Ingeniería de Servidores (2024-2025)
Grado en Ingeniería Informática
Universidad de Granada

Bloque 2 : Simulación de Carga de Trabajo y Monitorización

Índice

OBJETIVOS MÍNIMOS

1. Contenedores. Funcionamiento y relación con Virtualización.
2. Docker. Ejecución de contenedores y orquestación con compose.
3. Phoronix Test Suite: descarga, ejecución y gestión de resultados.
4. Similación de cargas de trabajo con en entornos Web empleando AB y JMeter.
5. Monitorización de prestaciones empleando los recursos del OS y herramientas básicas como `top`.
6. Sistema de ficheros /proc. Propósito y principales contenidos relacionados con al monitorización de Linux.
7. Generación y consulta de los ficheros de logs del sistema.
8. Papel de `logrotate` en la gestión de logs. Opciones básicas de configuración y ejecución.
9. Programación de tareas periódicas de administración con `cron`.
10. Monitorización a escala con Prometheus + Grafana.

Lecciones

1. Contenedores con Docker. Phoronix Test Suite. AB y Jmeter para la ejecución de carga web.
2. Monitorización. Herramientas del SO.
3. Prometheus + Grafana para la monitorización de servicios web y métricas de infraestructura.

Competencias que se trabajarán

Competencias Básicas

CB2. Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.

Específicas de la Asignatura

R1. Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sis- temas informáticos, asegurando su fiabilidad, seguridad y calidad,

conforme a principios éticos y a la legislación y normativa vigente.

R2. Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

R5. Conocimiento, administración y mantenimiento de sistemas, servicios y aplicaciones informáticas.

Competencias Específicas del Título:

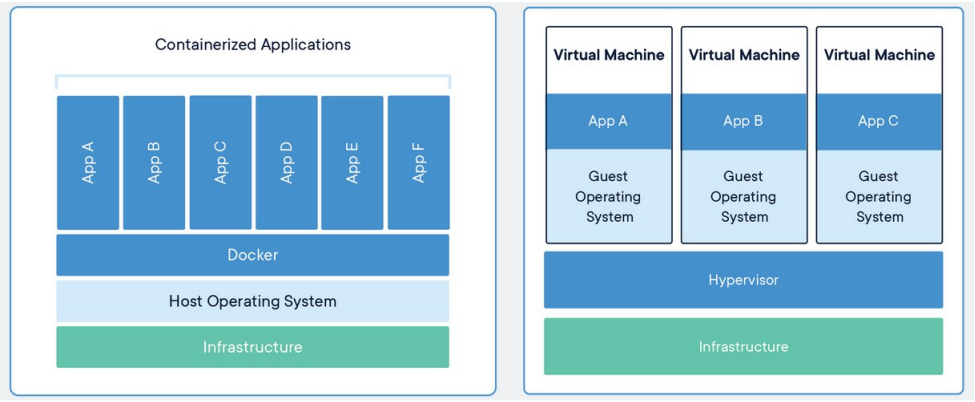
E4. Capacidad para definir, evaluar y seleccionar plataformas hardware y software para el desarrollo y la ejecución de sistemas, servicios y aplicaciones informáticas.

Competencias Transversales o Generales:

T2. Capacidad de organización y planificación así como capacidad de gestión de la Información.

1.- Breve introducción a los contenedores: docker

Ya hicimos referencia a los contenedores en el bloque 1 de prácticas, cuando hablamos del uso de las máquinas virtuales usando la analogía de proceso y hebra. Con ese conocimiento es suficiente para realizar la práctica si bien vamos a ilustrar esa compartición con la siguiente figura:



El contenedor presenta a la aplicación una nueva capa abstracción sobre el sistema operativo haciéndola homogénea, facilitando la portabilidad y, sobre todo, el encapsulamiento de información. La principal ventaja respecto al uso de máquinas virtuales es el ahorro en la función de Hypervisor (que es reemplazado por el S.O. de la máquina física) y el ahorro del S.O. invitado (que es reemplazado por la aplicación docker). Como consecuen- cias directas los contenedores ocupan menos espacio , requieren menos recursos y son más veloces en el arranque.

Aunque hay muchas tecnologías que implementan este nivel de abstracción (LXC, Open- VZ, RKT, Virtuozzo, etc.), Docker [1]es una de las más populares a día de hoy. No obstante, esta popularidad puede verse perjudicada por la reciente compra (13 de Noviembre 2019) por parte de Mirantis (o afectar a la parte abierta del proyecto...).

Debido a este cambio, es obligado mencionar tecnologías como **podman [2]**que nos permiten ejecutar contenedores con la misma sintaxis que docker y sin necesidad de tener un servicio como demonio con los riesgos de seguridad que eso conlleva.

Otra herramienta complementaria a `docker compose` [3] como citaremos adelante es buildah.io [4]. Como artículo complementario a este hecho es interesante leer el artículo "Its time to say goodbye to Docker"

La guía de "Getting Started with Docker" [24] es un buen punto de

partida para empezar a conocer esta tecnología.

Además, en "Play with Docker" [6] tiene cantidad de recursos para aprender los elementos básicos (y más avanzados). Concretamente, en "Frist Alpine Linux Containers" [5] puede ejecutar el "hola mundo" sin necesidad de instalar nada en su máquina anfitriona e ir familiarizándose con esta tecnología. Se le recomienda que realice el tutorial para entender mejor lo que haremos a continuación.

1.1.- Microservicios

La popularidad de los contenedores ha ido ligada al éxito de un tipo de arquitectura conocida como microservicios (*microservices*) [7]. En este tipo de arquitecturas tenemos distintos servicios que se ejecutan de forma independiente unos de otros y se comunican mediante APIs o sistemas de mensajería (RabbitMQ, Kafka, Qpid, p.ej.).

Las ventajas de esta arquitectura son varias pero la flexibilidad y escalabilidad (vertical y horizontal) quizá sean las más destacables.

1.2.- Instalación de Docker en el SO

A efecto de realización de los ejercicios prácticos, Docker puede instalarse en su equipo personal. Opcionalmente, puede instalarlo en una MV con Rocky 9. En este último caso, la configuración será la misma definida para las MV en el Bloque 1 de prácticas.

La instalación de Docker Engine es relativamente sencilla [8]. Para el caso de Rocky, emplee las instrucciones de CentOS 9.

La ejecución de Docker debe estar disponible para usuarios distintos de root. Asegúrese de realizar las acciones de configuración oportunas. La ejecución como root no se considerará realizada a efectos de completitud de las prácticas.

En caso de instalación en su equipo personal (con entorno gráfico), opcionalmente, puede elegir la opción de instalar Docker Desktop [9] que acompaña el engine con algunas utilidades gráficas de administración:

1.2.1.- Ejercicio Evaluable. Ejecutar "Hello World"

Instale Docker en su ordenador anfitrión o en una MV y ejecute el contenedor "Hello World" disponible en: https://hub.docker.com/_/hello-world.

2.- Benchmarks

Benchmarks hay muchos [10] para cada tipo de servicio (p.ej. Para DNSs:

NameBench y GRC's DNS Benchmark,) pero puede resultar más interesante programar uno para analizar algún parámetro concreto que deseemos. Siempre debemos tener en cuenta un mínimo de cuestiones al hacerlo:

- Objetivo del benchmark.
- Métricas (unidades, variables, puntuaciones, etc.).
- Instrucciones para su uso.
- Ejemplo de uso analizando los resultados.

2.1.- OpenBenchmarking

OpenBenchmarking [11] es un repositorio de benchmarks de creados bajo licencia opensource. Son un excelente punto de partida e inspiración para desarrollar sus propios benchamrks.

Phoronix [12] es la plataforma de OpenBenchmarking para ejecutar un conjunto de benchmarks.

La aplicación puede instalarse [13] en su equipo anfitrión empleando paquetes Debian/Ubuntu, en una MV Rokcy empleando el instalador Universal para Linux, en Windows o empleando contenedores [14]. Esta última opción es la recomendada en las prácticas.

2.1.1.- Ejercicio Opcional. Ejecutar un Benchmark.

El alumno/a debe ser capaz de utilizar Phoronix Test Suite para:

- Descargar, instalar y ejecutar un benchmark de su elección.
- Debe poder almacenar y recuperar los resultados de múltiples ejecución del benchmark.
- Debe ser capaz de explicar el objetivo del benchmark y de los resultados obtenidos.

2.1.2.- Ejercicio Opcional. Comparar benchmark en contenedor y en vm.

El alumno/a ejecutará el mismo benchmark sobre: su ordenador personal, docker en su ordenador personal y máquina virtual. Comente y justifique las posibles diferencias en los resultados obtenidos.

2.2.- Apache Benchmark

Dentro de los benchmarks más populares para servidores web podemos encontrar la herramienta para "Apache HTTP server benchmarking" [15] (comando ab).

Concretamente, su misión es mostrar cuantas peticiones por segundo el

servidor de HTTP (puede ser Apache, Nginx, IIS o cualquier otro) es capaz de servir.

Entre de los parámetros básicos que usted debe conocer encontramos la opción que especifica la concurrencia así como el número de peticiones. Debe ser capaz de ejecutar una prueba de carga y explicar de forma razonada los resultados obtenidos [16]

2.2.1.- Ejercicio Opcional. Carga Http con AB.

Ejecute el benchmark sobre los servidores Http empleados en las prácticas del Bloque 1 (Apache y Nginx). Se deja a la decisión del alumno/a la definición de los parámetros del benchmark, pero se valorará que sea capaz de definir un entorno de ejecución que eleve significativamente la carga de los servidores.

El alumno/a debe ser capaz de describir en detalle los resultados obtenidos así como razonar las posibles diferencias entre los servidores.

3.- Simulación de Carga con Jmeter

Aunque tenemos muchas posibilidades usando `ab` de manera correcta, hay otras herramientas que nos permiten hacer tests más complejos, concretamente vamos a ver Jmeter [17].

Este software se autodefine como una aplicación "*...designed to load test functional behavior and measure performance...*", es decir, no establecen ningún tecnicismo concreto ya que pueden medir y generar carga de forma genérica para varios elementos.

Algunas características interesantes de Jmeter es que puede crear concurrencia real debido a la posibilidad de usar varias hebras dentro de la mismas CPU así como distribuir la distribución del esfuerzo de carga entre varios equipos en red local.

Además, existen proveedores de servicios de carga en la nube, como Flood [18] que permite escalar y distribuir la carga de forma global.

La posibilidad de ejecutarse en modo de línea de comandos permite aligerar la carga de la máquina que está generando las peticiones al servidor además de permitir la automatización de ciertos tests.

También es muy interesante la funcionalidad que nos permite configurar como proxy de nuestro navegador a Jmeter para que vaya registrando nuestra navegación como usuarios y luego podamos replicar esas peticiones de manera automática y paralela.

Otras herramientas populares para la generación de pruebas de carga son: Gatling [19], Locust [20], K6 [21] o Artillery [22]

3.1.- Instalación de la aplicación para el test con Jmeter

La aplicación que emplearemos para el ejercicio de prueba de carga se encuentra en el repositorio de GitHub: https://github.com/davidPalomar-ugr/iseP4JMeter.git

Puede realizar clonar el repositorio empleando GIT o descargarlo en forma de archivo comprimido.

Tras esto, tendremos un directorio nuevo: iseP4JMeter al cual podremos acceder y levantar la aplicación con:

```
> cd iseP4JMeter

> docker compose up
```

El servicio se lanza en primer plano mostrando los logs de ejecución de sus componentes (incluidas las llamadas HTTP) lo que puede ser muy útil para depurar incidencias. Si lanzamos docker compose con la opción -d el servicio continuará en segundo plano.

3.2.- Detalles sobre el dockerfile y `docker-compose`

El *dockerfile* es el archivo donde indicamos todos los comandos necesarios para crear una imagen de docker y las acciones que debe llevar a cabo sobre esta (copiar archivos, instalar paquetes, modificar configuraciones). Desde un punto de vista muy abstracto podríamos decir que es el *playbook* de Ansible que aplica docker al levantar el contenedor.

En este dockerfile, dos elementos muy relevantes son la configuración del contenedor en lo que a red y almacenamiento se refiere.

En la aplicación sobre la que aplicaremos la carga tenemos dos contenedores que se configuran con los respectivos dockerfiles:

Contenido del dockerfile de la máquina que contiene la aplicación (en node.js):

```
FROM node:16.13.0-stretch
RUN mkdir -p /usr/src/app
COPY . /usr/src/app
EXPOSE 3000
WORKDIR /usr/src/app
RUN ["npm", "install"]
ENV NODE_ENV=production
CMD ["npm","start"]
```

En este archivo, indicamos la imagen a partir de la que crear el contenedor, node, con la etiqueta 16.13.0-stretch, que indica la versión de Node (16.13.0) y la imagen de Linux base a utilizar (Debian Stretch).

Tras descargar la imagen (en caso de que no esté descargada ya) ejecutará un comando (con RUN) para crear un directorio y copiar los archivos del repositorio en el contenedor.

Posteriormente, indicamos el puerto en el que queramos que escuche el contenedor (EXPOSE), esto está relacionado con la opción -p al ejecutar el contenedor ya que podemos hacer un mapeo del puerto del anfitrión ejecutando los contenedores con los que están exponiendo éstos.

Por defecto, la red empleado por los contenedores Docker es bridge [23]. Esta forma de red virtual tiene un comportamiento similar a al modo "Bridge" visto en el bloque 1 para VirtualBox. Permite que el contenedor acceda a los recursos de red del "anfitrión".

Por último se indica que la aplicación comience su ejecución tras configurar una variable de entorno.

Contenido del dockerfile de la máquina que contiene la base de datos (mongodb):

```
FROM mongo:6
COPY ./scripts/* /tmp/
RUN chmod 755 /tmp/initializeMongoDB.sh
WORKDIR /tmp
CMD ./initializeMongoDB.sh
```

En este caso, la imagen de partida es de mongo, una vez disponible, se copian unos scripts del repositorio al contenedor y se ejecutan (tras cambiar los permisos correspondientes). Por último se lanza un script para rellenar la base de datos.

Como ya hemos comentado, la herramienta para configurar varios contenedores de manera que den un único servicio es `docker compose`. Cuando la invocamos, ésta lee el archivo `docker-compose.yml` que especifica los siguientes elementos:

```
version: '2.0'
services:
#MongoDB based in the original Mongo Image
mongodb:
image: mongo:6
ports:
- "27017:27017"

# Initialize mongodb with data
mongodbinit:
build: ./mongodb
links:
- mongodb

# Nodejs App
nodejs:
build: ./nodejs
ports:
- "3000:3000"
links:
- mongodb
```

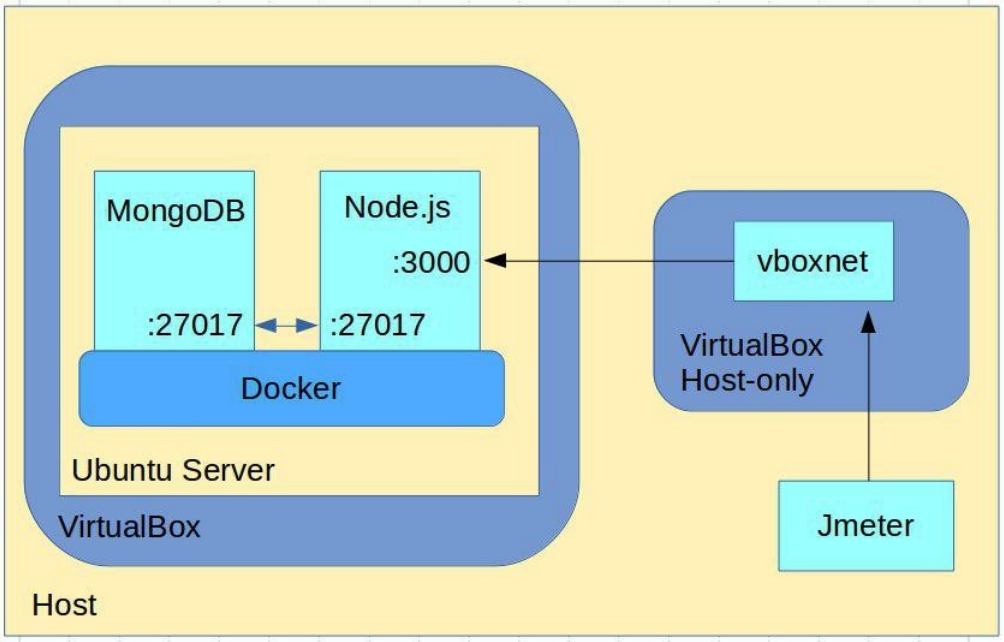


Figura 2.3: Descripción de los niveles de abstracción y los distintos elementos en ejecución.

Como siempre, le remitimos a la documentación oficial para ver los numerosos detalles sobre estos archivos [25][26][27]

-

3.3.- Ejercicio Obligatorio. Simulación de Carga Http con JMeter.

El repositorio https://github.com/davidPalomar-ugr/iseP4JMeter.git contiene la aplicación objeto de la prueba de carga y una descripción de los

requerimientos sobre la carga a simular. Siga las instrucciones y elabore la prueba de carga con Jmeter.

El alumno/a deberá realizar todo el ejercicio en un directorio que contendrá todos los artefactos necesarios para la ejecución de la prueba de carga con Jmeter. Dentro de la prueba de carga, los paths a los archivos se definirán de forma relativa para que la ejecución sea independiente de la localización del directorio. Como validación final, el alumno/a debe ser capaz de ejecutar la prueba de carga por linea de comandos (sin interfaz grafica) desde cualquier directorio de su equipo.

4.- Monitoring

La monitorización es la piedra angular de la operación de servicios IT y base de modernas metodologías de administración como: DevOps o Site Reliability Engineering.

4.1.- Top

Top [28][29] una herramienta clásica para obtener métricas de procesos y sobre el uso de los recursos de computo del servidor donde se ejecuta. Por motivos históricos, su amplia disponibilidad en distintas plataformas y el hecho de presentar una interfaz de texto compatible con entornos no gráficos, hacen que esta herramienta esté presente en el toolbox de los administradores de sistemas.

Htop es una versión con capacidades de visualización extendidas [30].

Top muestra un listado de métricas que por su amplia difusión se referencian más allá del ámbito Linux. El alumno/a debe familiarizarse con las mismas, siendo capaz de explicar su significado y justificar su valor considerando la carga actual del sistema.

Lo que hace la herramienta top, realmente, es reunir y presentar los datos de monitorización ofrecidos por el sistema operativo. Por ejemplo, los comandos: `uptime`, `free` o `vmstat` son alternativas a top que emplean las mismas fuentes. Al reducir el ámbito de la información devuelta y hacerlo por stdout, pueden emplearse para la automatización de scripts.

Una fuente fundamental de información para estas utilidades es el sistema de ficheros `/proc` [40]. En este directorio encontramos el origen de la información sobre monitorización. Por ejemplo: `loadavg`, `meminfo`, `uptime`, `vmstats`, `cpuinfo`. Mientras que `/proc/<processId>` contiene información detallada sobre los procesos en ejecución.

4.1.1.- Ejercicio Opcional. Stress + Top.

Simule una carga de trabajo en su equipo anfitrión o en un MV empleando el programa stress [31].

Monitorice con Top (o Htop) la evolución de la carga y justifique razonadamente los valores obtenidos.

4.2.- Programación de tareas periódicas con cron.

En el ámbito de la administración de servidores es muy común la ejecución de tareas periódicas de mantenimiento. Por ejemplo, para comprobar la salud de los sistemas, rotar los logs (como veremos más adelante) o realizar tareas de limpieza. Además, en el contexto de la monitorización, es muy común la ejecución de tareas periódicas para la toma de muestras.

Aunque existen soluciones más potentes (como los `systemd timers` [36]), por motivos históricos y por su amplia difusión, `cron` [37][38] sigue siendo una referencia obligatoria para cualquier administrador de sistemas.

4.2.1.- Ejercicio Opcional. Tarea periódica con cron.

Empleando `cron` y el utilidad `logger`. Programe una tarea periódica en su espacio de usuario que genere un mensaje de log ccon la etiqueta "ISE" y cuyo texto tenga la forma "<SusIniciales>: <Fecha y hora actual> – <Carga actual del Sistema>".

4.3.- Logs del Sistema.

Una forma básica de monitorización es analizar los logs generados por el sistema operativo o los servicios instalados en el servidor.

En Linux los logs se almacenan en el directorio /var/logs.

Este directorio contiene tanto logs del sistema linux (por ejemplo:messages, secure, lastlog, btmp, ...) como de servicios instalados (com los logs de Apache Httpd o Nginex, vistos en prácticas anteriores).

En algunos casos, el formato es texto plano y pueden consultarte con cualquier comando de tratamiento de textos (por ejemplo: secure, messages, cron) . Son muy comunes: cat, grep, more, less o tail.

Podrá apreciar como en sistemas con cierta historia, existen varias copias del archivos de logs con el mismo prefijo. Por ejemplo, cron, cron-20240222, cron-20240225, etc. Estos archivos se corresponden con "rotaciones" del log original (cron en el ejemplo anterior). Los archivos de logs se rotan para mantener un tamaño reducido que permita su consulta. De otra forma, un archivo de logs (por ejempo, de Httpd access.log) puede llegar a ser tan grande que no se pueda consultar en el equipo.

El programa responsable de realizar esta labor de rotado es `logrotate`

[39]. Este programa funciona programado como tarea periódica en cron (que veremos un poco más adelante). Logrotate permite, entre otras funciones, definir el rotado por tamaño o antigüedad, comprimir los archivos rotados o ejecutar tareas en el momento de la rotación. Por ejemplo, notificar al servicio origen sobre la acción de rotado. El fichero de configuración por defecto es /etc/logrotate.conf, pero las configuraciones específicas de los servicios las puede encontrar en /etc/logrotate.d/.

Apache Httpd rotatelogs es otra herramienta específica para realizar el rotado de los archivos de logs. Esta herramienta se ha vuelto muy popular por que no procisa notificar/reinicar el servicio original y porque es muy ligera y fácil de integrar con cualquier programa que emplee la salida estándar (stdout) para logs.

En algunos casos, los logs tienen formato binario y deben consultarse empleando comandos específicos. Por ejemplo, los comandos last y who muestra los accesos registrados en el archivo binario btmp, wtmp.

En Linux systemd centraliza la recepción y posible almacenamiento de todos los logs. El servicio systemd-journald está activo por defecto para el registro y la consulta del la actividad o "journal" del sistema. Por defecto, la información del journal se almacena de forma volátil (en memoria y en /run), de forma que se pierde al reiniciar el servidor. Para almacenarlos de forma permanente, debe modificarse la configuración en /etc/systemd/jornald.conf. En Rocky 9 la persistencia por defecto es "auto" por lo que, simplemente, creando el directorio /var/log/journal, se activa la persistencia no volátil.

Para consultar el journal del sistema debe emplear el comando journalctl [34][35]

Existen soluciones específicas para la gestión de grandes volúmenes de logs. Por ejemplo, procedentes de grandes instalaciones de servidores. En general, facilitan la consulta agregada, la extracción de métricas y eventos así como la correlación temporal entre equipos. Algunos ejemplos a destacar son: Elastic + Kibana + Logstash (ELK stack) [32] o Datadog [33].

4.3.1.- Ejercicio Opcional. Logs de arranque del sistema.

Consulte los logs del último arranque de una MV Rocky empleando journalctl. Concretamente, presente mensajes de niveles warning o más importantes.

4.4.- Grafana + Prometheus

Grafana [43] se autodefine como la pila de observabilidad (cualquier cosa que desee monitorizar). Para construir su pila tiene distintos componentes que se

estructuran en: dashboards, paneles y alertas.

Es una solución de código abierto con posibilidad de contratar su servicio alojado en la nube y de manera autogestionada para empresas con mayor presupuesto.

Dentro de las empresas que han adoptado esta tecnología encontramos algunas muy populares y muchos casos de éxito de implantación tienen un artículo asociado explicando cómo la han adoptado, se le anima a revisar algunos de éstos que suelen incluir algún vídeo explicativo (como, por ejemplo, el de una famosa empresa de citas [41])

Como suele ser habitual, tenemos la posibilidad de jugar con una demo disponible en: [42]

Grafana proporciona los servicios de visualización y alertas sobre los datos de telemetría, **pero no realiza por si misma la extracción de esta información**.

Para ello emplearemos Prometheus [44], una herramienta diseñada específicamente para extraer y almacenar la información de monitorización de forma eficiente y fácilmente adaptable a las necesidades de cada proyecto.

Alguno de los aspectos más interesantes de Prometheus es su optimización para la gestión de series temporales [45], los modelos de datos abiertos y personalizables para las necesidades del proyecto [46] y las de métricas estandarizadas [47].

Aunque Prometheus dispone de soluciones para visualizar los datos de monitorización y ejecutar alarmas, su funcionalidad es muy limitada.

Es por ello que Grafana (Visualización + Alarmas) junto con Prometheus (Recolección de Datos y Almacenamiento) son una combinación muy habitual en los Stacks Tecnológicos de muchas organizaciones IT.

Grafana y Prometheus pueden instalarse como servicios en una MV Rocky o en su ordenador anfitrión (en caso de ser compatible). Para ello, consulte las guías de instalación de cada servicio [48][49]. No obstante, la **forma preferida** de ejecución para estas prácticas será en forma de contenedores **Docker**.

En un directorio de su elección (por ejemplo: `progra`), dos archivos con los siguientes contenidos:

docker-compose.yml:

```
---
version: "3"
services:
  prometheus:
    image: prom/prometheus:v2.50.0
    ports:
      - 9090:9090
    volumes:
      - ./prometheus_data:/prometheus
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"

  grafana:
    image: grafana/grafana:9.1.0
    ports:
      - 4000:3000
    volumes:
      - ./grafana_data:/var/lib/grafana
    depends_on:
      - prometheus
```

prometheus.yml:

```
---
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: "prometheus_service"
    static_configs:
      - targets: ["prometheus:9090"]
```

Las interfaces web de Prometheus y Grafana estarán accesibles usando un navegador web (se recomienda Chrome o Firefox debido a algunos errores conocidos en Safari) en el equipo donde corre docker engine (por ejemplo, si es su equipo anfitrión será "localhost") en los puertos 9090 y 4000 respectivamente. Por ejemplo, `http://localhost:9090`.

La configuración proporcionada de Prometheus es un punto de partida. Solo contiene el scraping del propio servidor Prometheus. Debe definir Prometheus como un Datasource de Grafana para finalizar la integración [50]. Recuerde que dentro de un entorno de docker compose, los servicios se pueden acceder empleando el nombre declarado en la configuración. En este caso, el servicio de Grafana puede acceder al servicio de Prometheus empleando la url `http://prometheus:9090`.

4.4.1.- Ejercicio Obligatorio. Monitorización con Grafana + Prometheus.

Monitorización de Servidor Linux

Emplear la plataforma Prometheus + Grafana instalada para monitorizar las prestaciones de un servidor Rocky corriendo en una VM. El alumno/a puede elegir los componentes de Prometheus y Grafana que prefiera o crear nuevos componentes por si mismo/a. No obstante, se sugiere

emplear como base el exporter de Linux para Prometheus[56][57], configurado como un servicio [51] y emplear como base algún dashboard predefinido para Grafana [52][53]. Siga las instrucciones de cada dashboard para posibles ajustes en Prometheus. En Granafa vaya a `Dashboards → Import` y proporcionar el `Id` del dashboard.

El dashboard debe recibir como identificador, el nombre y apellidos del alumno/a en CamelCase junto con el sufijo "Linux". Por ejemplo, mariaGarciaPerezLinux. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: %CPU (MGP).

El alumno/a debe extender el dashboard anterior para incorporar indicadores sobre el el nivel de activación ("Activo"/"Inactivo", 1/0) de los servicios: SSHD y Apache Httpd en el equipo Linux monitorizado.

Además, deberá agregar un nuevo panel sobre el nivel de uso total de la CPU en tanto por ciento (%). A este panel se le asociará una alarma para que se dispare cuando la media del uso de CPU supere el 75% de CPU durante 5 minutos. Ponga de manifiesto el funcionamiento de la alarma empleando alguna herramienta de carga de las vistas en clase (por ejemplo, `stress`).

Para poner de manifiesto el funcionamiento de la monitorización, se adjuntará una memoria en la que se presenten:

- Descripción de la secuencia de pasos realizada para ejecutar el exporter de Linux. Con capturas de pantalla de los pasos seguidos para su ejecución y/o configuración.
- Capturas de pantalla de los monitores de Sshd y Httpd poniendo de manifiesto su comportamiento cuando los servicios están activos e inactivos.
- Captura de pantalla del monitor de uso de CPU antes y después de lanzar la carga de CPU.
- Captura de pantalla del comando empleado para disparar la carga de CPU.
- Captura de pantalla que ponga de manifiesto el disparo de la alarma asociada al monitor de CPU.

Monitorización de API WEB

La aplicación empleada en apartado anterior para la prueba de carga, expone en el path "/metrics" los indicadores de NodeJS para Prometheus. Para más información, el exporter de Prometheus de la API Web se ha generado empleando los componentes estandar: prom-client [54]y express-prom-bundle [55].

Cree un nuevo Dashboard con algunas de las métricas expuestas. Para el dashboard emplee como nombre su nombre y apellidos en CamelCase seguido del sufijo API. Por ejemplo, anaTorrentRamonetAPI. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: %Memoria (ATR).

Cree un monitores para las siguientes métricas:

- Tiempos de respuesta de los endpoints de la API (`http_request_duration_seconds_bucket`)
- Memoria disponible (`nodejs_heap_size_total_bytes`) vs la usada actualmente

(`nodejs_heap_size_used_bytes`)
- Uso de CPU (`process_cpu_seconds_total`)

Realice una memoria de prácticas en la que se ponga de manifiesto la ejecución de la prueba de carga diseñada para Jmeter y se aprecie el efecto de la misma en los monitores anteriormente descritos.

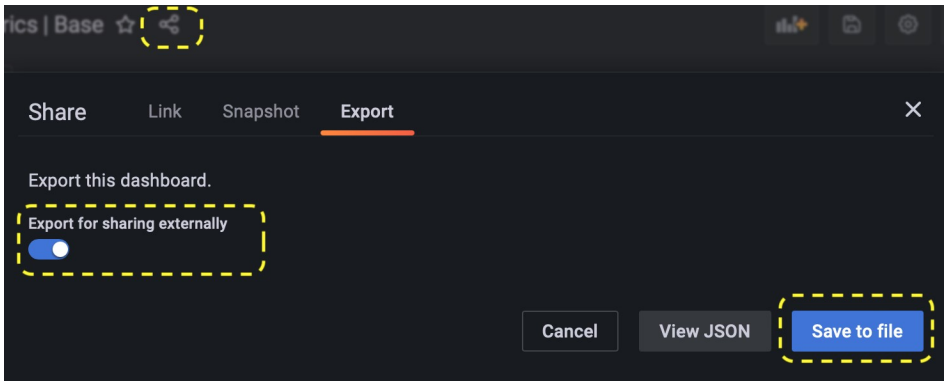
Entregables para este ejercicio:

Memoria en formato PDF con las capturas solicitadas en la monitorización del servidor linux.

Memoria en formato PDF con las capturas solicitadas en la monitorización de la API web.

Configuración de scraping de Prometheus (vista en : prometheus.yaml)

Definición de los dos dashboards de Grafana. Para esto último, debe emplear la opción de "Share Dashboard or Panel" y exportar el Dashboard con la opción "Export for sharing externally".



4.2.- Otras soluciones de monitorización.

Existe un amplio catálogo de soluciones al respecto, algunos ejemplos a destacar por su amplia difusión son los siguientes.

Munin

Munin (significa memoria) está disponible en http://munin-monitoring.org/.

Para su instalación puede hacerlo compilando el código fuente alojado en la página o a través del paquete disponible en el repositorio EPEL (http://fedoraproject.org/wiki/EPEL/es).

"¿Cómo puedo utilizarestos paquetes adicionales?" es el título de la subsección dentro de la página donde se explica cómo activar el repositorio que contiene los paquetes. Tan solo debe instalar un .rpm y podrá usar yum para instalar munin.

En http://demo.munin-monitoring.org/ puede ver una demo del sistema de monitorización en funcionamiento.

Nagios y Naemon

Es otro software muy usado para monitorizar sistemas. Recientemente ha pasado por una transformación de proyecto de la comunidad a proyecto empresarial. Debido a esto, ha aparecido Naemon http://naemon.org que acaba de hacer una nueva release.

Para veruna demo, podemos identificarnos a través de la demo que hace uso de la popular interfaz Thruk (https://demo.thruk.org/demo/thruk/cgi-bin/login.cgi?demo/thruk).

Ganglia

Es un proyecto alojado en http://ganglia.sourceforge.net/ que monitoriza sistemas

de cómputo distribuidos (normalmente para altas prestaciones) y permite una gran escalabilidad.

Como puede verse en su página, importantes instituciones y compañías lo usan (p.ej. UC Berkely, MIT, Twitter, . . .).

Netdata

Otra empresa que ha ido desarrollando un producto un poco más enfocado a nicho ha sido Netdata. Su especialización ha sido la minimización del impacto de la solución de monitorización pudiendo observar los datos en tiempo real.

Recientemente han incorporado otros aspectos (ya presentes en otras soluciones de monitorización) como la inclusión de aprendizaje automático para generar alertas. Además, conscientes de la popularización de Grafana y viendo que es una solución que abarca más aspectos, han desarrollado una integración para esta otra solución de monitorización.

Elastic (pila ELK)

Aunque no empezó su desarrollo como una herramienta de monitorización sino como una de búsqueda de información, el ecosistema ha crecido y se ha definido la pila ELK: Elastic + Logstash + Kibana.

Gracias a estos tres componentes, se puede almacenar información (logstash) mediante lo que denominan beats, indexarla y hacer búsquedas eficientes (Elastic) y visualizarla en paneles (Kibana).

Existen más plugins y es una solución bastante popular por sus integraciones, no obstante, es software privativo.

ZABBIX

Es otro programa que permite monitorizar el sistema también de código abierto.

Su instalación es relativamente sencilla ya que solo hay que añadir los repositorios (visto en la práctica anterior) e instalarlo con el gestor de paquetes.

Referencias

1:Docker Container Runtime, https://www.docker.com/products/container-runtime/
2:Podman Open Source Container Runtime, https://podman.io/
3:Docker Compose Overview, https://docs.docker.com/compose/
4:buildah tools, https://github.com/containers/buildah/blob/main/README.md
24:Docker Getting Started, https://docs.docker.com/get-started/
6:Play with Docker, https://training.play-with-docker.com/
5:Hello World with Docker, https://training.play-with-docker.com/ops-s1-hello/
7:Microservices, https://martinfowler.com/articles/microservices.html
8:Install Docker Engine, https://docs.docker.com/engine/install/
9:Docker Desktop, https://docs.docker.com/desktop/
10:Benckmarks in Sourceforte, https://sourceforge.net/directory/linux/?q=benchmark
11:OpenBenchmarking, https://openbenchmarking.org/
12:Phoronix Test Suite, https://www.phoronix-test-suite.com/
13:Phoronix Installation, https://www.phoronix-test-suite.com/?k=downloads
14:Phoronix Docker Image, https://hub.docker.com/r/phoronix/pts/
15:Apache Benchmark, https://httpd.apache.org/docs/2.4/programs/ab.html
16:How to use Apache Benchmark for web server performance testing, https://www.datadoghq.com/blog/apachebench/
17:Jmeter Home Page, https://jmeter.apache.org/
18:Flood Home Page, https://www.flood.io/
19:Gatling Home Page, https://gatling.io/
20:Locust Home, https://locust.io/%20
21:K6 Load Testing, https://k6.io/
22:Artillery Home Page, https://www.artillery.io/
23:Docker Networking, https://docs.docker.com/network/
25:Dockerfile Reference, https://docs.docker.com/reference/dockerfile/
26:Compose Reference, https://docs.docker.com/compose/compose-file/compose-file-v3/
27:Dockerfile Best Practices, https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
28:Top Man page, https://man7.org/linux/man-pages/man1/top.1.html
29:Top Guide, https://www.booleanworld.com/guide-linux-top-command/
30:Htop Guide, https://docs.rockylinux.org/gemstones/htop/
40:/proc filesystem, https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html
31:Stress Tool, https://www.tecmint.com/linux-cpu-load-stress-test-with-stress-ng-tool/
36:Managing Timers, https://documentation.suse.com/smart/systems-management/html/systemd-working-with-timers/index.html
37:Automating processes with cron and crontab, https://docs.rockylinux.org/guides/automation/cron_jobs_howto/
38:Cron expression calculator, https://crontab.guru/
39:How to rotate logs with logrotate, https://www.ubuntumint.com/rotate-logs-with-logrotate-linux/
34:How to use Journalctl , https://www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulate-systemd-logs
35:Query the systemd journal, https://documentation.suse.com/sles/12-SP5/html/SLES-all/cha-journalctl.html

32:ELK características , https://www.elastic.co/es/elastic-stack/features
33:Datadog Home, https://www.datadoghq.com/
43:Grafana Oss Home, https://grafana.com/oss/grafana/
41:Tinder & Grafana, https://grafana.com/blog/2019/03/26/tinder-grafana-a-love-story-in-metrics-and-monitoring/?plcmt=footer
42:Play Grafana, https://play.grafana.org/
44:Prometheus Docs, https://prometheus.io/docs/introduction/overview/
45:Prometheus Overview, https://prometheus.io/docs/introduction/overview/
46:Prometheus Data Model, https://prometheus.io/docs/concepts/data_model/
47:Prometheus Metrics, https://prometheus.io/docs/concepts/metric_types/
48:Prometheus Getting Started, https://prometheus.io/docs/prometheus/latest/getting_started/
49:Install Grafana on RHEL or Fedora, https://grafana.com/docs/grafana/latest/setup-grafana/installation/redhat-rhel-fedora/
50:Grafana support for Prometheus, https://prometheus.io/docs/visualization/grafana/
56:Prometheus Exporter, https://prometheus.io/docs/guides/node-exporter/
57:Github Prometheus Node Exporter, https://github.com/prometheus/node_exporter
51:How to monitor Linux Servers using Prometheus Node Exporter, https://devopscube.com/monitor-linux-servers-prometheus-node-exporter/
52:Linux Host Metrics Dashboard, https://grafana.com/grafana/dashboards/10180-kds-linux-hosts/
53:Node Exporter Full Dashboard, https://grafana.com/grafana/dashboards/1860-node-exporter-full/
54:Exporter de Prometheus para Nodejs, https://github.com/siimon/prom-client
55:Exporter de Prometheus para Express , https://www.npmjs.com/package/express-prom-bundle