# Perbandingan Platform Orkestrasi Kontainer pada Aplikasi Real-Time di Lingkungan Cloud Pribadi

Yunus Dhanzky Handitra[1], I Made Suartana[2]

[1,2] Program Studi Teknik Informatika/Teknik Informatika, Universitas Negeri Surabaya
[1]yunus.22081@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

***Abstrak*— Dalam topik komputasi modern yang terus berkembang, kebutuhan akan pemrosesan data yang cepat, efisien, dan fleksibel telah menjadi krusial, terutama pada aplikasi real-time seperti pemantuan Internet of things (IoT). Aplikasi real-time tersebut bergantung pada kemampuan sistem untuk merespon secara instan dan mengolah data dalam jumlah yang besar dengan latensi yang minimal. Fenomena ini tidak hanya membutuhkan infrastruktur yang kuat, tetapi juga adaptif dan responsif terhadap fluktuasi beban kerja. Salah satu pendekatan yang dilakukan adalah dengan menggunakan orkestrasi kontainer, yang memungkinkan pengelolaan aplikasi secara terdistribusi dengan kemampuan autoscaling. Docker Swarm dan K3s merupakan dua platform orkestrasi yang menawarkan pendekatan dalam skalabilitas, pengelolaan resource, dan kompleksitas operasional. Penelitian ini bertujuan untuk membandingkan performa Docker Swarm dan K3s pada lingkungan private cloud menggunaan arsitektur pemrosesan pesan secara real-time. Dimana fokus pengujian yang dilakukan ada pada latensi, throughput, penggunaan cpu, penggunaan memori, skalabilitas, dan fault tolerance. Hasil pengujian yang dilakukan dengan menggunakan beban sebesar 100, 500, dan 1000 pesan/detik, termasuk skenario kegagalan node, menunjukkan bahwa Docker Swarm dan K3s mampu menangani workload real-time dengan baik. Namun, K3s menunjukkan kestabilan latensi dan realibilitas yang lebih baik, sedangkan Docker Swarm memiliki penggunaan cpu yang lebih baik dan pengelolaan resource yang lebih ringan. Berdasarkan hasil pengujian dapat disimpulkan bahwa, K3s lebih sesuai untuk aplikasi yang membutuhkan kestabilan performa dan latensi yang lebih rendah, sedangkan Docker Swarm dapat menjadi alternatif yang baik untuk sistem yang mengutamakan efisiensi cpu dan orkestrasi ringan.**

***Kata Kunci*— Docker Swarm, K3s, Private Cloud, Orkestrasi Kontainer, Aplikasi Real-Time.**

## I. PENDAHULUAN

Perkembangan dalam komputasi awam bergerak cukup cepat, sehingga kebutuhan akan pemrosesan data yang cepat dan efisien dan fleksibel menjadi hal yang krusial terutama pada aplikasi real-time. Sektor-sektor seperti pemantuan Internet of Things (IoT), analisis data streaming, sistem peringatan dini, dan aplikasi interaktif lainnya sangat bergantung dengan kemampuan sistem utnuk mampu merespon sistem secara instan dan mengolah data dengan volume yang besar dengan latensi yang rendah. Selain itu, karakteristik data sensor IoT yang masif dan datang secara kontinu menuntut efisiensi yang tinggi pada arsitektur komputasi di tingkat edge guna menekan beban latensi jaringan [1]. Fenomena ini menuntut infrastruktur yang tidak hanya kuat, tetapi juga adaptif dan responsif terhadap fluktuasi beban kerja

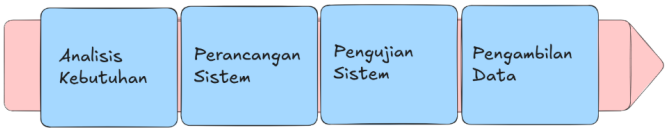
Untuk memenuhi tuntutan tersebut, platform orkestrasi kontainer telah muncul sebagai solusi fundamental. Teknologi ini memungkinkan otomatisasi penuh dalam pengelolaan siklus hidup kontainer, meliputi penjadwalan, penempatan workload, penyeimbangan beban (load balancing), pemulihan dari kegagalan (self-healing), dan yang terpenting, scaling aplikasi sesuai kebutuhan. Berbagai platform sudah dikembangkan untuk tujuan ini, dengan Docker Swarm (solusi orkestrasi bawaan docker) dan K3s (distribusi Kubernetes yang ringan) menjadi pilihan yang populer yang menawarkan keseimbangan antara fungsionalitas dan kemudahan operasional. Masing-masing platform ini menawarkan pendekatan arsitektur dan kapabilitas yang unik, sehingga performanya dapat bervariasi secara signifikan, terutama dalam konteks aplikasi real-time. Perbdaan mendasar ini terlihat pada overhead penggunaan resource Kubernetes yang berbeda dibandingkan dengan kesederhanaan Docker Swarm saat menangani replikasi layanan [2].

Dalam konteks lingkungan Private Cloud, yaitu infrastruktur yang dikelola secara internal oleh organisasi atau institusi [3] (dalam kasus ini, laboratorium atau server lokal), pemilihan platfom orkestrasi menjadi lebih krusial. Private cloud menawarkan kontrol penuh dan potensi latensi lebih rendah dibanding dengan public cloud, tetapi juga menuntut keahlian dalam manajemen infrastruktur. Terlebih lagi, efisiensi virtualisasi berbasis kontainer sangat dipengaruhi oleh isolasi resource dan mekanisme komunikasi jaringan internal host, yang menjadi penentu utama dalam menjaga stabilitas latensi aplikasi [4]. Oleh karena itu, perbandingan mendalam antara platform orkestrasi ringan ini dalam skenario private cloud untuk aplikasi real-time, khususnya dalam hal latensi end-to-end, kemampuan skalabilitas dinamis, dan efisiensi pemanfaatan sumber daya, menjadi sangat relevan dan mendesak.

Penelitian ini bertujuan untuk mengisi kesenjangan tersebut dengan melakukan perbandingan performa dari dua platform orkestrasi kontainer, yaitu Docker Swarm dan K3s. Fokus utama dari perbandingan performa Docker Swarm dan K3s dalam lingkungan private cloud adalah latensi, skalabilitas, dan efisiensi sumber daya saat menangani beban kerja aplikasi real-time yang disimulasikan dari aliran data sensor IoT.

## II. METODOLOGI PENILITIAN

Penelitian ini menggunakan pendekatan dengan metode kuantitatif eksperimental, di mana penelitian ini akan membandingkan dua platform yaitu Docker Swarm dan K3s, yang memiliki cara dan pengembangan orkestrasi dari suatu kontainer yang berbeda di setiap platformnya.




Gbr. 1 Alur Metode Penelitian

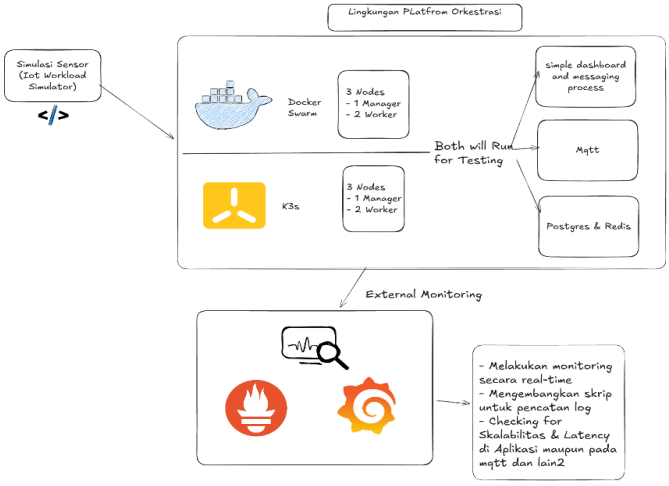
Dari Gbr. 1 terlihat bahwa alur penelitian yang dilakukan dengan analisis kebutuhan, melakukan perancangan sistem baik dari virtualisasi, pengembangan aplikasi, dan platform orkestrasi, pengujian sistem yang sudah dirancang, dan terakhir pengambilan dan analisis data dari pengujian yang dilakukan sebelumnya.

### *A. Analisis Kebutuhan*

Analisis ini dilakukan untuk mengidentifikasi kebutuhan tertentu agar penilitian yang dilakukan dengan baik. Pada Gbr. **1** diperlukan rancangan sistem yang dilakukan dalam penilitian maka hasil analisis kebutuhan yang dilakukan diperlukan dua kebutuhan yaitu kebutuhan perangkat keras yang digunakan dengan spesifikasi kurang lebih sebagai berikut, Intel Xeon E-2236 dengan RAM 32 GB dan SSD 500Gb. Selain itu, dibutuhkan juga kebutuhan perangkat lunak yang dimana kebutuhan sebagai berikut: OS Ubuntu Server 24.04 LTS, Canonical LXD untuk Virtualisasi, Platform Orkestrasi berupa Docker Swarm dan K3s, aplikasi uji berupa simulator IoT, MQTT Broker, dan layanan pemrosesan pesan, dan yang terakhir monitoring external dengan prometheus dan grafana.

### *B. Rancangan Penelittian*

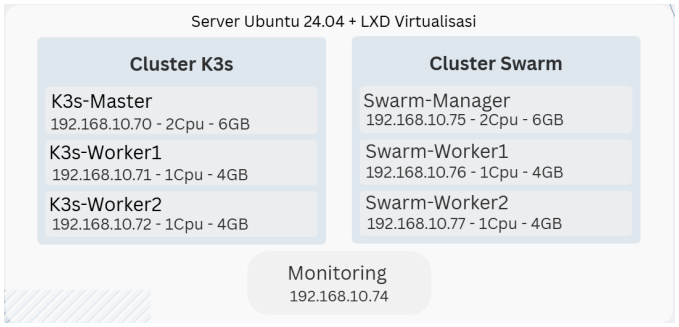
Penelitian ini akan mengikuti rancangan eksperimen sebagai berikut:




Gbr. 2 Rancangan Workflow Sistem

Pada Gbr. 2 diatas adalah rancangan bagaimana ruang lingkup penelitian dibuat.

1) *Virtual Machine Setup:* Pada tahap awal perancangan sistem ini dimulai dengan mempersiapkan Canonical LXD terlebih pada server. Canonical LXD ini digunakan untuk membuat virtual machine dan untuk konfigurasi network pada setiap VM yang akan digunakan pada kedua platform cluster dan monitoring node.




Gbr. 3 Topologi Virtual Machine

Pada Gbr. 3 Topologi Virtual Machine terlihat bahwa virtual machine yang digunakan nantinya akan berjumlah 7, dengan tiap cluster pada Docker Swarm maupun K3s akan memiliki 3 virtual machine dan 1 virtual machine yang digunakan untuk monitoring external. Semua dari virtual machine itu berasal dari satu server yang ditambahkan dengan Canonical LXD.

2) *Instalasi Docker Swarm dan K3s:* Pada tahap ini adalah kelanjutan dari setup setelah selesai mempersiapkan virtual machine untuk setiap cluster dan monitoring. Instalasi dimulai dari instalasi K3s dengan mendownload dan instalasi skrip yang disediakan langsung oleh K3s *curl -sfL https://get.k3s.io | sh -s - server --write-kubeconfig-mode 644,* untuk instalasi pada node master. Selanjutny adalah instalasi pada node worker dengan menggunakan command berikut *curl -sfL https://get.k3s.io | K3S URL=https://192.168.10.70:6443 K3S_TOKEN=Node-token-from-master sh -,* command diatas untuk instalasi k3s pada node worker dan menghubungkan ke node master.

Selanjutnya adalah melakukan instalasi pada Docker Swarm dengan menggunakan command berikut *docker swarm init --advertise-addr 192.168.10.75,* yang digunakan untuk instalasi pada node master. Selanjutnya adalah instalasi pada node worker dengan menggunakan command berikut *docker swarm join --token <token> 192.168.10.75:2377,* yang dimana untuk menginisiasi swarm dengan melakukan join pada node master.

*3) Pengembangan Aplikasi:* aplikasi yang dikembangkan terdiri dari tiga. Ketiga aplikasi yang dikembangkan ini nantinya digunakan untuk melaksanakan pengujian terhadap kedua platform orkestrasi nantinya. Aplikasi pertama yang dikembangkan adalah IoT generator yang mana digunakan untuk melakukan pengiriman pesan ke platform orkestrasi yang dapat disesuaikan pengiriman pesan per detiknya. Pengembangan aplikasi selanjutnya adalah aplikasi pemrosesan pesan, aplikasi ini digunakan untuk memproses pesan yang diterima oleh broker yang dimana akan direkam telemetrik pemrosesan pesan yang diterima. Aplikasi terakhir yang dikembangkan adalah aplikasi dashboard sederhana yang digunakan untuk memperlihatkan pesan yang diterima secara langsung oleh aplikasi pemrosesan pesan.

4) *Deployment Aplikasi Pada Cluster:* Setelah selesai pada pengembangan, selanjutnya adalah deployment aplikasi yang

akan digunakan pada cluster dengan menggunakan file deklaratif YAML dengan penempatan yang disesuaikan.

Stateful services yang dimana memiliki komponen yang sensitif akan dideploy statis pada Manager Node, komponen aplikasi stateful seperti MQTT Broker, PostgreSQL, dan redis dimana dibuat statis untuk menjaga persistensi data.

Stateless services yang dimana akan dideploy secara dinamis pada master node dan worker node untuk memancing mekanisme load balancing internal dari kedua platform orkestrasi. Skalabilitas replika pada kontainer atau pod yang akan meningkat ketika beban naik. Komponen aplikasi yang stateless adalah aplikasi pemrosesan pesan dan IoT dashboard sederhana.

Telemetri agent merupakan komponen aplikasi yang akan digunakan untuk memaparkan telemetrik hardware dan kontainer dari pada cluster platform orkestrasi ditempatkan ke monitoring node diluar cluster. Komponen aplikasi telemetri agent yang dideploy adalah Node-Exporter dan Cadvisor yang dideploy pada tiap node yang ada di cluster.

### C. *Pengujian Sistem*

Perngujian sistem yang akan dilakukan berfokus pada pengujian beban, pengujian skalabilitas, dan pengujian ketahanan.

1) *Metrik Pengujian:* Metrik pengujian ini nantinya akan berfokus pada performa pengiriman data dan juga pada penggunaan resource pada kedua platform. Metrik yang akan diuji nantinya adalah Persentil, Throughput, CPU, dan Memory.

Persentil adalah angka yang menunjukkan posisi suatu titik data dalam kumpulan data numerik dengan menunjukkan persentase dari kumpulan data tersebut yang memiliki nilai lebih kecil [5]. Persentil ini nantinya akan digunakan untuk menguji performa dari pemrosesan data di dalam platform orkestrasi yaitu pada latensinya. Persentil yang digunakan nantinya ada P50, P95, dan P99, dalam salah satu P95 konteksnya adalah bagaimana waktu yang dibutuhkan oleh 95 persen data yang diproses.

Throughput dalam framework aplikasi end-to-end yang dijelaskan dalam matematika sebagai rate bits per detik atau pesan per detik pada data yang terkirim daripada pengirim dengan penerima data dalam waktu tertentu [6]. Dalam konteks pengujian yang dilakukan throughput ini digunakan untuk melihat bagaimana kemampuan dari Docker Swarm dan K3s dalam memproses data IoT secara real-time pada berbagai tingkat pengujian yang dilakukan.

CPU adalah chip perangkat keras yang berguna sebagai otak dari suatu infrastruktur komputer yang di mana berfungsi untuk memanajemen performa keseluruhan infrastruktur komputer [7]. Pada konteks metrik pengujian ini dimana penggunaan CPU diperlihatkan ketika melakukan pengujian pada platform orkestrasi baik pada level hardware dan kontainer.

Memori atau yang biasa dikenal sebagai RAM (Random Access Memory) merupakan tempat untuk menyimpan data dan memproses data dari aplikasi sementara yang dimana akan menghilang ketika komputer atau server dimatikan [8]. Pada konteks pengujian ini akan dilihat efisiensi pada penggunaan memori oleh platform orkestrasi baik pada level hardware maupun kontainer.

*2) Pengujian Beban dan Skalabilitas:* Pada skenario pengujian beban ini dirancang untuk menguji kemampuan load balancing dan mekanisme scaling pada Docker Swarm dan K3s. Pengujian ini dilakukan secara bertahap dengan beban yang meningkat dari IoT generator dengan tiga tingkatan, yaitu 100 pesan per detik, 500 pesan per detik, dan 1000 pesan per detik.

*3) Pengujian Ketahanan dan Skalabilitas:* Pada skenarion pengujian ketahanan ini dirancang untuk melihat bagaimana kemampuan self-healing dan kelangsungan layanan kedua platform saat salah satu node tiba-tiba mati. Pengujian ini dilakukan nantinya pada saat cluster menerima beban pengiriman 1000 pesan per detik, salah satu node worker akan dimatikan secara paksa.

### D. *Pengumpulan Data*

Proses pengumpulan data telemetrik dan performa kluster dilakukan secara otomatis dan tersentralisasi secara real-time dengan menggunakan ekosistem monitoring open-source. Telemetrik scraping dilakukan dengan menggunakan Prometheus dari hasil log aplikasi dan sistem, sistem scraping ini mengambil dari endpoint dari Node-Exporter dan Cadvisor di setiap cluster pada Docker Swarm dan K3s. Seluruh data metrik yang sudah terkumpul dalam basis waktu disimpan ke dalam database internal prometheus yang kemudian dihubungkan ke Grafana melalui PromQL Data Source. Grafana ini digunakan untuk merangkum, menghitung, dan mengvisualisasikan seluruh aktivitas performa infrastruktur ke dalam dashboard analisis selama skenario pengujian berlangsung.

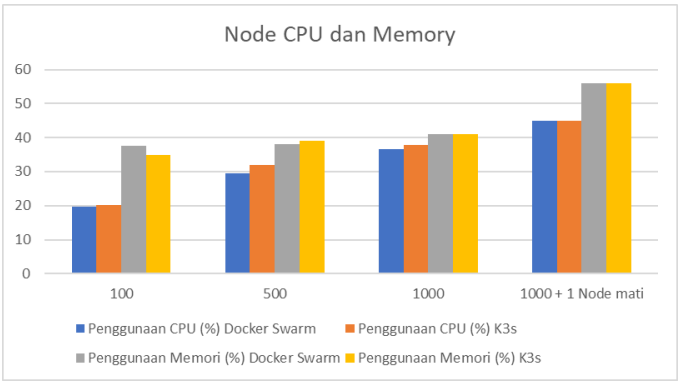
### E. *Analisis Data*

Proses analisis data yang sudah terkumpul sebelumnya akan dianalisis untuk menarik kesimpulan. Analisis data dilakukan dengan cara menghitung rata-rata dan dibandingkan penggunaan CPU dan Memory footprint pada kedua platform orkestrator pada tiap pengujian yang dilakukan. Selain itu, analisis data juga dilakukan pada persentil yang dimana digunakan untuk menghitung latensi. Persentil latensi yang dianalisis adalah P50 yang digunakan untuk latensi umum, P95 dan P99 yang digunakan untuk latensi terburuk untuk mengukur tingkat konsistensi, stabilitas, dan determinisme jaringan pada kedua platform dalam memproses data IoT secara real-time.

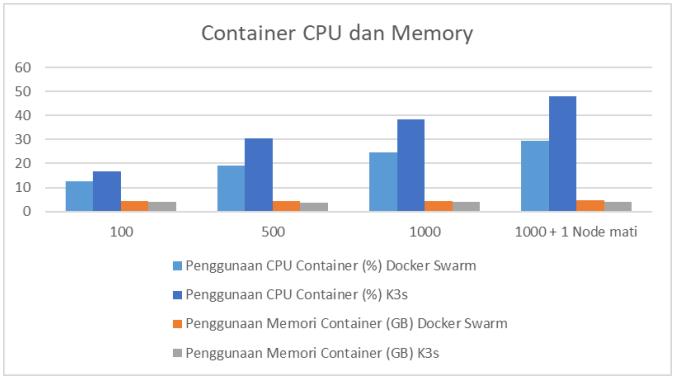
## III. HASIL DAN PEMBAHASAN

### A. *Performa*

1) *Penggunaan Resource:* penggunaan resource ini nantinya akan terbagi pada penggunan CPU, Memori, dan network. Pada hasil penggunaan CPU dan Memori di level hardware maupun kontainer kedua platform memiliki tren yang jelas dan hampir sama dari tiap pengujian yang dilakukan.




Gbr. 4 Node CPU dan Memori




Gbr. 5 Kontainer CPU dan Memori

Pada Gbr. 4 dan Gbr. 5 terlihat jelas bahwa penggunaan CPU pada level node dan container mengalami tren kenaikan yang sama pada setiap pengujian yang dilakukan pada Docker Swarm dan K3s. Penggunaan CPU pada level node di kedua platform memiliki nilai yang hampir sama dengan perbedaan sekitar 2% atau lebih pada pengujian 500 pesan per detik dan 1000 pesan per detik. Sedangkan pada penggunaan CPU di level kontainer K3s memiliki nilai hampir 10% lebih tinggi daripada Docker Swarm, seperti pada pengujian 500 pesan per detik Docker Swarm memiliki penggunaan CPU sebesar 19.6% sedangkan K3s 30.6%.

Pengunaan memori pada level node dan level kontainer juga mengalami kenaikan disetiap pengujian yang dilakukan. Pada level node penggunaan memori Docker Swarm dan K3s juga hampir sama, dimana kedua platform menggunakan memori sekitar 35% hingga 41% pada saat beban kerja normal dan melonjak ke nilai yang sama 54% saat pengujian beban. Sedangkan pada level kontainer, K3s memiliki penggunaan yang lebih optimal yang dimana sekitar 3,8 GB hingga 4,0 GB selama pengujian. Sebaliknya, Docker Swarm menunjukkan tren yang lebih tinggi, sekitar 4,2 GB dan naik hingga 4,6 GB selama pengujian. Untuk lebih detail penggunaan kedua ada pada TABEL I dan TABEL II.

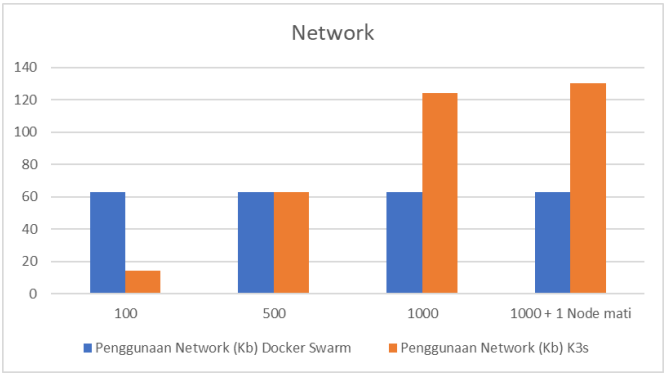
TABEL I
Penggunaan CPU dan Memori pada Node

| Pengujian ( Pesan/detik) | Penggunaan CPU (%) | | Penggunaan Memori (%) | |
|---|---|---|---|---|
| | Docker Swarm | K3s | Docker Swarm | K3s |
| 100 | 19.8 | 20.1 | 37.5 | 35 |
| 500 | 29.5 | 31.9 | 38 | 39 |
| 1000 | 36.6 | 37.8 | 41 | 41 |
| 1000 + 1 Node mati | 45 | 45 | 56 | 56 |

TABEL II
Penggunaan CPU dan Memori pada Kontainer

| Pengujian ( Pesan/detik) | Penggunaan CPU Container (%) | | Penggunaan Memori Container (GB) | |
|---|---|---|---|---|
| | Docker Swarm | K3s | Docker Swarm | K3s |
| 100 | 12.6 | 16.6 | 4.2 | 4.1 |
| 500 | 19 | 30.6 | 4.2 | 3.8 |
| 1000 | 24.5 | 38.5 | 4.5 | 3.9 |
| 1000 + 1 Node mati | 29.5 | 48 | 4.6 | 4 |

Selanjutnya adalah perbandingan penggunaan network sepanjang pengujian yang dilakukan pada Docker Swarm dan K3s.




Gbr. 6 Network

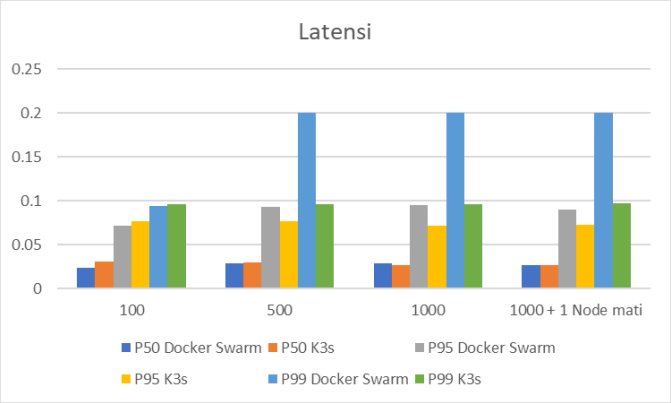
TABEL III
Penggunaan Network

| Pengujian ( Pesan/detik) | Penggunaan Network (Kb) | |
|---|---|---|
| | Docker Swarm | K3s |
| 100 | 63 | 14.4 |
| 500 | 63 | 63 |
| 1000 | 63 | 124 |
| 1000 + 1 Node mati | 63 | 130 |

Pada TABEL III dan Gbr. 6 terlihat bahwa penggunaan network yang dialami oleh K3s mengalami peningkatan

disetiap pengujian yang dilakukan. Penggunaan network dari K3s dimulai dari 14.4 Kb dan meningkat pada penggunaan 1000 pesan/detik hingga 124 Kb, hingga pada puncaknya saat pengujian ketahanan hingga 130 Kb. Sebaliknya, Docker Swarm yang memiliki nilai network yang pasif pada 63 Kb, hal ini terjadi dikarenakan tidak adanya telemetri yang terkirim dari Node-Exporter karena bentuk network yang lebih simpel dibanding K3s.

2) *Latensi dan Jitter:* Pada latensi dan jitter akan dibandingkan bagaimana performa Docker Swarm dan K3s pada end-to-end latensi dari pengiriman hingga aplikasi pemrosesan data.




Gbr. 7 Latensi Docker Swarm dan K3s

TABEL IV
Latensi Docker Swarm dan K3s

| Pengujian (Pesan/detik) | Platform | P50 (s) | P95 (s) | P99 (s) |
|---|---|---|---|---|
| 100 | K3s | 0.0304 | 0.0771 | 0.0964 |
| 100 | Swarm | 0.0232 | 0.0715 | 0.0943 |
| 500 | K3s | 0.0302 | 0.0771 | 0.0965 |
| 500 | Swarm | 0.0288 | 0.0934 | 30 |
| 1000 | K3s | 0.0262 | 0.072 | 0.0964 |
| 1000 | Swarm | 0.0286 | 0.0948 | 30 |
| 1000 + 1 Node mati | K3s | 0.0267 | 0.073 | 0.0967 |
| 1000 + 1 Node mati | Swarm | 0.0265 | 0.0897 | 30 |

Pada Gbr. 7 dan TABEL IV diperlihatkan perbedaan yang cukup signifikan antara Docker Swarm dan K3s, terutama pada saat pengujian dengan beban 500 dan 1000 pesan/detik. Pada saat pengujian dengan beban 100 pesan/detik, Docker Swarm memiliki nilai latensi P50 lebih rendah 23.2 ms dibanding K3s 30.4 ms. Sebaliknya, ketika pengujian beban meningkat pada 500 dan 1000 pesan/detik, K3s memilik latensi ekstrem P99 pada 96.4 ms, diman Docker Swarm mengalami penurunan performa yang sangat signifikan. Walaupun pada Docker Swarm latensi pada P50 dan P95 masih mendekati K3s, tetapi latensi ekstrem P99 meningkat hingga batas histogram 30 detik hingga pengujian ketahanan. Untuk memperlihat perbedaan dengan jelas, latensi ekstrem P99 diberi ambang batas 0.2 detik pada Gbr. 7. Pada kedua platform memiliki jitter yang tidak terlalu tinggi pada latensi P50 dan P95, yang berbeda hanya pada latensi P99 Docker Swarm yang melonjak tinggi dari pengujian 100 pesan/detik hingga pengujian ketahanan.

3) *Success Rate dan Throughput:* Pada tahap ini adalah perbandingan dari Docker Swarm dan K3s dalam throughput dan keberhasilan pemrosesan data ketika beban yang bertambah.

TABEL V
Throughput Docker Swarm dan K3s

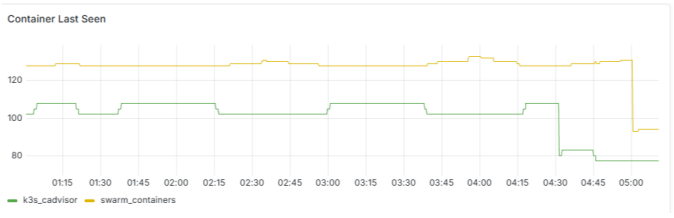
| Pengujian (Pesan/detik) | Throughput Pesan (Pesan/detik) | |
|---|---|---|
| | Docker Swarm | K3s |
| 100 | 100 | 100 |
| 500 | 500 | 500 |
| 1000 | 1000 | 1000 |
| 1000 + 1 Node mati | 1000 | 1000 |

TABEL VI
Success Rate Docker Swarm dan K3s

| Platform | Total Pesan Terkirim | Pesan Berhasil diproses | Pesan gagal diproses | Success rate |
|---|---|---|---|---|
| K3s | ± 3,6 Juta | ± 3,6 Juta | 0 | 100% |
| Swarm | ± 3,6 Juta | ± 3,6 Juta | 4 | 99.9% |

Pada TABEL V, terlihat bahwa Docker Swarm dan K3s dapat menunjukkan nilai throughput yang stabil pada setiap pengujian. Akan tetapi, pada TABEL VI terlihat perbedaan yang sangat jelas terhadap reliabilitas antara Docker Swarm dan K3s, pada kurang lebih 3.6 juta pesan yang terkirim. K3s dapat meraih keberhasilan pemrosesan pesan 100% tanpa mengalami kegagalan pada setiap pengujian yang dilakukan. Sebaliknya, Docker Swarm hanya meraih keberhasilan pemrosesan pesan 99.9% dimana terdapat 4 kegagalan pemrosesan pesan yang terjadi. Keempat pemroesan pesan ini terjadi pada setiap awal mulai pengujian ketika network menerima pesan yang membuat queue tertahan hingga ada pemroesan pesan yang gagal.

4) *Skalabilitas:* Pada tahap ini adalah perbandingan skalabilitas dari Docker Swarm dan K3s pada setiap pengujian yang dilakukan.




Gbr. 8 Skalabilitas Docker Swarm dan K3s

Pada Gbr. 8 terlihat bahwa kedua platform mengalami skalabilitas seiring dengan peningkatan beban pengiriman

pesan. K3s memiliki mekanisme skalabilitas Horizontal Pod Autoscaler (HPA) yang cenderung agresif, HPA ini menghitung penggunaan CPU kontainer dan mereplikasi kontainer berdasar maksimal replikasi atau hingga penggunaan CPU stabil. Sehingga pada skalabilitasnya K3s sudah mengalami replikasi pada saat menerima pesan meskipun pada beban 100 pesan/detik, yang dimana mempengaruhi penggunaan CPU hingga mencapai puncaknya 48%, tetapi bisa mengoptimalkan penggunaan memori kontainer stabil 4 GB.

Sedangkan, Docker Swarm menggunakan skrip custom untuk melakukan autoscaling yang dimana menghitung CPU dengan mengambil rata-rata semuanya. Sehingga pada pengujian beban 100 pesan/detik tidak adanya replikasi yang terjadi dan penggunaan CPU kontainer terlampau lebih rendah 29.5%. Akan tetapi, penggunaan memori yang lebih tinggi 4.6 GB karena kurangnya resource saat tejadi peningkatan beban. Selain itu, kedua platform mampu mengembalikan pod/kontainer yang hilang pada saat pengujian ketahanan dengan waktu kurang lebih 1 hingga 2 menit.

### B. *Kemudahan instalasi dan Deployment*

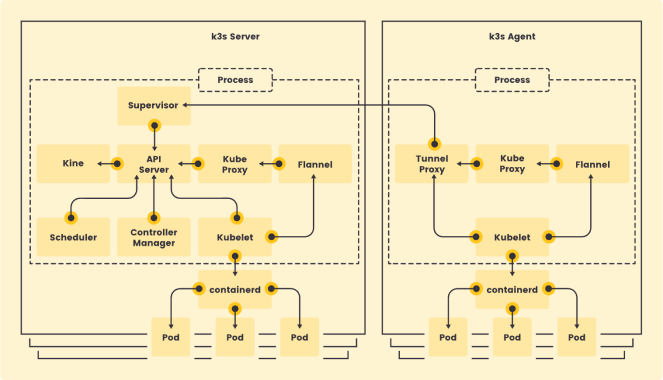
Perbandingan Selanjutnya adalah kemudahan instalasi dari kedua platform, Docker Swarm dan K3s. K3s memiliki cara instalasi yang terlampau lebih mudah daripada K8S, dokumentasi dari K3s memberikan cara instalasi dengan menggunakan binary yang sudah disediakan mereka dan instal dengan skrip yang berada di dalamnya juga. Command yang digunakan untuk instalasi K3s adalah ”*curl -sfL https://get.k3s.io | sh -s - server --write-kubeconfig-mode 644”*. Selanjutnya adalah instalasi pada node worker dengan command yang hampir sama seperti sebelumnya ” *curl -sfL https://get.k3s.io | K3S URL=https://192.168.10.70:6443 K3S_TOKEN=Node-token-from-master sh –*“ hanya berbeda pada k3s url dan k3s token untuk mengenalkan node masternya.

Untuk deployment pada K3s memiliki metode yang lebih kompleks, dimana ketika melakukan deploy satu kontainer/pod dibutuhkan setidaknya 4 file manifest seperti, deployment, konfigurasi pod, service, dan Horizontal Pod Autosclaing policy, yang dimana untuk deployment perlu di apply semuanya satu per satu. Semua itu membuat K3s memiliki pembelajaran yang lebih tinggi, tetapi itu bisa membuat K3s menjadi platform orkestrasi pada skala perusahaan.

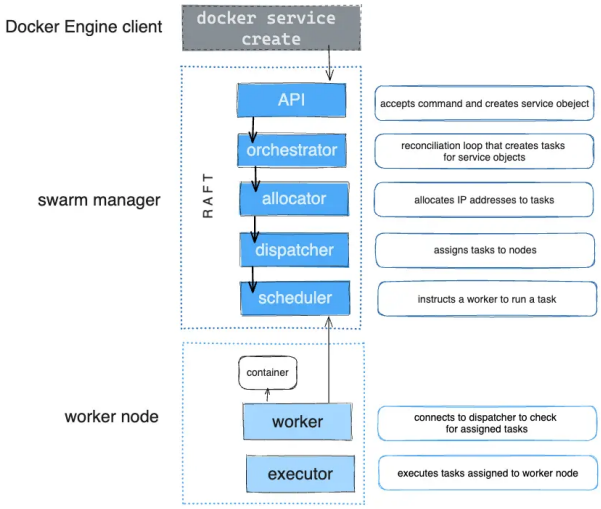
Sebaliknya, Docker Swarm memilki inisiasi yang terlampau lebih mudah dari pada K3s, yang dimana menggunakan command “*docker swarm init*” untuk memulai swarm pada node master. Sedangkan untuk node worker Docker Swarm hanya membutuhkan command yang langsung tersedia dari hasil command docker swarm init “*docker swarm join --token <token> 192.168.10.75:2377*” untuk dapat bergabung ke dalam cluster dari node master. Berbeda dengan K3s, Docker Swarm memiliki tingkat pemebelajran yang lebih rendah, karena untuk deployment satu aplikasi tidak perlu banyak melakukan konfigurasi abstraksi api. Untuk melakukan deploy pun hanya perlu memerlukan satu command dengan compose file yang sudah diperisapkan. Hal itu membuat Docker swarm menjadi tempat yang sesuai untuk melakukan prototyping, local environment yang minimum, dan infrastruktur yang memiliki resource yang terbatas.

### C. *Telemetrik dan Komunikasi*

Perbandingan akan berfokus pada arsitektur kedua platform yang dimana dapat menjelaskan perbedaan komunikasi internal antara Docker Swarm dan K3s dan juga telemetrik yang terekspos antara Docker Swarm dan K3s.




Gbr. 9 Arsitektur K3s




Gbr. 10 Arsitektur Docker Swarm

Pada Gbr. 9 dan Gbr. 10 terlihat bahwa arsitektur dari K3s dan Docker Swarm, yang memiliki pola komunikasi dan mekanisme yang berbeda di dalam satu cluster. K3s beroperasi dengan pendekatan server-agent, dimana agent node berkomunikasi dengan control-plane pada server node melalui koneksi websocket yang persisten dan internal client-side load balancer [9]. Bagaimana data terkirim K3s menggunakan flannel Container Network Interface (CNI) dengan backend VXLAN untuk mendistribusikan subnet ipv4 antar node [10]. Kompenen ini bekerja sama dengan CoreDNS untuk service discovery, kube-proxy, untuk konfigurasi iptables pada host, serta ServiceLB bawaan untuk mengabstrasikan traffic yang masuk lintas pod dijadwalkan secara dinamis.

Sebaliknya, Docker Swarm menggunakan pendekatan manager-worker yang digerakkan oleh Raft Consensus Protocol [11]. Dalam artsitektur ini manager node berfungsi untuk penjadwal terpusat yang memecah definisi service yang dideklarasikan menjadi beberapa task yang disebarkan ke worker node [12]. Docker swarm menggunakan konektivitas layanan global menggunakan mesin jaringan multi-layer bawaan terdiri dari ingress network untuk distribusi lalu lintas menyerah, serta antarmuka docker_gwbridge untuk menghubungkan setiap daemon docker tingkat node.

Untuk eksposur metrik dan agregasi telemetri kedua platform memiliki cara yang berbeda. K3s menunjukkan ekosistem pemantuan cloud-native dan mengintegrasikan eksposur metrik kontainer dan runtime secara bawaan di dalamnya. Sehingga pengumpulan metrik pada K3s berfokus pada Node-Exporter untuk mengambil metrik CPU, Memori, dan Network I/O pada level node. Sedangkan, Docker Swarm menerapkan model pencatatan terdesentralisasi statistik kontainer yang disimpan pada disk local setiap daemon dan tidak adanya API aggregator untuk ekspos metrik pusat cluster. Sehingga untuk mendapat telemetrik pada Docker Swarm diperlukan cAdvisor yang diarahkan pada docker socket untuk mengambil kontainer dan Node-Exporter untuk mengambil metrik CPU, Memori, dan Network I/O pada level node.

## IV. KESIMPULAN

Berdasar dari penilitian yang sudah dilakukan terhadapa platform orkestrasi kontainer Docker Swarm dan K3s dalam aplikasi real-time di lingkungan private cloud, dapat disimpulkan beberapa hal. Secara umum kedua platform dapat menjalankan aplikasi real-time dengan baik dan mampu menangani beban 1000 pesan/detik tanpa adanya bottleneck pada sistem. Namun kedua platform memiliki performa yang berbeda dimana K3s memiliki performa latensi yang lebih baik dan konsisten pada latensi P50, P95, dan P99 di semua pengujian. Dari sisi penggunaan resource Docker Swarm memiliki nilai yang lebih kecil dan optimal terutama pada penggunaan CPU di kontainer. Pada throughput dan success rate kedua platform menunjukkan performa yang seimbang dengan tingkat keberhasil mendekati 100% pada Docker Swarm dan tanpa kegagalan pada K3s.

Pada platform yang lebih optimal pada aplikasi real-time, K3s dapat dikatakan sebagai platform yang lebih optimal untuk aplikasi real-time daripada Docker Swarm. Hal itu dikarenakan latensi dari K3s yang menunjukkan nilai dan kestabilan lebih rendah dibandingkan dengan Docker Swarm, terutama pada latensi P99 yang mencapai ambang batas histogram 30 detik. Tetapi Docker Swarm juga bisa digunakan sebagai alternatif ringan karena penggunaan resource yang relatif lebih rendah dari pada K3s.

Pada sisi skalabilitas dan ketahanan dari kedua platform Docker Swarm dan K3s, mampu menunjukkan skalabilitas replika dengan baik dari awal pengujian saat throughput meningkat sebanding dengan peningkatan beban. Kedua platform memiliki mekanisme scaling yang berbeda, dimana K3s menggunakan Horizontal Pod Autoscaler (HPA) bawaan dari kubernetes sedangkan Docker Swarm harus menggunakan autoscaler skrip kustom dari pihak ketiga. Meskipun begitu pada pengujian ketahanan yang dilakukan Docker Swarm dan K3s mampu mempertahankan operasional sistem dan mengembalikan kontainer/pod yang mati saat salah satu node dimatikan, dimana K3s memiliki autoscaler yang lebih agresif dibanding Docker Swarm.

Kesimpulan akhir yang dapat diambil adalah K3s lebih unggul dalam aspek latensi, kestabilan performa, dan reliabilitas sehingga lebih sesuai digunakan untuk aplikasi real-time. Sedangkan Docker Swarm memiliki keunggulan dalam efisiensi pengunaan CPU dan memiliki kelemahan dalam latensi yang ekstrem membuatnya tidak sesuai dalam aplikasi real-time, akan tetapi lebih sesuai jika digunakan untuk aplikasi yang berfokus pada efisiensi resource.

## V. REFERENSI

[1] R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access,* 2017.

[2] V. Marella, "Comparative Analysis of Container Orchestration Platforms: Kubernetes vs. Docker Swarm," *International Journal of Scientific Research in Science and Technology,* pp. 526-543, 2024.

[3] K. Schmidt, C. Phillips and A. Chuvakin, "Logging And Log Management The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management," Waltham, Syngress, 2013, p. 384.

[4] R. Morabito, J. Kjällman and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015,* pp. 386-393, 2015.

[5] S. Eldrige, "Percentile | Definition, Quartile, & Facts | Britannica," 10 4 2026. [Online]. Available: https://www.britannica.com/topic/percentile. [Accessed 19 5 2026].

[6] J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach EIGHTH EDITION," Pearson Education Limited, 2022, p. 73.

[7] G. S. Nikolic, B. R. Dimitrijevic, T. R. Nikolic and M. K. Stojcev, "A Survey of Three Types of Processing Units: CPU, GPU and TPU," *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST 2022,* 2022.

[8] "All about computer memory - Microsoft Support," Microsoft, [Online]. Available: https://support.microsoft.com/en-us/windows/all-about-computer-memory-79a75280-86ff-457c-bfbd-8439e7f7f88b. [Accessed 19 5 2026].

[9] "Architecture | K3s," 21 May 2026. [Online]. Available: https://docs.k3s.io/architecture.

[10] "GitHub - flannel-io/flannel: flannel is a network fabric for containers, designed for Kubernetes · GitHub," [Online]. Available: https://github.com/flannel-io/flannel. [Accessed 21 5 2026].

[11] "Raft consensus in swarm mode | Docker Docs," [Online]. Available: https://docs.docker.com/engine/swarm/raft/. [Accessed 21 5 2026].

[12] D. D. Author, "How services work | Docker Docs," [Online]. Available: https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/. [Accessed 21 5 2026].