PRÁCTICA INTEGRAL: DESPLIEGUE Y OBSERVABILIDAD DE MICROSERVICIOS

Cristóbal Suárez Abad

OPTATIVA - 2º ASIR

Contenido

Práctica Integral: Despliegue y Observabilidad de Microservicios

1. Introducción y Objetivos

En el mundo real, un administrador de sistemas no solo "instala" servicios, sino que debe asegurar su **disponibilidad** y **rendimiento**. En esta práctica, realizarás el aprovisionamiento de una aplicación multi-contenedor y configurarás un stack profesional de monitorización basado en la arquitectura **Prometheus + Grafana**.

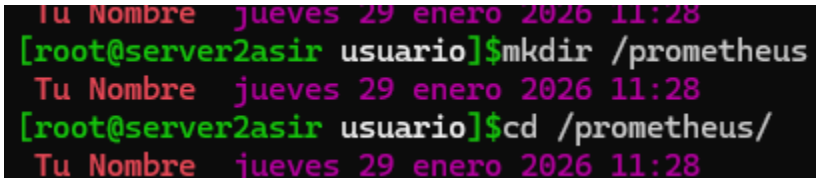
Objetivos técnicos:

- Desplegar una arquitectura de microservicios mediante **Infraestructura como Código (Docker Compose)**.
- Entender la arquitectura de **pull-metrics** de Prometheus.
- Implementar **Exporters** para obtener datos del sistema y de los contenedores.

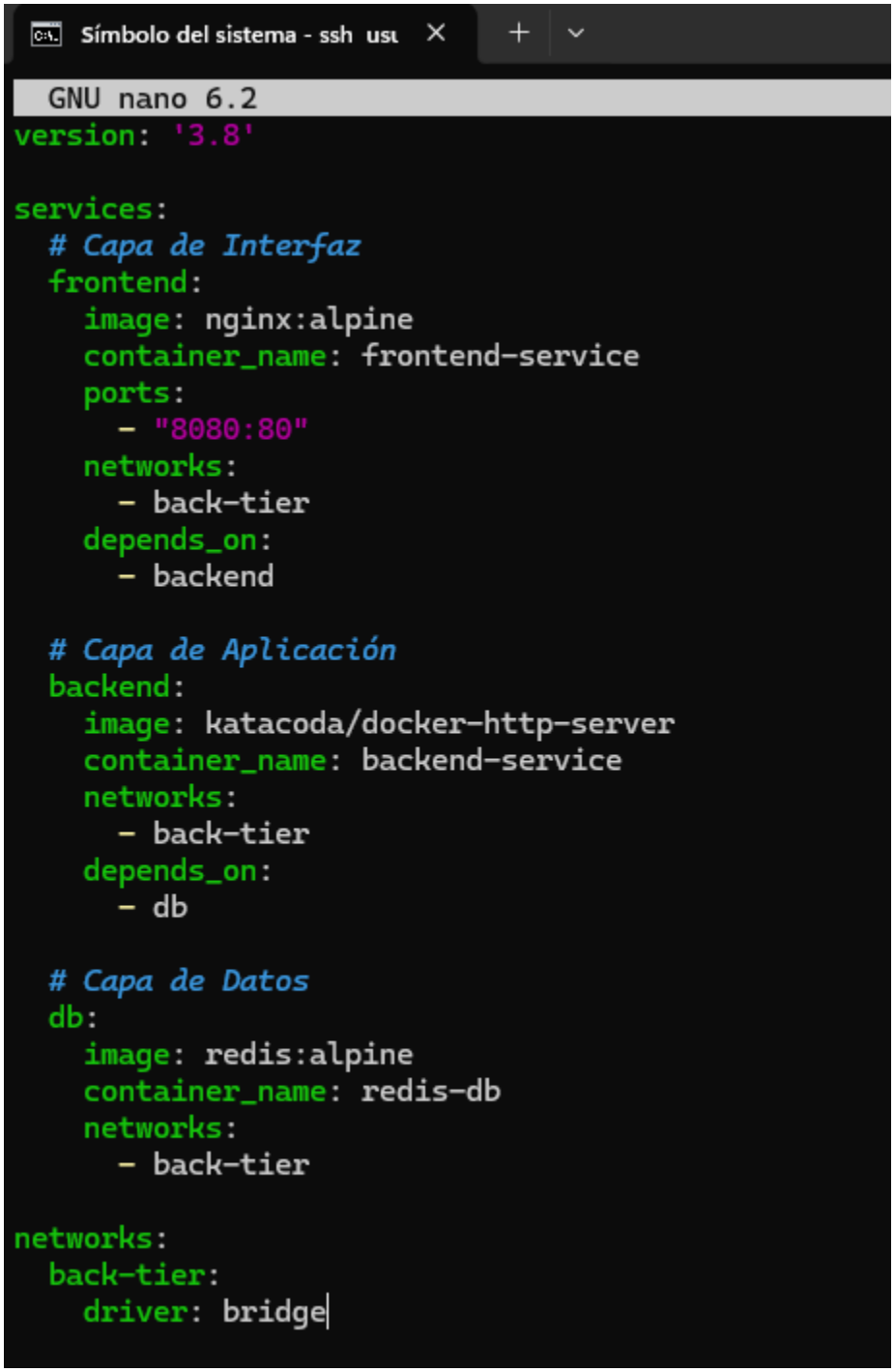
2. Fase 1: Aprovisionamiento de la Aplicación

Como no tenemos una app previa, tu primera tarea es desplegar la infraestructura base.

1. **El Escenario:** Crea un archivo `docker-compose.yml` que levante los siguientes servicios:
   - **Frontend:** Una imagen de `nginx:alpine`.
   - **Backend:** Una imagen de ejemplo (puedes usar `katacoda/docker-http-server` o una app sencilla en Python).
   - **Base de Datos:** Un contenedor de `redis:alpine`.
2. **Configuración de Red:** Todos los servicios deben estar en una red interna llamada `back-tier`.



nano docker-compose.yml



```
version: '3.8'

services:
  # Capa de Interfaz
  frontend:
    image: nginx:alpine
    container_name: frontend-service
    ports:
      - "8080:80"
    networks:
      - back-tier
    depends_on:
      - backend

  # Capa de Aplicación
  backend:
    image: katacoda/docker-http-server
    container_name: backend-service
    networks:
      - back-tier
    depends_on:
      - db

  # Capa de Datos
  db:
    image: redis:alpine
    container_name: redis-db
    networks:
      - back-tier

networks:
  back-tier:
    driver: bridge
```

Comprobamos que funciona:

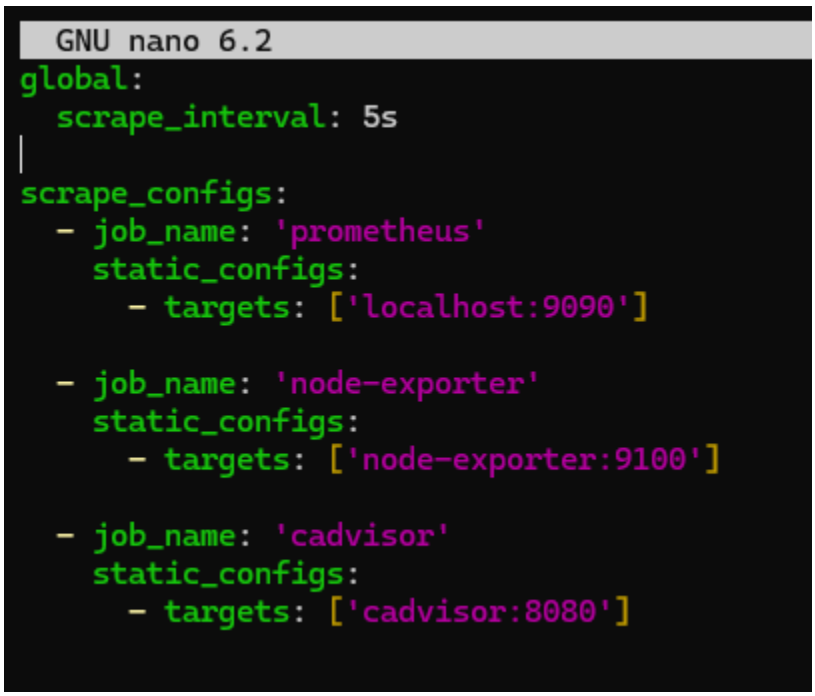
http://10.2.7.99:8080

3. Fase 2: El Stack de Monitorización

Ahora que la app "negocio" funciona, vamos a montar el sistema de vigilancia.

1. **Arquitectura Prometheus:** Añade Prometheus al `docker-compose.yml`.
   - Debes crear un archivo de configuración `prometheus.yml`.
   - Configura el `scrape_interval` a 5 segundos para que las gráficas sean fluidas en clase.
2. **Despliegue de Exporters (Los "Agentes"):** Prometheus no sabe leer la CPU del host por sí solo. Debes añadir:
   - **Node Exporter:** Para métricas de la máquina virtual (Host).
   - **cAdvisor:** Para métricas específicas de cada contenedor (memoria usada por Redis, CPU del Nginx, etc.).

En la misma carpeta donde tenemos el **docker-compose.yml**, creamos el archivo **prometheus.yml**



Añadimos esto al docker-compose.yml

```
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    networks:
      - back-tier

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--collector.filesystem.ignored-mount-points=^/(sys|proc|dev|host|etc)($$|/)'
    networks:
      - back-tier

  cadvisor:
    image: gcr.io/cadvisor/cadvisor:latest
    container_name: cadvisor
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    networks:
      - back-tier

networks:
  back-tier:
    driver: bridge
```

NOTA: Se ha añadido “restart: unless-stopped” a los contenedores para no tener que levantarlos manualmente cada vez que iniciemos el servidor.

3. **Verificación:** Accede a `http://localhost:9090` y comprueba en el menú *Status -> Targets* que todos los componentes aparecen en verde (**UP**).

http://10.2.7.99:9090/targets