Analisis Perbandingan Kinerja High Availability Pada Cluster Docker Swarm Dan K3S

Dani Maulana Ferdiansyah[1], Agus Prihanto[2]

[1,2] Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]danimaulanaf@gmail.com
[2]agusprihanto@unesa.ac.id

***Abstrak*— Penelitian ini berjudul "Analisis Perbandingan Kinerja *High Availability* pada Cluster Docker Swarm dan K3S." Penelitian ini berangkat dari latar belakang bahwa containerization telah menjadi paradigma penting dalam pengembangan aplikasi *modern*, dengan *high availability* menjadi krusial untuk kontinuitas layanan. Docker Swarm dan K3S merupakan dua platform orkestrasi container yang populer, namun belum ada kajian komprehensif yang membandingkan kinerja keduanya dalam konteks high availability. Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja high availability pada cluster Docker Swarm dan K3S, dengan fokus pada skalabilitas, kehandalan, kinerja, dan efisiensi sumber daya. Metode penelitian yang digunakan adalah eksperimen komparatif, di mana kedua platform akan diuji dalam kondisi yang sama untuk mengukur performa berdasarkan metrik CPU dan memori. Teknik analisis data melibatkan analisis kuantitatif terhadap data kinerja yang diperoleh dari pengujian. Landasan teori mencakup konsep containerization, orkestrasi container, dan prinsip high availability. Docker Swarm dan K3S akan dievaluasi berdasarkan kemampuan mereka dalam menjaga ketersediaan layanan selama situasi kegagalan (fail over) dan penggunaan sumber daya yang efisien.**

Hasil penelitian ini diharapkan memberikan wawasan yang berharga bagi profesional IT dan organisasi dalam memilih platform orkestrasi container yang paling sesuai dengan kebutuhan mereka.

***Kata Kunci — Server, Container Orchestration, High Availability*.**

I. PENDAHULUAN

Dalam era modern infrastruktur IT, penggunaan containerization telah menjadi prasyarat yang penting bagi pengembangan aplikasi yang fleksibel dan dapat diandalkan. Seiring dengan itu, kebutuhan akan high availability dalam menjaga kontinuitas pelayanan menjadi semakin mendesak. Dua platform orkestrasi container yang menonjol, Docker Swarm dan K3S, menawarkan solusi untuk manajemen cluster container, namun belum ada kajian yang menyeluruh yang membandingkan kinerja keduanya dalam konteks high availability. Oleh karena itu, penelitian ini bertujuan untuk menyelidiki dan menganalisis perbandingan kinerja antara Docker Swarm dan K3S, dengan fokus pada aspek-aspek seperti skalabilitas, kehandalan, kinerja, dan efisiensi sumber daya. Melalui pemahaman yang mendalam terhadap kelebihan dan kelemahan masing-masing platform, diharapkan penelitian ini akan memberikan wawasan yang berharga bagi profesional IT dan organisasi dalam memilih platform yang paling sesuai dengan kebutuhan mereka untuk infrastruktur container yang handal dan efisien.

Containerization telah menjadi paradigma dalam pengembangan perangkat lunak *modern*. Konsep ini memungkinkan aplikasi beserta dependensinya dikemas secara terisolasi ke dalam unit yang disebut container [2]. Docker, sebagai salah satu platform container paling populer, telah mengubah cara kita membangun, menyampaikan, dan menjalankan aplikasi. Pendekatan ini memberikan fleksibilitas yang besar dalam menyusun dan mendistribusikan aplikasi di berbagai lingkungan, dari lingkungan pengembangan hingga produksi.

Dalam infrastruktur IT, terutama untuk aplikasi dan layanan yang penting, kebutuhan akan *high availability* menjadi esensial. High availability menjamin bahwa layanan tetap tersedia dan berfungsi meskipun terjadi kegagalan pada salah satu komponen sistem. Ini menghindari downtime yang dapat merugikan, memastikan kontinuitas operasional, dan memenuhi harapan pengguna akan ketersediaan layanan yang tak terputus [4]. Oleh karena itu, implementasi high availability menjadi prioritas utama bagi organisasi dalam merancang infrastruktur IT mereka.

Docker Swarm dan K3S adalah dua platform orkestrasi container yang memiliki peran penting dalam memfasilitasi manajemen dan pengelolaan cluster container. Docker Swarm, sebagai bagian dari ekosistem Docker, menyediakan alat untuk mengelola cluster Docker dengan pendekatan yang lebih sederhana. Di sisi lain, K3S merupakan proyek open-source yang dikembangkan oleh Rancher Labs, menawarkan solusi Kubernetes yang ringan dan mudah diimplementasikan, terutama untuk kasus penggunaan edge dan IoT. Kedua platform ini menawarkan cara yang berbeda dalam menyusun dan mengelola cluster container, dengan berbagai kelebihan dan kekurangan masing-masing.

Dalam konteks judul skripsi, "High Availability (HA)" merujuk pada kemampuan cluster kontainer yang dibangun menggunakan Docker Swarm dan K3s untuk tetap beroperasi dan menyediakan layanan tanpa gangguan, bahkan jika terjadi kegagalan pada salah satu kontainer di dalamnya. Jadi, HA dalam skripsi mengacu pada kemampuan sistem untuk

menjaga ketersediaan layanan dengan waktu henti minimum dalam cluster kontainer.

Dalam analisis, akan dibandingkan bagaimana Docker Swarm dan K3s mengimplementasikan konsep High Availability (HA) ini, khususnya dalam hal replikasi dan manajemen kontainer. Akan dievaluasi seberapa efektif kedua

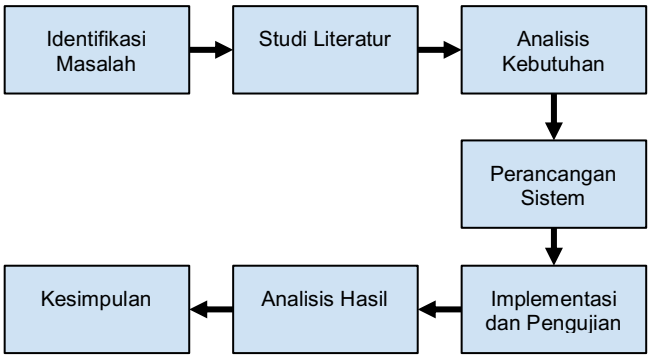
platform ini dalam mempertahankan ketersediaan layanan dalam cluster, termasuk dalam situasi kegagalan kontainer.

Tujuan utama dari penelitian ini adalah untuk melakukan analisis perbandingan kinerja antara Docker Swarm dan K3S dalam konteks high availability cluster. Beberapa aspek yang akan dievaluasi termasuk skalabilitas, kehandalan, kinerja, dan efisiensi sumber daya. Melalui penelitian ini, diharapkan dapat memberikan pemahaman yang lebih baik tentang kelebihan dan kelemahan masing-masing platform, sehingga organisasi dapat membuat keputusan yang lebih terinformasi dalam merancang dan mengelola infrastruktur container mereka.

II. METODOLOGI PENELITIAN

A. Metodologi Penelitian

Pada gambar 1 Dengan fokus pada analisis perbandingan kinerja *High Availability* pada *Cluster* antara Docker Swarm dan K3S, metode yang diterapkan pada penelitian ini adalah metode *experimental design*. Pendekatan eksperimental ini dimaksudkan untuk memungkinkan analisis yang komprehensif dan objektif terhadap kinerja kedua platform orkestrasi container tersebut dalam konteks cluster tingkat. Dengan menggunakan pendekatan ini, bertujuan untuk mendapatkan pemahaman yang mendalam tentang keunggulan dan kelemahan masing-masing platform, serta potensi penggunaannya dalam implementasi *High Availability* pada *Cluster*.



Gbr. 1 Experimental Design

1) Identifikasi Masalah

Dalam era komputasi modern, High Availability Cluster menjadi krusial dalam memastikan ketersediaan dan keandalan sistem. Namun, implementasi serta manajemen cluster semacam ini seringkali menemui tantangan yang kompleks. Oleh karena itu, penelitian ini bertujuan untuk membandingkan kinerja antara dua teknologi populer dalam membangun High Availability Cluster, yaitu Docker Swarm dan K3s. Dengan demikian, dapat diidentifikasi keunggulan dan kelemahan masing-masing platform untuk membantu pemilihan teknologi yang tepat dalam implementasi cluster.

2) Studi Literatur

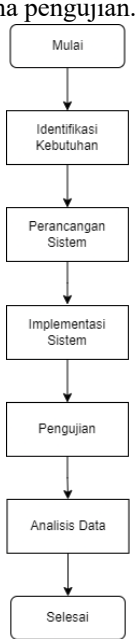
Dalam kajian literatur ini, akan dilakukan tinjauan mendalam terhadap konsep High Availability Cluster serta teknologi Docker Swarm dan K3s. Dalam konteks ini, kelebihan dan kelemahan dari kedua teknologi akan dievaluasi secara komprehensif berdasarkan literatur terkait dan penelitian sebelumnya yang relevan. Tujuannya adalah untuk memperoleh pemahaman yang mendalam tentang aspek-aspek kunci yang mempengaruhi kinerja dan keandalan cluster.

3) *Analisis* Kebutuhan

Analisis kebutuhan akan fokus pada identifikasi persyaratan yang diperlukan untuk membangun dan mengelola High Availability Cluster menggunakan Docker Swarm dan K3s. Selain itu, faktor-faktor yang memengaruhi kinerja dan keandalan kedua sistem akan dievaluasi secara kritis. Dengan memahami kebutuhan dan faktor-faktor yang terlibat, diharapkan dapat merancang eksperimen yang tepat untuk membandingkan kinerja keduanya.

4) *Perancangan* Sistem

Pada gambar 2 Perancangan sistem akan mencakup arsitektur dari High Availability Cluster yang akan dibangun menggunakan Docker Swarm dan K3s. Rancangan eksperimental yang terstruktur juga akan disusun untuk menguji kinerja dan keandalan kedua sistem. Selain itu, parameter-parameter yang relevan akan ditetapkan untuk pengukuran kinerja selama pengujian.



Gbr. 2 Perancangan Sistem

5) Implementasi dan Pengujian

Langkah selanjutnya adalah implementasi sistem berdasarkan rancangan yang telah dibuat dan melakukan pengujian yang terstruktur. Proses ini akan melibatkan pengujian kinerja dan keandalan kedua sistem dalam berbagai skenario beban kerja. Data yang terkumpul akan dianalisis untuk mengevaluasi kinerja relatif dari Docker Swarm dan K3s.

6) Analisis Hasil

Analisis hasil akan melibatkan pengolahan data yang dikumpulkan selama pengujian. Hasilnya akan digunakan untuk membandingkan kinerja antara Docker Swarm dan K3s, serta menginterpretasikan temuan yang diperoleh dari eksperimen. Dengan demikian, dapat ditarik kesimpulan yang kuat terkait dengan perbandingan kinerja kedua sistem.

7) Kesimpulan

Pada bagian kesimpulan, temuan utama dari penelitian ini akan dirangkum dan dianalisis secara menyeluruh. Kesimpulan yang ditarik akan mencakup implikasi praktis dari perbandingan kinerja antara Docker Swarm dan K3s, serta saran untuk pengembangan dan implementasi lebih lanjut dalam konteks penggunaan nyata.

B. Analisis Kebutuhan Sistem

Dalam merancang infrastruktur yang tangguh dan efisien, kebijaksanaan dalam menentukan spesifikasi hardware dan perangkat lunak adalah langkah krusial. Dalam konteks cluster, penyesuaian yang cermat terhadap kebutuhan masing-masing *node* adalah kunci untuk mencapai keseimbangan yang optimal antara kinerja dan efisiensi. Berikut merupakan kebutuhan yang dibagi menjadi beberapa bagian, yaitu:

1) Kebutuhan Perangkat Keras (*Hardware*)

Perangkat keras yang diperlukan guna tujuan penelitian yaitu Laptop sebagai uji coba dengan spesifikasi berikut:

Processor : AMD Ryzen 7 6800H
RAM : 16 GB
SSD : 1 TB
Sistem Operasi : Ubuntu

2) Kebutuhan Perangkat Lunak (*Software*)

Perangkat lunak berfungsi untuk pengoperasian sistem pada penelitian ini. Pada penelitian ini, sistem operasi yang digunakan berbasis GNU/Linux seperti berikut:

Sistem Operasi : Ubuntu 24

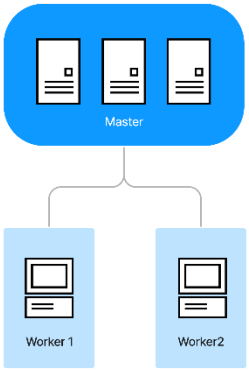
3) Kebutuhan *Virtual Machine*

Virtual machine (VM) diperlukan untuk menjalankan simulasi dan uji coba pada penelitian ini. Berikut adalah kebutuhan VM untuk penelitian ini:

Hypervisor : AMD Ryzen 7 6800H
CPU : 2 Core
RAM : 2 GB
Penyimpanan : 20 GB
Sistem Operasi : Ubuntu 22 LTS *Server*

C. Perancangan Sistem

Pada gambar 3 Perancangan sistem, fokus utama adalah pada desain dan implementasi *High Availability Cluster* menggunakan Docker Swarm dan K3s. Cluster yang akan dibangun terdiri dari tiga *node master* dan dua *node worker*. Pertama-tama, akan dilakukan perancangan arsitektur untuk kedua solusi tersebut. Untuk Docker Swarm, tiga *node master* akan diatur sedemikian rupa untuk membentuk sebuah cluster yang dapat mengelola dan mendistribusikan beban kerja dengan efisien. Sementara itu, K3s akan diimplementasikan dengan pola yang sama, di mana tiga *node master* akan membentuk konsensus dan pengelolaan terhadap *worker node* dilakukan.



Gbr. 3 Arsitektur Perancangan Sistem

D. Perancangan Pengujian

Perancangan pengujian merupakan tahap kunci dalam mengevaluasi kinerja *High Availability* pada Cluster menggunakan Docker Swarm dan K3s. Pengujian akan dilakukan dengan mempertimbangkan beberapa skenario beban kerja yang berbeda untuk menilai respons dan kinerja kedua platform. Adapun metode pengujian yang dilakukan pada penelitian ini yaitu :

1) Pengujian *High Availability*

Uji *High Availability* dilakukan dengan sengaja menonaktifkan satu *node* pada *server*. Tujuannya adalah untuk mengevaluasi keberhasilan dalam menangani *Downtime*. Adapun pengujian yang dilakukan sebagai untuk melihat tingkat keberhasilan *Availability*.

2) Pengujjian *Performance*

Setiap skenario beban kerja akan dijalankan secara berurutan dan diukur dengan parameter-parameter kinerja yang relevan, seperti waktu penggunaan CPU serta memori dilakukan pada saat *stress request server* dalam menangani *client*. Pengukuran ini akan dilakukan secara berulang untuk memastikan konsistensi hasil.

E. Kajian Pustaka

1) *High Availability*

High Availability (Kehandalan Tinggi) adalah konsep dalam teknologi informasi yang merujuk pada kemampuan sistem atau layanan untuk tetap tersedia dan beroperasi secara normal selama periode waktu tertentu, biasanya diukur dalam persentase waktu (misalnya, 99,99% uptime). Tujuan dari *High Availability* adalah untuk mengurangi atau menghilangkan waktu down time sebuah sistem (galat), sehingga meningkatkan ketersediaan layanan dan mengurangi dampak dari kegagalan sistem. Sistem dan layanan yang menggunakan konsep ini harus sepenuhnya otomatis, artinya, tidak ada campur tangan manusia yang diperlukan agar konsep ketersediaan tinggi dapat bekerja dengan baik [9].

Rumus umum untuk mengukur Kehandalan Tinggi (*High Availability*) adalah dengan pendekatan "nine (sembilan)", yang dinyatakan sebagai persentase waktu kerja selama periode tertentu [6]. Rumus tersebut dapat direpresentasikan sebagai berikut:

Availability : $\left(\frac{Total\ Time - Downtime}{Total\ Time}\right) x\ 100\%$

Availability : Persentase waktu di mana sistem atau layanan beroperasi dengan baik.
Total Time : Total waktu dalam periode pengukuran (biasanya dalam jam atau hari).
Downtime : Total waktu di mana sistem atau layanan tidak tersedia atau mengalami gangguan (biasanya dalam jam atau hari).

Sebagai contoh, Sebuah perusahaan menggunakan sistem basis data yang sangat penting untuk operasional sehari-hari mereka. Mereka menerapkan *High Availability* dengan cara mengkonfigurasi replikasi data antara dua atau lebih server basis data. Dengan replikasi ini, setiap perubahan data yang dilakukan pada server utama secara otomatis disalin ke server cadangan. Jika server utama mengalami kegagalan, operasi bisnis dapat segera diarahkan ke server cadangan tanpa kehilangan data atau waktu henti yang signifikan. Meskipun ini bukan *High Availability* cluster tradisional, ini adalah contoh penerapan *High Availability* dalam infrastruktur yang kritis untuk memastikan kontinuitas operasi bisnis.

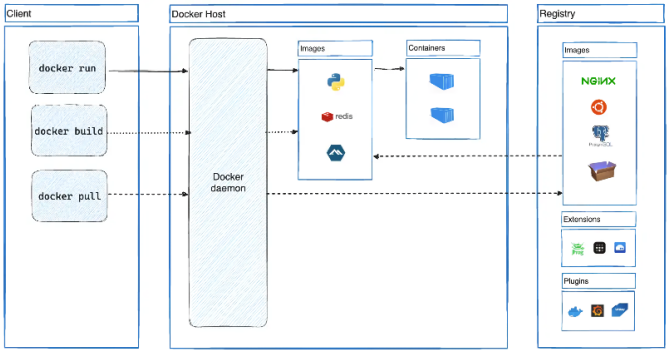
2) Docker

Docker adalah platform terbuka untuk mengembangkan, mengelola, dan menjalankan aplikasi. Docker memungkinkan untuk memisahkan aplikasi dari infrastruktur sehingga dapat mengelola perangkat lunak dengan cepat. Dengan Docker, infrastruktur dapat dikelola dengan cara yang sama seperti mengelola aplikasi. Dengan memanfaatkan metodologi Docker untuk melakukan pengelolaan, pengujian, dan penyebaran kode, dapat mengurangi penundaan secara signifikan antara menulis kode dan menjalankannya di produksi.

Docker menyediakan kemampuan untuk mengemas dan menjalankan aplikasi dalam lingkungan yang terisolasi secara longgar yang disebut kontainer. Isolasi dan keamanannya memungkinkan menjalankan banyak kontainer secara simultan pada suatu hos. Kontainer ringan dan berisi semua yang diperlukan untuk menjalankan aplikasi, sehingga tidak perlu bergantung pada apa yang terinstal di hos. Kontainer dapat dibagikan saat bekerja, memastikan bahwa setiap orang mendapatkan kontainer yang sama yang berfungsi secara konsisten.

Docker mempermudah pengembangan dengan menyediakan lingkungan terstandarisasi melalui kontainer lokal, memungkinkan pengembang untuk mengakses aplikasi dan layanan dengan mudah. Kontainer sangat cocok untuk alur kerja integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD).

Pada gambar 4 Docker menggunakan arsitektur klien-server. Klien Docker berbicara dengan daemon Docker, yang melakukan pekerjaan berat dalam membangun, menjalankan, dan mendistribusikan kontainer Docker. Klien Docker dan daemon Docker dapat berjalan pada sistem yang sama, atau dapat menyambungkan klien Docker ke daemon Docker jarak jauh. Klien Docker dan daemon berkomunikasi menggunakan API REST, melalui soket UNIX atau antarmuka jaringan. Klien Docker lainnya adalah Docker Compose, yang memungkinkan bekerja dengan aplikasi yang terdiri atas sekumpulan kontainer.



Gbr. 4 Arsitektur Docker

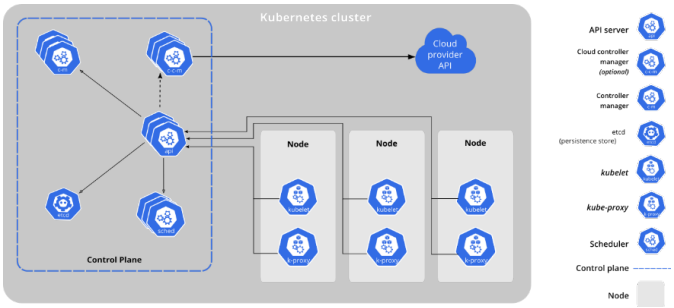
3) Docker Swarm

Docker Swarm adalah alat orkestrasi yang dikembangkan oleh Docker untuk mengelola dan menyusun kontainer Docker dalam lingkungan cluster. Dengan Docker Swarm, Anda dapat mengelola sekelompok host Docker sebagai satu sumber daya terpusat yang disebut sebagai "swarm". Ini memungkinkan Anda untuk mengatur aplikasi yang berjalan dalam kontainer di sejumlah host yang terpisah secara terkoordinasi [7].

4) K8s

K8s adalah singkatan dari Kubernetes. Kubernetes adalah sebuah platform sumber terbuka yang digunakan untuk mengelola aplikasi yang berjalan dalam sebuah lingkungan berbasis container. Platform ini dikembangkan oleh Google dan dirilis ke masyarakat sebagai proyek open-source [8]. Kubernetes menawarkan berbagai fitur yang memungkinkan pengguna untuk secara efisien mengelola, men-deploy, dan men-skala aplikasi containerized.

Pada gambar 5 Klaster Kubernetes terdiri atas sekumpulan mesin pekerja, yang disebut node, yang menjalankan aplikasi dalam kontainer [9]. Setiap klaster memiliki setidaknya satu node pekerja. Node pekerja meng-host Pod yang merupakan komponen beban kerja aplikasi. Control plane mengelola node pekerja dan Pod di dalam kluster. Dalam lingkungan produksi, control plane biasanya berjalan di beberapa komputer dan kluster biasanya menjalankan beberapa node, sehingga memberikan toleransi terhadap kesalahan dan ketersediaan yang tinggi.



Gbr. 5 Komponen Kubernetes

Komponen control plane membuat keputusan terpusat tentang cluster (misalnya, penjadwalan), serta mendeteksi dan merespons peristiwa cluster (misalnya, memulai pod baru saat bidang replika Deployment tidak terpenuhi).

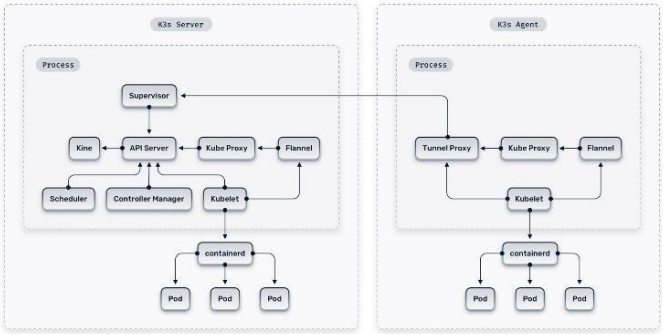
Komponen control plane dapat dijalankan di mesin mana pun dalam klaster. Namun, untuk mempermudah, skrip penyiapan biasanya memulai semua komponen control plane di mesin yang sama, dan tidak menjalankan kontainer pengguna di mesin ini.

5) K3s

K3s adalah distribusi Kubernetes ringan yang dibuat oleh Rancher Labs, dan sepenuhnya disertifikasi oleh Cloud Native Computing Foundation (CNCF). K3s sangat tersedia dan siap produksi.

Secara sederhana, K3s adalah Kubernetes dengan bloat yang dihilangkan dan datastore pendukung yang berbeda. Meskipun demikian, penting untuk dicatat bahwa K3s bukanlah fork, karena K3s tidak mengubah fungsionalitas inti Kubernetes dan tetap dekat dengan Kubernetes stok.

Pada gambar 6 Tidak seperti Kubernetes tradisional, yang menjalankan komponen-komponennya dalam proses yang berbeda, K3s menjalankan control plane, kubelet, dan kube-proxy dalam satu proses Server atau Agen, dengan containerd yang menangani fungsi-fungsi siklus hidup kontainer.



Gbr. 6 Komponen K3s

6) Container

Container adalah unit yang ringan, portabel, dan mandiri yang mengemas aplikasi bersama dengan dependensi, pustaka, dan lingkungan runtime. Container memungkinkan aplikasi berjalan secara konsisten di berbagai lingkungan komputasi, menyederhanakan proses pengembangan, pengujian, dan penerapan. Mereka mengisolasi aplikasi dari sistem yang mendasarinya [1], memastikan bahwa setiap aplikasi berjalan di ruang pengguna khusus.

Container menggunakan kernel sistem operasi host tetapi mempertahankan sistem file mereka sendiri, yang berkontribusi pada efisiensi dan kemudahan penggunaannya. Platform kontainer yang populer termasuk Docker dan containerd, sementara alat orkestrasi seperti Kubernetes membantu mengelola dan menskalakan penerapan container.

7) Sistem Virtualisasi

Virtual system (VSYS) adalah teknologi virtualisasi yang membagi perangkat fisik menjadi beberapa perangkat logis yang independen. Setiap sistem virtual berfungsi sebagai perangkat nyata yang memiliki sumber dayanya sendiri dan menjalankan layanannya sendiri, yang dapat dikonfigurasi dan dikelola secara independen oleh administrator [3]. Ketika manajemen jaringan menjadi semakin kompleks, pengguna memiliki persyaratan yang lebih tinggi pada isolasi layanan, keamanan sistem, dan keandalan. Teknologi VSYS dapat digunakan untuk mengisolasi sumber daya dan layanan secara fisik, menyederhanakan penyebaran dan manajemen jaringan, serta meningkatkan keamanan dan keandalan sistem [5].

Dengan teknologi virtualisasi, administrator sistem publik dapat membuat sistem virtual pada firewall Huawei dan mengalokasikan sumber daya yang diperlukan (termasuk zona keamanan, antarmuka, alamat IP publik, dan sumber daya sesi) ke sistem virtual sehingga dapat bekerja secara independen sebagai firewall fisik.

Jika tidak ada sistem virtual yang dikonfigurasi pada firewall, firewall meneruskan paket berdasarkan kebijakan dan berbagai tabel (seperti tabel sesi, tabel alamat MAC, dan tabel perutean) dari sistem publik. Setelah sistem virtual dikonfigurasi pada firewall, setiap sistem virtual berfungsi sebagai perangkat independen dan memiliki kebijakan dan tabelnya sendiri untuk pemrosesan paket. Dalam hal ini, setelah menerima paket, firewall harus terlebih dahulu menentukan sistem virtual tujuan dari paket tersebut.

8) Kernel

Kernel adalah komponen utama dari sistem operasi yang mengelola operasi komputer dan perangkat keras. Pada dasarnya kernel mengelola operasi memori dan waktu CPU. Kernel merupakan komponen inti dari sebuah sistem operasi. Kernel bertindak sebagai jembatan antara aplikasi dan pemrosesan data yang dilakukan di tingkat perangkat keras menggunakan komunikasi antar-proses dan panggilan sistem.

Kernel dimuat pertama kali ke dalam memori saat sistem operasi dimuat dan tetap berada di dalam memori hingga sistem operasi dimatikan kembali. Kernel bertanggung jawab atas berbagai tugas seperti manajemen disk, manajemen tugas, dan manajemen memori.

9) Server

Sebuah server pada dasarnya adalah sebuah perangkat keras atau komputer yang menjalankan program server untuk menyediakan layanan kepada komputer lainnya melalui jaringan. Program yang berjalan di server tersebut

memungkinkan server untuk menerima permintaan, memprosesnya, dan mengirimkan respons kembali kepada komputer yang memintanya. Jadi, server bisa merujuk pada perangkat keras (komputer) yang menjalankan program server, atau bisa merujuk secara langsung kepada program server itu sendiri dan layanan kepada program komputer lainnya.

III. Hasil dan Pembahasan

A. Implementasi Sistem

1) Konfigurasi Docker Swarm

Pada gambar 7 berikut ini. Menentukan versi file Docker Compose yang digunakan.

```
version: '3.7'
```

Gbr. 7 Versi File Docker Compose

Pada gambar 8 Kode ini mendefinisikan layanan MySQL dalam Docker Compose dengan menggunakan gambar Docker MySQL versi 5.7. Layanan ini diatur untuk berjalan dalam mode replikasi, dengan hanya satu replika yang berjalan pada node dengan peran manager.

```
services:
  db:
    image: mysql:5.7
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.role == manager
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
```

Gbr. 8 Service Mysql

Pada gambar 9 Kode ini mendefinisikan layanan WordPress dalam Docker Compose dengan menggunakan *Image* Docker WordPress versi terbaru. Layanan ini diatur untuk berjalan dalam mode replikasi dengan tiga replika yang berjalan pada node dengan peran manager.

```
wordpress:
  image: wordpress:latest
  deploy:
    mode: replicated
    replicas: 3
    placement:
      constraints:
        - node.role == manager
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
    WORDPRESS_DB_NAME: wordpress
  volumes:
    - wordpress_data:/var/www/html
  ports:
    - "8000:80"
  networks:
    - wordpress
```

Gbr. 9 Service Wordpress

Pada gambar 10 Mendefinisikan dua volume untuk persistent storage: mysql_data dan wordpress_data.

```
volumes:
  mysql_data:
  wordpress_data:
```

Gbr. 10 Volumes

Pada gambar 11 wordpress: Membuat jaringan overlay bernama wordpress yang memungkinkan komunikasi antar layanan di berbagai host dalam sebuah cluster Swarm.

```
networks:
  wordpress:
    driver: overlay
```

Gbr. 11 Network

2) Konfigurasi K3s

Pada gambar 12Kode ini adalah konfigurasi Kubernetes yang terdiri dari Deployment untuk WordPress, PersistentVolumeClaim (PVC) untuk MySQL.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - image: mysql:5.7
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: your_mysql_root_password
        - name: MYSQL_DATABASE
          value: wordpress
        - name: MYSQL_USER
          value: your_mysql_username
        - name: MYSQL_PASSWORD
          value: your_mysql_password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pvc
```

Gbr. 12 Mysql K3s

Pada gambar 13 Deployment WordPress membuat tiga replika pod yang menjalankan container WordPress menggunakan gambar wordpress:latest.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
spec:
  replicas: 3
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
      - name: wordpress
        image: wordpress:latest
        ports:
        - containerPort: 80
        env:
        - name: WORDPRESS_DB_HOST
          value: mysql
        - name: WORDPRESS_DB_USER
          value: your_mysql_username
        - name: WORDPRESS_DB_PASSWORD
          value: your_mysql_password
        - name: WORDPRESS_DB_NAME
          value: wordpress
```

Gbr. 13 Wordpress K3s

B. Pengujian

Untuk melakukan analisis perbandingan kinerja (performance) antara Docker Swarm dan K3s dalam konteks *High Availability* (HA) Cluster, berikut adalah beberapa langkah detail yang bisa diambil untuk melakukan pengujian yang menyeluruh:

1) Pengujian *High Availability*

Penulis melakukan skema dengan melakukan shutdown untuk memicu failover. Setiap kali node master dimatikan, penulis akan mencatat waktu downtime-nya.

Pengujian dilakukan dengan skema failback yang menargetkan interval 14 menit selama 1 hari. Proses ini dimulai dengan secara sistematis mematikan node yang menyimpan pod/container, dilanjutkan dengan langkah-langkah untuk menyalakannya kembali. Setelah itu, siklus ini diulang setiap 14 menit untuk memberi peluang bagi perpindahan pod/container dari node yang dimatikan ke node lain dalam cluster. Downtime, yang merupakan periode di mana pod/container tidak tersedia, dihitung sebagai selisih antara waktu dimatikan dan waktu dinyalakan kembali.

Pada gambar 14 Berdasarkan hasil pengujian, terdapat 103 kali kejadian di mana sebuah node mengalami downtime. Untuk Docker Swarm, dengan rata-rata downtime container sebesar 5 detik, total waktu keseluruhan downtime container dalam satu hari adalah 515 detik. Dengan demikian, Availability Score untuk Docker Swarm sekitar 99.41%.. Sementara itu, untuk K3s, dengan rata-rata downtime container sebesar 350 detik, total waktu keseluruhan downtime container dalam satu hari adalah 36.874 detik. Ini menghasilkan Availability Score sekitar 57.32%. Dari data ini, dapat disimpulkan bahwa Docker Swarm memiliki ketersediaan yang lebih tinggi dibandingkan dengan K3s dalam periode pengujian tersebut.

Docker Swarm

$$\text{Availability} = \left(\frac{\textit{Total Time} - \textit{Downtime}}{\textit{Total Time}}\right) x\ 100\%$$

$$\text{Availability} = \left(\frac{86.400 - 515}{86.400}\right) x\ 100\%$$

$$\text{Availability} = \left(\frac{85.885}{86.400}\right) x\ 100\%$$

$$\text{Availability} = 99.41\%$$

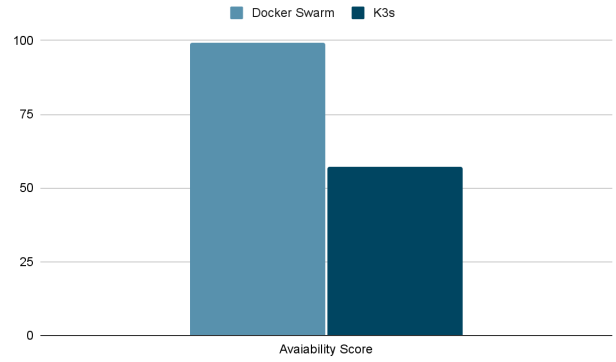
K3s

$$\text{Availability} = \left(\frac{\textit{Total Time} - \textit{Downtime}}{\textit{Total Time}}\right) x\ 100\%$$

$$\text{Availability} = \left(\frac{86.400 - 36.874}{86.400}\right) x\ 100\%$$

$$\text{Availability} = \left(\frac{49.526}{86.400}\right) x\ 100\%$$

$$\text{Availability} = 57.32\%$$



Gbr. 14 Availability Score

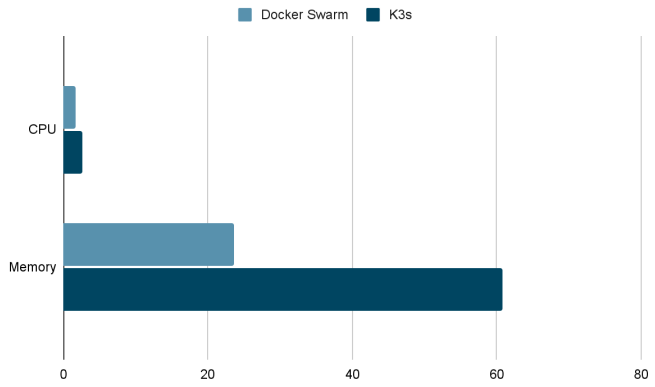
Penelitian menunjukkan bahwa Docker Swarm memiliki skor ketersediaan yang lebih tinggi (99.41%) dibandingkan dengan K3s (57.32%). Hal ini mungkin disebabkan oleh beberapa faktor yang memengaruhi kinerja *High Availability* kedua platform ini. Salah satunya adalah mekanisme deteksi kegagalan. Docker Swarm memiliki mekanisme deteksi kegagalan yang lebih cepat dan handal dibandingkan dengan K3s. Perbedaan ini dapat berasal dari implementasi deteksi kegagalan atau efisiensi algoritma deteksi yang digunakan oleh masing-masing platform. Kemudian, respons dan pemulihan terhadap kegagalan juga menjadi faktor penting. Docker Swarm lebih responsif dalam merespons dan melakukan pemulihan terhadap kegagalan dengan cepat. Ini bisa mencakup kemampuan Docker Swarm untuk secara otomatis memulai ulang kontainer yang gagal atau memindahkan beban kerja ke node lain dengan efisien

2) Pengujian *Performance*

Mengukur dan membandingkan pengMengukur dan membandingkan penggunaan CPU dan RAM pada cluster Docker Swarm dan K3s dalam kondisi idle (tanpa beban aplikasi yang berjalan) selama 60 menit. Pengujian ini bertujuan untuk memahami baseline resource utilization kedua sistem dalam kondisi tanpa beban dan pemulihan selama proses HA.n CPU dan RAM pada cluster Docker Swarm dan K3s dalam kondisi idle (tanpa beban aplikasi yang berjalan) selama 60 menit. Pengujian ini bertujuan untuk memahami baseline resource utilization kedua sistem dalam kondisi tanpa beban dan pemulihan selama proses HA.

Pada gambar 15 Pengujian idle adalah pengujian yang dilakukan pada sistem atau perangkat lunak dalam kondisi tanpa beban atau aktivitas yang signifikan. Tujuannya adalah untuk mengukur konsumsi sumber daya dasar seperti CPU, dan memori (RAM) ketika sistem berada dalam keadaan minimal operasi atau tidak melakukan tugas-tugas yang berat.

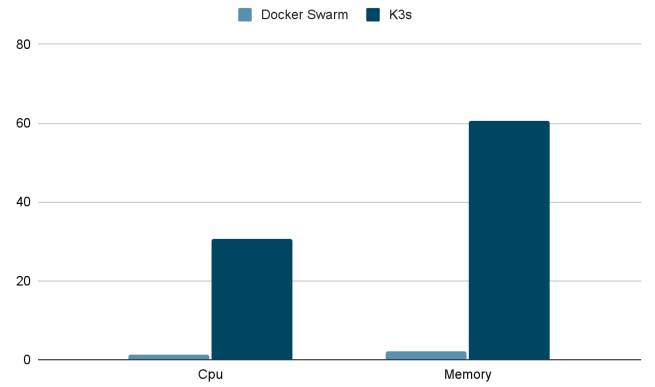
Gbr. 4 Contoh grafik garis



Gbr. 15 Pengujian performance CPU dan Memory

Dari hasil tersebut, terlihat bahwa pada pengujian ini, Docker Swarm menunjukkan penggunaan CPU yang lebih rendah dengan 1.73% dibandingkan dengan K3s yang mencapai 2.6%. Namun, terdapat perbedaan signifikan pada penggunaan RAM, dimana Docker Swarm hanya menggunakan 23.565% sementara K3s memerlukan 60.8%. Perbandingan ini menggambarkan efisiensi penggunaan sumber daya antara kedua platform tersebut.

Pada gambar 16 Dari hasil tersebut, terlihat bahwa pada pengujian ini, Docker Swarm menunjukkan penggunaan CPU yang lebih rendah dengan 1.73% dibandingkan dengan K3s yang mencapai 2.6%. Namun, terdapat perbedaan signifikan pada penggunaan RAM, dimana Docker Swarm hanya menggunakan 23.565% sementara K3s memerlukan 60.8%. Perbandingan ini menggambarkan efisiensi penggunaan sumber daya antara kedua platform tersebut.



Gbr. 16 Pengujian performance CPU dan Memory

Dari hasil pengujian ini, terlihat bahwa Docker Swarm menunjukkan penggunaan CPU yang lebih rendah sebesar 1.184%, sedangkan K3s memiliki penggunaan CPU sebesar 2%. Namun, terdapat perbedaan yang signifikan pada penggunaan RAM, dimana Docker Swarm hanya menggunakan 30.6% sementara K3s mencapai 60.7%. Hasil ini memberikan gambaran tentang efisiensi penggunaan sumber daya saat menggunakan kedua platform dalam skenario *High Availability*.

IV. KESIMPULAN

A. Kinerja *High Availability*

Dari hasil analisis yang dilakukan, dapat disimpulkan bahwa Docker Swarm dan K3s memiliki karakteristik yang berbeda dalam konteks *High Availability* (HA) Cluster. Ketersediaan Layanan (Availability) menjadi salah satu perbandingan utama antara kedua platform, dengan Docker Swarm menunjukkan tingkat ketersediaan yang lebih tinggi dibandingkan dengan K3s. Hal ini terlihat dari rata-rata downtime yang lebih rendah pada Docker Swarm, yang menghasilkan skor ketersediaan yang lebih tinggi.

B. Faktor yang mempengaruhi kinerja

Faktor-faktor teknis yang mempengaruhi kinerja *High Availability* (HA) Cluster pada Docker Swarm dan K3s sangatlah vital. Dalam hal arsitektur cluster, Docker Swarm mengadopsi model manager-worker yang memanfaatkan node manager untuk mengatur cluster dan node worker untuk menjalankan container, sementara K3s menyajikan pendekatan yang lebih ringan dengan arsitektur master-agent. Perbedaan ini mempengaruhi kemampuan kluster untuk mendeteksi dan mengatasi kegagalan dengan efisien. Selain itu, manajemen sumber daya juga menjadi faktor kunci, dengan Docker Swarm menggunakan algoritma spread dan binpack untuk mendistribusikan pod/container secara merata, sedangkan K3s memanfaatkan algoritma default Kubernetes yang lebih fokus pada efisiensi sumber daya. Algoritma scheduling yang berbeda juga memberikan pengaruh pada kinerja keseluruhan kluster, dengan Docker Swarm menawarkan fleksibilitas lebih dalam memilih algoritma scheduling.

C. Performa Cpu dan Memory

Selanjutnya, dalam hal penggunaan sumber daya, Docker Swarm juga menunjukkan keunggulan. Ketika dalam kondisi idle, Docker Swarm menggunakan CPU dan RAM dengan lebih efisien dibandingkan dengan K3s. Penggunaan CPU dan RAM Docker Swarm lebih rendah, menunjukkan kemampuan platform ini untuk mengoptimalkan sumber daya bahkan saat tidak ada beban aplikasi yang berjalan. Selama proses failover pun, Docker Swarm tetap menunjukkan efisiensi penggunaan sumber daya yang lebih baik dalam hal penggunaan CPU, meskipun terdapat peningkatan yang cukup besar pada penggunaan RAM, namun tetap lebih rendah daripada K3s. Hal ini menunjukkan bahwa Docker Swarm mungkin lebih efisien dalam mengelola sumber daya sistem saat berada dalam kondisi normal maupun saat terjadi kegagalan pada cluster

V. SARAN

Sebelum mengambil keputusan final dalam memilih platform untuk implementasi HA cluster, ada beberapa saran [10] yang perlu dipertimbangkan. Pertama, adalah pentingnya mempertimbangkan kebutuhan spesifik proyek. Jika kecepatan failover dan efisiensi penggunaan sumber daya adalah prioritas utama, Docker Swarm bisa menjadi pilihan yang lebih baik. Namun, jika lebih memperhatikan kemudahan penggunaan dan penyebaran yang ringan, K3s bisa menjadi opsi yang menarik.

Selanjutnya, lakukan evaluasi lebih lanjut terhadap aspek-aspek lain dari kedua platform, seperti skalabilitas, dan fitur tambahan. Meskipun Docker Swarm menunjukkan kinerja yang lebih baik dalam uji coba yang telah dilakukan, namun fitur-fitur spesifik dari K3s mungkin lebih sesuai dengan kebutuhan proyek yang lebih besar.

Selain itu, perhatikan juga ketersediaan dan dukungan komunitas untuk kedua platform. Dukungan komunitas yang kuat dapat membantu dalam mengatasi masalah dan menemukan solusi ketika mengimplementasikan dan mengelola HA cluster.

Dengan mempertimbangkan secara cermat faktor-faktor di atas, dapat membuat keputusan yang lebih terinformasi dan tepat dalam memilih antara Docker Swarm dan K3s untuk implementasi HA cluster sesuai dengan kebutuhan dan prioritas proyek.

REFERENSI

[1] Al Jawarneh, I. (2019). ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE.

[2] Bhardwaj, A., & Krishna, C. R. (2021). Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey. Arabian Journal for Science and Engineering, 46(9), 8585–8601. https://doi.org/10.1007/s13369-021-05553-3

[3] Cvetkovski Assoc, A. (2021). OPERATING SYSTEM VIRTUALIZATION IN THE EDUCATION OF COMPUTER SCIENCE STUDENTS.

[4] Ghatrehsamani, D., Denninnart, C., Bacik, J., & Amini Salehi, M. (2020, August 17). The Art of CPU-Pinning: Evaluating and Improving the Performance of Virtualization and Containerization Platforms. ACM International Conference Proceeding Series. https://doi.org/10.1145/3404397.3404442

[5] Hsu, C. H. (2020). Internet of Vehicles. Technologies and Services Toward Smart Cities (C.-H. Hsu, S. Kallel, K.-C. Lan, & Z. Zheng, Eds.; Vol. 11894). Springer International Publishing. https://doi.org/10.1007/978-3-030-38651-1

[6] Khatami, A. A., Purwanto, Y., & Ruriawan, M. F. (2020). High availability storage server with kubernetes. 2020 International Conference on Information Technology Systems and Innovation, ICITSI 2020 - Proceedings, 74–78. https://doi.org/10.1109/ICITSI50517.2020.9264928

[7] Naik, N. (2021, April 15). Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm. 15th Annual IEEE International Systems Conference, SysCon 2021 - Proceedings. https://doi.org/10.1109/SysCon48628.2021.9447123

[8] Senjab, K., Abbas, S., Ahmed, N., & Khan, A. ur R. (2023). A survey of Kubernetes scheduling algorithms. In Journal of Cloud Computing (Vol. 12, Issue 1). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1186/s13677-023-00471-1

[9] Šimon, M., Huraj, L., & Búčik, N. (2023). A Comparative Analysis of High Availability for Linux Container Infrastructures. Future Internet, 15(8). https://doi.org/10.3390/fi15080253