Judul:
Laporan Tugas Akhir: Rancangan dan Implementasi Infrastruktur Cloud untuk Organisasi Digital

Nama Peserta:
Muhammad Avi Majid Kaaffah

Tanggal Pengumpulan:
2 Agustus 2025

Bootcamp / Kelas / Fasilitator:
Bootcamp Cloud Engineer CompTIA Cloud+ – Facilitator: Jasson dan Sapto

I. Pendahuluan

1.1. Latar Belakang Studi Kasus

PT Digital Nusantara adalah sebuah perusahaan rintisan (startup) yang bergerak dinamis di sektor retail digital, dengan fokus utama pada pengembangan dan pengoperasian platform toko online serta layanan e-commerce yang inovatif. Didukung oleh tim yang terdiri dari sekitar 75 hingga 100 karyawan, PT Digital Nusantara saat ini berada pada fase pertumbuhan yang pesat, di mana ketergantungan pada infrastruktur teknologi yang tangguh dan adaptif menjadi sangat krusial untuk mempertahankan momentum bisnis.

Saat ini, seluruh operasional perusahaan **bergantung sepenuhnya pada infrastruktur on-premise** yang dikelola secara mandiri. Meskipun telah mendukung bisnis hingga ke titik ini, model operasional ini kini menjadi penghambat utama pertumbuhan dan menciptakan tantangan bisnis yang kritis:

- **Keterbatasan Skalabilitas:** Infrastruktur fisik on-premise tidak memiliki elastisitas untuk menopang lonjakan trafik musiman (misalnya, saat Harbolnas atau kampanye 12.12). Hal ini menyebabkan performa situs yang lambat dan seringkali *downtime*, yang berujung pada hilangnya pendapatan dan penurunan kepercayaan pelanggan.
- **Hambatan Inovasi:** Proses pengadaan dan penyediaan server baru untuk lingkungan development atau fitur baru memakan waktu berminggu-minggu. Siklus pengembangan yang lambat ini membuat perusahaan kalah cepat dari kompetitor dalam merespons tren pasar.
- **Tingginya Biaya Operasional dan Modal:** Perusahaan harus menanggung biaya modal (CAPEX) yang besar untuk pembelian perangkat keras dan biaya operasional (OPEX) yang tinggi untuk pemeliharaan, listrik, dan pendinginan data center.

Dinamika pasar retail digital yang sangat kompetitif, ditambah dengan ekspektasi pelanggan yang terus meningkat, menuntut PT Digital Nusantara untuk bertransformasi. Oleh karena itu, perusahaan telah menetapkan tujuan strategis untuk melakukan migrasi dari lingkungan on-premise ke cloud. Langkah ini bukan hanya tentang pemindahan infrastruktur, melainkan sebuah langkah fundamental untuk membangun fondasi digital yang akan memberikan keunggulan kompetitif nyata. Fondasi ini akan memungkinkan **peningkatan kelincahan bisnis** untuk meluncurkan produk dalam hitungan jam, **keunggulan operasional** dengan

mengalihkan fokus tim dari pemeliharaan ke inovasi, **pengalaman pelanggan yang superior** melalui performa aplikasi yang andal, serta **efisiensi biaya strategis** dengan mengubah belanja modal (CAPEX) menjadi biaya operasional (OPEX) yang dapat dioptimalkan.

1.2. Tujuan Umum Proyek

Proyek ini bertujuan untuk merancang, mengamankan, dan mengelola infrastruktur cloud yang efisien, aman, dan tahan gangguan bagi PT Digital Nusantara. Secara spesifik, proyek ini mencakup lima area fokus utama yang akan dibahas dalam laporan ini:

- **Perencanaan Arsitektur Awal:** Menentukan layanan cloud (IaaS, PaaS) dan model deployment (Hybrid Cloud sebagai fase transisi) yang paling sesuai, serta merancang komponen dasar untuk compute, storage, dan jaringan.
- **Implementasi Keamanan:** Menyusun kebijakan akses (IAM), menerapkan kontrol keamanan jaringan, dan memastikan enkripsi data untuk melindungi aset digital perusahaan dan data pelanggan.
- **Automasi Operasional:** Mensimulasikan dan merancang proses provisioning serta deployment infrastruktur dan aplikasi secara otomatis untuk meningkatkan efisiensi, konsistensi, dan kecepatan rilis.
- **Monitoring dan Observabilitas:** Merancang skema monitoring metrik utama (kinerja, ketersediaan, keamanan) dan mengidentifikasi tools yang relevan untuk memastikan performa dan keandalan sistem secara proaktif.
- **Perencanaan Kelangsungan Bisnis:** Mengembangkan strategi backup dan Disaster Recovery (DR) yang komprehensif, termasuk penentuan target RTO/RPO, untuk memastikan ketahanan sistem terhadap berbagai skenario kegagalan.

1.3. Kerangka Modul

Laporan ini disusun berdasarkan materi dan konsep yang tercakup dalam **modul 1 hingga 12 dari kurikulum CompTIA Cloud+ v4**. Pendekatan dan rekomendasi teknis yang disajikan dalam laporan ini juga sangat merujuk pada praktik dan pemahaman yang diperoleh dari

berbagai **Lab CompTIA**, yang berfungsi sebagai referensi implementasi praktis dalam lingkungan *cloud* yang sesungguhnya.

II. Mini Project 1 – Perancangan Infrastruktur Cloud Dasar

2.1 Analisis Kebutuhan dan Strategi

Berdasarkan kondisi PT Digital Nusantara yang saat ini beroperasi penuh di lingkungan on-premise, strategi migrasi yang paling realistis dan minim risiko adalah sebagai berikut:

- **Model Layanan: Kombinasi IaaS & PaaS**
    - **IaaS (Infrastructure as a Service):** Dipilih untuk migrasi aplikasi inti (core application) dengan metode *Lift and Shift*. Pendekatan ini memungkinkan pemindahan cepat dengan perubahan kode minimal, memberikan *quick win* dan mengurangi kurva pembelajaran tim.
    - **PaaS (Platform as a Service):** Dipilih untuk modernisasi database dan layanan pendukung lainnya. Dengan mengalihkan tugas manajemen database (patching, backup, scaling) ke penyedia cloud, tim developer dapat fokus pada pengembangan fitur yang memberikan nilai bisnis.
- **Model Deployment: Hybrid Cloud (Sebagai Fase Transisi)**
    - Model ini dipilih sebagai langkah pertama yang paling logis untuk keluar dari ketergantungan penuh pada on-premise. Ini memungkinkan migrasi bertahap, memanfaatkan investasi on-premise yang masih berjalan untuk beban kerja internal, dan memberikan waktu bagi tim untuk beradaptasi dengan teknologi cloud sebelum transisi penuh.

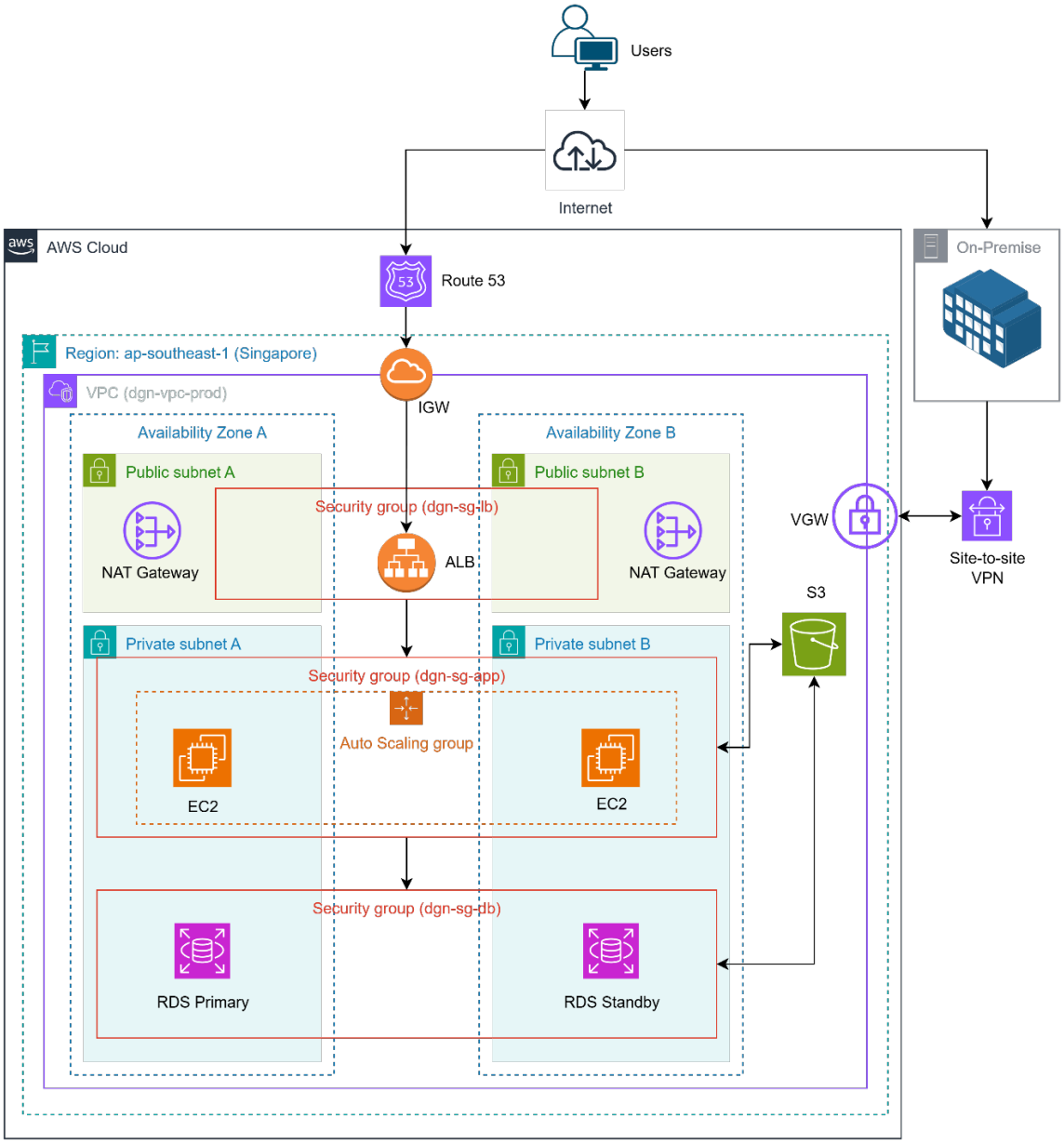
2.2 Desain Arsitektur Awal

Diagram berikut memvisualisasikan arsitektur hybrid yang direkomendasikan. Arsitektur ini dirancang dengan prinsip keamanan berlapis dan skalabilitas.

Penjelasan Detail Komponen Arsitektur:

- DNS (Domain Name System): Menggunakan layanan seperti AWS Route 53 atau Azure DNS. Ini adalah "pintu gerbang" utama yang menerjemahkan nama domain www.digitalnusantara.com ke alamat IP dari Load Balancer, sehingga dapat diakses oleh publik.

- **Koneksi Hybrid:** Menggunakan **Site-to-Site VPN** sebagai koneksi aman awal. Ini adalah solusi hemat biaya untuk menghubungkan data center on-premise dengan Virtual Private Cloud (VPC) dan memungkinkan komunikasi data yang terenkripsi.
- **Compute (IaaS):** Sebuah **Auto Scaling Group** akan mengelola sekelompok **Virtual Machines (VM)** yang berfungsi sebagai web/app server. Jika Load Balancer mendeteksi lonjakan trafik (misalnya, CPU di atas 70%), grup ini akan secara otomatis meluncurkan VM baru. Sebaliknya, jika trafik turun, VM yang tidak terpakai akan dimatikan untuk efisiensi biaya.



[Arsitektur Hybrid Cloud untuk PT Digital Nusantara]

- **Storage (PaaS & Object):**
    - **Managed Database (PaaS):** Data transaksional (pelanggan, pesanan) akan disimpan di layanan seperti AWS RDS atau Google Cloud SQL. Layanan ini menyediakan fitur ketersediaan tinggi (multi-AZ) dan backup otomatis, yang sangat sulit dan mahal untuk dicapai di lingkungan on-premise.
    - **Object Storage:** Aset statis seperti gambar produk, file CSS, dan video akan disimpan di layanan seperti AWS S3. Layanan ini sangat tahan lama, murah, dan dapat diintegrasikan dengan CDN untuk distribusi konten global.
- **Jaringan (VPC):** Jaringan virtual dibagi menjadi dua zona keamanan utama (subnet) untuk menerapkan prinsip keamanan berlapis (*defense-in-depth*):
    - **Public Subnet:** Zona ini adalah satu-satunya yang memiliki akses ke internet. Sesuai praktik terbaik keamanan, subnet ini hanya berisi **Application Load Balancer**. Fungsinya adalah sebagai gerbang tunggal yang menerima, menyaring, dan mendistribusikan lalu lintas.
    - **Private Subnet:** Zona ini sepenuhnya terisolasi dari internet. Di sinilah "mahkota permata" perusahaan berada:
        - **Application Server (EC2 Instances):** Server yang menjalankan kode aplikasi ditempatkan di sini untuk melindunginya dari akses langsung. Mereka hanya menerima lalu lintas yang aman dari Load Balancer.
        - **Database (Amazon RDS):** Berisi data paling sensitif dan hanya dapat diakses oleh Application Server.

2.3 Simulasi Provisioning

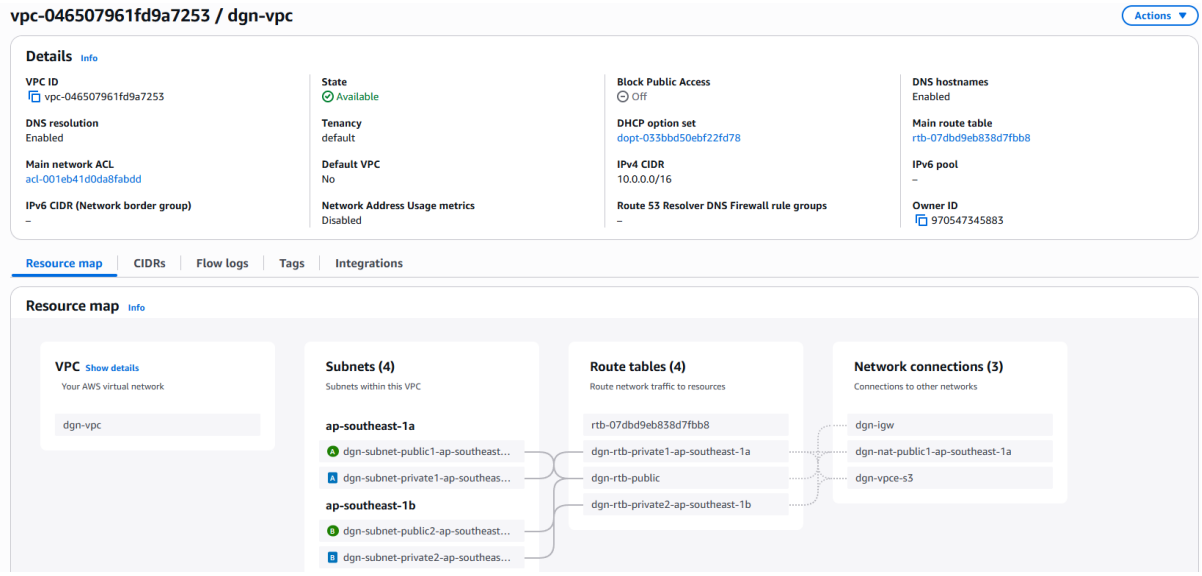
Berikut adalah panduan langkah demi langkah untuk melakukan provisioning sumber daya di konsol AWS dan Azure sesuai dengan arsitektur yang telah dirancang. Pemahaman proses ini penting sebagai fondasi untuk implementasi dan automasi di masa depan.

A. Platform Amazon Web Services (AWS)

Langkah 1: Membuat Jaringan (VPC) dan Subnet

1. Buka AWS Management Console, navigasi ke layanan **VPC**.

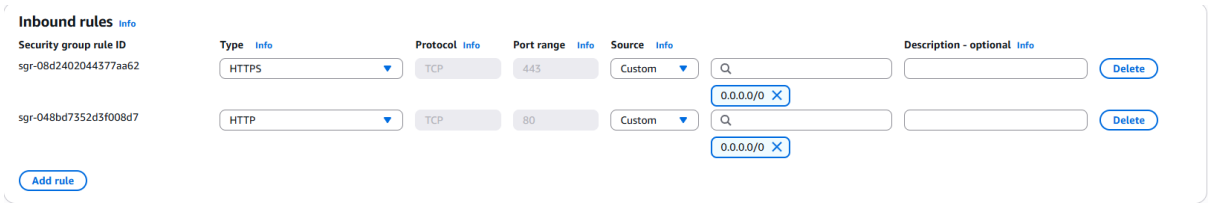
2. Klik **"Create VPC"** dan pilih opsi **"VPC and more"** untuk menggunakan wizard.
3. **Konfigurasi Dasar:**
    - Beri nama VPC (misal: dgn-vpc).
    - Tentukan blok alamat IPv4 CIDR (misal: 10.0.0.0/16).
4. **Konfigurasi Subnet:**
    - **Number of Availability Zones (AZs):** Pilih 2 untuk ketersediaan tinggi.
    - **Number of public subnets:** Pilih 2. Beri nama (dgn-subnet-public1-ap-southeast-1a, dgn-subnet-public2-ap-southeast-1b).
    - **Number of private subnets:** Pilih 2. Beri nama (misal: dgn-subnet-private1-ap-southeast-1a, dgn-subnet-private2-ap-southeast-1b).
5. **Konfigurasi Gateway:**
    - **NAT gateways:** Pilih In 1 AZ untuk efisiensi biaya awal. NAT Gateway diperlukan agar server di private subnet dapat mengakses internet untuk pembaruan perangkat lunak, tanpa bisa diakses dari internet.
    - **VPC endpoints:** Pilih S3 Gateway.
6. Klik **"Create VPC"**. Wizard ini akan secara otomatis membuat VPC, Subnet, Internet Gateway, NAT Gateway, dan Route Tables yang sesuai.



[Created VPC]

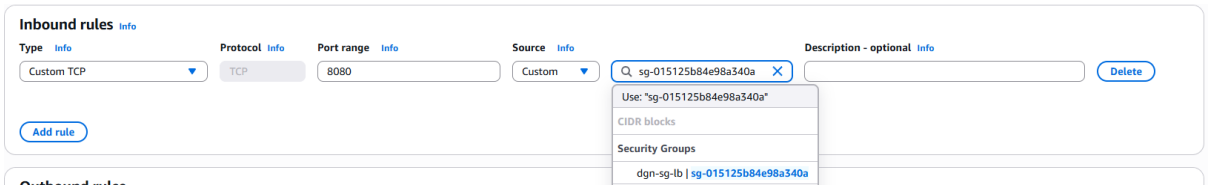
Langkah 2: Konfigurasi Keamanan Jaringan (Security Groups)

1. Di dashboard VPC, navigasi ke **Security Groups**.
2. Buat **3 Security Groups** dengan memastikan memilih **dgn-vpc** dan aturan yang spesifik:
    - **Nama:** dgn-sg-lb (Untuk Load Balancer)
        - **Inbound Rules:** Tambahkan aturan untuk HTTP (Port 80) dan HTTPS (Port 443) dengan **Source:** Anywhere-IPv4 (0.0.0.0/0).



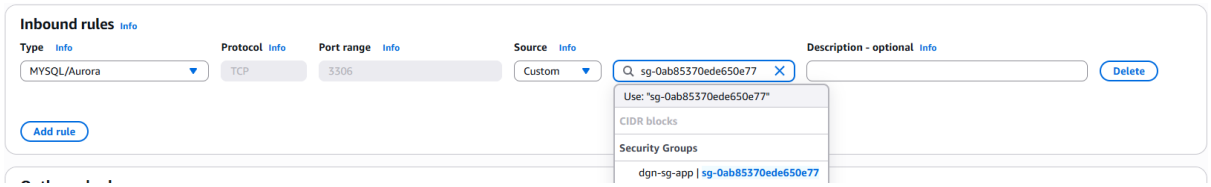
[Inbound Rules dgn-sg-lb]

- **Nama:** dgn-sg-app (Untuk Application Server)
    - **Inbound Rules:** Tambahkan aturan untuk port aplikasi Anda (misal: Custom TCP, Port 8080) dengan **Source:** dgn-sg-lb (pilih dari daftar security group). Ini adalah kunci keamanan 3-tier.



[Inbound Rules dgn-sg-app]

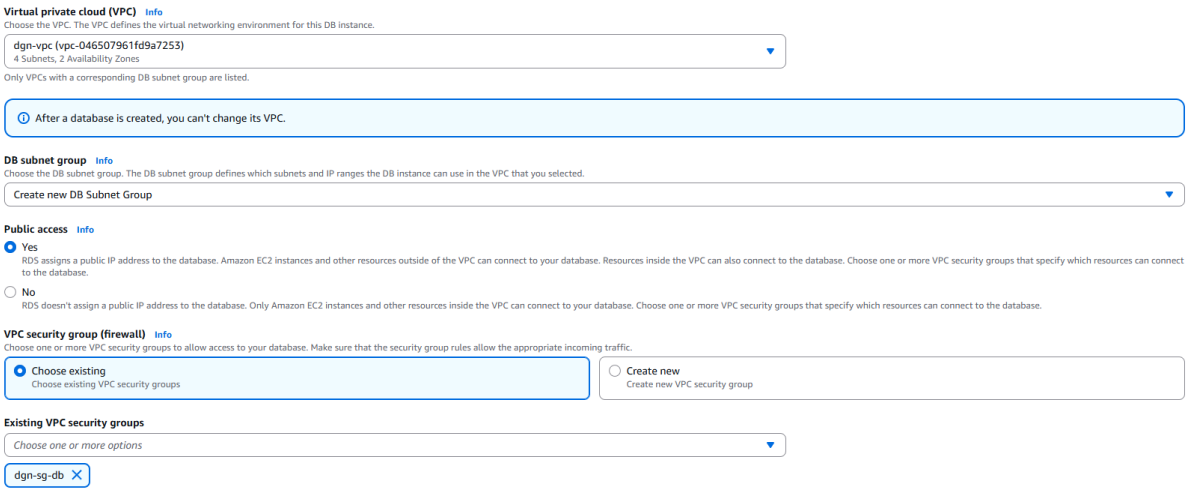
- **Nama:** dgn-sg-db (Untuk Database)
    - **Inbound Rules:** Tambahkan aturan untuk port database (misal: MySQL/Aurora, Port 3306) dengan **Source:** dgn-sg-app.



[Inbound Rules dgn-sg-db]

Langkah 3: Provisioning Database (RDS)

1. Navigasi ke layanan **RDS**, klik **"Create database"**.
2. Pilih "Standard Create" dan engine **MySQL**.
3. Pilih template **"Production"** dan aktifkan **Multi-AZ deployment** untuk ketersediaan tinggi.
4. **Konfigurasi Konektivitas:**
    - **Virtual Private Cloud (VPC):** Pilih dgn-vpc.
    - **DB subnet group:** RDS akan otomatis membuat grup subnet baru yang berisi **private subnet** Anda. Pastikan ini benar.
    - **Public access:** Pilih **"No"**.
    - **VPC security group:** Hapus yang default dan pilih dgn-sg-db.
5. Aktifkan **Automated backups** dan atur periode retensi.
6. Klik **"Create database"**.

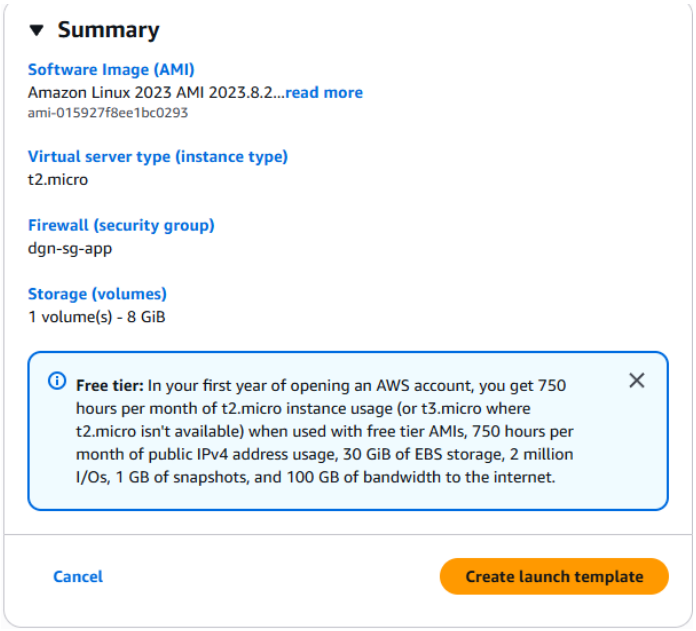


[Membuat instance RDS]

Langkah 4: Provisioning Application Server (EC2 Auto Scaling Group & ALB)

1. **Buat Launch Template:** Navigasi ke EC2 > Launch Templates.
    - Beri nama (misal: dgn-app-template).

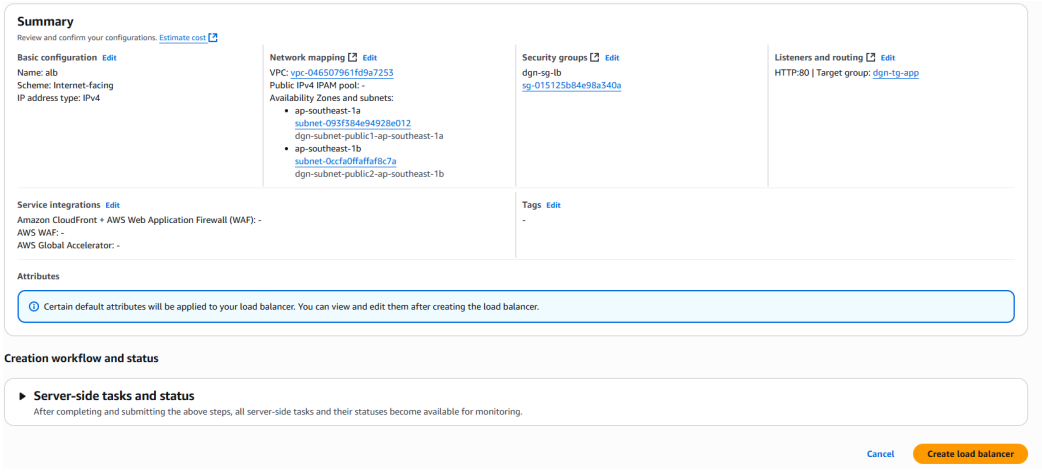
- Pilih AMI (misal: Amazon Linux 2).
- Pilih Instance Type (misal: t3.micro).
- Pilih Key Pair untuk akses SSH (jika diperlukan).
- Di bagian **Network settings**, pilih dgn-sg-app.



[Buat Launch Template]

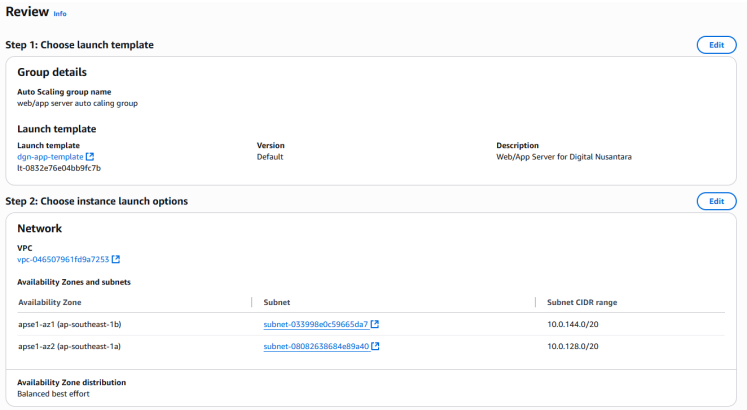
2. **Buat Target Group:** Navigasi ke EC2 > Target Groups.
   - Pilih target type **"Instances"**.
   - Beri nama (dgn-tg-app).
   - Pilih dgn-vpc dan protokol port HTTP:8080.
   - Konfigurasikan health check (misal: path /health).
3. **Buat Application Load Balancer (ALB):**
   - Navigasi ke EC2 > Load Balancers, klik **"Create Load Balancer"**.
   - Pilih **Application Load Balancer**.
   - **Konfigurasi Dasar:** Beri nama, pilih skema **"Internet-facing"**.
   - **Network mapping:** Pilih dgn-vpc dan centang **kedua public subnet**.

- **Security groups:** Hapus yang default dan pilih dgn-sg-lb.
- **Listeners and routing:**
  - Buat listener untuk **HTTP:80**. Atur *default action* untuk meneruskan (forward) trafik ke Target Group dgn-tg-app. Untuk tujuan latihan, kita hanya akan menggunakan HTTP. Listener HTTPS dapat ditambahkan kemudian setelah sertifikat SSL tersedia.

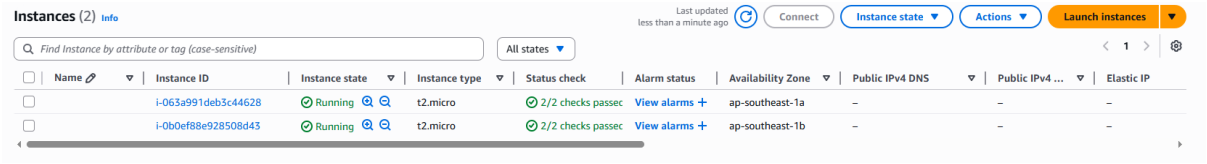


[Buat Application Load Balancer]

4. **Buat Auto Scaling Group:**
   - Navigasi ke EC2 > Auto Scaling Groups, klik **"Create Auto Scaling group"**.
   - Pilih Launch Template yang dibuat pada langkah 1.
   - Di bagian **Network**, pilih dgn-vpc dan centang **kedua private subnet**.
   - Di bagian **Load balancing**, pilih "Attach to an existing load balancer" dan pilih Target Group dgn-tg-app.
   - Atur kebijakan scaling (misal: Target tracking scaling policy berdasarkan CPU Utilization 70%) dan jumlah instance (Min: 2, Desired: 2, Max: 4).



[Buat Auto Scaling Group]



[Instance telah berhasil di-provision]

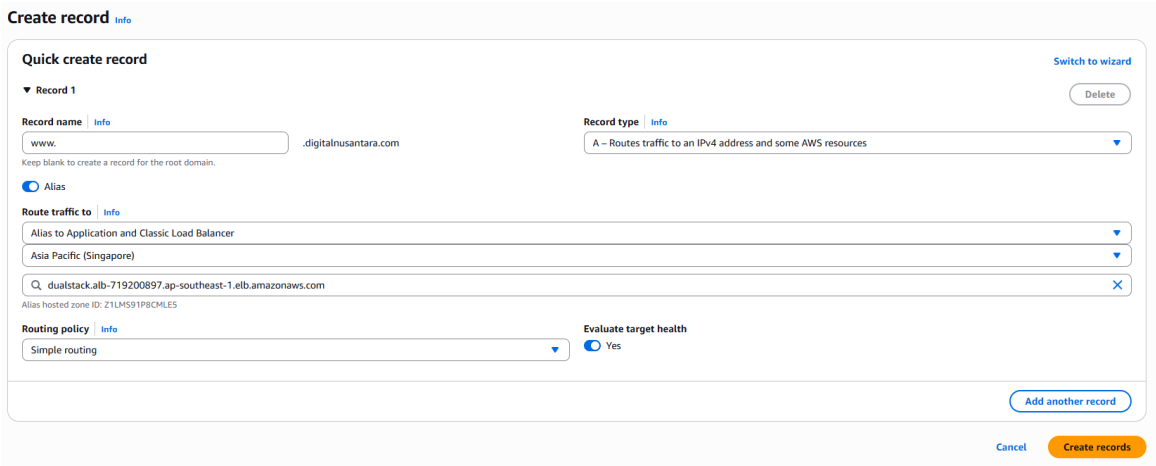
Langkah 5: Provisioning Koneksi Hybrid (Site-to-Site VPN)

1. Di dashboard VPC, navigasi ke **"Customer Gateways"** dan klik **"Create Customer Gateway"**. Masukkan nama dan alamat IP publik dari router/firewall on-premise Anda.
2. Navigasi ke **"Virtual Private Gateways"** dan klik **"Create Virtual Private Gateway"**. Beri nama, lalu setelah dibuat, pilih "Attach to VPC" dan kaitkan dengan dgn-vpc-prod.
3. Navigasi ke **"Site-to-Site VPN Connections"** dan klik **"Create VPN Connection"**.
    - Konfigurasi: Beri nama, pilih Virtual Private Gateway dan Customer Gateway yang telah dibuat.
    - Pilih "Static" dan masukkan blok alamat IP jaringan on-premise Anda.
4. Klik **"Create VPN Connection"**. Setelah koneksi tersedia, unduh file konfigurasi (Download Configuration) yang berisi *pre-shared keys* dan detail endpoint untuk diimplementasikan di perangkat on-premise.

Langkah 6: Konfigurasi DNS (Route 53)

1. Navigasi ke layanan **Route 53** dan masuk ke **"Hosted zones"**.

2. Jika belum ada, buat sebuah *hosted zone* untuk domain digitalnusantara.com.
3. Di dalam *hosted zone* tersebut, klik **"Create record"**.
4. Konfigurasi record untuk www.digitalnusantara.com:
    - **Record type:** A
    - **Value:** Aktifkan toggle **"Alias"**.
    - **Route traffic to:** Pilih "Alias to Application and Classic Load Balancer", pilih region, lalu pilih ALB yang telah dibuat pada Langkah 4.
5. Klik **"Create records"**.



[Buat A record dengan Alias ke ALB di Route 53]

B. Platform Microsoft Azure

Langkah 1: Membuat Jaringan (Virtual Network)

1. Buka Azure Portal, cari dan pilih **"Virtual networks"**.
2. Klik **"+ Create"**.
3. Konfigurasi: Pilih Resource Group, beri nama VNet (misal: dgn-vnet-prod), pilih Region.
4. Di tab **"IP Addresses"**, tentukan ruang alamat (misal: 10.0.0.0/16).
5. Buat dua subnet: public-subnet dan private-subnet.
6. Klik **"Review + create"**, lalu **"Create"**.

Langkah 2: Konfigurasi Keamanan Jaringan (Network Security Groups)

1. Cari dan pilih **"Network security groups (NSG)"**.
2. Buat **3 NSG** dengan aturan yang spesifik:
    - **Nama:** dgn-nsg-lb
        - **Inbound security rules:** Tambahkan aturan Allow dengan **Priority** 100, **Source:** Internet, **Destination port ranges:** 80,443, **Protocol:** TCP.
    - **Nama:** dgn-nsg-app
        - **Inbound security rules:** Tambahkan aturan Allow dengan **Priority** 100, **Source:** VirtualNetwork, **Destination port ranges:** 8080, **Protocol:** TCP.
    - **Nama:** dgn-nsg-db
        - **Inbound security rules:** Tambahkan aturan Allow dengan **Priority** 100, **Source:** VirtualNetwork (bisa diperketat dengan Application Security Group), **Destination port ranges:** 3306, **Protocol:** TCP.
3. Kaitkan dgn-nsg-lb ke **public-subnet**. Kaitkan dgn-nsg-app dan dgn-nsg-db ke **private-subnet**.

Langkah 3: Provisioning Database (Azure Database for MySQL)

1. Pilih **Flexible Server**.
2. Di tab **"Networking"**, pilih **"Private access (VNet Integration)"**. Pilih dgn-vnet-prod dan **private-subnet**.
3. Aktifkan **Geo-redundant backup** untuk ketahanan bencana.

Langkah 4: Provisioning Application Server (Virtual Machine Scale Sets & App Gateway)

1. **Buat Application Gateway (WAF V2 SKU):**
    - Buat subnet baru khusus untuk Application Gateway (misal: app-gateway-subnet).

- Konfigurasikan **Frontend** dengan IP Publik baru.
- Buat **Backend pool** (awalnya kosong, akan diisi oleh VMSS nanti).
- Konfigurasikan **Routing rules** yang menghubungkan listener HTTPS:443 ke backend pool.

2. **Buat Virtual Machine Scale Set (VMSS):**
   - Di tab **"Networking"**, pilih dgn-vnet-prod dan **pilih private-subnet**.
   - Di bagian **Load balancing**, pilih "Application Gateway" dan pilih App Gateway yang dibuat pada langkah 1. VMSS akan secara otomatis ditambahkan ke backend pool.
   - Di tab **"Scaling"**, atur jumlah instance awal dan kebijakan autoscaling.

Langkah 5: Provisioning Koneksi Hybrid (Site-to-Site VPN)

1. Cari dan pilih **"Virtual network gateways"** dan klik **"+ Create"**.
   - Konfigurasi: Beri nama, pilih dgn-vnet-prod, buat alamat IP Publik baru untuk gateway ini.
2. Cari dan pilih **"Local network gateways"** dan klik **"+ Create"**. Ini merepresentasikan jaringan on-premise Anda. Masukkan alamat IP publik dari perangkat VPN on-premise dan blok alamat IP jaringan lokal Anda.
3. Kembali ke Virtual Network Gateway yang telah dibuat, navigasi ke **"Connections"** dan klik **"+ Add"**.
   - Konfigurasi: Beri nama koneksi, pilih tipe "Site-to-site (IPsec)", pilih Local Network Gateway yang tadi dibuat.
   - Buat sebuah *shared key* (PSK) yang akan digunakan di kedua sisi koneksi.
4. Klik **"OK"**. Konfigurasi ini kemudian harus diimplementasikan pada perangkat VPN on-premise menggunakan IP publik dari Virtual Network Gateway dan *shared key* yang telah dibuat.

Langkah 6: Konfigurasi DNS (Azure DNS)

1. Cari dan pilih **"DNS zones"** dan klik **"+ Create"**.

2. Buat zona DNS baru untuk domain digitalnusantara.com.
3. Masuk ke zona DNS yang baru dibuat, klik **"+ Record set"**.
4. Konfigurasi record untuk www:
    - **Type:** A
    - **Alias record set:** Pilih "Yes".
    - **Alias type:** Pilih "Azure resource".
    - **Azure resource:** Pilih Application Gateway yang telah dibuat pada Langkah 4.
5. Klik **"OK"**.

2.4. Risiko & Troubleshooting Awal

Untuk memastikan proses transisi yang mulus dan meminimalkan gangguan pada operasional bisnis, strategi migrasi awal akan dilakukan secara bertahap. Kami merekomendasikan untuk memulai dengan **proyek percontohan (pilot project)**, yaitu memigrasikan beban kerja yang tidak kritis terlebih dahulu. Contohnya bisa berupa lingkungan staging, aplikasi internal, atau situs web statis.

Tujuan dari fase percontohan ini adalah untuk:

- **Memvalidasi Konektivitas:** Menguji dan memastikan koneksi Site-to-Site VPN antara on-premise dan cloud berjalan stabil dan memiliki performa yang memadai.
- **Menguji Konfigurasi Keamanan:** Memastikan semua Security Group, kebijakan IAM, dan kontrol jaringan berfungsi seperti yang dirancang.
- **Membangun Keahlian Tim:** Memberikan pengalaman langsung kepada tim teknis dalam mengelola sumber daya di lingkungan cloud.

Meskipun pendekatan bertahap ini dirancang untuk mengurangi risiko, setiap proyek migrasi tetap memiliki potensi tantangan. Berikut adalah identifikasi risiko utama yang mungkin dihadapi selama fase awal, beserta solusi dan rencana mitigasinya.

Risiko Potensial	Solusi / Mitigasi
Cost Overrun (Pembengkakan Biaya)	Aktifkan *billing alerts* dan buat *budget* di konsol cloud. Lakukan *right-sizing* sumber daya secara berkala. Manfaatkan diskon dengan komitmen (Reserved Instances/Savings Plans) untuk beban kerja yang dapat diprediksi.
Skills Gap (Kesenjangan Keahlian Tim)	Adakan pelatihan dan program sertifikasi cloud. Mulai dengan layanan PaaS yang lebih sederhana untuk mengurangi kompleksitas. Pertimbangkan untuk menggunakan jasa konsultan pada fase awal.
Latensi Jaringan (On-Premise ke Cloud)	Mulai dengan VPN. Jika performa tidak memadai untuk aplikasi kritis, lakukan analisis dan rencanakan anggaran untuk upgrade ke koneksi khusus (*Direct Connect/Interconnect*).
Kompleksitas Sinkronisasi Data	Gunakan layanan migrasi data yang disediakan cloud provider (misal: AWS DMS). Rencanakan proses *cutover* dengan cermat, lakukan uji coba, dan jadwalkan migrasi final pada jam sepi.

III. Mini Project 2 – Automasi, Keamanan, dan Ketahanan Cloud

3.1 Desain Automasi dan Praktik DevOps

Untuk mencapai kelincahan, konsistensi, dan kecepatan yang dibutuhkan di pasar kompetitif, PT Digital Nusantara harus mengadopsi praktik automasi dan DevOps. Ini mengubah cara infrastruktur dan aplikasi dikelola, dari proses manual menjadi proses berbasis kode yang otomatis.

A. Infrastructure as Code (IaC)

Fondasi dari automasi adalah memperlakukan infrastruktur sebagai kode. Alih-alih membuat sumber daya dengan mengklik di konsol, tim akan mendefinisikan seluruh arsitektur (VPC, VM, database, dll.) dalam file teks menggunakan alat seperti **Terraform** dan **CloudFormation**.

- **Manfaat Utama:**
    - **Konsistensi:** Menghilangkan masalah "works on my machine" dengan memastikan lingkungan development, staging, dan produksi identik.
    - **Versioning:** Setiap perubahan pada infrastruktur dilacak melalui Git, memungkinkan audit, kolaborasi, dan rollback yang mudah.
    - **Kecepatan:** Membangun ulang seluruh lingkungan dari awal dapat dilakukan dalam hitungan menit, bukan hari.
- **Simulasi dengan AWS CloudFormation:** CloudFormation adalah layanan native dari AWS yang memungkinkan Anda mendefinisikan infrastruktur dalam format file YAML atau JSON. Contoh template file YAML yang lengkap untuk membangun fondasi jaringan dan keamanan sesuai arsitektur yang dirancang dimuat **pada bab VI. Lampiran** laporan ini.

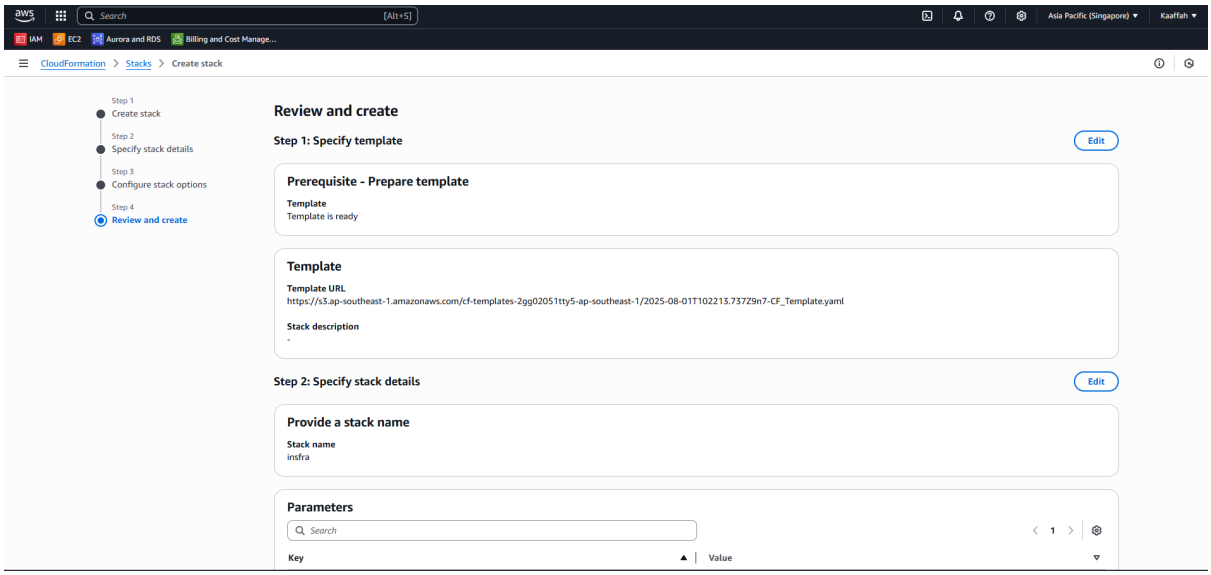
  Cara Kerja:

  a) Tim menyimpan file YAML di atas dalam repositori Git.

  b) Melalui AWS Console (atau command line), tim membuat sebuah "Stack" baru dan mengunggah template ini.

c) CloudFormation akan membaca template, memahami semua sumber daya yang perlu dibuat (VPC, Subnet, Security Groups) dan dependensinya, lalu secara otomatis melakukan provisioning dalam urutan yang benar.

d) Setiap pembaruan pada infrastruktur dilakukan dengan memperbarui file template, bukan dengan mengubah sumber daya secara manual.



[CloudFormation]

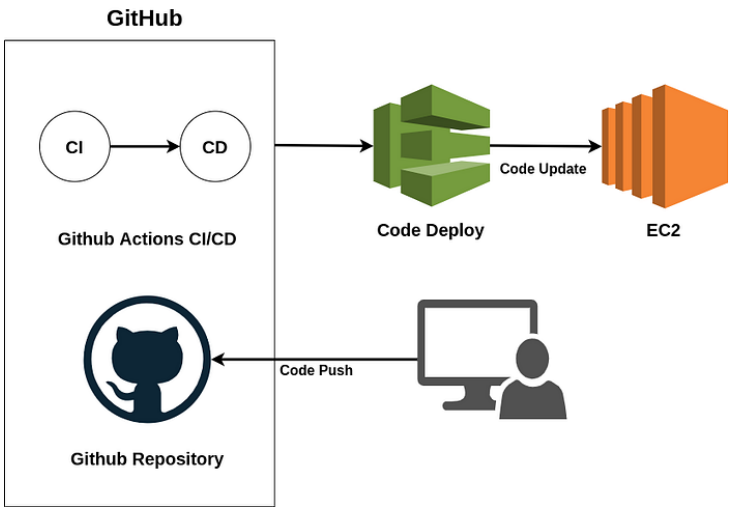
B. Configuration Management

Setelah infrastruktur di-provisioning oleh IaC, server-server tersebut masih perlu dikonfigurasi (misalnya, menginstal web server, menerapkan patch keamanan, atau mengatur file konfigurasi). Untuk ini, digunakan alat *Configuration Management* seperti **Ansible**.

- **Peran Ansible:**
    - **Automasi Konfigurasi:** Menjalankan *playbook* (skrip) untuk menginstal dan mengkonfigurasi perangkat lunak secara konsisten di semua server.
    - **Idempotent:** Ansible memastikan bahwa server selalu berada dalam keadaan yang diinginkan. Menjalankan playbook yang sama berulang kali tidak akan menyebabkan perubahan jika server sudah dalam kondisi yang benar.
    - **Orkestrasi:** Dapat digunakan untuk melakukan deployment aplikasi atau tugas-tugas operasional yang kompleks secara berurutan.

C. CI/CD Pipeline

CI/CD (Continuous Integration/Continuous Deployment) adalah "mesin" yang mengorkestrasi seluruh proses automasi, dari kode hingga produksi.



[CI/CD Github Actions dan AWS]

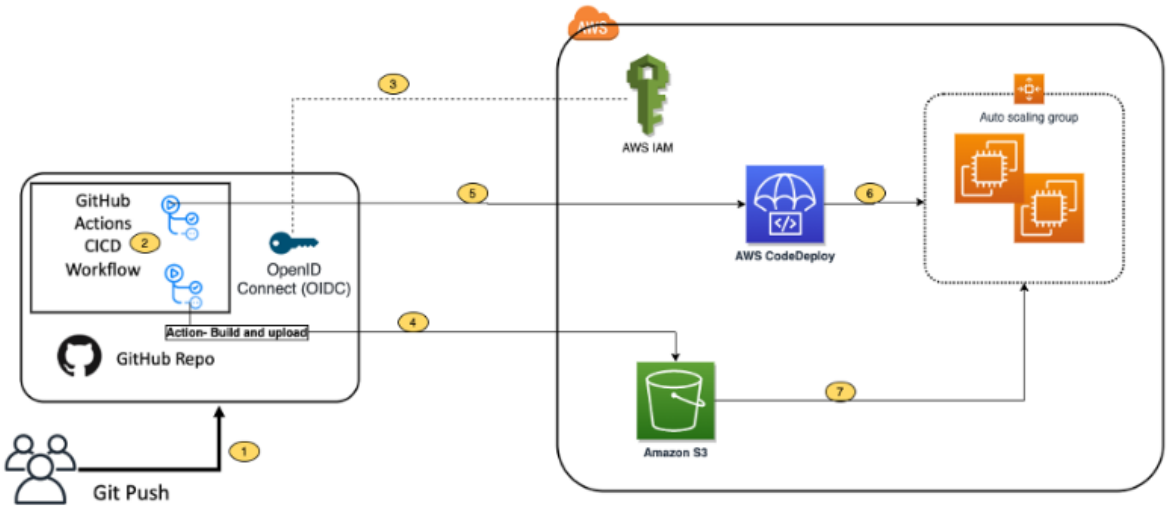
Alur Kerja Otomatis:

1. **Commit:** Developer menyimpan perubahan (kode aplikasi atau IaC) ke repositori Git.
2. **Build & Test (CI):** Aksi ini memicu pipeline (misalnya, Jenkins atau GitLab CI). Kode di-build, dan tes otomatis dijalankan.
3. **Deploy (CD):** Jika tes berhasil, pipeline akan:
    - Menjalankan **Terraform** untuk menyiapkan atau memperbarui infrastruktur.
    - Menjalankan **Ansible** untuk mengkonfigurasi server yang baru dibuat.
    - Men-deploy versi aplikasi terbaru ke server tersebut.

D. Simulasi Alur Kerja Otomatis (End-to-End)

Untuk memberikan gambaran yang lebih jelas, berikut adalah simulasi alur kerja rilis fitur baru menggunakan **GitHub Actions** yang terintegrasi dengan layanan AWS.

- **Skenario:** Seorang developer perlu merilis pembaruan pada halaman depan (frontend) aplikasi e-commerce.



[Simulasi alur kerja rilis fitur baru]

1. **Commit & Push:** Developer melakukan perubahan pada kode aplikasi di laptopnya, lalu melakukan git commit dan git push ke repositori di GitHub.
2. **Trigger GitHub Actions:** Aksi push ini secara otomatis (atau manual, tergantung konfigurasi) memicu *workflow* CI/CD yang telah didefinisikan dalam file YAML di dalam repositori.
3. **Autentikasi Aman ke AWS:** GitHub Actions menggunakan **OpenID Connect (OIDC)** untuk mendapatkan kredensial sementara dari **AWS IAM**. Ini adalah metode yang aman karena tidak perlu menyimpan *access key* jangka panjang di GitHub.
4. **Build & Upload Artefak:** *Workflow* di GitHub Actions akan menjalankan serangkaian perintah:
    - Menginstal dependensi aplikasi.
    - Menjalankan tes (unit test, linting).
    - Jika tes berhasil, kode aplikasi akan di-bundle menjadi sebuah artefak deployment (misalnya, file .zip).
    - Artefak ini kemudian diunggah ke bucket **Amazon S3** yang telah disiapkan khusus untuk menyimpan artefak.
5. **Panggil AWS CodeDeploy:** Setelah artefak berhasil diunggah, langkah terakhir di GitHub Actions adalah membuat panggilan API untuk memulai deployment baru di **AWS CodeDeploy**.

6. **Trigger Deployment Group:** CodeDeploy menerima perintah dan memulai proses deployment ke *Deployment Group* yang telah dikonfigurasi. *Deployment Group* ini menargetkan **Auto Scaling Group** yang menjalankan EC2 instance aplikasi kita.
7. **Deploy ke EC2 Instances:**
    - Agen CodeDeploy yang terpasang di setiap EC2 instance akan diberi tahu tentang adanya deployment baru.
    - Agen akan mengunduh artefak aplikasi dari bucket Amazon S3.
    - Agen akan menjalankan serangkaian skrip (didefinisikan dalam file appspec.yml) untuk menghentikan aplikasi versi lama, memasang versi baru, dan memulai kembali aplikasi. CodeDeploy akan melakukan ini secara bertahap (misalnya, satu per satu atau setengah-setengah) untuk memastikan tidak ada *downtime* (zero-downtime deployment).

Alur kerja ini memastikan proses rilis yang cepat, andal, dan sepenuhnya otomatis, memungkinkan PT Digital Nusantara untuk berinovasi dengan kecepatan tinggi.

3.2 IAM dan Keamanan Dasar

Fondasi keamanan di cloud dimulai dengan manajemen identitas dan akses yang ketat. Penerapkan **Principle of Least Privilege** sangat direkomendasikan, di mana setiap pengguna atau layanan hanya diberikan hak akses minimum yang mutlak diperlukan untuk melakukan tugasnya.

A. Struktur Peran dan Kebijakan Akses

Alih-alih memberikan izin langsung ke setiap pengguna, praktik terbaik adalah mengelola izin melalui **Grup** dan **Peran (Roles)**. Diagram berikut mengilustrasikan konsep ini:

Penjelasan Alur:

1. **Pengguna (Users):** Setiap individu (misalnya, Budi dari tim developer) memiliki akun pengguna sendiri.
2. **Grup (Groups):** Pengguna dengan fungsi pekerjaan yang sama dikelompokkan bersama (misalnya, Grup "Developers").

3. **Peran (Roles) & Kebijakan (Policies):** Izin tidak diberikan langsung ke grup. Sebaliknya, sebuah **Peran** (misalnya, DeveloperRole) dibuat. **Kebijakan Akses** (misalnya, DeveloperAccessPolicy) yang berisi daftar izin spesifik (seperti "boleh membaca dari S3" atau "boleh memulai ulang EC2 di lingkungan dev") dilekatkan pada peran tersebut.
4. **Asosiasi:** Grup "Developers" kemudian diberi izin untuk mengambil (assume) DeveloperRole. Dengan cara ini, semua izin terpusat pada peran, membuatnya lebih mudah dikelola dan diaudit.

Berikut adalah susunan peran awal yang direkomendasikan:

Role	Deskripsi Tugas	Contoh Hak Akses
CloudAdmin	Mengelola seluruh infrastruktur (create, update, delete).	Akses Penuh (Allow: *) dengan proteksi Multi-Factor Authentication (MFA) yang diwajibkan.
Developer	Akses ke repository kode, pipeline CI/CD, dan sumber daya di lingkungan development/staging.	Akses terbatas (hanya-baca) ke sumber daya produksi untuk tujuan debugging.
SupportEngineer	Memantau dan mendiagnosis masalah pada aplikasi dan infrastruktur produksi.	Akses Hanya-Baca (Read-Only) ke log, metrik, dan dashboard pemantauan produksi.

Implementasi IAM *Policy*

Berikut adalah contoh *policy* dalam format JSON yang dapat dilekatkan pada setiap peran untuk menerapkan izin secara spesifik.

a) ***Policy* untuk CloudAdmin:** *Best practice* untuk peran ini adalah dengan menggunakan *policy* terkelola AWS yang sudah ada.

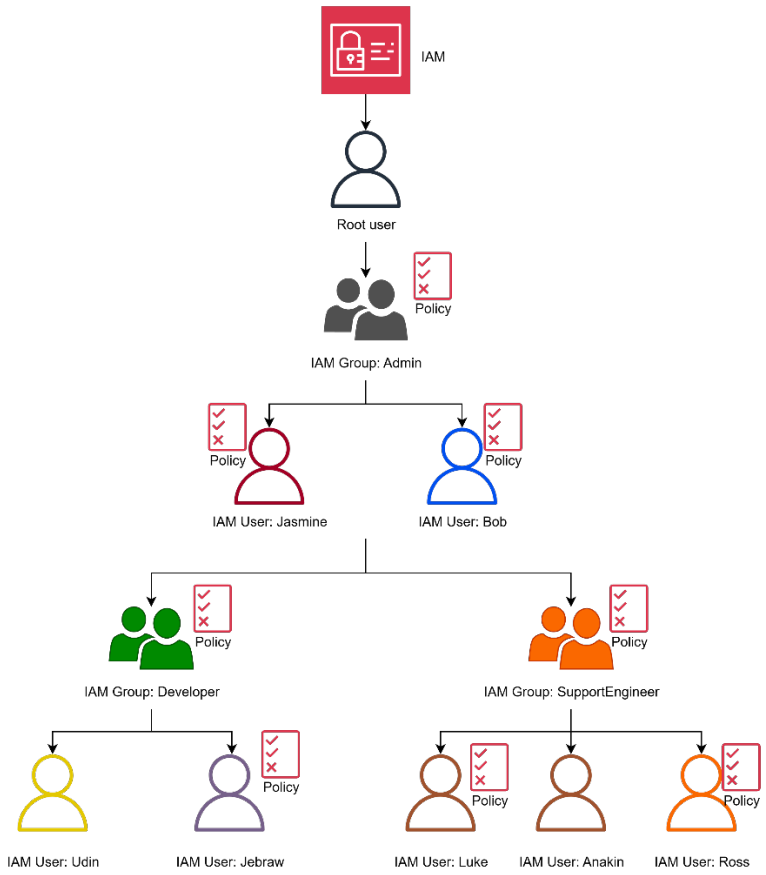
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

b) ***Policy* untuk Developer:** *Policy* ini memberikan akses penuh ke layanan CI/CD dan akses hanya-baca ke sumber daya produksi untuk troubleshooting.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCICDAccess",
            "Effect": "Allow",
            "Action": [
                "codecommit:*",
                "codebuild:*",
                "codedeploy:*",
                "codepipeline:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyProd",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "rds:Describe*",
                "logs:Get*",
                "logs:DescribeLogStreams"
            ],
            "Resource": "*"
        }
    ]
}
```

c) ***Policy* untuk SupportEngineer:** *Policy* ini secara ketat memberikan akses hanya-baca ke layanan yang relevan untuk pemantauan dan diagnosis masalah.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMonitoringAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Get*",
                "cloudwatch:Describe*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "elasticloadbalancing:Describe*",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```



[Visual IAM]

B. Kontrol Keamanan Jaringan & Data:

- **Web Application Firewall (WAF):** Ditempatkan di depan Load Balancer untuk memfilter lalu lintas berbahaya seperti SQL Injection, Cross-Site Scripting (XSS), dan serangan bot otomatis.
- **Enkripsi Data:**
    - **In-Transit:** Wajib menggunakan protokol HTTPS/TLS di semua titik komunikasi, baik dari klien ke cloud maupun antar layanan di dalam cloud.
    - **At-Rest:** Mengaktifkan enkripsi di semua layanan penyimpanan (disk VM, database, object storage) menggunakan kunci yang dikelola oleh layanan terpusat seperti AWS Key Management Service (KMS).
- **Layanan Audit dan Pencatatan (Wajib Aktif):** Layanan seperti **AWS CloudTrail** atau **Azure Monitor (Activity Log)** harus diaktifkan sejak awal. Layanan ini mencatat setiap panggilan API yang terjadi di akun ("siapa melakukan apa, dan kapan"). Log ini adalah fondasi yang tak ternilai untuk investigasi keamanan, audit kepatuhan, dan troubleshooting.

3.3 Monitoring dan Observabilitas

Dalam lingkungan cloud yang dinamis, pendekatan monitoring tradisional tidak lagi cukup. Kita harus beralih ke strategi **Observabilitas** yang proaktif, yang memungkinkan kita tidak hanya mengetahui *apa* yang salah, tetapi juga *mengapa* itu salah. Ini dicapai melalui tiga pilar utama:

- **Metrik (Metrics):** Data numerik yang dikumpulkan dari waktu ke waktu (misalnya, penggunaan CPU, latensi permintaan, jumlah error). Metrik sangat baik untuk memantau tren dan memicu peringatan (alert).
    - **Alat Utama: Amazon CloudWatch Metrics**. Layanan ini secara otomatis mengumpulkan metrik dari hampir semua layanan AWS.
- **Log:** Catatan berbasis teks dari setiap peristiwa yang terjadi di aplikasi atau infrastruktur (misalnya, log akses dari web server, log error dari aplikasi, log audit dari CloudTrail). Log sangat penting untuk *debugging* dan analisis forensik.

    - **Alat Utama: Amazon CloudWatch Logs**. Tim dapat mengirim semua log aplikasi dan sistem ke layanan ini untuk penyimpanan, pencarian, dan analisis terpusat.

- **Jejak (Traces):** Memberikan gambaran end-to-end dari sebuah permintaan saat bergerak melalui berbagai layanan dalam arsitektur. Misalnya, sebuah jejak dapat menunjukkan berapa lama waktu yang dihabiskan di Load Balancer, di Application Server, dan di Database untuk satu permintaan pengguna. Ini sangat kuat untuk mengidentifikasi *bottleneck* performa.
    - **Alat Utama: AWS X-Ray**.

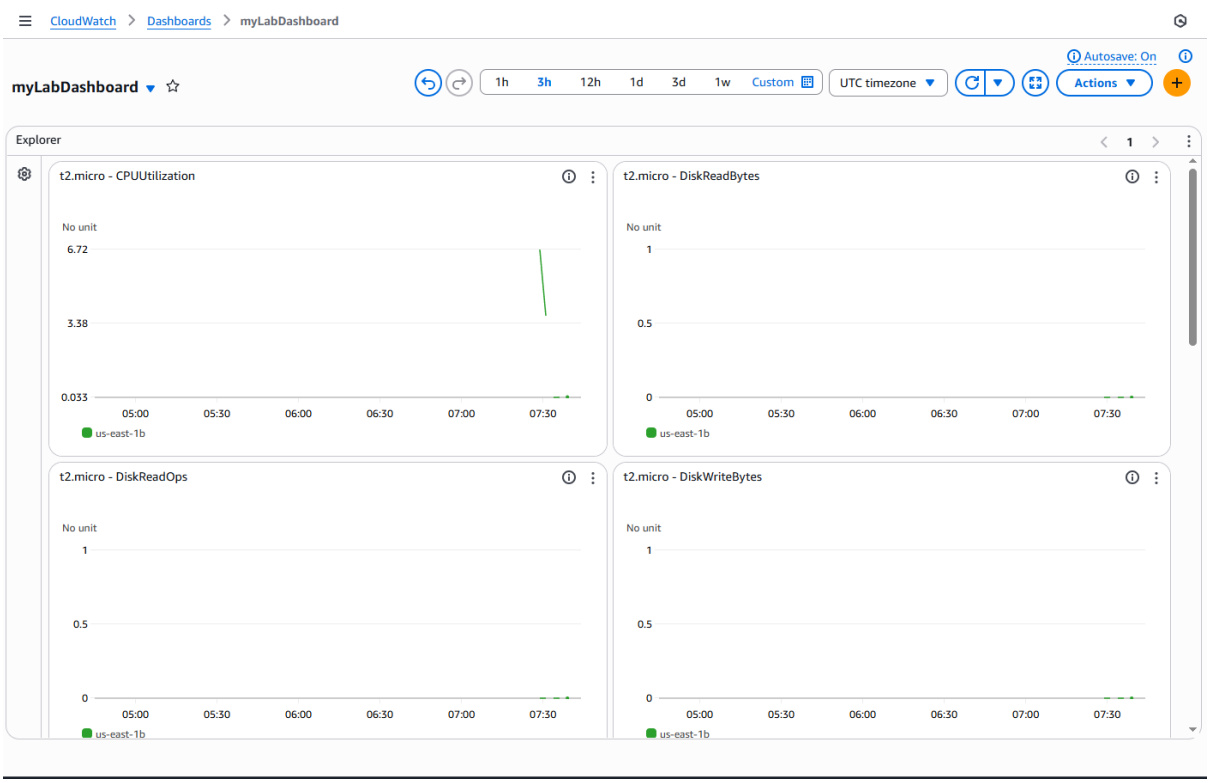
Implementasi Skema Monitoring:

1. **Pengumpulan Data:** Semua layanan AWS akan secara otomatis mengirim metrik ke CloudWatch. Tim developer akan mengkonfigurasi aplikasi untuk mengirim log ke CloudWatch Logs.
2. **Visualisasi (Dashboard):** Beberapa dashboard kunci yang harus dibuat:
    - **Dashboard Eksekutif:** Menampilkan metrik bisnis tingkat tinggi seperti *uptime*, jumlah transaksi, dan kepuasan pelanggan (Apdex score).
    - **Dashboard Operasional:** Menampilkan kesehatan infrastruktur secara real-time (CPU, memori, status health check ALB, koneksi database).
    - **Dashboard Aplikasi:** Menampilkan metrik spesifik aplikasi (tingkat error per endpoint, latensi API, jumlah pengguna aktif).
3. **Peringatan (Alerting):**
    - **Mekanisme:** Peringatan akan dikonfigurasi di **Amazon CloudWatch Alarms**. Ketika sebuah metrik melampaui ambang batas (misalnya, CPU > 80% selama 5 menit), alarm akan memicu notifikasi.
    - **Notifikasi:** Alarm akan mengirim pesan ke **Amazon SNS (Simple Notification Service)**. Dari SNS, notifikasi dapat disalurkan ke berbagai tujuan:
        - **Peringatan Kritis (P1):** Dikirim ke **PagerDuty** untuk membangunkan tim on-call.

- **Peringatan non-Kritis (P2/P3):** Dikirim ke kanal **Slack** tim operasional atau email.

Metrik Utama yang Dipantau:

Kategori	Metrik Utama	Contoh Alert
Availability	HTTP 5xx Error Rate (dari ALB), Health Check Status	P1 Alert (via PagerDuty) jika 5xx Error Rate > 1% selama 5 menit.
Performance	CPU Utilization (EC2 & RDS), Application Latency (p95, p99 dari ALB)	P2 Alert (via Slack) jika CPU Utilization > 80% selama 10 menit.
Security	Jumlah Login Gagal (dari log aplikasi), Aktivitas IAM Mencurigakan (dari CloudTrail)	P1 Alert (to Security Team) jika ada percobaan login gagal 5x dari IP yang sama dalam 1 menit.



[Dashboard CloudWatch di Lab]

3.4 Backup dan Disaster Recovery

Untuk memastikan kelangsungan bisnis dan melindungi dari kehilangan data, strategi Backup dan Disaster Recovery (DR) yang solid adalah komponen yang tidak bisa ditawar. Strategi ini harus menjawab dua pertanyaan kunci: "Berapa banyak data yang boleh hilang?" dan "Berapa lama sistem boleh mati?".

Definisi Metrik Kunci (RTO & RPO):

- **Recovery Point Objective (RPO):** Toleransi maksimal terhadap kehilangan data.
    - **Target: 15 Menit**. Ini berarti, dalam skenario terburuk, perusahaan siap kehilangan data transaksi maksimal 15 menit terakhir. Target ini dicapai melalui backup database yang berkelanjutan.
- **Recovery Time Objective (RTO):** Waktu maksimal yang dibutuhkan untuk memulihkan layanan setelah insiden terjadi.
    - **Target: 1 Jam**. Ini berarti, platform e-commerce harus kembali online dan beroperasi penuh dalam waktu satu jam setelah deklarasi bencana.

Strategi Backup Otomatis:

Strategi backup akan diotomatisasi sepenuhnya menggunakan layanan AWS untuk memastikan konsistensi dan mengurangi risiko kesalahan manusia.

- **Application Servers (EC2):** Menggunakan layanan **AWS Backup** untuk membuat *backup plan* terpusat. Rencana ini akan secara otomatis membuat *snapshot* harian dari volume disk (EBS) semua EC2 instance. Snapshot ini akan disimpan selama 30 hari dan dapat digunakan untuk memulihkan server ke kondisi hari sebelumnya jika terjadi kerusakan.
- **Database (RDS):** Layanan RDS menyediakan dua mekanisme backup yang kuat:
    1. **Automated Snapshots:** Snapshot harian dari seluruh database akan dibuat dan disimpan secara otomatis.
    2. **Transaction Logs:** RDS secara terus-menerus mencadangkan log transaksi (biasanya setiap 5 menit). Kombinasi snapshot dan log transaksi inilah yang memungkinkan fitur **Point-in-Time Recovery (PITR)**, yang krusial untuk memenuhi RPO 15 menit.

- **Aset Statis (S3):**
    - **S3 Versioning:** Diaktifkan pada bucket S3 untuk melindungi dari penghapusan atau penimpaan file yang tidak disengaja. Setiap versi file akan disimpan, memungkinkan pemulihan yang mudah.
    - **Cross-Region Replication (CRR):** Untuk data yang paling kritis, CRR akan diaktifkan untuk secara otomatis menyalin setiap file baru ke bucket S3 di region AWS yang berbeda (misalnya, dari Singapura ke Jakarta). Ini adalah bagian penting dari strategi DR.

Rencana Pemulihan Bencana (Disaster Recovery Plan):

Arsitektur Multi-AZ yang dirancang sudah memberikan ketahanan terhadap kegagalan satu data center (Availability Zone). Rencana DR ini berfokus pada skenario yang lebih besar, seperti kegagalan layanan di seluruh AZ.

- **Skenario:** Kegagalan total pada Availability Zone A, yang berisi RDS Primary.
- **Langkah-langkah Pemulihan:**
    1. **Deteksi & Failover Otomatis:**
        - **Database:** Layanan RDS Multi-AZ akan secara otomatis mendeteksi kegagalan RDS Primary. Dalam beberapa menit, RDS akan mempromosikan RDS Standby di Availability Zone B menjadi Primary yang baru dan memperbarui endpoint DNS-nya.
        - **Aplikasi:** Auto Scaling Group, yang sudah berjalan di kedua AZ, akan mendeteksi bahwa instance di AZ A tidak sehat. Ia akan secara otomatis menghentikan instance tersebut dan meluncurkan instance baru di Availability Zone B untuk mempertahankan kapasitas yang diinginkan.
    2. **Verifikasi:** Tim operasional akan menerima notifikasi dari CloudWatch dan bertugas untuk memverifikasi bahwa failover telah berhasil dan aplikasi berjalan normal di Availability Zone B.
    3. **Pemulihan (Remediasi):** Setelah AZ A kembali pulih, tim akan memastikan RDS dan EC2 kembali ke konfigurasi Multi-AZ yang seimbang.

Rencana ini memanfaatkan automasi bawaan AWS untuk mencapai RTO di bawah 1 jam dengan intervensi manual yang minimal. Rencana ini harus diuji secara berkala (minimal setahun sekali) melalui *DR Drill* untuk memastikan semua prosedur berjalan seperti yang diharapkan.

IV. Analisis Biaya dan Rekomendasi Strategis

4.1 Estimasi dan Perbandingan Biaya

Berikut adalah estimasi biaya bulanan untuk arsitektur yang diusulkan di tiga penyedia cloud utama (harga *on-demand*, region Singapura). Perlu ditekankan bahwa ini adalah titik awal; biaya aktual akan bergantung pada penggunaan dan optimasi.

Layanan	AWS (Amazon Web Services)	Azure (Microsoft)	GCP (Google Cloud Platform)
Virtual Machines (2x)	$55	$60	$52
Managed Database (1x)	$25	$30	$28
Object Storage (100 GB)	$2.30	$2.00	$2.60
Load Balancer	$22.50	$25	$20
Data Transfer Out (500 GB)	$45	$42.50	$42.50
VPN Gateway	$36	$40	$38
Lainnya (WAF, Logs, Backup, DNS)	$55	$63.20	$55
TOTAL ESTIMASI BULANAN	$240.80	$262.70	$238.10
TOTAL ESTIMASI TAHUNAN	$2,889.60	$3,152.40	$2,857.20

Kesimpulan Biaya: GCP menawarkan harga on-demand sedikit lebih rendah untuk skenario ini. Namun, selisihnya tidak signifikan. **Rekomendasi kami adalah memulai dengan platform di mana tim memiliki keahlian paling banyak atau ekosistem layanannya paling sesuai.** Biaya ini dapat **diturunkan 30-70%** dengan komitmen penggunaan (Savings Plans/Reserved Instances).

4.2 Refleksi Profesional

Pelajaran utama dari proses perancangan ini adalah bahwa transformasi cloud lebih dari sekadar pemindahan teknologi; ini adalah perubahan strategi operasional dan budaya. Tantangan utama bukan pada pemilihan layanan, melainkan pada bagaimana merancangnya secara terintegrasi untuk mencapai tujuan bisnis. Solusinya terletak pada pendekatan bertahap (hybrid), mengutamakan keamanan sejak hari pertama (*security by design*), dan merangkul automasi sebagai kunci efisiensi dan kelincahan.

4.3 Rekomendasi Langkah Lanjutan (*Road Map*)

Kami merekomendasikan *road map* implementasi dalam tiga fase untuk mencapai maturitas cloud.

- **Fase 1: Implementasi Fondasi (0-3 Bulan)**
    - Membangun arsitektur dasar yang dijelaskan dalam laporan ini menggunakan IaC.
    - Melakukan migrasi awal (Rehost) aplikasi non-kritis untuk validasi.
    - Membangun pipeline CI/CD dasar untuk satu aplikasi.
    - Melakukan training intensif dan sertifikasi dasar untuk tim teknis.
- **Fase 2: Optimalisasi Kinerja & Biaya (3-9 Bulan)**
    - **Peningkatan Kinerja:** Mengimplementasikan **Content Delivery Network (CDN)** untuk mempercepat waktu muat halaman secara global dan mengadopsi **Caching Layer (Redis/Memcached)** untuk mengurangi beban database.
    - **Optimalisasi Biaya:** Melakukan *right-sizing* sumber daya secara agresif dan membeli **Reserved Instances/Savings Plans** untuk beban kerja stabil.
    - **Peningkatan Operasional:** Mengimplementasikan manajemen patch otomatis menggunakan **AWS Systems Manager** atau **Azure Automation**.
    - **Modernisasi Awal:** Memulai proses **Replatforming** untuk database dan aplikasi pendukung lainnya.
- **Fase 3: Maturitas & Inovasi (9+ Bulan)**

- **Peningkatan Keamanan (DevSecOps):** Mengintegrasikan pemindaian keamanan (SAST/DAST) ke dalam pipeline CI/CD, mengimplementasikan **Manajemen Rahasia (AWS Secrets Manager/Azure Key Vault)**, dan mengaktifkan **Deteksi Ancaman Otomatis (AWS GuardDuty)**.
- **Eksplorasi Serverless:** Menggunakan **AWS Lambda** atau **Azure Functions** untuk tugas-tugas spesifik (misalnya, pemrosesan gambar) guna meningkatkan efisiensi.
- **Modernisasi Aplikasi dengan Kontainer dan Orkestrasi:** Memulai proses **Refactoring** aplikasi monolitik menjadi layanan-layanan mikro (*microservices*). Setiap layanan akan dibungkus (*containerized*) menggunakan **Docker** dan dikelola oleh platform orkestrasi seperti **Kubernetes (menggunakan layanan terkelola seperti Amazon EKS atau Azure AKS)** atau **Amazon ECS**. Langkah ini adalah kunci untuk mencapai kelincahan deployment, skalabilitas granular, dan ketahanan sistem yang superior.

V. Daftar Istilah (Glossary)

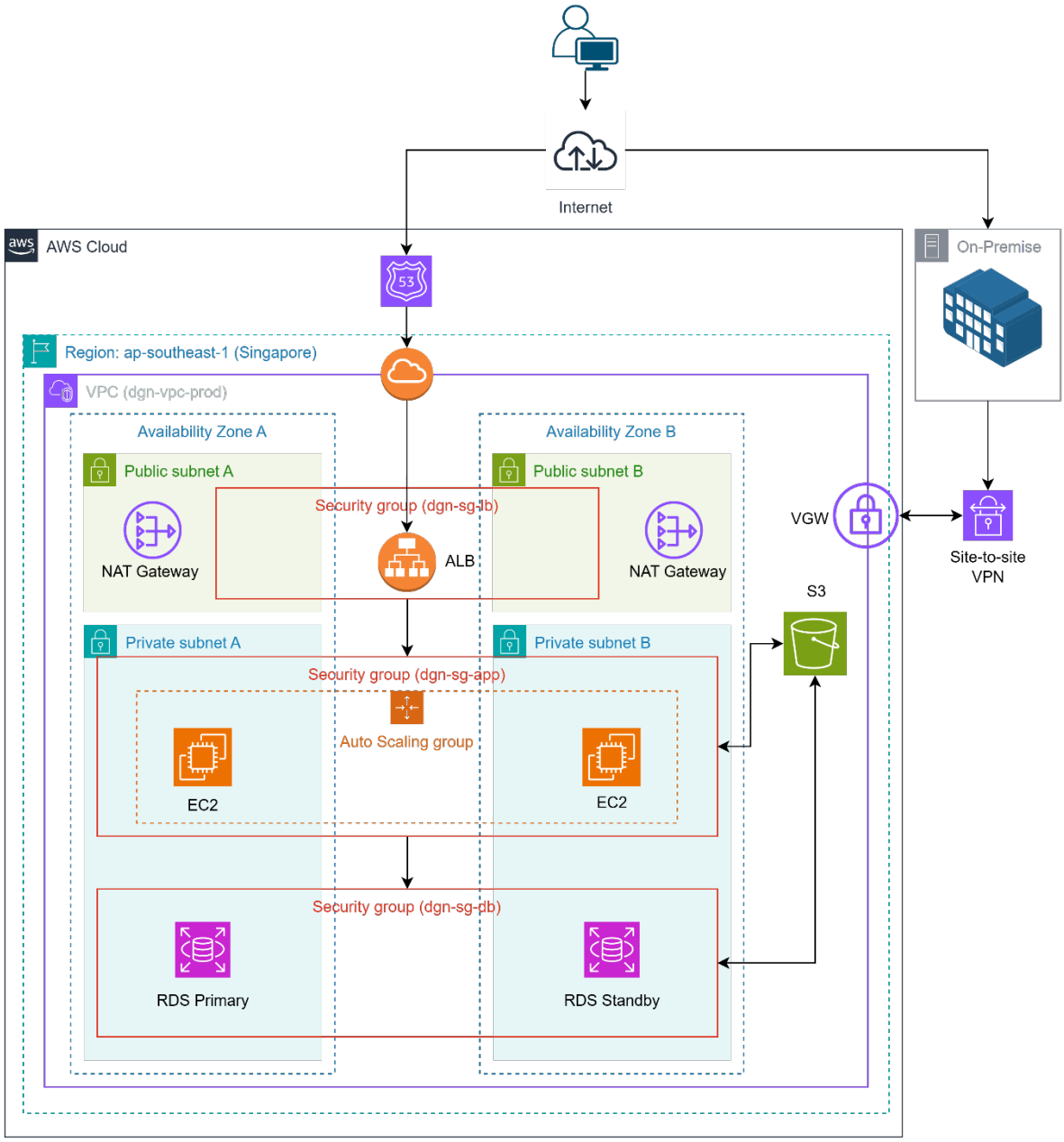
Istilah	Deskripsi
CI/CD	*Continuous Integration/Continuous Deployment*. Praktik automasi untuk membangun, menguji, dan men-deploy kode secara cepat dan andal.
IaaS	*Infrastructure as a Service*. Model layanan cloud di mana penyedia menyewakan infrastruktur IT dasar (server, storage, jaringan).
PaaS	*Platform as a Service*. Model layanan cloud yang menyediakan platform untuk mengembangkan dan menjalankan aplikasi tanpa mengelola infrastruktur di bawahnya.
RPO	*Recovery Point Objective*. Titik waktu maksimal di masa lalu di mana data harus dapat dipulihkan. Mengukur toleransi kehilangan data.
RTO	*Recovery Time Objective*. Durasi waktu maksimal yang ditargetkan untuk memulihkan layanan setelah terjadi insiden.
VPC	*Virtual Private Cloud*. Jaringan pribadi yang terisolasi secara logis di dalam public cloud.
WAF	*Web Application Firewall*. Firewall yang melindungi aplikasi web dari serangan umum pada lapisan aplikasi.

VI. Daftar Referensi

1. CompTIA Cloud+ Certification Study Guide (Exam CV0-004).
2. Dokumentasi Resmi Amazon Web Services (aws.amazon.com/documentation).
3. Dokumentasi Resmi Microsoft Azure (docs.microsoft.com/en-us/azure).
4. Dokumentasi Resmi Google Cloud Platform (cloud.google.com/docs).
5. NIST Cybersecurity Framework.
6. AWS Well-Architected Framework.

VI. Lampiran (Opsional)

- Diagram infrastruktur



- Template insfrastructure.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Description: >
  CloudFormation template lengkap untuk arsitektur 3-tier PT Digital
Nusantara.
  Membuat Jaringan, Keamanan, Load Balancer, dan Auto Scaling Group.
```

```yaml
Parameters:
  VpcCIDR:
    Type: String
    Default: 10.0.0.0/16
  PublicSubnetACIDR:
    Type: String
    Default: 10.0.1.0/24
  PublicSubnetBCIDR:
    Type: String
    Default: 10.0.2.0/24
  PrivateSubnetACIDR:
    Type: String
    Default: 10.0.10.0/24
  PrivateSubnetBCIDR:
    Type: String
    Default: 10.0.11.0/24
  LatestAmiId:
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-
x88_64-gp2'
  InstanceType:
    Type: String
    Default: t3.micro

Resources:
  # --- 1. FONDASI JARINGAN & KEAMANAN ---
  VPC:
    Type: AWS::EC2::VPC
    Properties:
     CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: dgn-vpc

  InternetGateway:
    Type: AWS::EC2::InternetGateway
  AttachGateway:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref VPC
      InternetGatewayId: !Ref InternetGateway

  PublicSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
```

```
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnetACIDR
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      MapPublicIpOnLaunch: true
  PublicSubnetB:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnetBCIDR
      AvailabilityZone: !Select [ 1, !GetAZs '' ]
      MapPublicIpOnLaunch: true

  PrivateSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: !Ref PrivateSubnetACIDR
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
  PrivateSubnetB:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: !Ref PrivateSubnetBCIDR
      AvailabilityZone: !Select [ 1, !GetAZs '' ]

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: AttachGateway
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
  PublicSubnetARouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetA
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetBRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetB
      RouteTableId: !Ref PublicRouteTable

  NatGatewayEIP:
```

```
    Type: AWS::EC2::EIP
    Properties:
      Domain: vpc

  NatGateway:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGatewayEIP.AllocationId
      SubnetId: !Ref PublicSubnetA # NAT Gateway resides in a public
subnet
      Tags:
        - Key: Name
          Value: dgn-nat-gateway

  PrivateRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: dgn-private-rtb

  PrivateRoute:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway

  PrivateSubnetARouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PrivateSubnetA
      RouteTableId: !Ref PrivateRouteTable

  PrivateSubnetBRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PrivateSubnetB
      RouteTableId: !Ref PrivateRouteTable

  LoadBalancerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Allow HTTP traffic to Load Balancer"
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
```

```
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0

  AppServerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Allow traffic from Load Balancer"
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 8080
          ToPort: 8080
          SourceSecurityGroupId: !Ref LoadBalancerSecurityGroup

  # --- 2. STACK APLIKASI ---
  AppLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: dgn-alb
      Subnets: [!Ref PublicSubnetA, !Ref PublicSubnetB]
      SecurityGroups: [!Ref LoadBalancerSecurityGroup]
      Scheme: internet-facing

  AppTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: dgn-tg-app
      VpcId: !Ref VPC
      Protocol: HTTP
      Port: 8080
      HealthCheckProtocol: HTTP
      HealthCheckPath: /health
      TargetType: instance

  HttpListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      LoadBalancerArn: !Ref AppLoadBalancer
      Protocol: HTTP
      Port: 80
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref AppTargetGroup

  AppLaunchTemplate:
    Type: AWS::EC2::LaunchTemplate
    Properties:
```

```
      LaunchTemplateName: dgn-app-template
      LaunchTemplateData:
        ImageId: !Ref LatestAmiId
        InstanceType: !Ref InstanceType
        SecurityGroupIds: [!Ref AppServerSecurityGroup]
        # UserData: (bisa ditambahkan skrip instalasi di sini)

  AppAutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      VPCZoneIdentifier: [!Ref PrivateSubnetA, !Ref PrivateSubnetB]
      LaunchTemplate:
        LaunchTemplateId: !Ref AppLaunchTemplate
        Version: !GetAtt AppLaunchTemplate.LatestVersionNumber
      MinSize: '2'
      MaxSize: '4'
      DesiredCapacity: '2'
      TargetGroupARNs: [!Ref AppTargetGroup]

Outputs:
  LoadBalancerDNS:
    Description: DNS name dari Application Load Balancer
    Value: !GetAtt AppLoadBalancer.DNSName
```

- Template IAM Policy

Policy untuk CloudAdmin:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Policy untuk Developer:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCICDAccess",
            "Effect": "Allow",
```

```
            "Action": [
                "codecommit:*",
                "codebuild:*",
                "codedeploy:*",
                "codepipeline:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyProd",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "rds:Describe*",
                "logs:Get*",
                "logs:DescribeLogStreams"
            ],
            "Resource": "*"
        }
    ]
}
```

Policy untuk SupportEngineer:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMonitoringAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Get*",
                "cloudwatch:Describe*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "elasticloadbalancing:Describe*",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```