

**Centro de Enseñanza Técnica Industrial**

**Desarrollo de Software**

**Operación Continua**

**Jesús Alberto Aréchiga Carrillo**

**22310439 6N**

**Profesor**

**José Francisco Pérez Reyes**

**Mayo 2025**

**Guadalajara, Jalisco**

## Implementación de un Sistema de Monitoreo con Prometheus

Para asegurar la operación continua, es necesario contar con una herramienta de monitoreo que recoja métricas en tiempo real, alerte sobre anomalías y facilite la visualización gráfica del comportamiento de cada microservicio.

## ¿Por qué Prometheus?

Prometheus es un sistema de monitoreo y almacenamiento de series temporales diseñado para entornos de microservicios.

* Recopilación basada en “pull”, donde el servidor Prometheus interroga periódicamente a los microservicios para recopilar métricas expuestas en formato estándar.
* Lenguaje de consultas PromQL, que permite definir consultas y reglas complejas para calcular indicadores y disparar alertas.
* Alertmanager integrado, que gestiona notificaciones cuando las métricas superan umbrales definidos.
* Ecosistema amplio, con exportadores para Kubernetes, Docker, Linux, bases de datos, y bibliotecas clientes para instrumentar cada lenguaje de programación.

## Monitoreo en Storix

La arquitectura de monitoreo con Prometheus se estructura en los siguientes componentes**:**

1. **Servidor Prometheus**
   * Ejecuta la recolección de métricas desde cada microservicio y desde componentes de infraestructura.
   * Almacena series temporales en su base de datos interna de alto rendimiento.
   * Evalúa las reglas de alerta definidas mediante PromQL y envía alertas a Alertmanager.
2. **Exportadores**
   * node\_exporter: Recolecta métricas del sistema operativo de cada nodo (CPU, memoria, disco, red).
   * Advisor: Extrae métricas de contenedores Docker/Kubernetes, como uso de CPU y memoria por contenedor.
   * Instrumentación interna: Cada microservicio en Storix expone métricas personalizadas (número de solicitudes, latencia, errores) en un endpoint /metrics mediante bibliotecas oficiales de Prometheus.
3. **Alertmanager**
   * Recibe las notificaciones de alerta generadas por el servidor Prometheus tras evaluar las “alert rules”.
   * Gestiona la agrupación de alertas, los silenciamientos y reenvíos a canales externos.

## Implementación de Prometheus

Para la implementación de Prometheus se utilizan servidores en Linux dedicados.

Primero se descarga la versión de Prometheus deseada, para buscar la versión se puede entrar a la página de descargas o al repositorio en Github para saber la versión más reciente. En este caso, se utiliza la versión 2.44

### Descarga de Prometheus

wget https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz

O con curl:

curl -LO https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz

### Después se instalan los binarios: cd prometheus-2.44.0.linux-amd64

sudo mv prometheus promtool /usr/local/bin/

sudo mkdir -p /etc/prometheus /var/lib/prometheus

sudo mv console\*.yml prom\* /etc/prometheus/

sudo mv consoles/ console\_libraries/ /etc/prometheus/

### Se crea el usuario y se asignan los permisos:

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

sudo chown -R prometheus:prometheus /usr/local/bin/prometheus /usr/local/bin/promtool

sudo chown -R prometheus:prometheus /etc/prometheus /var/lib/prometheus

### Configurar y arrancar el servicio:

Se crea /etc/systemd/system/prometheus.service con el siguiente contenido:

[Unit]

Description=Prometheus Service

After=network.target

[Service]

User=prometheus

ExecStart=/usr/local/bin/prometheus \

--config.file=/etc/prometheus/prometheus.yml \

--storage.tsdb.path=/var/lib/prometheus

[Install]

WantedBy=multi-user.target

### Se recarga y se corre el servicio de Prometheus:

sudo systemctl daemon-reload

sudo systemctl enable prometheus

sudo systemctl start prometheus

## Configuración de servicios de Prometheus

### Definir los “Jobs” que Prometheus debe sondear

En el archivo prometheus.yml se configura:

global:

scrape\_interval: 15s

evaluation\_interval: 30s

# Alerting

rule\_files:

- /etc/prometheus/alerts.yml

scrape\_configs:

# Prometheus se auto-monitorea

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

# Monitoreo de nodos del clúster (node\_exporter)

- job\_name: 'node\_exporter'

static\_configs:

- targets: ['node1:9100', 'node2:9100', 'node3:9100']

# Microservicios de Storix (asumimos cada despliegue expone /metrics)

- job\_name: 'storix-services'

kubernetes\_sd\_configs:

- role: pod

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_pod\_label\_app]

regex: 'storix-.\*'

action: keep

- source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_scrape]

regex: 'true'

action: keep

- source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_port]

target\_label: \_\_address\_\_

replacement: '$1:$/metrics'

- source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_path]

target\_label: \_\_metrics\_path\_\_

replacement: '$1'

### Definición de alertas con PromQL

En el archivo alerts.yml se definen las reglas:

groups:

- name: storix\_alerts

rules:

- alert: HighCPUUsage

expr: avg(node\_cpu\_seconds\_total{mode!="idle"}) by (instance) > 0.85

for: 2m

labels:

severity: warning

annotations:

summary: "Uso de CPU alto en {{ $labels.instance }}"

description: "El nodo {{ $labels.instance }} ha reportado un uso de CPU mayor al 85% durante 2 minutos."

- alert: ServiceDown

expr: up{job="storix-services", service=~"storix-.\*"} == 0

for: 1m

labels:

severity: critical

annotations:

summary: "Servicio abajo: {{ $labels.service }}"

description: "El microservicio {{ $labels.service }} no responde desde hace más de 1 minuto."

- alert: HighErrorRate

expr: increase(storix\_http\_requests\_total{code=~"5.."}[5m]) / increase(storix\_http\_requests\_total[5m]) > 0.05

for: 5m

labels:

severity: critical

annotations:

summary: "Alta tasa de errores (5xx) en storix-services"

description: "Más del 5% de las solicitudes en los últimos 5 minutos han generado errores 5xx."

### Configuración de Alertmanager

Prometheus gestiona la detección, pero Alertmanager se encarga de la notificación. En el archivo alertmanager.yml se configura:

global:

resolve\_timeout: 5m

route:

receiver: "team-slack"

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 3h

receivers:

- name: "team-slack"

slack\_configs:

- api\_url: 'https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX'

channel: '#alerts'

send\_resolved: true

- name: "oncall-email"

email\_configs:

- to: 'oncall@storix-company.com'

from: 'prometheus@storix-company.com'

send\_resolved: true

inhibit\_rules:

- source\_match:

severity: "critical"

target\_match:

severity: "warning"

equal:

- service

Y se despliega el Alertmanager agregando en prometheus.yml:  
alerting:

alertmanagers:

- static\_configs:

- targets:

- 'alertmanager.monitoring.svc.cluster.local:9093'

Ahora al acceder a <http://IPPrometheus:9090>, se puede acceder a la interfaz gráfica que brinda prometheus. En el enlace se sustituye “IPPrometheus” por la dirección IP del equipo o la máquina virtual que tenga el servicio de prometheus corriendo.

Con esta implementación de **Prometheus**, Storix contará con un sistema de monitoreo sólido y eficaz que no sólo detecta y notifica proactivamente incidentes, sino que también respalda la visibilidad continua del rendimiento de los microservicios y de la infraestructura subyacente.