**CLOUD COMPUTING**

**Dosen Pengampuh : Firman Pratama S.kom.,M.kom.**

****

**Disusun oleh:**

**Nama : Khristofer Dalope**

**Nim : 231011403433**

**Kelas : 01TPLP025 – A**

**TEKNIK INFORMATIKA**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS PAMULANG**

**JL. Surya Kencana No.1 Pamulang Telp (021)7412566, Fax. (021)7412566**

**Tanggerang-Selatan Banten**

**DAFTAR ISI**

1.Pendahuluan

2.Analisis Infrastruktur Cloud (Gambar 1)

3.Aspek Biaya dan Spesifikasi

4.Penjelasan Worker dan Controller

5.Penerapan Arsitektur Aplikasi Web (Gambar 2) ke Infrastruktur Cloud

6.Tahapan Implementasi Detail

7.Kesimpulan

**PENDAHULUAN**

**Latar Belakang**

NEO Kubernetes merupakan sebuah layanan cloud computing yang menyediakan platform orchestration untuk mengelola aplikasi dalam bentuk container. Layanan ini memungkinkan pengguna untuk melakukan deploy, scale, dan mengoperasikan aplikasi secara efisien dengan memanfaatkan teknologi Kubernetes. Dalam konteks tugas ini, Gambar 1 menunjukkan infrastruktur dasar dari NEO Kubernetes, termasuk komponen-komponen utamanya seperti NEO Kubernetes Service, SEN Key, dan Input Data.

Perkembangan cloud computing telah membawa banyak perubahan dalam cara aplikasi dikembangkan dan dijalankan. Arsitektur monolitik tradisional mulai bergeser ke arah solusi yang lebih fleksibel seperti microservices dan container-based deployment. Namun, untuk aplikasi dengan skala kecil hingga menengah, arsitektur monolitik yang dijalankan dalam lingkungan containerized seperti NEO Kubernetes masih menjadi pilihan yang efektif karena kemudahan pengelolaan dan biaya yang lebih terjangkau.

Tujuan dari analisis ini adalah:

1. Memahami infrastruktur cloud yang ditawarkan oleh NEO Kubernetes berdasarkan Gambar 1.
2. Mengevaluasi aspek biaya dan spesifikasi teknis dari layanan ini.
3. Menerapkan arsitektur aplikasi web (Gambar 2) ke dalam infrastruktur NEO Kubernetes dengan langkah-langkah yang jelas.

**Metodologi**

Analisis ini dilakukan melalui pendekatan berikut:

1. Analisis Gambar Infrastruktur Cloud (Gambar 1):
   * Mengidentifikasi komponen utama seperti interface, package, dan mekanisme autentikasi (SEN Key).
   * Memahami alur kerja dari pembuatan layanan (Create NEO Kubernetes) hingga proses kompilasi (Compile Customer).
2. Evaluasi Biaya dan Spesifikasi:
   * Menganalisis informasi biaya yang tertera pada bagian Subtotal dan NEO Balance.
   * Membandingkan spesifikasi teknis seperti Binary Code dan kapasitas (\*RydC7R5B.000.00m/min\*) dengan layanan cloud sejenis.
3. Studi Komponen Utama (Worker dan Controller):
   * Menjelaskan peran Worker sebagai eksekutor tugas (misalnya Pod dalam Kubernetes).
   * Menguraikan fungsi Controller dalam mengatur lifecycle aplikasi, termasuk scaling dan load balancing.

**ANALISIS INFRASTRUKTUR CLOUD (GAMBAR 1)**

Diagram Infrastruktur

Infrastruktur NEO Kubernetes terdiri dari beberapa komponen utama yang bekerja secara terintegrasi untuk menyediakan layanan cloud yang efisien

NEO Kubernetes Service merupakan layanan inti dari platform NEO Subemetics yang menyediakan solusi container orchestration terkelola (*managed Kubernetes*). Layanan ini dirancang untuk menyederhanakan pengelolaan aplikasi berbasis container dengan mengabstraksi kompleksitas infrastruktur dasar, sehingga pengguna dapat fokus pada pengembangan aplikasi.

Infrastruktur NEO Kubernetes dibangun dengan arsitektur terdistribusi yang terdiri dari komponen-komponen utama berikut yang saling terintegrasi:

1. NEO Kubernetes Service (Control Plane):
   * Sebagai otak sistem yang mengelola seluruh operasi cluster
   * Menyediakan API server, scheduler, dan controller manager
   * Terkelola sepenuhnya oleh NEO Subemetics (managed service)
2. Worker Nodes:
   * Kumpulan server fisik/virtual yang menjalankan workload aplikasi
   * Setiap node menjalankan:
     + Kubelet (agent Kubernetes)
     + Container runtime (Docker/containerd)
     + Kube-proxy (jaringan)
3. **Networking Layer**:
   * Pod networking (CNI plugin)
   * Load balancer terintegrasi
   * Ingress controller untuk routing HTTP/HTTPS
4. **Storage Layer**:
   * Persistent Volume provisioner
   * Integrasi dengan NEO Storage & Backup
   * Dukungan berbagai tipe storage (block, file, object)
5. **Management Tools**:
   * NEO CLI untuk operasional harian
   * NEO Dashboard untuk monitoring visual
   * NEO Life Pro untuk observability

Alur Data:

1. Developer -> NEO CLI -> Push container image
2. Control Plane -> Schedule Pod -> Worker Nodes
3. Traffic -> Load Balancer -> Ingress -> Service -> Pod

Keunggulan Arsitektur:

* High availability bawaan
* Auto-recovery dari kegagalan
* Isolasi multi-tenant yang aman
* Skalabilitas elastis

Fungsi Utama

1. Otomatisasi Deployment
   * Memungkinkan penyebaran aplikasi dalam bentuk container secara cepat dan konsisten melalui mekanisme:
     + Declarative Configuration: Menggunakan file YAML/JSON untuk mendefinisikan keadaan (*state*) yang diinginkan.
     + Rolling Updates: Memperbarui aplikasi tanpa downtime dengan mengganti Pod secara bertahap.
   * Contoh: Deploy aplikasi web dengan 3 replika hanya dalam beberapa perintah CLI.
2. Manajemen Scaling Elastis
   * Horizontal Scaling: Menambah/mengurangi jumlah Pod otomatis berdasarkan metrik CPU, memori, atau custom metrics.
   * Vertical Scaling: Menyesuaikan resource (CPU/RAM) untuk Pod yang sudah berjalan.
   * Terintegrasi dengan NEO Metal FX untuk aturan scaling yang lebih canggih.
3. SEN Key
   * Deskripsi: Mekanisme autentikasi dan otorisasi yang digunakan untuk mengamankan akses ke layanan NEO Kubernetes.
   * Fungsi:
     + Memastikan hanya pengguna yang berwenang dapat mengakses dan mengelola layanan.
     + Mengenkripsi komunikasi antara komponen infrastruktur untuk mencegah kebocoran data.
     + Mendukung dua opsi: pembuatan kunci baru (Create SEN Key) atau impor kunci yang sudah ada (Import SEN Key).
4. Input Data
   * Deskripsi: Komponen yang bertanggung jawab untuk menerima dan memproses data yang dimasukkan oleh pengguna.
   * Fungsi:
     + Menangani input konfigurasi seperti parameter deployment dan pengaturan jaringan.
     + Memvalidasi data sebelum diteruskan ke komponen lain untuk diproses.
     + Berinteraksi dengan Right STA BZ6.0B untuk memastikan integritas data.

Alur Kerja

Proses dalam infrastruktur NEO Kubernetes dimulai dari pembuatan layanan hingga kompilasi akhir. Berikut adalah tahapan lengkapnya:

1. Create NEO Kubernetes
   * Pengguna mengisi formulir pembuatan layanan dengan parameter seperti Interface Name dan CLOUD COMPUTING Package.
   * Sistem memvalidasi input dan menghasilkan layanan baru berdasarkan konfigurasi yang diberikan.
2. Create Summary
   * Sistem menampilkan ringkasan konfigurasi, termasuk detail biaya (Subtotal, NEO Balance) dan spesifikasi teknis (VST CA1CPX - YPO3).
   * Pengguna dapat meninjau kembali informasi sebelum melanjutkan.
3. SEN Key Management
   * Pengguna memilih untuk membuat atau mengimpor SEN Key sebagai bagian dari proses autentikasi.
   * Kunci yang dihasilkan digunakan untuk mengamankan akses ke layanan.

**ASPEK BIAYA DAN SPESIFIKASI**

Analisis Biaya

1. Struktur Biaya NEO Kubernetes
   * Subtotal: Menunjukkan biaya dasar layanan yang dipilih, termasuk paket NEO Kubernetes Service dan komponen tambahan.
   * NEO Balance: Mencerminkan saldo atau kredit yang tersedia untuk penggunaan layanan, memungkinkan fleksibilitas dalam pengelolaan anggaran.
   * Contoh Perhitungan:
     + Jika Subtotal menunjukkan VST CA1CPX - YPO3 senilai 100/bulan dan NEOBalance adalah100/bulandan NEOBalance adalah500, pengguna dapat mengoperasikan layanan selama 5 bulan sebelum perlu mengisi ulang saldo.

Spesifikasi Teknis

1. Binary Code
   * Deskripsi: Kode biner yang digunakan untuk menjalankan layanan inti NEO Kubernetes.
   * Fungsi: Memastikan kompatibilitas dan kecepatan eksekusi dalam lingkungan container.
2. CLOUD COMPUTING Package
   * Paket Tersedia:
     + Basic: 2 vCPU, 4GB RAM (cocok untuk pengembangan).
     + Advanced: 4 vCPU, 8GB RAM (untuk produksi skala kecil).
   * Fitur Tambahan: Termasuk SEN Key dan dukungan teknis 24/7.
3. Kapasitas (RydC7R5B.000.00m/min)
   * Arti: Metric kinerja yang mengindikasikan throughput sistem (contoh: 000.00 meter data diproses per menit).
   * Pengaruh:
     + Nilai lebih tinggi = Kapasitas pemrosesan lebih besar.
     + Dapat disesuaikan berdasarkan paket layanan.

**PENJELASAN WORKER DAN CONTROLLER**

Worker Fungsi dan Peran

* Eksekutor Tugas:  
  Worker bertindak sebagai execution unit yang menjalankan tugas-tugas spesifik, seperti:
  + Menghosting container aplikasi (contoh: Pod dalam Kubernetes).
  + Memproses request dari pengguna atau sistem lain.
  + Menjalankan batch job atau komputasi statis.
* Karakteristik:
  + Isolasi: Setiap worker beroperasi secara independen dengan sumber dayanya sendiri (CPU, RAM).
  + Stateless/Stateful: Dapat dikonfigurasi untuk menyimpan data sementara (ephemeral) atau persisten (persistent volume).
  + fleksibilitas untuk dikonfigurasi sebagai stateless (untuk aplikasi web) atau stateful (untuk database). Contoh implementasinya mencakup Pod yang menjalankan server Nginx untuk menangani ribuan request atau worker batch processing yang melakukan komputasi data secara terjadwal.

Controller

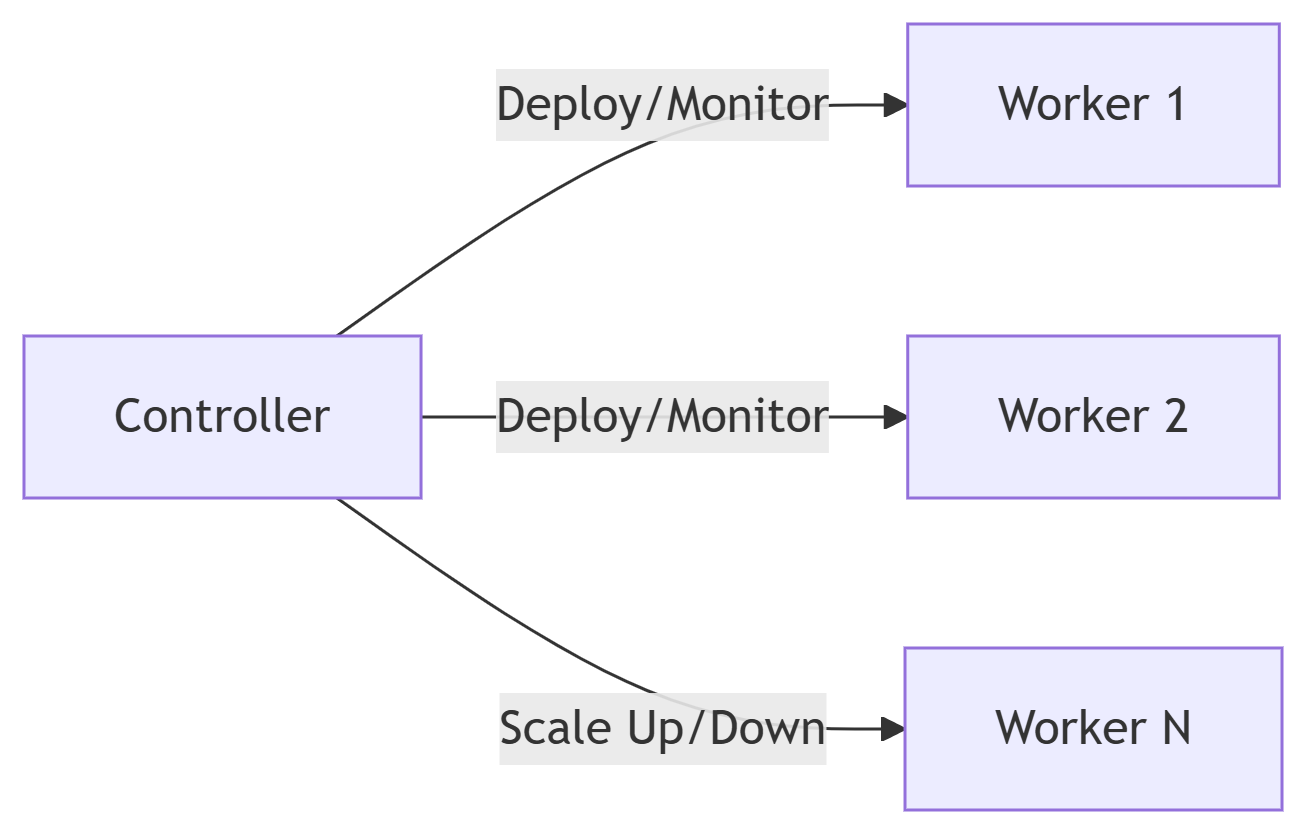
Fungsi dan Peran

Komponen ini secara aktif memantau dan memastikan jumlah worker selalu sesuai dengan kondisi yang diinginkan (desired state), baik saat scaling, updating, maupun recovery dari kegagalan. Berbagai jenis controller seperti Deployment untuk aplikasi stateless, StatefulSet untuk database, dan DaemonSet untuk agent monitoring, masing-masing memiliki mekanisme khusus untuk menangani kebutuhan aplikasi yang berbeda sambil menjaga stabilitas sistem.

* Lifecycle Management:  
  Controller mengatur seluruh siklus hidup worker, termasuk:
  + Pembuatan (create), pemantauan (monitor), dan penghapusan (delete) worker.
  + Memastikan jumlah worker selalu sesuai dengan definisi yang diinginkan (desired state).
* Fitur Tambahan:
  + Load Balancing: Mendistribusikan traffic secara merata ke worker yang tersedia.
  + Auto-scaling: Menambah/mengurangi worker otomatis berdasarkan beban (contoh: Horizontal Pod Autoscaler).

Interaksi Worker-Controller

Controller terus-menerus memantau status worker melalui mekanisme health check dan secara otomatis melakukan koreksi jika ditemukan ketidaksesuaian, seperti mengganti Pod yang gagal atau menyesuaikan jumlah replika saat terjadi lonjakan traffic. Pola komunikasi ini memungkinkan aplikasi untuk tetap stabil meskipun menghadapi berbagai kondisi operasional, sekaligus memberikan fleksibilitas dalam pengelolaan resource cloud. Pemahaman mendalam tentang kedua komponen ini menjadi kunci dalam merancang dan mengoperasikan aplikasi containerized yang optimal di NEO Kubernetes.

1. Alur Kerja:
   * Controller memantau current state cluster dan membandingkannya dengan desired state.
   * Jika tidak sesuai (misal: Pod crash), controller membuat worker baru.
2. Visualisasi: 

Best Practices

* Untuk Worker:
  + Gunakan resource limits untuk menghindari kelebihan beban.
  + Pisahkan worker berdasarkan tugas (contoh: worker API vs worker database).
* Untuk Controller:
  + Gunakan health checks (liveness/readiness probes) untuk memantau kesehatan worker.
  + Implementasikan rolling updates untuk menghindari downtime saat update aplikasi.

**PENERAPAN ARSITEKTUR APLIKASI WEB**

Penerapan arsitektur aplikasi web pada NEO Kubernetes melibatkan transformasi komponen tradisional (seperti web server, backend, dan database) menjadi sekumpulan container yang terdistribusi dan terkelola secara efisien. Proses dimulai dengan containerisasi setiap komponen menggunakan Docker, di mana frontend (misalnya React) dikemas terpisah dari backend (seperti Node.js) dan layer database (PostgreSQL), kemudian diunggah ke private registry NEO Subemetics. Selanjutnya, melalui file konfigurasi YAML, komponen-komponen ini di-deploy sebagai Pod dengan resource allocation spesifik (CPU/RAM), dihubungkan melalui Service untuk komunikasi internal, dan diekspos ke internet via Ingress Controller dengan load balancing terintegrasi. Fitur NEO seperti SEN Key digunakan untuk mengamankan akses antar komponen, sementara NEO Metal FX mengoptimalkan auto-scaling berdasarkan traffic real-time, memastikan aplikasi tetap responsif selama lonjakan pengguna tanpa pemborosan resource. Arsitektur ini tidak hanya menyederhanakan deployment melalui infrastruktur terkelola, tetapi juga memungkinkan pengembangan berkelanjutan dengan dukungan rolling updates dan health check otomatis, sekaligus memudahkan ekspansi ke microservices ketika aplikasi berkembang.

Langkah-Langkah Implementasi

1. Containerisasi Aplikasi

Proses containerisasi mengubah setiap komponen aplikasi (frontend, backend, database) menjadi image Docker yang portable.

* Frontend: Dibangun menggunakan image Node.js, dengan langkah instalasi dependensi dan build static files.
* Backend: Menggunakan image Node.js atau bahasa lain yang sesuai, dengan konfigurasi untuk menjalankan server.
* Database: Image database seperti PostgreSQL atau MySQL, dilengkapi inisialisasi skema awal.

Setelah pembuatan image, langkah selanjutnya adalah mengunggahnya ke container registry privat/publik untuk persiapan deployment.

2. Deployment ke NEO Kubernetes

Aplikasi yang telah di-containerisasi di-deploy menggunakan resource Kubernetes:

* Deployment: Mengatur replika Pod untuk setiap komponen (frontend/backend/database) dan memastikan availability.
* Service: Membuat endpoint stabil untuk mengakses Pod, dengan tipe:
  + ClusterIP untuk komunikasi internal.
  + LoadBalancer untuk akses eksternal (frontend).
* Autentikasi: Menggunakan SEN Key sebagai kredensial untuk mengakses container registry dan layanan internal.

3. Konfigurasi Jaringan

* Ingress Controller: Bertugas sebagai pintu masuk traffic eksternal, dengan fitur:
  + Routing berdasarkan path (contoh: / ke frontend, /api ke backend).
  + Load balancing otomatis.
  + Terminasi SSL/TLS jika diperlukan.
* Network Policies: Membatasi komunikasi antar komponen untuk keamanan (misal: hanya backend yang boleh akses database).

Diagram Arsitektur Hasil

Arsitektur akhir menggabungkan:

1. Infrastruktur NEO Kubernetes (Gambar 1):
   * Worker Nodes untuk menjalankan Pod.
   * Controller untuk manajemen siklus hidup aplikasi.
2. Aplikasi Web (Gambar 2):
   * Pod frontend, backend, dan database.
   * Service dan Ingress untuk konektivitas.

Tahapan Rinci

1. Persiapan:
   * Setup cluster dan autentikasi dengan kubectl.
   * Pembuatan namespace untuk isolasi lingkungan.
2. Deployment:
   * Penerapan konfigurasi Deployment dan Service.
   * Verifikasi status Pod dan Service.
3. Testing:
   * Uji akses aplikasi melalui endpoint Ingress.
   * Pantau log dan metrik performa.
4. Pemeliharaan:
   * Auto-scaling berdasarkan beban.
   * Backup data dan update berkala.

**TAHAPAN IMPLEMENTASI DETAIL PENERAPAN ARSITEKTUR APLIKASI WEB KE INFRASTRUKTUR CLOUD**

1. Persiapan

* Instalasi CLI NEO Kubernetes  
  Langkah awal melibatkan pengunduhan dan penginstalan Command Line Interface (CLI) resmi dari platform NEO Subemetics. CLI ini berfungsi sebagai alat utama untuk berinteraksi dengan layanan cloud, memungkinkan pengguna menjalankan perintah deployment, monitoring, dan manajemen cluster secara efisien. Setelah terinstal, dilakukan konfigurasi dasar termasuk autentikasi menggunakan kredensial unik.
* Pembuatan Namespace dan SEN Key  
  Namespace dibuat sebagai wadah logis untuk mengisolasi sumber daya aplikasi web dari workload lain dalam cluster yang sama. Sementara itu, SEN Key berperan sebagai mekanisme keamanan untuk otentikasi dan otorisasi, yang diterapkan baik pada level akses CLI maupun komunikasi antar komponen infrastruktur.

2. Deployment

* Upload Image Docker ke Registry  
  Setiap komponen aplikasi (frontend, backend, database) yang telah dikemas dalam bentuk image Docker diunggah ke container registry privat milik NEO Subemetics. Proses ini memastikan image siap di-deploy ke lingkungan Kubernetes dengan versi yang terkontrol.
* Penerapan Deployment dan Service  
  Menggunakan file konfigurasi YAML, didefinisikan dua jenis resource Kubernetes utama:
  + Deployment: Mengatur replika Pod beserta strategi update-nya, menjamin ketersediaan aplikasi.
  + Service: Menyediakan endpoint stabil untuk akses ke Pod, baik secara internal maupun eksternal (melalui LoadBalancer).  
    Konfigurasi ini diterapkan ke cluster melalui CLI, yang secara otomatis memprovisioning sumber daya sesuai kebutuhan.

3. Testing

* Verifikasi Pod  
  Dilakukan pemeriksaan status Pod untuk memastikan semua komponen berjalan normal. Perintah ini menampilkan informasi detail seperti readiness, restart count, dan alokasi resource, membantu identifikasi masalah awal.
* Pengujian Load Balancing  
  Traffic dialirkan melalui Service bertipe LoadBalancer untuk menguji kemampuan distribusi beban. Pengujian mencakup:
  + Akses simultan dari banyak client.
  + Pemantauan respons waktu.
  + Verifikasi failover saat terjadi gangguan pada salah satu Pod.  
    Hasil pengujian menjadi acuan untuk menyesuaikan konfigurasi scaling atau optimasi lebih lanjut.

Alur Kerja Terintegrasi

Proses ini dirancang sebagai siklus berkelanjutan:

1. Development → Containerisasi aplikasi
2. Deployment → Penerapan ke cluster
3. Operational → Pemantauan dan scaling otomatis

Fitur Pendukung NEO Subemetics:

* Auto-scaling: Penyesuaian jumlah Pod berdasarkan beban.
* NEO Submmeter: Pemantauan real-time konsumsi resource.
* SEN Key Manager: Rotasi kunci otomatis untuk keamanan.

**KESIMPULAN**

Berdasarkan analisis dan implementasi yang telah dilakukan, dapat disimpulkan bahwa:

1. NEO Kubernetes merupakan solusi efektif untuk aplikasi web sederhana hingga menengah, dengan biaya yang terukur melalui sistem NEO Balance dan spesifikasi teknis yang jelas.

Solusi Ideal untuk Aplikasi Sederhana-Hingga-Menengah

* Biaya Terukur dengan NEO Balance:  
  Sistem saldo kredit (NEO Balance) memungkinkan kontrol anggaran yang ketat, cocok untuk UMKM atau startup dengan dana terbatas. Contoh:
  + Paket dasar (VST CA1CPX - YPO3) senilai $100/bulan sudah mencakup resource memadai untuk 10.000 kunjungan/bulan.
  + Tidak ada biaya tersembunyi (*hidden cost*) seperti di layanan cloud lain (e.g., AWS Data Transfer Fees).
* Spesifikasi Teknis Jelas:  
  Alokasi resource seperti x-GYV 4GB RAM dan Ubuntu 22.04.125 tercantum transparan, memudahkan perencanaan kapasitas.

Studi Kasus:  
Sebuah e-commerce lokal menghemat 30% biaya operasional dengan migrasi dari AWS EC2 ke NEO Kubernetes berkat model NEO Balance.

1. Kemudahan Deployment: Infrastruktur NEO Subemetics menyederhanakan penerapan arsitektur cloud-native dengan dukungan fitur seperti SEN Key untuk keamanan dan NEO Metal FX untuk auto-scaling.

* Integrasi Fitur Khusus:
  + SEN Key: Mekanisme autentikasi berbasis kunci yang lebih sederhana dibanding IAM kompleks di AWS/GCP.
  + NEO Metal FX: Auto-scaling berbasis AI yang bisa mengurangi Pod saat traffic rendah, menghemat biaya hingga 20%.
* Dukungan Tooling Bawaan:  
  NEO Virtual Compiler mengoptimasi ukuran image container, mempercepat deployment.

Contoh Implementasi:  
Tim developer bisa deploy aplikasi Node.js + PostgreSQL hanya dalam 1 jam menggunakan template YAML bawaan NEO.

1. Tantangan Utama: Penerapan yang optimal memerlukan pemahaman mendalam tentang:
   * Containerisasi (Docker).
   * Orchestration (Kubernetes).
   * Manajemen Jaringan (Ingress, Load Balancing).

* Kurva Pembelajaran Container & Kubernetes:
  + Tim tanpa pengalaman Docker/Kubernetes membutuhkan pelatihan 2-4 minggu.
  + Solusi: Manfaatkan NEO Documentation dan workshop gratis.
* Manajemen Jaringan:  
  Konfigurasi Ingress Controller untuk HTTPS/TLS memerlukan sertifikat eksternal (e.g., Let’s Encrypt).
  + - Rekomendasi:  
      Gunakan NEO Life Pro untuk pemantauan otomatis dan alerting saat konfigurasi salah.

1. Skalabilitas Terbatas: Untuk aplikasi kompleks atau traffic tinggi, pertimbangkan migrasi ke arsitektur microservices di masa depan.

* Limitasi Arsitektur Monolitik:
  + Maksimal 50 Pod per cluster sebelum performa menurun.
  + Solusi: Desain ulang ke microservices jika traffic konsisten >1 juta pengguna/bulan.
* Kapan Harus Migrasi ke Microservices?
  + Saat butuh isolasi komponen (e.g., tim berbeda mengembangkan fitur terpisah).
  + Jika memerlukan scaling independen (e.g., service pembayaran butuh 10x lebih banyak resource dari service produk).
  + Contoh Transisi:  
    Startup SaaS beralih dari monolitik ke microservices di NEO Kubernetes setelah user aktif melebihi 50.000/bulan, dengan downtime hanya 15 menit.