Introducción a Terraform y Salt Project

Unidad 06. Monitorización con Prometheus y Grafana

Autor: Sergi García

Actualizado Noviembre 2025

Licencia

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

ÍNDICE

Unidad 06. Monitorización con Prometheus y Grafana

1. 📘 Introducción a la monitorización en entornos IaC

La **monitorización** es la capacidad de entender el estado interno de un sistema complejo a partir de los datos que este genera: métricas, logs y trazas. En entornos modernos gestionados con **Infraestructura como Código (IaC)**, como Terraform, esta observabilidad es esencial para garantizar que los servicios desplegados no solo existan, sino que además funcionen correctamente, sean estables y estén optimizados.

Terraform permite describir y crear infraestructura de manera declarativa, pero no incorpora de forma nativa un sistema de monitoreo. Por tanto, una vez desplegada la infraestructura, necesitamos herramientas complementarias que proporcionen visibilidad sobre:

- **Rendimiento de los recursos** (CPU, memoria, red, disco).
- **Disponibilidad de los servicios** (estado de contenedores, latencia, tiempo de respuesta).
- **Alertas proactivas** ante fallos o degradación.
- **Históricos de comportamiento** para análisis y planificación.

Aquí es donde entran en juego **Prometheus** y **Grafana**, dos herramientas de código abierto ampliamente adoptadas en entornos DevOps y de observabilidad moderna.

2. 📊 Prometheus: recolección de métricas

Prometheus es un **sistema de monitorización basado en series temporales** (time series database). Fue diseñado originalmente por SoundCloud y hoy es un proyecto graduado de la **Cloud Native Computing Foundation (CNCF)**, al igual que Kubernetes.

Sus principales características son:

- 🕵️ **Scraping activo (modelo pull)**: Prometheus se conecta a los endpoints de los servicios (*targets*) y recolecta métricas a intervalos regulares.
- 🧮 **Lenguaje de consultas PromQL**: permite analizar, filtrar y combinar métricas en tiempo real.
- 💾 **Base de datos interna optimizada**: almacena métricas históricas sin necesidad de sistemas externos.
- ⚙️ **Fácil integración con exporters**: pequeñas utilidades que exponen métricas en formato estándar (por ejemplo, node_exporter para métricas del sistema operativo o cAdvisor para contenedores Docker).

Un ejemplo simple de configuración de Prometheus es el siguiente bloque prometheus.yml

```
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: "node_exporter"
    static_configs:
      - targets: ["node_exporter:9100"]

  - job_name: "prometheus"
```

```
    static_configs:
      - targets: ["prometheus:9090"]
```

Cada *target* define una fuente de métricas, en este caso el propio Prometheus y el contenedor node_exporter que expone datos del sistema.

3. 📈 GRAFANA: VISUALIZACIÓN Y ANÁLISIS

Grafana es una plataforma de visualización de datos que complementa a Prometheus. Permite crear **dashboards interactivos** que transforman las métricas en gráficos, tablas, indicadores y alertas visuales.

Principales características:

- 🎨 **Dashboards dinámicos**: personalizables y exportables (formato JSON).
- 🔗 **Integración con múltiples fuentes**: Prometheus, Loki, Elasticsearch, InfluxDB, etc.
- 🔔 **Alertas configurables**: se pueden definir umbrales visuales o enviar alertas por correo, Slack o webhook.
- 👥 **Gestión de usuarios y roles**: autenticación, permisos y control de acceso.

Ejemplo de panel básico creado a partir de métricas de Prometheus:

```
rate(node_cpu_seconds_total{mode!="idle"}[1m])
```

Esta consulta en PromQL muestra el uso de CPU promedio en los últimos 60 segundos.
Grafana permite graficar esta métrica, añadir leyendas, colores, límites y comparativas entre hosts.

4. 📘 RELACIÓN CON TERRAFORM

Terraform no "ejecuta" Prometheus ni Grafana, pero sí puede **automatizar su despliegue** como parte del ciclo de infraestructura, garantizando que ambos servicios existan, estén configurados y conectados.
Esto es clave en IaC: no solo crear la infraestructura funcional (como WordPress o MySQL), sino también la **infraestructura de observabilidad** que la acompañe.

Así, con Terraform se pueden definir contenedores de Docker, redes y volúmenes para Prometheus y Grafana, como:

```
resource "docker_container" "prometheus" {
  image = "prom/prometheus:latest"
  ports {
    internal = 9090
    external = 9090
  }
  mounts {
    type   = "bind"
    source = abspath("${path.module}/prometheus.yml")
    target = "/etc/prometheus/prometheus.yml"
  }
}

resource "docker_container" "grafana" {
```

```
  image = "grafana/grafana:latest"
  ports {
    internal = 3000
    external = 3000
  }
  depends_on = [docker_container.prometheus]
}
```

Terraform se convierte así en el **punto único de orquestación** que define no solo la aplicación, sino también su capa de métricas y visualización.

5. 🔗 Integración Prometheus – Grafana – Terraform

🔹 Prometheus como colector de métricas

Prometheus actúa como centro de recolección (scraper). Cada pocos segundos (definido por scrape_interval), consulta las fuentes de datos —conocidas como targets— y extrae información sobre el estado de los sistemas.
Estas métricas se almacenan internamente y se pueden consultar usando PromQL (Prometheus Query Language). Por ejemplo:

- node_load1 → carga promedio del sistema en 1 minuto.
- node_memory_MemAvailable_bytes → memoria libre disponible.
- rate(node_network_receive_bytes_total[5m]) → tráfico de red promedio recibido en 5 minutos.

Terraform puede **automatizar completamente** el despliegue de Prometheus, junto con sus configuraciones y redes Docker, lo que garantiza que los targets estén correctamente definidos desde el inicio.

🔹 Grafana como capa de visualización

Grafana no almacena métricas; su rol es consultar y mostrar los datos provenientes de Prometheus.
Gracias a la modularidad de Terraform, ambos servicios se pueden levantar dentro de la misma red Docker para que Grafana acceda a Prometheus mediante su hostname interno (prometheus:9090).
La relación puede resumirse así:

Rol	Herramienta	Protocolo	Ejemplo
Recolector de métricas	Prometheus	HTTP pull	/metrics
Visualización y análisis	Grafana	REST API	http://prometheus:9090

Terraform define esta interconexión en el bloque de configuración de Grafana:

```
resource "docker_container" "grafana" {
  name  = "grafana"
  image = "grafana/grafana:latest"
  networks_advanced {
    name = docker_network.wp_net.name
  }

  env = [
```

```
    "GF_SECURITY_ADMIN_USER=${var.grafana_user}",
    "GF_SECURITY_ADMIN_PASSWORD=${var.grafana_password}"
  ]
}
```

Una vez desplegado, el usuario solo necesita acceder a **Grafana → Data Sources → Prometheus** y configurar la URL interna (http://prometheus:9090).

⚙️ Flujo de datos entre los componentes

El ciclo de vida y flujo general de observabilidad se puede visualizar así:

[Terraform] → crea → [Docker Network] → conecta →
├── WordPress + MySQL (servicio)
├── Node Exporter (métricas del host)
├── Prometheus (recolector)
└── Grafana (visualización)

🔁 Flujo de información:

1. Node Exporter expone métricas del sistema en :9100/metrics.
2. Prometheus las consulta y almacena.
3. Grafana lee los datos desde Prometheus mediante PromQL.
4. El usuario los visualiza en paneles y gráficos personalizados.

📈 Dashboards JSON: infraestructura reproducible

Uno de los mayores beneficios de Grafana es que los dashboards se definen en **JSON**. Esto permite tratarlos también como código (IaC), versionarlos y replicarlos en distintos entornos.

Ejemplo simple del archivo “dashboard-basic.json”:

```
{
  "title": "WordPress Basic Metrics",
  "panels": [
    {
      "type": "graph",
      "title": "CPU Usage",
      "targets": [
        {
          "expr": "rate(node_cpu_seconds_total{mode!='idle'}[1m])",
          "legendFormat": "CPU"
        }
      ]
    }
  ]
}
```

Terraform puede incluso desplegar Grafana **ya con el dashboard importado** mediante su sistema de *provisioning*, integrando completamente la capa visual dentro del flujo de IaC.

6. 🧠 Conclusiones y beneficios del enfoque

El uso conjunto de **Terraform + Prometheus + Grafana** representa una práctica moderna de **infraestructura observable como código (IoC)**. Esto va más allá de la simple Infraestructura como Código (IaC): no solo creamos los recursos, sino que también codificamos **su supervisión, visualización y control operativo**.

🔹 1. Monitorización como parte del despliegue

Tradicionalmente, la observabilidad se añadía después del despliegue. Con Terraform, ahora se puede **declarar la observabilidad desde el inicio**, asegurando que cada servicio (por ejemplo, WordPress, MySQL, Redis, etc.) se entregue acompañado de su propio monitoreo.

Ventajas:

- Evita la "zona ciega" de infraestructuras sin métricas.
- Estandariza configuraciones de Prometheus y Grafana.
- Facilita el mantenimiento en equipos DevOps distribuidos.

🔹 2. Reproducibilidad y consistencia total

Al definir **infraestructura y monitoreo como código**, cualquier entorno (desarrollo, pruebas, staging o producción) puede reproducirse con el mismo nivel de instrumentación.
Esto garantiza que las métricas y los dashboards sean idénticos entre entornos, reduciendo errores humanos y diferencias no documentadas.

💡 *Ejemplo práctico:* Un dashboard que mide la carga del servidor en el entorno de desarrollo se comportará igual en el entorno productivo, porque ambos se construyen a partir del mismo archivo JSON versionado en Git.

🔹 3. Escalabilidad sin complejidad adicional

Terraform permite añadir nuevos servicios (contenedores o nodos) sin modificar manualmente Prometheus o Grafana. Basta con ampliar las variables o recursos de Terraform y aplicar los cambios con "terraform apply".

🔹 4. Integración natural en flujos DevOps / SRE

El paradigma **DevOps moderno** busca automatizar no solo el despliegue, sino también la observación, corrección y mejora continua.
Prometheus y Grafana son herramientas nativas del ecosistema **SRE (Site Reliability Engineering)**, donde la monitorización se convierte en un **indicador directo de salud y calidad de servicio (SLOs/SLIs)**.

Terraform aporta:

- Versionado de la infraestructura.
- Modularización y reutilización.
- Integración continua (CI/CD) con herramientas como Jenkins, GitHub Actions o GitLab CI.

Prometheus y Grafana aportan:

- Métricas detalladas para detectar degradación de rendimiento.
- Dashboards que permiten anticipar incidencias antes de afectar al usuario final.
- Posibilidad de establecer alertas automáticas.

🔹 5. Buenas prácticas recomendadas

Categoría	Recomendación
Rutas	Usar siempre abspath() para rutas absolutas en montajes Docker.
Contraseñas	No incluirlas directamente en main.tf; usar variables o terraform.tfvars.
Dashboards	Mantener los JSON versionados junto al código Terraform.
Alertas	Configurar alertas simples en Prometheus y Grafana para CPU o RAM.
Modularización	Separar monitorización y aplicación en módulos (modules/monitoring, modules/app).
Persistencia	Usar volúmenes Docker para evitar pérdida de datos al destruir el entorno.

7. 🌐 Webgrafía recomendada

🔸 Documentación oficial

- 📊 Grafana Docs → https://grafana.com/docs/grafana/latest/
- 📈 Prometheus Docs → https://prometheus.io/docs/introduction/overview/

🔸 Recursos DevOps y ejemplos

- HashiCorp Learn: *Monitor Infrastructure with Terraform*
  - https://developer.hashicorp.com/terraform/tutorials
- Grafana Academy: *From Prometheus to Dashboard*
  - https://grafana.com/tutorials
- CNCF Prometheus Case Studies
  - https://www.cncf.io/projects/prometheus/

🔸 Repositorios y referencias

- Ejemplo oficial: *Terraform + Prometheus + Grafana on Docker*
  - https://github.com/vegasbrianc/prometheus
- Blog HashiCorp: *Infrastructure as Code meets Observability* https://www.hashicorp.com/blog