Sistem Terdistribusi (ST)

Pertemuan 9 : Studi Kasus Sistem Cloud

Alauddin Maulana Hirzan

Fakultas Teknologi Informasi dan Komunikasi
Universitas Semarang

Outline

Konsep Dasar Analisis Kebutuhan Sistem

Definisi

Analisis kebutuhan sistem merupakan proses sistematis untuk mengidentifikasi, memahami, dan mendokumentasikan kebutuhan pengguna serta batasan sistem sebelum tahap perancangan dilakukan. Dalam konteks sistem terdistribusi, proses ini menjadi lebih kompleks karena melibatkan banyak komponen yang tersebar, komunikasi jaringan, serta dependensi antar layanan. Tujuan utamanya adalah memastikan bahwa sistem yang dikembangkan mampu memenuhi kebutuhan pengguna secara tepat, efisien, dan dapat diskalakan.

Contoh:

Pada sistem cloud untuk e-learning, analisis kebutuhan mengidentifikasi bahwa pengguna membutuhkan akses materi secara real-time, sistem harus mendukung ribuan pengguna simultan, serta memiliki fitur autoscaling ketika terjadi lonjakan trafik.

Kategori Kebutuhan Sistem

Kebutuhan sistem dibagi menjadi dua kategori utama:

1. Functional Requirements: Menjelaskan fungsi atau layanan yang harus disediakan sistem.
2. Non-Functional Requirements: Menjelaskan kualitas sistem seperti performa, keamanan, dan keandalan.

Pada sistem terdistribusi, non-functional requirements sangat krusial karena berkaitan dengan latency, fault tolerance, dan consistency antar node.

Contoh:

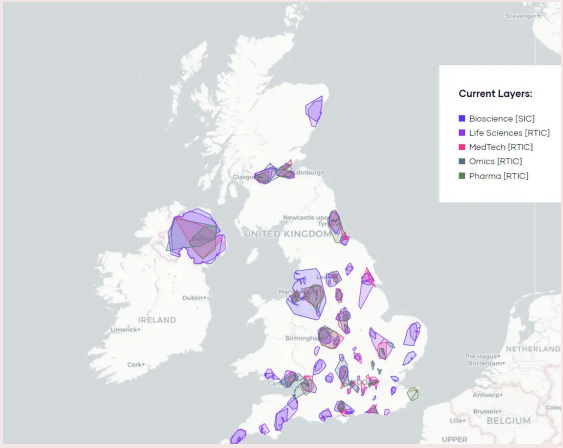
- Functional : Sistem harus dapat menyimpan dan mengambil data pengguna.
- Non-Functional : Waktu respon sistem < 2 detik dan sistem tetap berjalan meskipun satu server gagal.

Tantangan Analisis pada Sistem Terdistribusi

Analisis kebutuhan pada sistem terdistribusi memiliki beberapa tantangan utama:

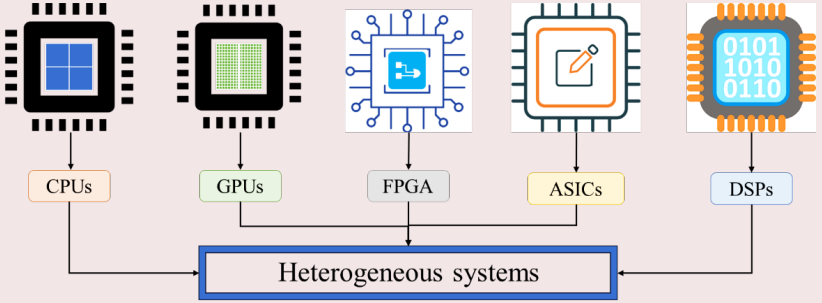
1. Distribusi geografis : Node berada di lokasi berbeda
2. Heterogenitas sistem : Perbedaan platform, OS, dan bahasa pemrograman
3. Konsistensi data : Sinkronisasi antar node sulit dijaga
4. Reliabilitas jaringan : Ketergantungan pada koneksi jaringan
5. Skalabilitas : Sistem harus mampu menangani pertumbuhan pengguna

Peta Distribusi



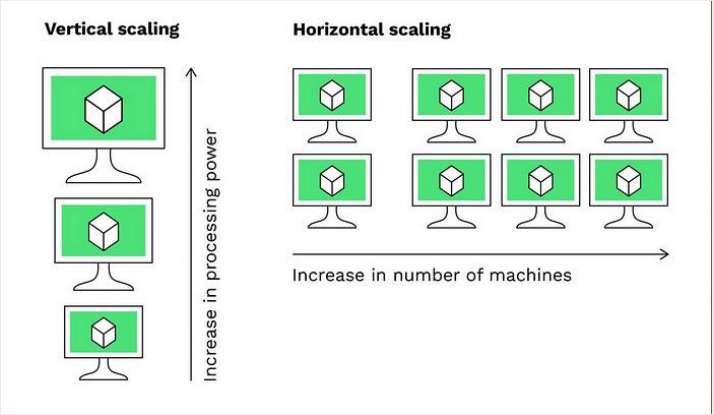
Ilustrasi

Heterogenitas Sistem



Ilustrasi

Skalabilitas



Platform E-Commerce Berbasis Cloud

Definisi

Platform e-commerce berbasis cloud merupakan sistem terdistribusi yang melayani transaksi online secara real-time dengan jumlah pengguna yang dinamis. Sistem ini memanfaatkan infrastruktur cloud untuk mencapai skalabilitas tinggi, ketersediaan layanan, dan performa yang stabil. Arsitektur umumnya terdiri dari frontend (web/mobile), backend service, serta database yang tersebar untuk mendukung transaksi dan penyimpanan data.

Contoh:

Marketplace seperti sistem penjualan online yang menangani ribuan transaksi per detik, termasuk fitur katalog produk, keranjang belanja, pembayaran, dan tracking pesanan.

Komponen Utama Sistem Cloud

Sistem cloud terdistribusi umumnya terdiri dari beberapa komponen utama:

- Client (Mobile/Web) : Antarmuka pengguna untuk mengakses sistem
- Backend Service (API) : Menyediakan logika bisnis dan komunikasi antar layanan
- Database Terdistribusi : Menyimpan data secara terdistribusi untuk meningkatkan availability dan fault tolerance
- Cloud Infrastructure : Infrastruktur komputasi seperti VM, container, dan orchestration (misalnya Kubernetes)

Contoh:

Pada sistem e-commerce:

- Client → aplikasi Android
- Backend → REST API untuk transaksi
- Database → NoSQL cluster untuk data produk

Arsitektur Umum: Microservices

Arsitektur microservices membagi sistem menjadi layanan-layanan kecil yang independen dan saling terhubung melalui API. Setiap service memiliki tanggung jawab spesifik dan dapat dikembangkan serta di-deploy secara terpisah. Pendekatan ini meningkatkan skalabilitas dan fleksibilitas, tetapi menambah kompleksitas dalam komunikasi dan manajemen sistem.

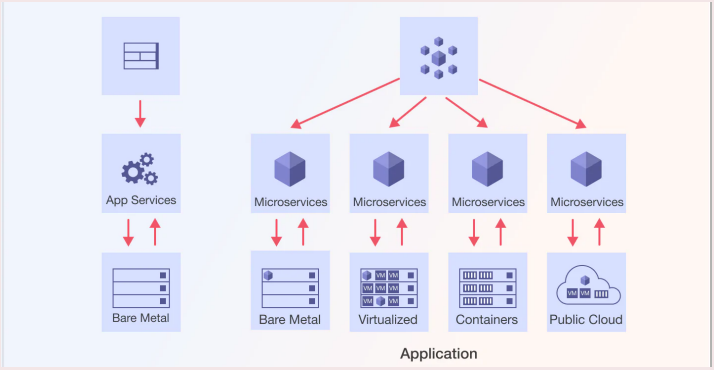
Contoh:

Pada e-commerce:

1. Service User
2. Service Product
3. Service Payment
4. Service Order

Ilustrasi

Microservices



Identifikasi Kebutuhan Sistem

Definisi

Identifikasi kebutuhan sistem merupakan tahap lanjutan dari analisis kebutuhan yang bertujuan merinci fungsi dan kualitas sistem secara operasional. Pada sistem cloud, kebutuhan tidak hanya berfokus pada fitur, tetapi juga pada kemampuan sistem dalam menangani beban dinamis, distribusi layanan, dan integrasi antar komponen.

Contoh:

Pada sistem e-commerce, kebutuhan tidak hanya “melakukan transaksi”, tetapi juga “mampu menangani 10.000 pengguna secara bersamaan”.

Functional Requirements

1. Autentikasi Pengguna
2. Manajemen Data
3. API Komunikasi
4. Proses Bisnis

Functional Requirements

Authentikasi Pengguna

Sistem harus menyediakan mekanisme autentikasi untuk memastikan hanya pengguna yang sah dapat mengakses layanan. Ini mencakup login, logout, serta manajemen sesi.

Contoh:

Pengguna login menggunakan email dan password, kemudian sistem menghasilkan token (JWT) untuk akses ke API.

Functional Requirements

Manajemen Data

Sistem harus mampu melakukan operasi CRUD *(Create, Read, Update, Delete)* terhadap data. Dalam sistem terdistribusi, data dapat disimpan di beberapa node.

Contoh:

Admin dapat menambah produk, pengguna dapat melihat katalog, dan sistem menyimpan data transaksi dalam database terdistribusi.

Functional Requirements

API Komunikasi

Backend menyediakan API sebagai jembatan komunikasi antara client dan service lain. API harus terdefinisi dengan jelas dan konsisten.

Contoh:

Endpoint `/api/products` untuk mengambil data produk, `/api/orders` untuk membuat pesanan baru.

Functional Requirements

Proses Bisnis Utama

Sistem harus mengimplementasikan alur utama sesuai domain aplikasi (business logic). Pada sistem cloud, proses ini sering didistribusikan ke beberapa service.

Contoh:

Pada e-commerce: alur pemesanan $\rightarrow$ validasi stok $\rightarrow$ pembayaran $\rightarrow$ konfirmasi $\rightarrow$ pengiriman.

Non-Functional Requirements

- Scalability (Horizontal Scaling)
- Availability dan Reliability
- Performance (Latency Rendah)
- Security

Non-Functional Requirements

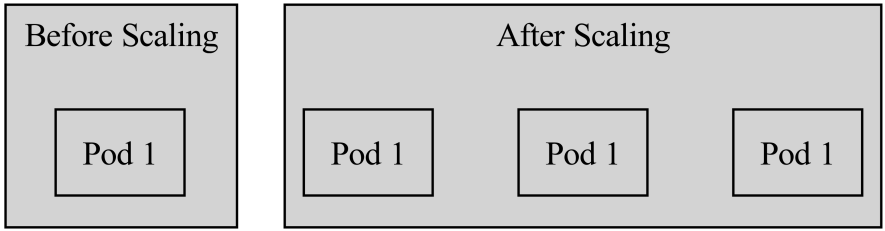
Scalability

Sistem harus mampu meningkatkan kapasitas dengan menambah jumlah instance/server (bukan hanya meningkatkan spesifikasi hardware).

Contoh:

Menambah container backend saat jumlah pengguna meningkat (auto-scaling pada Kubernetes).

Ilustrasi



Non-Functional Requirements

Availability dan Reliability

Availability: Sistem harus selalu tersedia (misalnya uptime 99.9%) Reliability: Sistem harus tetap berfungsi dengan benar meskipun terjadi kegagalan sebagian komponen

Contoh:

Jika satu server gagal, traffic dialihkan ke server lain tanpa mengganggu pengguna.

Ilustrasi

Uptime	Downtime	
	per month	per year
100%	0m	0m
99.999%	0.4m	5m
99.99%	4m	52m
99.9%	43m	326m

Non-Functional Requirements

Performance

Sistem harus memberikan respon cepat, terutama untuk aplikasi real-time. Latency menjadi faktor penting dalam pengalaman pengguna.

Contoh:

Waktu respon API < 200 ms untuk request data produk.

Non-Functional Requirements

Security

Keamanan merupakan aspek kritis dalam sistem cloud:

- Enkripsi: Data dienkripsi saat transmisi (TLS/HTTPS) maupun saat disimpan
- Authentication & Authorization: Validasi identitas dan pengaturan hak akses

Contoh:

Penggunaan HTTPS untuk komunikasi dan role-based access control (RBAC) untuk membatasi akses admin dan user.

Kebutuhan Spesifik Cloud

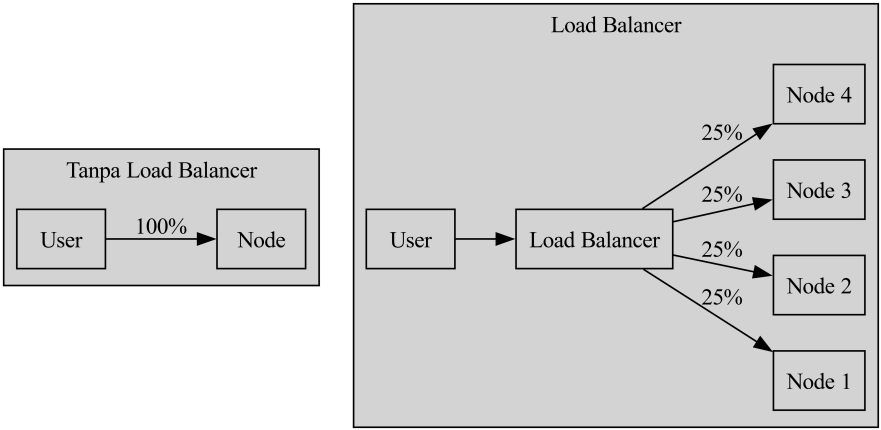
Sistem cloud memiliki kebutuhan tambahan yang tidak selalu ada pada sistem tradisional:

- Elastic Resource Allocation: Penyesuaian resource secara dinamis
- Load Balancing: Distribusi beban ke beberapa server
- Auto-Scaling: Penambahan/pengurangan instance otomatis
- Multi-Region Deployment: Deployment di beberapa lokasi geografis untuk meningkatkan availability

Contoh:

Aplikasi di-deploy di beberapa region (Asia dan Eropa), sehingga pengguna mendapatkan latency lebih rendah dan sistem tetap berjalan saat satu region down.

Ilustrasi


Tanpa Load Balancer
User
100%
Node
Load Balancer
User
Load Balancer
25%
25%
25%
25%
Node 4
Node 3
Node 2
Node 1

Konsep Pemetaan Kebutuhan ke Cloud

Definisi

Mapping kebutuhan ke arsitektur cloud merupakan proses menerjemahkan kebutuhan sistem menjadi komponen infrastruktur dan layanan cloud yang spesifik. Pendekatan ini memastikan setiap requirement memiliki representasi teknis yang jelas, sehingga mempermudah implementasi, deployment, dan scaling pada sistem terdistribusi.

Contoh:

Kebutuhan “high availability” dipetakan menjadi deployment multi-instance dengan load balancer dan multi-region.

Mapping ke Compute (VM / Container)

Kebutuhan komputasi (compute) berkaitan dengan eksekusi aplikasi dan layanan backend. Dalam sistem modern, compute biasanya diimplementasikan menggunakan Virtual Machine (VM) atau container.

Mapping:

- Functional (API, business logic) $\rightarrow$ Containerized services
- Scalability $\rightarrow$ Container orchestration (horizontal scaling)
- Reliability $\rightarrow$ Multiple instances (replication)

Mapping ke Storage (Object / Block / Database)

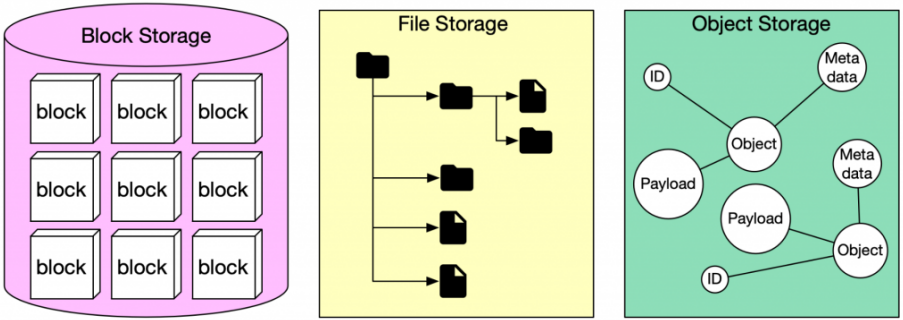
Kebutuhan penyimpanan dipetakan berdasarkan jenis data dan pola akses:

- Object Storage: Untuk file statis (gambar, video)
- Block Storage: Untuk kebutuhan disk pada VM
- Database Terdistribusi: Untuk data aplikasi yang membutuhkan konsistensi dan query

Mapping:

- Data user $\rightarrow$ Database (SQL/NoSQL)
- File produk $\rightarrow$ Object storage
- Backup $\rightarrow$ Distributed storage

Ilustrasi


Block Storage
block
block
block
block
block
block
block
block
block
File Storage
Object Storage
ID
Meta data
Object
Payload
Meta data
Payload
Object
ID

Mapping ke Networking

Kebutuhan komunikasi antar layanan dipenuhi melalui komponen networking:

- Virtual network (VPC)
- Subnet dan routing
- Firewall dan security group
- API communication (HTTP, gRPC, messaging)

Mapping:

- Security → Firewall rules dan network isolation
- Performance → Low-latency routing
- Availability → Redundant network paths

Desain Arsitektur Cloud

Desain arsitektur mencakup komponen penting dalam sistem terdistribusi:

1. Load Balancer:
   - Mendistribusikan traffic ke beberapa instance untuk meningkatkan availability dan performance
   - Contoh: Request user diarahkan ke server dengan beban paling rendah
2. Service Registry:
   - Menyimpan daftar service yang aktif agar service lain dapat menemukannya secara dinamis
   - Contoh: Microservices mendaftarkan dirinya saat startup
3. API Gateway:
   - Pintu masuk utama untuk semua request client, mengatur routing, autentikasi, dan rate limiting
   - Contoh: Semua request dari mobile app melewati gateway sebelum diteruskan ke service terkait

Lastly

Terima Kasih