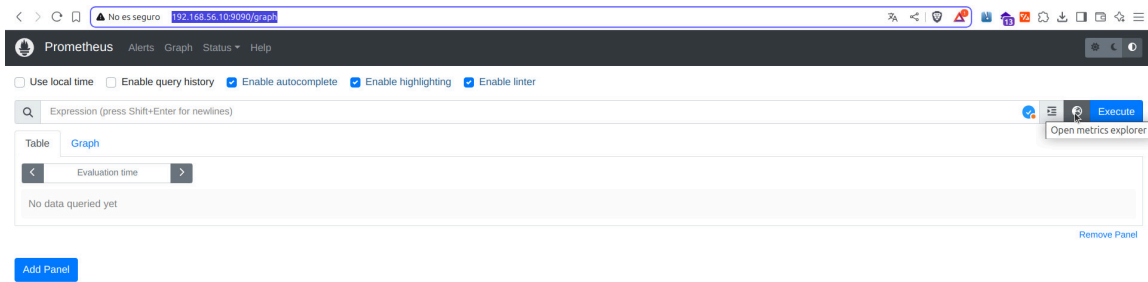


Figura 8.13\_Monitor\_Kafka: Grafana y Prometheus 8

18. He generado un pequeño dashboard para que puedas importarlo y usarlo. Haz las modificaciones que creas oportunas. Recuerda, este dashboard está configurado y debe ser usado para nuestra configuración de JMX Exporter + Kafka 4.0 + `kafka-kraft-3_0_0.yml`. Puedes [descargar el json](#) e importarlo como nuevo dashboard
19. Observa la captura de como se vería para este pequeño ejemplo. Al no añadir ningún topic, consumidor,... no están completos los datos del dashboards. Este ejemplo nos sirve para ver como funcionan los dashboards y preparar una plantilla. En el siguiente ejemplo ejecutaremos un ejemplo más completo para verlo en funcionamiento.

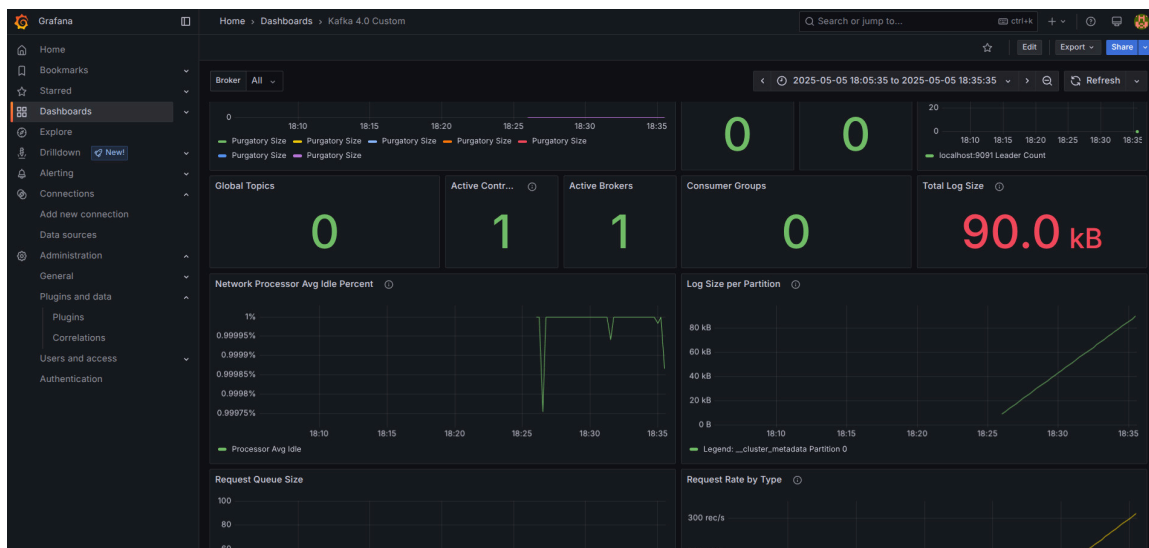


Figura 8.14\_Monitor\_Kafka: Grafana y Prometheus 9

## 10. Ejemplo 2. JMX + Prometheus + Grafana

Para ver grafana en funcionamiento, vamos a ejecutar el [Ejemplo 3](#) de Kafka de la unidad anterior, donde configuramos 3 controller con 3 brokers, 1 topic junto a un consumidor y un productor.



**Severs y puertos de kafka para el ejemplo**

Para este caso, para poder realizar el ejemplo, tenemos 3 opciones

1. Levantar un server en cada una de las máquinas que tenemos y usar el mismo puerto para JMX
2. Levantar todos los servers de Kafka en una máquina y usar un puerto para cada uno de ellos
3. Levantar todo el stack con docker (Se desarrollará en la siguiente Unidad)

Ambos casos tienen sus dificultades. En el primero necesitamos configurar el cluster de Kafka cada uno en un nodo, por ejemplo, en master 1 controller, y un broker en cada nodo (nodo1-> broker1, nodo2 -> broker2,...). En el segundo, el script de inicio de los servers de kafka debe tener mayor configuración, ya que debemos indicar en el script que arranque cada server en un puerto diferente.

Nosotros, elegiremos la segunda opción, así podemos establecer un ejemplo (NO EN PRODUCCIÓN) del funcionamiento de grafana con un posible cluster con quorum dinámico de varios controllers y brokers.

1. Necesitamos cambiar el script de arranque de los servidores de Kafka. Para ello vamos a hacer una copia del ya existente y vamos modificarlo. Este nuevo script será el que usemos para este ejemplo.

```
1 cp /opt/kafka_2.13-4.0.0/bin/kafka-server-start.sh /opt/kafka_2.13-4.0.0/bin/kafka-server-start_ejemplo2_mon_kafka.sh
```

2. Abrimos el script

```
1 nano /opt/kafka_2.13-4.0.0/bin/kafka-server-start_ejemplo2_mon_kafka.sh
```

3. Sustituimos la línea de configuración de nuestro ejemplo anterior por el siguiente código. Hemos elegido los puertos **11000** para exponer los datos a Prometheus. Puedes usar los que quieras

```
1 # --- INICIO DE MODIFICACIÓN PARA JMX EXPORTER ---
2
3 # Establecemos KAFKA_OPTS por defecto si no está definido
4 KAFKA_OPTS=${KAFKA_OPTS:-}
5
6 # Encontrar la ruta al archivo server.properties (o cualquier archivo
7 .properties)
8 PROPERTIES_FILE=""
9 for arg in "$@"; do
10     if [[ "$arg" == *.properties ]]; then
11         PROPERTIES_FILE="$arg"
12         break
```

```

12     fi
13 done
14
15 # Variables para JMX Exporter
16 JMX_AGENT_PATH="/opt/kafka_2.13-4.0.0/libs/jmx_prometheus_javaagent-
17 1.2.0.jar"
18 # Ruta al archivo de configuración del JMX Exporter. Si no usas uno,
19 déjala vacía pero mantén el ':' en el argumento final.
20 JMX_CONFIG_FILE="/opt/kafka_2.13-4.0.0/config/jmx-exporter-kafka.yml"
21
22 # Lógica para determinar el puerto y configurar KAFKA_OPTS
23 NODE_ID=""
24 JMX_PORT=""
25 JMX_JAVAAGENT_ARG=""
26
27 # 1. Intentar extraer node.id del archivo properties
28 if [ -n "$PROPERTIES_FILE" ] && [ -f "$PROPERTIES_FILE" ]; then
29     NODE_ID=$(grep "^[:space:]*node.id[:space:]*=" "$PROPERTIES_FILE"
30 | head -1 | sed "s/^[[:space:]*node.id[:space:]*=//" | cut -d'#' -f1 | tr -
31 d '[:space:]')
32 fi
33
34 # 2. Validar NODE_ID y calcular JMX_PORT
35 if ! [ "$NODE_ID" =~ ^[0-9]+$ ]; then
36     echo "WARNING: JMX Exporter: No se pudo determinar el node.id del
37 archivo $PROPERTIES_FILE o no es un número válido ('$NODE_ID')." >&2
38     echo "WARNING: JMX Exporter NO será cargado para esta instancia de
39 Kafka." >&2
40 # 3. Validar si el JAR del JMX Exporter existe y es legible
41 elif [ ! -r "$JMX_AGENT_PATH" ]; then
42     echo "ERROR: JMX Exporter: El archivo JAR NO se encuentra o no es
43 legible: $JMX_AGENT_PATH" >&2
44     echo "WARNING: JMX Exporter NO será cargado para esta instancia de
45 Kafka." >&2
46 # 4. Validar si el archivo de configuración existe y es legible (si se
47 especificó uno)
48 elif [ -n "$JMX_CONFIG_FILE" ] && [ ! -r "$JMX_CONFIG_FILE" ]; then
49     echo "ERROR: JMX Exporter: El archivo de configuración NO se
50 encuentra o no es legible: $JMX_CONFIG_FILE" >&2
51     echo "WARNING: JMX Exporter NO será cargado para esta instancia de
52 Kafka." >&2
53 # 5. Si todas las validaciones pasan, construir el argumento y configurar
54 KAFKA_OPTS
55 else
56     JMX_BASE_PORT=11000 # Puerto base, JMX_PORT = JMX_BASE_PORT + NODE_ID
57     JMX_PORT=$((JMX_BASE_PORT + NODE_ID))
58
59     # Construir el argumento del javaagent: =puerto:/ruta/config (o
60 =puerto:)
61     JMX_JAVAAGENT_ARG="-javaagent:$JMX_AGENT_PATH=$JMX_PORT"
62     if [ -n "$JMX_CONFIG_FILE" ]; then
63         JMX_JAVAAGENT_ARG="$JMX_JAVAAGENT_ARG:${JMX_CONFIG_FILE}" #
64 Añadir : y la ruta del archivo
65     else
66         JMX_JAVAAGENT_ARG="$JMX_JAVAAGENT_ARG:" # Si no hay archivo,
67 añadir solo el :
68     fi
69 fi

```

```

54
55
56     # Añadir el argumento del javaagent a KAFKA_OPTS
57     if [ -z "$KAFKA_OPTS" ]; then
58         export KAFKA_OPTS="$JMX_JAVAAGENT_ARG"
59     else
60         export KAFKA_OPTS="$KAFKA_OPTS $JMX_JAVAAGENT_ARG" # Añadir un
espacio antes por si ya hay opciones
61     fi
62
63     echo "DEBUG: JMX Exporter: Configurado en puerto $JMX_PORT para
node.id=$NODE_ID." >&2
64     echo "DEBUG: JMX Exporter: KAFKA_OPTS resultante: '$KAFKA_OPTS'" >&2
65 fi
66
67 # --- FIN DE MODIFICACIÓN PARA JMX EXPORTER ---

```

4. Hacemos una copia del fichero de configuración `prometheus.yml` para adaptarlo a esta configuración.

```

1 cp /opt/prometheus-2.53.4/prometheus.yml /opt/prometheus-
2.53.4/prometheus_ejemplo2_mon_kafka.yml

```

5. Abrimos el archivo de configuración

```

1 nano /opt/prometheus-2.53.4/prometheus_ejemplo2_mon_kafka.yml

```

6. Añadimos los diferentes *scrapings*. La configuración de los endpoint a partir del puerto 11000.

```

1 - job_name: "kafka"
2
3     # metrics_path defaults to '/metrics'
4     # scheme defaults to 'http'.
5
6     static_configs:
7         - targets: [
8             "localhost:11001", # Controller 1 (node.id=1)
9             "localhost:11002", # Controller 2 (node.id=2)
10            "localhost:11003", # Controller 3 (node.id=3)
11            "localhost:11005", # Broker 1 (node.id=5)
12            "localhost:11006", # Broker 2 (node.id=6)
13            "localhost:11007"  # Broker 3 (node.id=7)
14        ]

```

7. El resultado sería.

```

1 # my global config
2 global:
3     scrape_interval: 15s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.

```

```

4   evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
is every 1 minute.
5   # scrape_timeout is set to the global default (10s).
6
7   # Alertmanager configuration
8   alerting:
9     alertmanagers:
10      - static_configs:
11        - targets:
12          # - alertmanager:9093
13
14  # Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
15  rule_files:
16    # - "first_rules.yml"
17    # - "second_rules.yml"
18
19  # A scrape configuration containing exactly one endpoint to scrape:
20  # Here it's Prometheus itself.
21  scrape_configs:
22    # The job name is added as a label `job=<job_name>` to any timeseries
scraped from this config.
23    - job_name: "prometheus"
24
25      # metrics_path defaults to '/metrics'
26      # scheme defaults to 'http'.
27
28      static_configs:
29        - targets: ["localhost:9090"]
30
31    - job_name: "kafka"
32
33      # metrics_path defaults to '/metrics'
34      # scheme defaults to 'http'.
35
36      static_configs:
37        - targets: [
38          "localhost:11001", # Controller 1 (node.id=1)
39          "localhost:11002", # Controller 2 (node.id=2)
40          "localhost:11003", # Controller 3 (node.id=3)
41          "localhost:11005", # Broker 1 (node.id=5)
42          "localhost:11006", # Broker 2 (node.id=6)
43          "localhost:11007" # Broker 3 (node.id=7)
44        ]

```

## 8. Arrancamos el ejemplo

## 9. Generamos los uuid

```

1  #Generamos un cluster UUID y los IDs de los controllers
2  KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
3  CONTROLLER_1_UUID="$(bin/kafka-storage.sh random-uuid)"
4  CONTROLLER_2_UUID="$(bin/kafka-storage.sh random-uuid)"
5  CONTROLLER_3_UUID="$(bin/kafka-storage.sh random-uuid)"

```

## 10. Damos formato a los controllers de Kafka indicando los controllers iniciales (Recuerda la documentación)

```

1  #Formateamos los directorios de log indicando los controllers iniciales
2  #sudo rm -r /opt/kafka/ejemplo3/logs/*
3  #controller1
4  bin/kafka-storage.sh format --cluster-id ${KAFKA_CLUSTER_ID} \
5  --initial-controllers
6  "1@localhost:9096:${CONTROLLER_1_UUID},2@localhost:9097:${CONTROLLER_2_UUID},3@localhost:9098:${CONTROLLER_3_UUID}" \
7  --config /opt/kafka/ejemplo3/config/controller1.properties
8  #controller2
9  bin/kafka-storage.sh format --cluster-id ${KAFKA_CLUSTER_ID} \
10 --initial-controllers
11 "1@localhost:9096:${CONTROLLER_1_UUID},2@localhost:9097:${CONTROLLER_2_UUID},3@localhost:9098:${CONTROLLER_3_UUID}" \
12 --config /opt/kafka/ejemplo3/config/controller2.properties
13 #controller3
14 bin/kafka-storage.sh format --cluster-id ${KAFKA_CLUSTER_ID} \
15 --initial-controllers
16 "1@localhost:9096:${CONTROLLER_1_UUID},2@localhost:9097:${CONTROLLER_2_UUID},3@localhost:9098:${CONTROLLER_3_UUID}" \
    --config /opt/kafka/ejemplo3/config/controller3.properties

```

## 8. Damos formato a los logs de los brokers

```

1  #Formateamos los directorios de log de los brokers
2  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
3  /opt/kafka/ejemplo3/config/broker1.properties
4  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
5  /opt/kafka/ejemplo3/config/broker2.properties
6  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
7  /opt/kafka/ejemplo3/config/broker3.properties

```

## 9. Iniciamos los servers(3 controllers y 3 brokers) cada uno en una terminal

```

1  #Ejecutamos los servidores Kafka (uno en cada terminal)
2  #Observa en el log al arrancar que indica que Levanta JMX y el puerto
   correspondiente al node.id
3  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/controller1.properties
4  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/controller2.properties
5  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/controller3.properties
6  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/broker1.properties
7  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/broker2.properties
8  bin/kafka-server-start_ejemplo2_mon_kafka.sh
   /opt/kafka/ejemplo3/config/broker3.properties

```

10. Comprobamos que se ha expuesto correctamente el endpoint en 9091 :

```

1  sudo ss -tunelp | grep 1100*

```

```

1  tcp    LISTEN 0      3            *:11006      *:*
   users: (("java",pid=11101,fd=115)) uid:1000 ino:39171 sk:4
   cgroup:/user.slice/user-1000.slice/session-1.scope v6only:0 <->
2  tcp    LISTEN 0      3            *:11007      *:*
   users: (("java",pid=11562,fd=115)) uid:1000 ino:40394 sk:5
   cgroup:/user.slice/user-1000.slice/session-7.scope v6only:0 <->
3  tcp    LISTEN 0      3            *:11005      *:*
   users: (("java",pid=10685,fd=115)) uid:1000 ino:38804 sk:6
   cgroup:/user.slice/user-1000.slice/session-3.scope v6only:0 <->
4  tcp    LISTEN 0      3            *:11002      *:*
   users: (("java",pid=9819,fd=115)) uid:1000 ino:37271 sk:7
   cgroup:/user.slice/user-1000.slice/session-5.scope v6only:0 <->
5  tcp    LISTEN 0      3            *:11003      *:*
   users: (("java",pid=10266,fd=115)) uid:1000 ino:38499 sk:8
   cgroup:/user.slice/user-1000.slice/session-4.scope v6only:0 <->
6  tcp    LISTEN 0      3            *:11001      *:*
   users: (("java",pid=9428,fd=115)) uid:1000 ino:38008 sk:9
   cgroup:/user.slice/user-1000.slice/session-2.scope v6only:0 <->
7  tcp    LISTEN 0      50          [::ffff:127.0.0.1]:9093  *:*
   users: (("java",pid=11101,fd=156)) uid:1000 ino:39338 sk:c
   cgroup:/user.slice/user-1000.slice/session-1.scope v6only:0 <->
8  tcp    LISTEN 0      50            *:36061      *:*
   users: (("java",pid=11101,fd=120)) uid:1000 ino:39174 sk:e
   cgroup:/user.slice/user-1000.slice/session-1.scope v6only:0 <->

```

11. Iniciamos Prometheus. Vamos a su directorio /opt/prometheus-2.53.4 . Levantamos prometheus

```

1  ./prometheus --config.file=prometheus_ejemplo2_mon_kafka.yml

```

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:11003/metrics	UP	instance="localhost:11003" job="kafka"	4.500s ago	495.525ms	
http://localhost:11005/metrics	UP	instance="localhost:11005" job="kafka"	11.623s ago	2.388s	
http://localhost:11006/metrics	UP	instance="localhost:11006" job="kafka"	13.612s ago	3.459s	
http://localhost:11007/metrics	UP	instance="localhost:11007" job="kafka"	13.642s ago	3.370s	
http://localhost:11001/metrics	UP	instance="localhost:11001" job="kafka"	12.892s ago	2.821s	
http://localhost:11002/metrics	UP	instance="localhost:11002" job="kafka"	1.598s ago	463.221ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	3.920s ago	10.047ms	

Figura 8.15\_Monitor\_Kafka: Ejemplo2: WebUI Prometheus

## 12. Iniciamos Grafana

```
1 systemctl start grafana-server
```

## 13. Creamos el topic con factor de replica 3 y 3 particiones. El topic debe conectarse a un broker.

```
1 bin/kafka-topics.sh --create --topic financial-transactions --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3
```

## 14. Podemos ver la descripción del topic creado

```
1 bin/kafka-topics.sh --describe --topic financial-transactions --bootstrap-server localhost:9092
```

```
1 bin/kafka-topics.sh --describe --topic financial-transactions --bootstrap-server localhost:9092
2 Topic: financial-transactions TopicId: MjtMKW06RNYR0aZWDOiIdg
PartitionCount: 3 ReplicationFactor: 3Configs: segment.bytes=1073741824
3 Topic: financial-transactionsPartition: 0 Leader: 5 Replicas:
5,6,7 Isr: 5,6,7 Elr: LastKnownElr:
4 Topic: financial-transactionsPartition: 1 Leader: 6 Replicas:
6,7,5 Isr: 6,7,5 Elr: LastKnownElr:
5 Topic: financial-transactionsPartition: 2 Leader: 7 Replicas:
7,5,6 Isr: 7,5,6 Elr: LastKnownElr:
```

## 15. Comprobamos el dashboard de Grafana

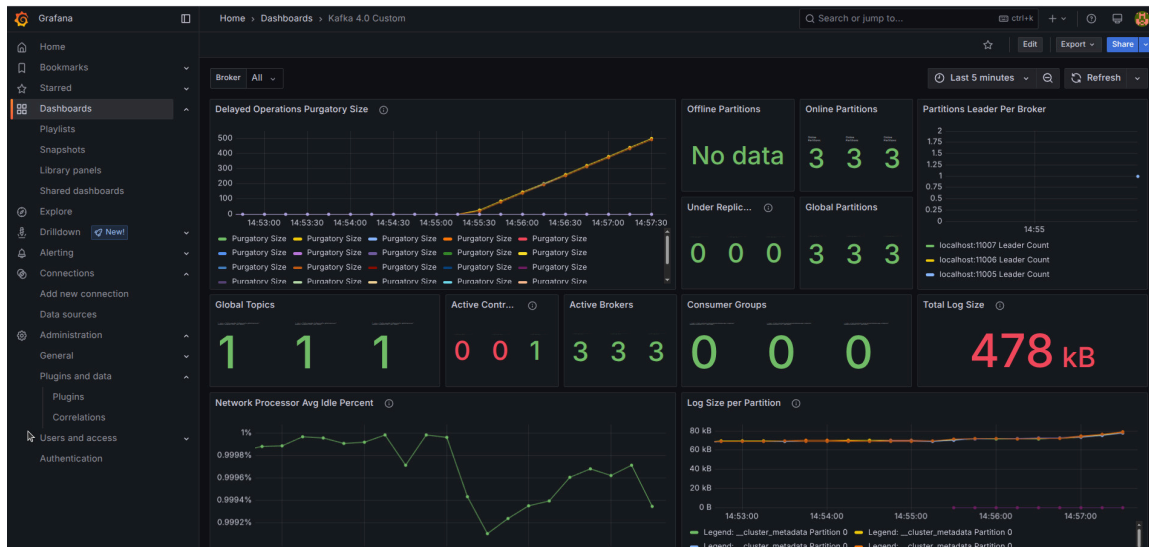


Figura 8.16\_Monitor\_Kafka: Ejemplo2: Dashboard Grafana

## 16. Lanzamos el productor

```
1 python3 /opt/kafka/ejemplo3/producer.py
```

## 17. Lanzamos el consumidor

```
1 python3 /opt/kafka/ejemplo3/consumer.py
```



Animación 8.1\_Monitor\_Kafka: Ejemplo 2