MODUL 9

CONTAINER MONITORING WITH PROMETHEUS & GRAFANA

9.1 Topik Pembahasan

1. *Platform open-source docker*.
2. Layanan *container*.
3. Sistem monitoring dengan Prometheus.
4. Integrasi Prometheus dan Grafana.

9.2 Tujuan Praktikum

1. Menginstal Docker dan Docker Compose pada Ubuntu di AWS EC2.
2. Deploy Prometheus, Node Exporter, dan Grafana menggunakan Docker Compose.
3. Monitoring sistem menggunakan Prometheus dan Grafana

9.3 Alat dan Bahan

1. Laptop

9.4 Dasar Teori

9.4.1 Docker

Gambar 1. Docker

Docker merupakan sebuah platform yang menyediakan kemampuan untuk mengemas dan menjalankan sebuah aplikasi dalam sebuah lingkungan terisolasi yang disebut dengan container. Dengan adanya isolasi dan keamanan yang memadai memungkinkan kamu untuk menjalankan banyak container di waktu yang bersamaan pada *host* tertentu.

Docker diperkenalkan pada tahun 2013 oleh *Solomon Hykes* pada acara *PyCon.*

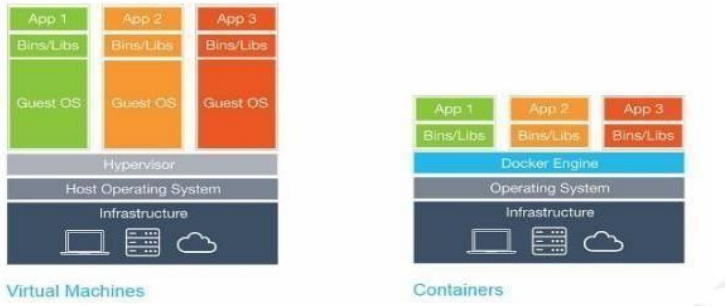
Beberapa bulan setelahnya docker secara resmi diluncurkan, tepatnya pada tahun 2014. Semenjak itu *docker* menjadi sangat populer di kalangan *developer* luar negeri, tetapi belum terlalu populer di Indonesia. Sekarang, *docker* dapat dijalankan pada sistem operasi *MacOS*, *Windows*, dan *Linux*.

Kelebihan *docker* antara lain sistem container berbasis *open source*, mudah dibuat dan mendistribusikan *images*, dapat menjalankan images yang sama di setiap *container* yang telah dibuat, dan pemanfaatan sumber daya yang lebih baik. Kekurangannya, sulit mengelolah data persisten dalam sebuah wadah.

9.4.2 Container

Container merupakan sebuah istilah teknologi software yang mengemas dan mengisolasi applikasi secara virtual untuk mempermudah software deployment. Berbeda dengan konsep traditional *virtual machine*, *container* tidak membutuhkan *dedicated operating system* (OS Kernel) tetapi *container kernel* dapat di pergunakan secara bersamaan.

Container memungkinkan aplikasi dapat di pindah-pindahkan dengan cepat dan andal dari satu lingkungan komputasi ke lingkungan komputasi lainnya. *Container* merupakan paket *software* yang ringan, mandiri, dan dapat dieksekusi. Paket *container* mencakup semua yang diperlukan untuk menjalankan aplikasi: kode, runtime, alat sistem, pustaka sistem, danpengaturan.



Gambar 2 Ilustrasi perbedaan Virtual Machine dan Contai

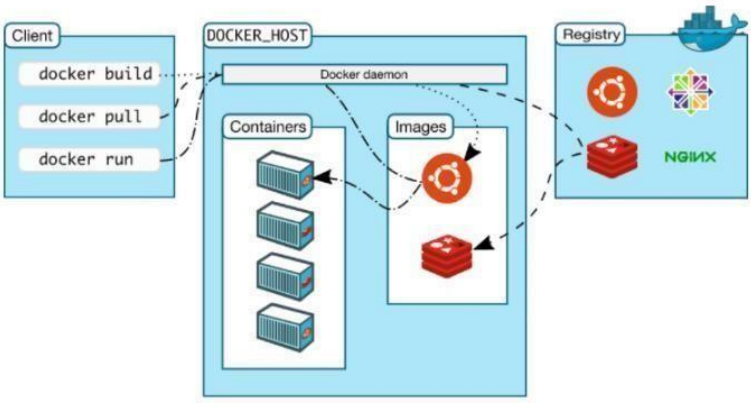
Pada VM, alokasi *resource* dilakukan pada awal instalasi sehingga ketika ada 2 atau lebih VM yang sudah ditentukan *resource*-nya dan salah satunya kehabisan resource, maka VM yang kehabisan resource tidak dapat mengambil resource dari VM lain. Jika di Container, alokasi resource dapat dilakukan oleh *host server* sehingga host dapat melakukan pengambilan *resource* pada *hardware* sesuai yang dibutuhkan container itu sendiri. Pada

VM memiliki *Hypervisor* yang digunakan untuk menjalankan sistem di VM. Jika tidak ada hypervisor, maka VM tidak dapat berjalan. Di sisi lain, *container* dapat menjalankan programsecara langsung di OS itu sendiri.

9.4.3 Fungsi Docker

1. Mempermudah Pengembangan Aplikasi. *Docker* bisa mempermudah pekerjaan developer ketika mengembangkan aplikasi. Alasannya, *Docker* lebih hemat *resource* dan mampu menyediakan *environment* yang stabil untuk dijalankan di perangkat apapun, mulai dari *cloud server* hingga komputer pribadi.
2. Menyederhanakan Konfigurasi. *Docker* tidak memiliki *overhead* sehingga *developer* bisa menjalankan aplikasi yangdiuji tanpa konfigurasi tambahan
3. Mendukung *Multitenancy*. *Docker* cocok digunakan untuk membuat aplikasi berstruktur *multitenance* seperti *Software as a Service* (SaaS). Anda bisa membuat lebih dari satu *environment* yang terisolasi dan menjalankan objek aplikasi untuk setiap tenant.
4. Meningkatkan Sumber Daya dengan Cepat. Dengan *docker*, peningkatan sumber daya perangkat dapat dilakukan dengan cepat sehingga durasi pengembangan software lebih singkat.

9.4.4 Prinsip Kerja Docker



Gambar 3 Ilustrasi arsitektur client-server dalam prinsip kerja Docker

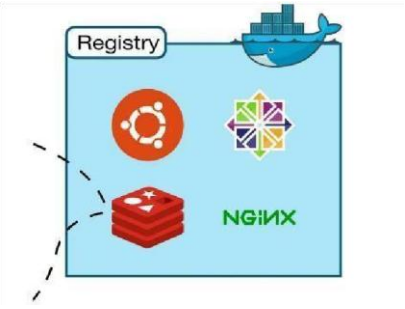
AWS *Docker* bertindak sebagai alat yang digunakan untuk menjalankan container. Container ini bertindak seperti virtual machine, yang seperti simulasi komputer yang berjalan di dalam komputer asli pengembang. Pada *virtual machine* ini, nantinya semua kode sistem tersimpan untuk menjalankan simulasi seolah-olah adalah operasi sistem utama. *Docker* bertindak melakukan virtualisasi sistem operasi di dalam sistem operasi *host*. *Docker*

membangun *container* berdasarkan gambar yang berisi kode program. Gambar atau *images* ini ditumpuk satu sama lain untuk lantas membangun pengaturan yang lengkap. Gambar bertumpuk dapat berbagi gambar inti yang sama, seperti cabang-cabang dari batang pohon yang sama.

Proses ini dapat dimisalkan ketika seorang pengembang ingin menguji tampilan situs *web* baru di sebuah *browser web* berbeda, tetapi pengembang tidak ingin langsung melakukan instalasi setiap browser ke komputernya. Melakukan hal ini dapat menyebabkan masalah dengan peramban pribadi pengembang. Dalam pengujian inilah nantinya *docker* berguna sebagai tempat uji coba. *Docker* bekerja menggunakan sistem arsitektur *client-server*. Dalam hal ini nantinya klien akan berkomunikasi dengan apa yang disebut *daemon docker* atau proses pengelolaan *docker images*, *container*, *network*, dan *volume* penyimpanan. *Docker daemon* nantinya akan menerima permintaan pemrosesan dari *Docker Engine Rest* API yang berguna untuk interaksi dan bisa diakses oleh klien melalui *hypertext transfer protocol* (HTTP).

Klien *docker* yang lain adalah *docker compose* yang dapat memungkinkan pengembang untuk bekerja dengan aplikasi yang terdiri dari sekumpulan container. Dalam ilustrasi di atas container yang dipakai adalah Ubuntu, sehingga nantinya akan menempel secara interaktif ke sesi baris perintah yang dimiliki pengembang. Saat pengembang mulai menjalankan perintah terhadap container *Ubuntu* maka akan terjadi konfigurasi oleh *docker* dan pembuatan *container* baru. Selanjutnya, *docker* akan mengalokasikan dokumen sistem ke dalam container sebagai lapisan terakhirnya. Hal ini memungkinkan container yang sedang berjalan untuk membuat atau memodifikasi dokumen dan direktori ke sistem dokumen lokal.

9.4.5 Docker Registry



Gambar 4 Docker Registry

Registry adalah sebuah *repository* atau sebuah penyimpanan image terpusat, kasarnya *image* yang sudah dibuat oleh orang orang atau mungkin kalian ikut membuat

sebuah image, maka *registry* adalah sebuah tempat yang tepat untuk menyimpan *images* kalian disebarkan.

Contoh dari *Docker Registry* sebagai berikut:

1. *Harbour*, opsi sumber terbuka lainnya.
2. *Docker Hub*, *registry* resmi berbasis *cloud* dan *docker*. Berbeda dengan *docker registry open source*, *Docker Hub* adalah layanan terkelola sepenuhnya yang hanya tersedia dari *Docker*.
3. *JFrog Artifactory*, manajer biner yang dapat digunakan untuk meng*hosting* gambar *container* baik lokal maupun di *cloud*.
4. *Amazon Elastic Container Registry*, layanan registri *docke*r di *cloud* AWS.
5. *Azure Container Registry*, solusi registrer kontainer utama yang dihosting di *Azure*.
6. *Registry Container Google Cloud*.

9.4.6. Prometheus

Prometheus adalah sistem monitoring open-source yang digunakan untuk mengumpulkan, menyimpan, dan menganalisis data metrik dari server, aplikasi, hingga perangkat jaringan. Dengan Prometheus, Anda dapat memantau berbagai parameter penting seperti penggunaan CPU dan memori, jumlah request dan response aplikasi, waktu respon layanan, serta status kesehatan (health check) dari suatu service. Prometheus bekerja dengan metode pull, yaitu secara aktif mengambil data dari target yang telah dikonfigurasi. Semua data yang dikumpulkan disimpan dalam bentuk time-series, sehingga mudah dianalisis secara historis untuk mengetahui pola performa sistem maupun mendeteksi anomali.

Fitur yang Prometheus Sediakan

Prometheus menyediakan beberapa fitur utama untuk pemantauan dan pemberitahuan pada sistem dan layanan, termasuk:

- Pengumpulan metric: Prometheus dapat mengambil metric dari berbagai sumber, termasuk sistem dan layanan yang berjalan pada mesin yang sama, atau pada mesin lain dalam jaringan.
- Penyimpanan waktu-seri: Prometheus menyimpan semua metric yang dikumpulkan dalam basis data waktu-seri, memungkinkan untuk mudah mengkueri dan menganalisis data.
- Pemberitahuan: Prometheus memungkinkan pengguna untuk mendefinisikan aturan untuk menghasilkan pemberitahuan berdasarkan metric yang dikumpulkan. Pemberitahuan ini dapat dikirimkan ke berbagai saluran notifikasi, seperti email atau Slack.
- Bahasa kueri: Prometheus mencakup bahasa kueri yang dibangun dalam, PromQL, untuk mengkueri dan menganalisis metric yang disimpan.
- Dasbor dan visualisasi: Prometheus dapat diintegrasikan dengan alat visualisasi dan dasbor lainnya, seperti Grafana, untuk menampilkan metrik yang dikumpulkan dalam format grafis.
- Fleksibilitas: Prometheus sangat dapat dikonfigurasi dan dapat digunakan di berbagai lingkungan, mulai dari sistem simpul tunggal kecil hingga sistem terdistribusi besar.

9.4.6. Grafana

Grafana adalah platform visualisasi open-source yang sangat populer di kalangan sysadmin dan DevOps. Ia mampu menampilkan data dari berbagai sumber seperti Prometheus, Loki, InfluxDB, dan banyak lagi. Dengan Grafana, kamu bisa membuat dashboard interaktif yang menampilkan metrik server, aplikasi, hingga log, semuanya dalam satu tampilan yang elegan.

Cara Kerja Grafana

Dashboard analytics ini bekerja dengan mengambil data dari berbagai sumber, mengolahnya, lalu menampilkan hasilnya dalam bentuk visual yang mudah dipahami. Berikut adalah tahapan utama cara kerja Grafana:

1. Menghubungkan ke Sumber Data

Langkah pertama adalah menyambungkan Grafana ke sumber data. Misalnya, jika kamu ingin memantau performa server menggunakan Prometheus, kamu bisa menghubungkannya ke Grafana sebagai sumber data utama.

2. Membuat Query untuk Mengambil Data

Setelah sumber data terhubung, kamu bisa menggunakan query editor untuk menulis kueri yang akan mengambil data dari sumber tersebut. Platform ini menyediakan antarmuka intuitif yang memudahkan pengguna dalam menulis kueri.

3. Menyusun Dashboard dan Panel

Setelah data berhasil diambil, kamu bisa menyusunnya dalam sebuah dasbor interaktif. Kamu bisa menambahkan berbagai panel untuk menampilkan data dalam bentuk grafik, tabel, heatmap, atau indikator lainnya.

4. Mengatur Alerting dan Notifikasi

Jika kamu ingin mendapatkan pemberitahuan saat ada sesuatu yang tidak normal dalam sistem, kamu bisa mengatur alerting rules. Tools ini akan terus memantau kondisi yang telah ditentukan dan mengirim notifikasi jika ada pelanggaran batas yang sudah diset.

5. Mengakses dan Menganalisis Data Secara Real-Time

Setelah semuanya diatur, dashboard analytics ini akan terus memperbarui data secara real-time, memungkinkan pengguna untuk melihat tren dan pola dari waktu ke waktu.

LANGKAH PRAKTIKUM

1. Buat instance EC2 (Ubuntu 24.04 LTS).
2. Pastikan security group membuka port:
   - `22` → SSH
   - `9090` → Prometheus
   - `9100` → Node Exporter
   - `3000` → Grafana
3. Login ke EC2 via SSH.
4. Install Docker & Docker Compose
5. Buat folder project

```
mkdir ~/monitoring-project
cd ~/monitoring-project
```

6. Buat file docker-compose.yml:

```
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    restart: always

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    ports:
      - "9100:9100"
    restart: always

  grafana:
    image: grafana/grafana-oss:latest
    container_name: grafana
    ports:
      - "3000:3000"
    restart: always
```

7. Buat file prometheus.yml di folder yang sama:

```
global:
  scrape_interval: 15s
```

```yaml
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['prometheus:9090']

  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter:9100']
```

8. Jalankan container: docker ps
9. Buka http://<EC2-IP>:9090/targets
10. Pastikan muncul 2 target:
    - `prometheus:9090` → UP
    - `node-exporter:9100` → UP
11. Setup Grafana
    - Buka `http://<EC2-IP>:3000`
    - Login: `admin` / `admin` (default)
    - Tambahkan Data Source:
    - Pilih **Prometheus**
    - URL: `http://prometheus:9090` (jika di container network)
    - Save & Test → harus sukses

12. Buat dashboard Grafana
13. Klik + → **Dashboard** → **Add new panel**
14. Pilih metric, misal:
    - CPU → `node_cpu_seconds_total`
    - Memory → `node_memory_MemAvailable_bytes`
    - Disk → `node_disk_io_time_seconds_total`
    - Network → `node_network_receive_bytes_total`
15. Klik **Apply** → grafik akan muncul
16. Selesai