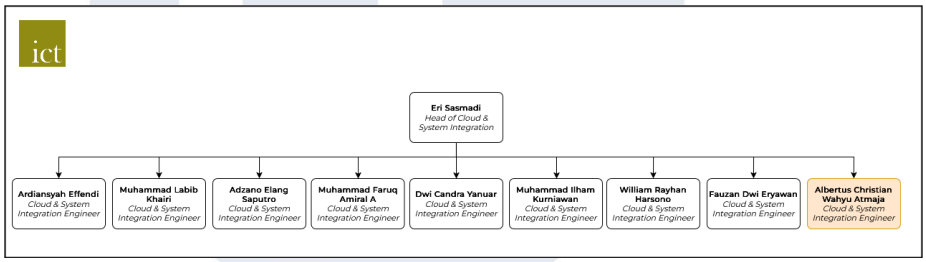
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Bagian ini menjelaskan posisi atau kedudukan yang dijalani selama kerja magang di PT Infracom Technology, serta mekanisme koordinasi kerja yang diterapkan dalam pengerjaan tugas dan proyek di bawah bimbingan pembimbing lapangan.

3.1.1 Kedudukan



Gambar 3.1 Posisi & Kedudukan di PT Infracom Technology

Gambar 3.1 menampilkan lokasi dimana tugas magang dilaksanakan dengan posisi sebagai *Cloud & System Integration Engineer* dalam divisi *Cloud & System Integration*. Kedudukan ini berada di bawah bimbingan dan pengawasan langsung dari Bapak Eri Sasmadi, selaku *Head of Cloud & System Integration*. Sebagaimana digambarkan pada Gambar 3.1, posisi ini tidak bersifat independen, melainkan terintegrasi penuh sebagai anggota tim yang berkolaborasi secara aktif dengan para *Cloud & System Integration Engineer* senior lainnya.

Divisi Cloud & System Integration memegang peranan sentral dalam arsitektur teknologi perusahaan, yang bertanggung jawab untuk merancang, mengimplementasikan, dan mengelola solusi infrastruktur *cloud* bagi klien. Dalam kedudukannya, tanggung jawab diberikan untuk mendukung berbagai proyek implementasi dan otomatisasi infrastruktur,

khususnya pada platform *Oracle Cloud Infrastructure (OCI)* dan *Amazon Web Services (AWS)*.

3.1.2 Rincian tugas dan tanggung jawab utama

1. Perancangan dan Implementasi Infrastruktur *Multi-Cloud* (OCI & AWS)

    a. Membantu merancang arsitektur dasar *cloud* sesuai dengan kebutuhan proyek dan praktik terbaik industri.

    b. Mengimplementasikan komponen inti *Landing Zone* di OCI dan AWS untuk memastikan tata kelola (*governance*), keamanan, dan segmentasi jaringan yang standar.

    c. Melakukan penyediaan (*provisioning*) dan konfigurasi sumber daya *cloud* seperti *Virtual Cloud Networks (VCN)*, *Virtual Private Cloud (VPC)*, serta *Identity and Access Management (IAM)*.

2. Otomatisasi Infrastruktur Menggunakan *Terraform* (*Infrastructure as Code*)

    a. Menulis, menguji, dan memelihara skrip *Terraform* untuk mengotomatiskan proses penyediaan infrastruktur.

    b. Memastikan kode *IaC* yang dikembangkan bersifat modular, dapat digunakan kembali (*reusable*), dan konsisten.

    c. Mengelola *state file Terraform* untuk melacak kondisi infrastruktur yang telah diterapkan.

3. Pengelolaan, Integrasi, dan Pemecahan Masalah

    a. Mendukung integrasi antara berbagai layanan *cloud* untuk menciptakan solusi yang terpadu.

b. Melakukan analisis dan pemecahan masalah (*troubleshooting*) terkait isu konfigurasi atau konektivitas.

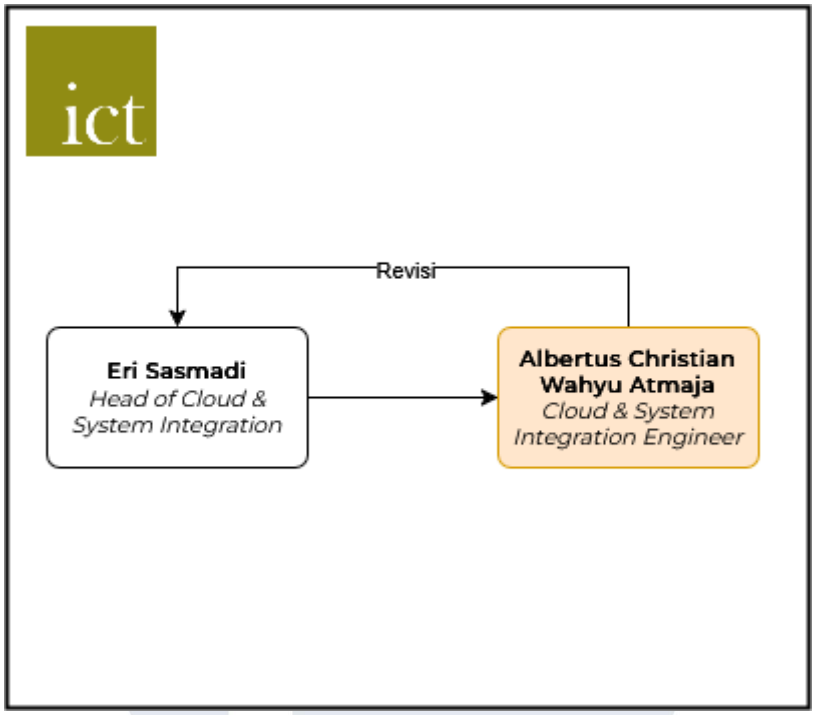
c. Membantu dalam proses dokumentasi teknis terkait arsitektur dan konfigurasi.

3.1.3 Koordinasi

Bagian Alur koordinasi pekerjaan selama magang dirancang agar efektif dan kolaboratif, memastikan setiap tugas berjalan sesuai arahan dan target yang ditetapkan. Proses ini melibatkan koordinasi vertikal dengan atasan dan koordinasi horizontal dengan sesama anggota tim.

Koordinasi utama bersifat vertikal, di mana arahan, tugas, dan bimbingan langsung dari *Head of Cloud & System Integration* serta para insinyur senior di dalam tim. Melaporkan kemajuan pekerjaan dan tantangan yang dihadapi secara berkala kepada atasan. Umpan balik dan evaluasi diberikan secara berkelanjutan untuk memastikan proyek berjalan sesuai rencana.

Secara horizontal, berkolaborasi secara harian dengan anggota tim *Cloud & System Integration Engineer* lainnya. Kolaborasi ini bertujuan untuk menyelesaikan tugas bersama, berbagi temuan teknis, dan memastikan keselarasan dalam pengerjaan proyek.



Gambar 3.2 Bagan Alur Koordinasi

Gambar 3.2 merupakan alur koordinasi, dan untuk mendukung alur koordinasi tersebut, diterapkan sistem kerja hibrida. Pada dua bulan pertama, kegiatan dilaksanakan secara luring penuh (*onsite*) di kantor untuk mempercepat proses adaptasi dan memungkinkan koordinasi tatap muka yang intensif. Selanjutnya, jadwal kerja diubah menjadi dua hari di kantor dan tiga hari dari rumah, dengan jam kerja terstruktur dari pukul 08:00 hingga 17:00 WIB.

Seluruh proses komunikasi dan koordinasi didukung oleh berbagai alat kolaborasi. Diskusi strategis dan bimbingan mendalam seringkali dilakukan saat sesi tatap muka di kantor, sementara platform seperti Microsoft Teams dan WhatsApp digunakan secara intensif untuk koordinasi harian, pendelegasian tugas, dan pemecahan masalah cepat, memastikan komunikasi tetap lancar meskipun tim bekerja dari lokasi yang berbeda.

3.2 Tugas yang Dilakukan

Pada Tabel 3.1 berisi Program magang di PT Infracom Technology berlangsung selama enam bulan, dari 4 Juni 2025 hingga 3 Desember 2025. Selama

periode ini, keterlibatan aktif dilakukan dalam berbagai tugas yang berkaitan dengan implementasi infrastruktur *cloud*, otomatisasi dengan *Infrastructure as Code*, dan *system integration*. Program magang ini memberikan kesempatan untuk menerapkan pengetahuan teknis dalam lingkungan dunia nyata yang dinamis, meningkatkan keterampilan pemecahan masalah, dan memperoleh pengalaman langsung dalam mengembangkan serta mengelola sistem berbasis *cloud*.

Salah satu tugas utama yang diberikan selama magang adalah mengimplementasikan arsitektur *Landing Zone* pada platform Oracle Cloud Infrastructure (OCI). Tugas ini melibatkan perancangan fondasi *cloud* yang terstandarisasi, aman, dan dapat diskalakan untuk menampung seluruh beban kerja di masa depan. anggung jawab yang diemban adalah mengimplementasikan komponen-komponen inti seperti konfigurasi jaringan (*Virtual Cloud Network*), manajemen identitas dan akses (*Identity and Access Management*), dan aturan keamanan. Melalui proses ini, dikembangkan pemahaman mendalam tentang bagaimana arsitektur yang terstruktur dapat meningkatkan efisiensi, konsistensi, dan keamanan dalam lingkungan *cloud* berskala korporat.

Selain itu, peran penting dimainkan dalam menerapkan otomatisasi untuk merampingkan proses penyediaan dan konfigurasi infrastruktur. Fokus utamanya adalah memanfaatkan Terraform sebagai alat *Infrastructure as Code* (IaC). Dengan penulisan skrip Terraform, tugas-tugas manual yang repetitif dan rawan kesalahan dibantu untuk digantikan dengan proses otomatis yang dapat diulang dan dikontrol melalui versi. Implementasi otomatisasi ini tidak hanya meningkatkan produktivitas tim, tetapi juga memastikan setiap lingkungan *cloud* yang di-*deploy* memiliki konfigurasi yang identik dan sesuai dengan standar yang telah ditetapkan.

Pengalaman langsung yang diperoleh selama magang memberikan wawasan berharga ke dalam praktik rekayasa infrastruktur *cloud* di dunia nyata, termasuk pemecahan masalah yang sistematis, strategi optimalisasi, dan alur kerja pengembangan profesional. Kemampuan untuk bekerja sebagai bagian dari tim, sambil menerima bimbingan langsung dari supervisor dan insinyur senior, secara signifikan meningkatkan keterampilan teknis dan analitis yang dimiliki. Seluruh

pengetahuan yang diperoleh selama magang diharapkan dapat menjadi fondasi yang kuat untuk karier di bidang rekayasa *cloud* di masa depan.

Tabel 3.1 Detail Pekerjaan yang Dilakukan

No	Aktivitas	Waktu Mulai	Waktu Selesai
1	Perkenalan Kantor, pemberian *jobdesc*, dan pengenalan teman satu divisi	4 Juni 2025	5 Juni 2025
2	Pembuatan *private web server* menggunakan bastion di OCI.	5 Juni 2025	5 Juni 2025
3	Implementasi Terraform dengan OCI.	9 Juni 2025	9 Juni 2025
4	Membuat web hosting dengan menggunakan terraform.	10 Juni 2025	11 Juni 2025
5	Membuat *landing zone* dengan menggunakan terraform.	12 Juni 2025	13 Juni 2025
6	Implementasi oci core landing zone	16 Juni 2025	8 September 2025
7	Simulasi migrasi 5 *app* dari *on-prem* ke AWS	14 Agustus 2025	28 Agustus 2025

3.3 Uraian Pelaksanaan Kerja

Secara umum, pekerjaan yang dilakukan selama magang berpusat pada implementasi infrastruktur *cloud*, otomatisasi dengan pendekatan *Infrastructure as Code (IaC)*, dan integrasi sistem. Program ini memberikan kesempatan untuk menerapkan pengetahuan teknis dalam lingkungan korporat yang dinamis, meningkatkan keterampilan pemecahan masalah, serta memperoleh pengalaman langsung dalam mengembangkan dan mengelola sistem berbasis *cloud* yang aman dan efisien.

Dalam mendukung kelancaran seluruh tugas tersebut, berikut merupakan serangkaian perangkat keras dan perangkat lunak sebagai berikut:

Perangkat Lunak (*Software*)

1. Oracle Cloud Infrastructure (OCI): Platform *cloud computing* utama yang digunakan untuk merancang dan mengelola infrastruktur proyek.
2. Amazon Web Services (AWS): Platform *cloud computing* kedua yang digunakan untuk simulasi migrasi dan implementasi arsitektur *multi-cloud*.
3. Visual Studio Code: *Editor* kode utama yang digunakan untuk menulis, mengelola, dan melakukan *debugging* skrip *Terraform*.
4. Google Chrome: Peramban web untuk mengakses konsol manajemen OCI dan AWS.
5. Gmail & Microsoft Teams: Media komunikasi formal untuk laporan, dokumentasi, dan koordinasi proyek.
6. WhatsApp: Sarana komunikasi informal untuk diskusi cepat dan koordinasi harian dengan tim.

Perangkat Keras (*Hardware*)

1. Prosesor: AMD Ryzen 3 5300U
2. RAM: 12GB

3.3.1 Proses Pelaksanaan

Pada bagian ini, akan dipaparkan secara terperinci proses pelaksanaan dari proyek-proyek dan tugas utama yang menjadi tanggung jawab selama program magang, mulai dari tahap perencanaan hingga implementasi.

3.3.1.1 Perkenalan Kantor, pemberian jobdesc, dan pengenalan teman satu divisi

Selama minggu pertama magang di PT Infracom Technology, yang berlangsung dari tanggal 4 Juni 2025 hingga 3 Desember 2025, dijalani fase orientasi dan pengenalan terhadap

lingkungan kerja perusahaan. Fase pengenalan ini, yang berlangsung pada tanggal 4 dan 5 Juni 2025, bertujuan agar dapat membiasakan diri dengan alur kerja dan budaya perusahaan, khususnya di dalam divisi Cloud & System Integration. Pada hari pertama, secara langsung dikenalkan kepada seluruh anggota tim di dalam divisi, yang memungkinkan proses adaptasi dan kolaborasi awal berjalan dengan baik.

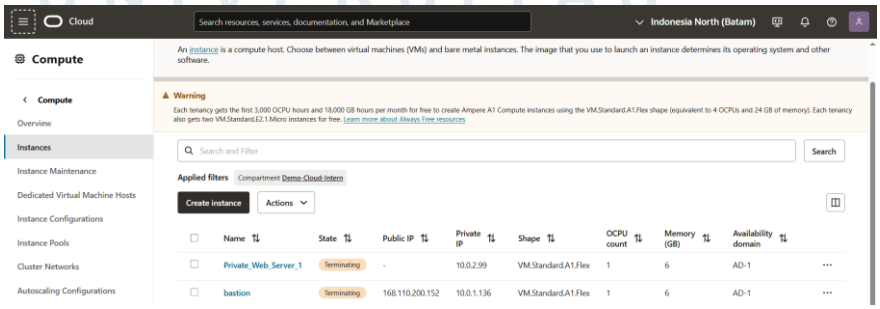
Setelah sesi perkenalan, Head of Cloud & System Integration selaku supervisor memberikan pengarahan terperinci mengenai ruang lingkup pekerjaan magang dan tugas-tugas spesifik yang akan menjadi tanggung jawab selama enam bulan ke depan. Tanggung jawab utama tersebut meliputi implementasi arsitektur OCI dan AWS *Landing Zone*, melakukan otomatisasi penyediaan infrastruktur menggunakan Terraform (*Infrastructure as Code*), serta mendukung berbagai proyek integrasi sistem yang sedang berjalan. Pengarahan ini juga mencakup penjelasan mengenai jadwal proyek, hasil yang diharapkan, dan standar teknis yang harus dipenuhi untuk memastikan memiliki pemahaman yang jelas tentang perannya.

Fase perkenalan ini sangat membantu dalam memahami alur kerja teknis di dalam divisi *Cloud Engineering* dan bagaimana setiap proyek infrastruktur dikelola dari awal hingga akhir. Selain itu, pengarahan ini memberikan wawasan tentang praktik terbaik dalam rekayasa *cloud*, metodologi manajemen proyek, dan ekspektasi dalam memberikan solusi infrastruktur yang andal dan aman. Setelah orientasi diselesaikan, proyek yang ditugaskan secara resmi mulai dikerjakan, dimulai dengan tahap analisis kebutuhan dan perancangan awal struktur kode Terraform untuk implementasi *Landing Zone*.

3.3.1.2 Deployment Private Web Server di OCI menggunakan Bastion

Tugas awal dalam program magang ini adalah *deployment* sebuah web server privat di Oracle Cloud Infrastructure (OCI) dengan menggunakan Bastion *host* sebagai gerbang akses yang aman. Tujuan utamanya adalah untuk membangun pemahaman fundamental mengenai konsep jaringan, keamanan akses, dan konfigurasi server secara manual di OCI. Pelaksanaan tugas dimulai dengan membangun fondasi jaringan melalui pembuatan sebuah *Virtual Cloud Network* (VCN) yang terisolasi. Di dalam VCN ini, dikonfigurasi dua jenis subnet: subnet publik untuk menempatkan Bastion host dan subnet privat untuk menempatkan web server utama, yang sepenuhnya terisolasi dari akses internet langsung demi keamanan.

Kegunaannya mengatur alur lalu lintas data, dua jenis *gateway* dikonfigurasikan. Sebuah *Internet Gateway* (IGW) dihubungkan ke subnet publik untuk memungkinkan Bastion diakses dari internet, sementara sebuah NAT *Gateway* (NGW) dihubungkan ke subnet privat. Fungsi NAT *Gateway* ini krusial, karena memungkinkan web server untuk mengakses internet (misalnya, untuk pembaruan perangkat lunak) tanpa bisa diakses kembali dari luar.



Gambar 3.3 Daftar Compute Instance untuk Bastion dan Web Server

Berdasarkan Gambar 3.3 dapat dilihat bahwa dua *Compute Instance* (mesin virtual) telah berhasil di-*deploy*. *Instance* pertama dengan nama "bastion" memiliki alamat IP publik (168.110.200.152) yang menandakan posisinya di subnet publik. *Instance* kedua dengan nama "Private_Web_Server_1" tidak memiliki alamat IP publik dan hanya memiliki alamat IP privat (10.0.2.99), yang membuktikan bahwa *instance* tersebut ditempatkan dengan benar di dalam subnet privat dan terisolasi dari internet.

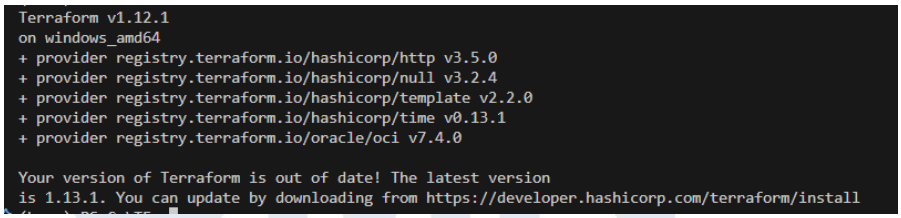
Lapisan keamanan kemudian diterapkan melalui konfigurasi *Security List*, yang berfungsi sebagai *firewall* virtual. Aturan ditetapkan untuk hanya mengizinkan lalu lintas SSH (port 22) dari internet ke Bastion. Akses ke web server privat dibatasi secara ketat, hanya diizinkan dari alamat IP internal Bastion. Untuk mengelola web server tersebut, teknik SSH *Tunneling* (atau *port forwarding*) diterapkan melalui Bastion. Metode ini memastikan bahwa tidak ada *port* manajemen yang terbuka langsung ke internet, sehingga seluruh sesi manajemen terenkapsulasi dengan aman. Hasil akhir dari tugas ini adalah sebuah arsitektur jaringan yang aman dan teruji, di mana aset utama (web server) terlindungi dengan baik dan hanya dapat diakses melalui prosedur yang aman.

3.3.1.3 Implementasi Infrastructure as Code dan Development Environment

Tahap awal program magang difokuskan untuk membangun fondasi pengetahuan yang kokoh dan menyiapkan lingkungan pengembangan (*development environment*) yang profesional untuk otomatisasi infrastruktur *cloud*. Fase persiapan yang metodis ini sangat krusial, karena menjadi dasar bagi seluruh tugas implementasi teknis yang akan dilakukan selanjutnya. Proses ini terbagi menjadi dua kegiatan utama yang saling melengkapi: pendalaman konsep teoretis

mengenai teknologi yang digunakan dan penyiapan praktis dari perangkat kerja (*toolchain*) yang diperlukan.

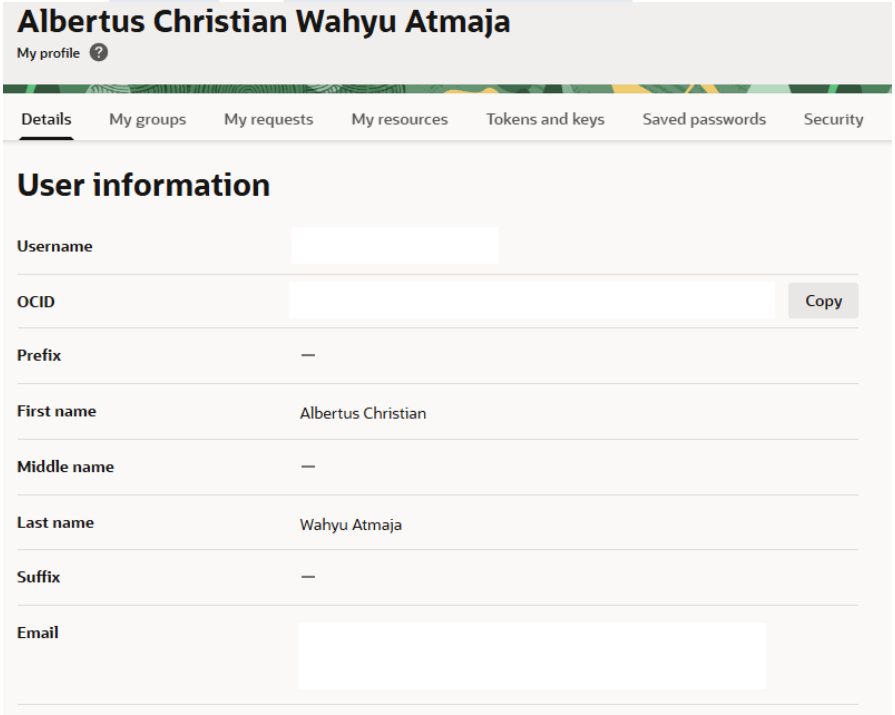
Pada aspek teoretis, dilakukan pembelajaran komprehensif mengenai konsep *Infrastructure as Code (IaC)*, memahami keunggulannya yang fundamental dibandingkan penyediaan manual, seperti mengurangi *human error*, meningkatkan kecepatan *deployment*, dan memfasilitasi audit melalui kontrol versi (*version control*). Pembelajaran dilanjutkan dengan pendalaman *Terraform* sebagai *tool IaC* utama, mencakup alur kerja fundamentalnya (*init, plan, apply, destroy*), konsep *providers* untuk berinteraksi dengan API vendor, cara mendefinisikan *resources*, serta penggunaan *variables* dan *outputs* untuk kode yang dinamis dan modular. Di sisi lain, mempelajari dasar-dasar arsitektur *Oracle Cloud Infrastructure (OCI)*, termasuk konsep inti seperti *Identity and Access Management (IAM)* untuk kontrol akses, struktur jaringan dengan *Virtual Cloud Network (VCN)*, serta layanan komputasi dasar seperti *Compute Instances*.



Gambar 3.4 Hasil Verifikasi Terraform Terinstall

Setelah fondasi teoretis terbentuk, kegiatan dilanjutkan dengan penyiapan lingkungan pengembangan lokal. Proses ini dimulai dengan instalasi dan konfigurasi *Visual Studio Code* sebagai *editor* utama, yang kemudian diperkaya dengan *extension* spesifik seperti HashiCorp Terraform. *Extension* ini sangat membantu dalam meningkatkan produktivitas dengan menyediakan fitur *syntax highlighting*, *autocompletion*, dan validasi kode secara *real-time*.

Langkah selanjutnya adalah instalasi *Terraform* dan *OCI Command Line Interface (CLI)*. Seperti yang ditunjukkan pada Gambar 3.4, verifikasi melalui perintah terraform init berhasil mengonfirmasi bahwa Terraform v1.12.1 dan OCI provider v7.4.0 telah terpasang dengan benar dan siap digunakan. Struktur direktori proyek juga disiapkan mengikuti praktik terbaik, termasuk pembuatan berkas .gitignore untuk mengecualikan data sensitif seperti *file* variabel rahasia (*.tfvars) dan direktori .terraform dari repositori Git.



Gambar 3.5 Kunci OCID

Langkah final dari tahap persiapan adalah mengkonfigurasi konektivitas yang aman antara lingkungan lokal dengan platform OCI. Proses ini melibatkan pembuatan sepasang kunci API (*API key pair*) menggunakan OpenSSL, di mana *public key* diunggah ke konsol OCI pada profil pengguna, sementara *private key* disimpan secara aman di mesin lokal. Setelah *public key* diunggah, OCI akan menyediakan *fingerprint* sebagai penanda unik. Kredensial ini,

bersama dengan *User OCID* (seperti yang ditampilkan pada profil di Gambar 3.5), *Tenancy OCID*, dan *region*, kemudian dimasukkan ke dalam *file* konfigurasi OCI CLI (~/.oci/config). Pengaturan ini sangat penting untuk memungkinkan *Terraform* melakukan autentikasi secara terprogram dan aman saat menjalankan perintah untuk mengelola sumber daya di OCI. Hasil akhir dari tahap ini adalah sebuah lingkungan pengembangan yang terkonfigurasi sepenuhnya dan pemahaman konseptual yang solid, menjadi modal utama untuk memulai implementasi Proyek *Landing Zone*.

3.3.1.4 Demo Implementasi Web Server dan Otomatisasi Terraform

Tugas ini merupakan salah satu pilar utama dalam program magang, dirancang secara komprehensif untuk menjembatani pemahaman konseptual dengan implementasi praktis infrastruktur *cloud*. Tujuannya adalah untuk mengalami secara langsung evolusi proses kerja, mulai dari perancangan dan *deployment* manual yang fundamental hingga mencapai otomatisasi penuh menggunakan pendekatan *Infrastructure as Code (IaC)* dengan Terraform. Proses ini dibagi menjadi dua fase utama yang saling membangun: perancangan arsitektur konseptual dan implementasi melalui otomatisasi.

1. **Tahap 1: Perancangan Arsitektur Manual dan Konseptual**

   Tahap Sebelum menulis satu baris kode pun, langkah pertama adalah merancang arsitektur jaringan dan komputasi secara konseptual. Fase ini mensimulasikan proses *deployment* manual di OCI Console untuk mendefinisikan setiap komponen yang dibutuhkan. Fondasi utamanya adalah perancangan *Virtual Cloud Network* (VCN) sebagai jaringan virtual privat yang terisolasi, yang didefinisikan dengan blok

CIDR utama 10.0.0.0/16. Di dalam VCN ini, dirancang dua lapisan jaringan (*subnet*) untuk keamanan: sebuah Subnet Publik dengan CIDR 10.0.1.0/24 yang akan menjadi lokasi Bastion Host sebagai *jump host* aman dari internet, dan sebuah Subnet Privat dengan CIDR 10.0.2.0/24 untuk menempatkan *Web Server*, melindunginya dari akses internet langsung.

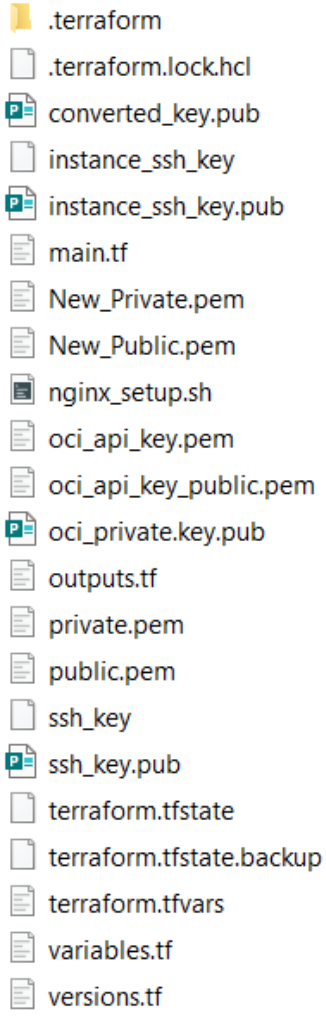
Untuk mengatur konektivitas, arsitektur ini merancang penggunaan dua jenis *gateway*. Sebuah Internet Gateway (IGW) akan dihubungkan ke Subnet Publik untuk menyediakan akses internet dua arah. Sementara itu, untuk memungkinkan *Web Server* di Subnet Privat mengakses internet keluar (misalnya untuk mengunduh pembaruan paket) tanpa bisa diakses dari luar, dirancang sebuah NAT Gateway (NGW). Implementasi keamanan menjadi fokus utama, di mana dirancang aturan-aturan spesifik pada Security Lists untuk membatasi akses secara ketat, seperti hanya mengizinkan trafik SSH (port 22) dari internet ke *Bastion Host*, dan trafik HTTP (port 80) secara internal dari *Bastion* ke *Web Server*. Konsep SSH tunneling juga dirancang sebagai metode untuk mengakses *web server* di jaringan privat secara aman melalui koneksi terenkripsi yang dilewatkan melalui *Bastion Host*. Fase perancangan ini menghasilkan sebuah *blueprint* arsitektur yang detail dan menjadi dasar untuk tahap otomatisasi.

2. Tahap 2: Implementasi Otomatisasi Penuh dengan Terraform

Fase kedua adalah menerjemahkan seluruh rancangan arsitektur dari tahap konseptual ke dalam kode Terraform yang deklaratif. Proses ini menunjukkan kekuatan IaC dalam

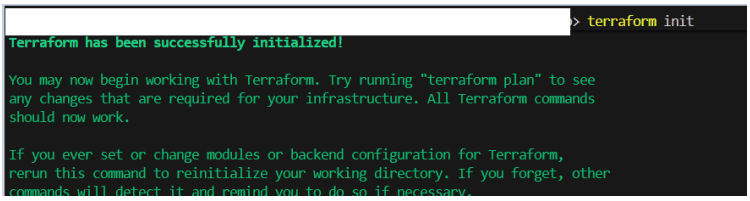
menciptakan infrastruktur yang konsisten, dapat diulang, dan terdokumentasi dengan sendirinya.

a. Struktur Proyek dan Inisialisasi



Gambar 3.6 Struktur Folder Proyek

Langkah pertama dalam otomatisasi adalah mempersiapkan struktur proyek. Struktur direktori proyek, seperti yang terlihat pada Gambar 3.6, diorganisir secara logis untuk memisahkan konfigurasi, variabel, dan skrip.



Gambar 3.7 Terminal Output dari Terraform Init untuk Web

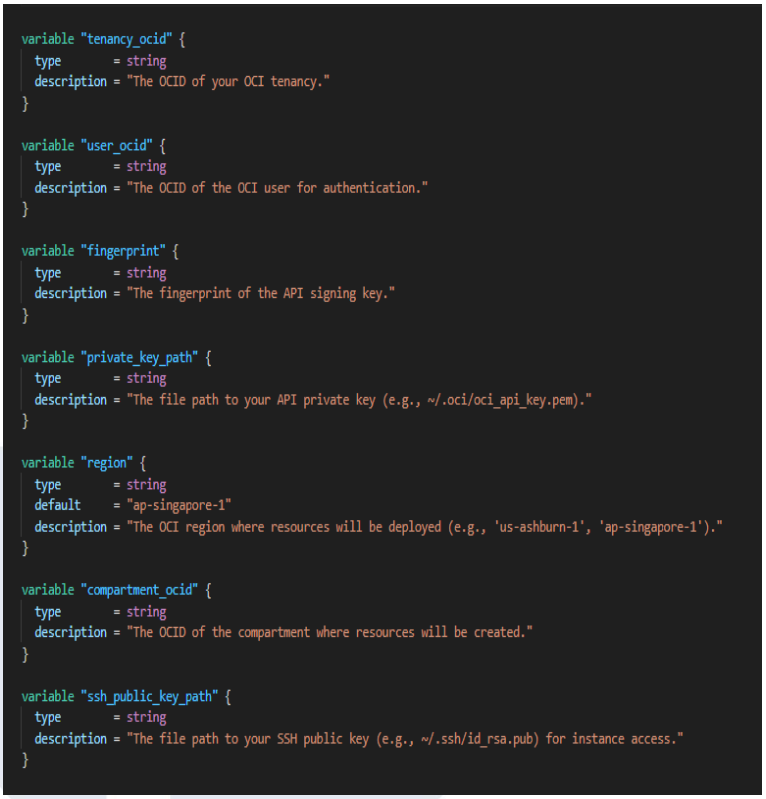
Setelah struktur disiapkan, proses dimulai dengan inisialisasi Terraform melalui perintah terraform init. Perintah ini mengunduh *provider* yang diperlukan, dalam hal ini adalah *provider* OCI, dan menyiapkan direktori kerja. *Output* dari proses ini, seperti yang ditunjukkan pada Gambar 3.7, mengonfirmasi bahwa semua *plugin* yang dibutuhkan telah berhasil diunduh dan siap digunakan.

b. Definisi Infrastruktur sebagai Kode (Infrastructure as Code)

```
resource "oci_core_vcn" "web_server_vcn" {
  compartment_id = var.compartment_ocid
  display_name   = "VCN-WebServer-Demo"
  cidr_block     = "10.0.0.0/16"
}
```

Gambar 3.8 Terrafom oci_core_vcn untuk Web

Seluruh arsitektur yang dirancang sebelumnya kini didefinisikan dalam *file* main.tf. Setiap komponen infrastruktur dipetakan ke sumber daya Terraform yang spesifik. Definisi VCN menggunakan sumber daya oci_core_vcn dengan blok CIDR yang telah ditentukan. Gambar 3.8 menunjukkan potongan kode yang mendefinisikan VCN ini.



```
variable "tenancy_ocid" {
  type        = string
  description = "The OCID of your OCI tenancy."
}

variable "user_ocid" {
  type        = string
  description = "The OCID of the OCI user for authentication."
}

variable "fingerprint" {
  type        = string
  description = "The fingerprint of the API signing key."
}

variable "private_key_path" {
  type        = string
  description = "The file path to your API private key (e.g., ~/.oci/oci_api_key.pem)."
}

variable "region" {
  type        = string
  default     = "ap-singapore-1"
  description = "The OCI region where resources will be deployed (e.g., 'us-ashburn-1', 'ap-singapore-1')."
}

variable "compartment_ocid" {
  type        = string
  description = "The OCID of the compartment where resources will be created."
}

variable "ssh_public_key_path" {
  type        = string
  description = "The file path to your SSH public key (e.g., ~/.ssh/id_rsa.pub) for instance access."
}
```

Gambar 3.9 Variabel Terraform Web

Selanjutnya, konektivitas jaringan seperti Internet Gateway dan *Route Table* didefinisikan menggunakan sumber daya oci_core_internet_gateway dan oci_core_route_table. Aturan-aturan *firewall* yang dirancang sebelumnya diimplementasikan dalam blok oci_core_security_list, yang secara eksplisit mendefinisikan aturan *ingress* dan *egress* untuk port 22 dan 80. Kedua Subnet (publik dan privat) juga didefinisikan menggunakan oci_core_subnet, yang kemudian diasosiasikan dengan *Route Table* dan *Security List* yang sesuai. Terakhir, *Compute Instance* untuk *Bastion* dan *Web Server* didefinisikan menggunakan sumber daya oci_core_instance dengan *shape* yang ditentukan, yaitu VM.Standard.E5.Flex. Untuk membuat kode lebih fleksibel dan dapat digunakan kembali, parameter-parameter penting seperti CIDR block, *shape*

instance, dan nama-nama sumber daya didefinisikan dalam *file* variables.tf, seperti yang terlihat pada Gambar 3.9.

c. Konfigurasi Otomatis dengan Skrip *Provisioner*

```
#!/bin/bash
set -e # Exit immediately if a command exits with a non-zero status.
echo "Starting user data script..."

# --- Configure YUM/DNF repositories for CentOS Stream 8 (to vault.centos.org) ---
# This addresses the "Couldn't resolve host: mirrorlist.centos.org" error
echo "Configuring DNF repositories to vault.centos.org..."
sudo sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-Stream-*.repo || true
sudo sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-Stream-*.repo || true
sudo dnf clean all || true
sudo dnf makecache || true
echo "DNF repository configuration complete."

# Install EPEL (for some dependencies, optional but recommended)
sudo dnf install -y epel-release || true

# --- Install Nginx ---
echo "Installing Nginx..."
sudo dnf install -y nginx
echo "Nginx installation complete."

# --- Stop and Disable Apache HTTP Server (if installed by previous runs) ---
# Ensure no conflict with Nginx on port 80
echo "Stopping and disabling Apache HTTP Server (httpd)..."
sudo systemctl stop httpd || true # '|| true' prevents script from failing if httpd isn't running
sudo systemctl disable httpd || true
echo "Apache HTTP Server (httpd) operations complete."

# --- Update system packages ---
echo "Updating system packages..."
sudo dnf update -y
echo "System package update complete."

# --- Create custom index.html ---
echo "Creating custom index.html at /var/www/html/index.html..."
sudo mkdir -p /var/www/html
sudo sh -c 'cat > /var/www/html/index.html <<HTML_EOF
```

Gambar 3.10 nginx setup bagian 1

```
<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hello ICT!</title>
    <style>
        body {
            font-family: '\''Arial'\'', sans-serif;
            background-color: #f0f2f5;
            color: #333;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            margin: 0;
            padding: 20px;
            box-sizing: border-box;
        }
        .container {
            background-color: #ffffff;
            padding: 40px;
            border-radius: 12px;
            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
            text-align: center;
            max-width: 500px;
            width: 100%;
        }
        .logo {
            width: 150px; /* Ukuran logo bisa disesuaikan */
            height: auto;
            margin-bottom: 30px;
            border: 3px solid #eee; /* Opsional: bingkai tipis */
            border-radius: 8px; /* Opsional: sudut bingkai melengkung */
        }
        h1 {
            color: #007bff; /* Warna biru yang cerah */
            font-size: 2.5em;
            margin-bottom: 20px;
            font-weight: bold;
        }
        p {
            font-size: 1.2em;
            color: #555;
        }
    </style>
</head>
```

Gambar 3.11 nginx setup bagian 2

```
<body>
    <div class="container">
        <img src="https://media.licdn.com/dms/image/v2/C560BAQFhpP5YvCD3gw/company-logo_200_200/company-
        <h1>Hello ICT!</h1>
        <p>Selamat datang di halaman sederhana ini.</p>
    </div>
</body>
</html>
HTML_EOF'
echo "index.html created."

# ...existing code...
echo "index.html created."

# --- Fix permissions and SELinux context for Nginx web root ---
sudo chown -R nginx:nginx /var/www/html
sudo chmod -R 755 /var/www/html
sudo dnf install -y policycoreutils-python-utils || true
sudo semanage fcontext -a -t httpd_sys_content_t "/var/www/html(/.*)?"
sudo restorecon -Rv /var/www/html/
echo "Permissions and SELinux context set for /var/www/html"

# --- Configure Nginx ---
echo "Configuring Nginx..."

# Backup existing nginx.conf
sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak

# Create a new, minimal nginx.conf that only includes conf.d
# This avoids conflicts with the default server block
sudo sh -c 'cat > /etc/nginx/nginx.conf <<NGINX_CONF_EOF
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /run/nginx.pid;

events {
    worker_connections  1024;
}
```

Gambar 3.12 nginx setup bagian 3

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    access_log  /var/log/nginx/access.log;
    sendfile        on;
    tcp_nopush     on;
    keepalive_timeout  65;
    include /etc/nginx/conf.d/*.conf;
}
NGINX_CONF_EOF'
echo "Minimal nginx.conf created."

# Create Nginx config file for your site, using the instance_public_ip variable
sudo sh -c 'cat > /etc/nginx/conf.d/webserver.conf <<EOF
server {
    listen 80;
    server_name _;

    root /var/www/html;
    index index.html index.htm;
    try_files \$uri \$uri/ =404;

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
        internal;
    }
}
EOF'
echo "webserver.conf created."

# Optionally, disable default.conf if it exists (common on RHEL/CentOS)
if [ -f "/etc/nginx/conf.d/default.conf" ]; then
  sudo mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.disabled
fi

# --- Configure FirewallD ---
echo "Configuring FirewallD to allow HTTP..."
# Ensure firewalld is running and enabled before attempting to add rules
sudo systemctl enable --now firewalld || true
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload
echo "FirewallD configuration complete."
```

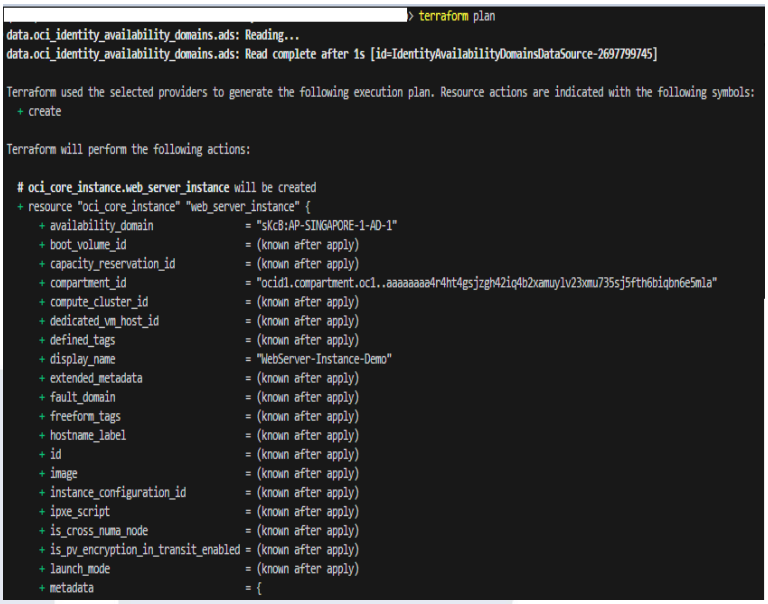
Gambar 3.13 nginx setup bagian 4

```
# --- Configure SELinux ---
# Install policycoreutils-python-utils if not present
echo "Installing policycoreutils-python-utils for SELinux management..."
sudo dnf install -y policycoreutils-python-utils || true
echo "Setting SELinux context for /var/www/html/..."
sudo semanage fcontext -a -t httpd_sys_content_t "/var/www/html(/.*)?"
sudo restorecon -Rv /var/www/html/
echo "SELinux context set."

# --- Start and Enable Nginx ---
echo "Starting and enabling Nginx..."
sudo systemctl start nginx
sudo systemctl enable nginx
echo "Nginx started and enabled."

echo "User data script execution complete."
```

Gambar 3.14 nginx setup bagian 5

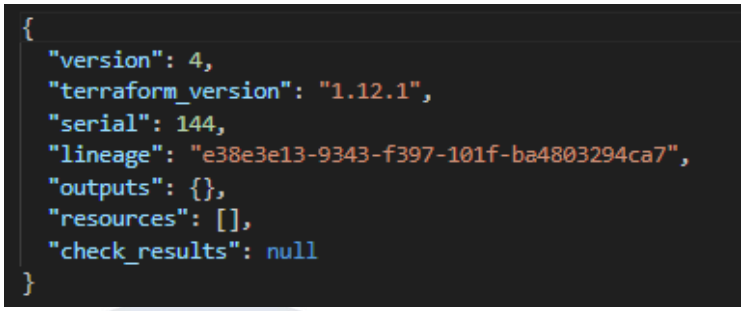
Dalam mengotomatiskan konfigurasi di dalam *Web Server*, Terraform memanfaatkan *provisioner* yang menjalankan skrip nginx_setup.sh. Skrip ini, yang isinya dapat dilihat pada Gambar 3.10, Gambar 3.11, Gambar 3.12, Gambar 3.13, dan Gambar 3.14, bertanggung jawab penuh untuk instalasi dan konfigurasi Nginx, pengaturan *firewall* internal menggunakan firewalld untuk mengizinkan trafik HTTP, membuat halaman index.html kustom dengan *branding* PT. ICT, serta memastikan izin SELinux sudah benar dan layanan Nginx berjalan secara otomatis saat *booting*.

d. Validasi dan Simulasi *Deployment*



Gambar 3.15 Terraform Plan untuk Web

Sebelum menerapkan perubahan apa pun, langkah krusial adalah menjalankan terraform plan. Perintah ini menyajikan rencana eksekusi yang sangat detail, menunjukkan sumber daya apa saja yang akan dibuat, diubah, atau dihancurkan. *Output* dari perintah ini, seperti yang terdokumentasi pada Gambar 3.15, berfungsi sebagai pengganti verifikasi di OCI Console. Ini adalah bukti konkret bahwa kode yang ditulis secara akurat mendefinisikan arsitektur yang diinginkan, dan memberikan kesempatan untuk meninjau setiap detail sebelum *deployment* sebenarnya.



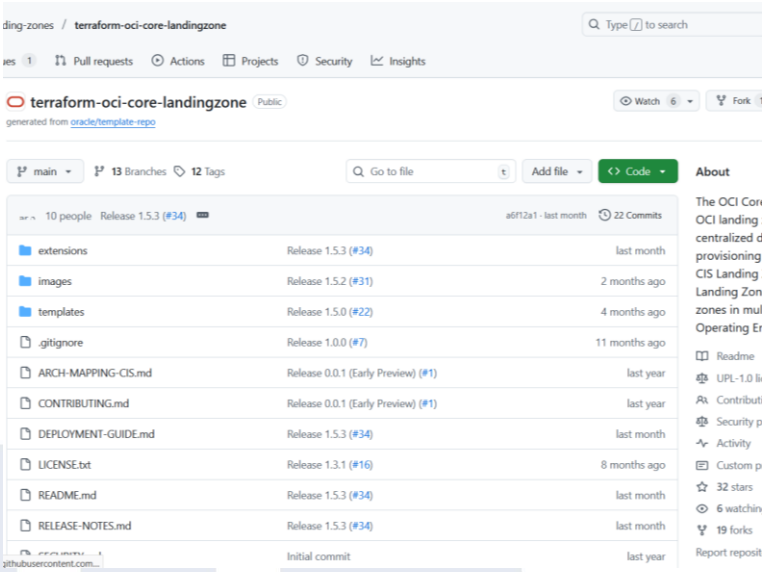
Gambar 3.16 terraform.tfstate untuk Web

Setelah *deployment*, Terraform akan mencatat semua sumber daya yang dibuatnya dalam sebuah *state file* bernama terraform.tfstate. *File* ini, yang strukturnya dapat dilihat pada Gambar 3.16, sangat penting karena berfungsi sebagai "sumber kebenaran" bagi Terraform untuk melacak infrastrukturnya.

3.3.1.5 Riset dan Implementasi OCI Core Landing Zone

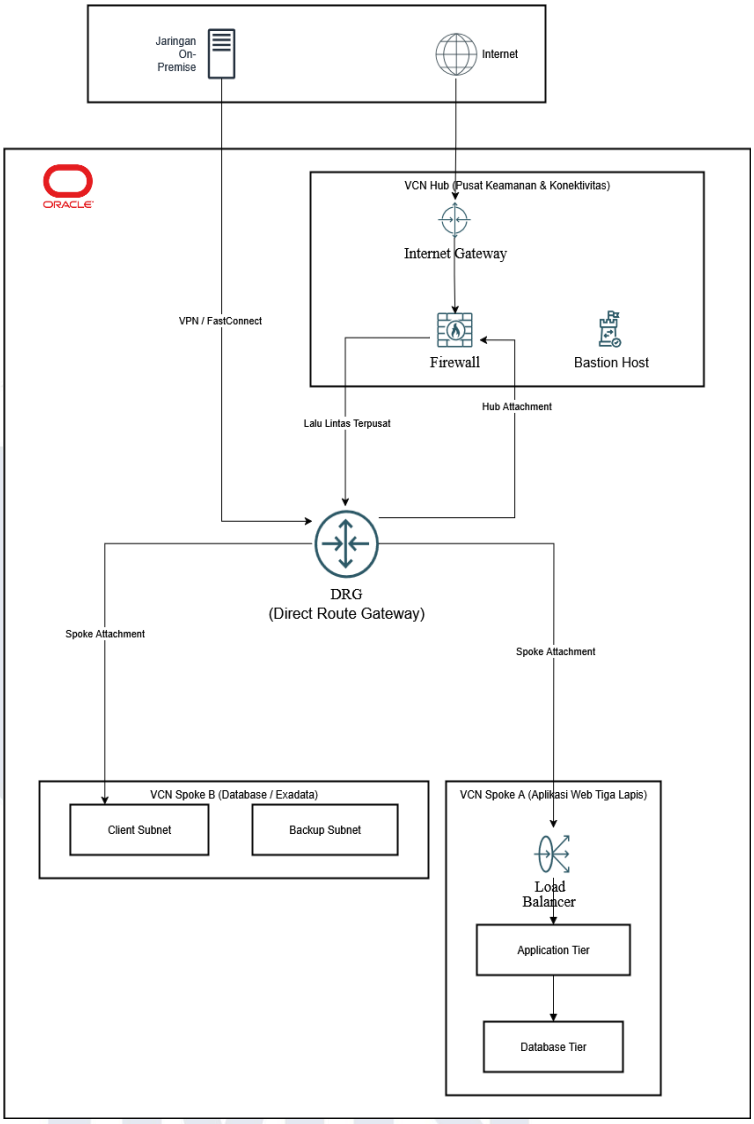
Proyek implementasi OCI Core Landing Zone merupakan puncak dari kegiatan magang, di mana seluruh pengetahuan teoretis dan keterampilan praktis yang telah dipelajari diintegrasikan untuk membangun sebuah arsitektur *cloud* berskala perusahaan (*enterprise-grade*). Pelaksanaan tugas ini dibagi menjadi beberapa fase metodis, mulai dari studi literatur dan analisis mendalam, dilanjutkan dengan implementasi teknis, hingga diakhiri dengan pembuatan dokumentasi yang komprehensif.

A. Fase Riset dan Analisis Arsitektur



Gambar 3.17 Halaman Repositori GitHub untuk Modul terraform-oci-core-landingzone

Seperti yang ditunjukkan pada Gambar 3.17, titik awal dari proyek ini adalah dilakukannya riset mendalam terhadap modul Terraform resmi dari Oracle, yaitu terraform-oci-core-landingzone. Modul ini tidak hanya dipelajari sebagai sebuah *template* biasa, melainkan dianalisis sebagai sebuah kerangka kerja standar industri yang dirancang untuk mengotomatiskan penyiapan lingkungan OCI yang aman dan terkelola. Ditemukan bahwa tujuan utama dari modul ini adalah untuk menerapkan fondasi infrastruktur yang secara inheren mematuhi CIS OCI Foundations Benchmark v2.0, yang merupakan kumpulan praktik terbaik keamanan yang diakui secara global. Dari analisis kode dan dokumentasi, dipetakan bahwa modul ini secara otomatis akan membuat dan mengkonfigurasi serangkaian sumber daya fundamental, termasuk di antaranya: IAM untuk tata kelola akses, Jaringan VCN yang tersegmentasi, *Cloud Guard* untuk deteksi ancaman, *Logging* terpusat untuk audit, serta *Security Zones* untuk penegakan kebijakan keamanan yang ketat.



Gambar 3.18 Ilustrasi Diagram Arsitektur Jaringan Hub & Spoke di OCI

Analisis arsitektur kemudian diperdalam untuk memahami berbagai topologi jaringan yang ditawarkan, dengan fokus utama pada arsitektur Hub & Spoke, seperti yang diilustrasikan pada Gambar 3.18. Arsitektur ini dipilih untuk dipelajari karena merupakan pola desain jaringan yang sangat umum digunakan di perusahaan besar. Dalam model ini, sebuah Virtual Cloud Network (VCN) Hub difungsikan sebagai gerbang utama untuk mengontrol semua lalu lintas data, baik yang menuju internet (arus *North-South*) maupun

antar VCN lainnya (arus *East-West*). Seluruh VCN aplikasi, yang disebut sebagai VCN Spoke, terhubung ke hub ini melalui sebuah Dynamic Routing Gateway (DRG). DRG ini bertindak sebagai *router* virtual terpusat yang canggih, memastikan semua komunikasi antar-jaringan dapat diinspeksi dan diamankan di satu titik. Selain itu, dipelajari juga bagaimana struktur kompartemen IAM yang logis diterapkan, di mana sumber daya dipisahkan berdasarkan fungsi (Jaringan, Keamanan, Aplikasi, Database) untuk mencerminkan pembagian tanggung jawab tim IT di dunia nyata.

B. Fase Implementasi Praktis dan Pengujian

Setelah fondasi teoretis terbentuk, fase implementasi praktis diawali dengan penyiapan fondasi identitas dan akses (IAM) menggunakan *template* custom-identity-domain untuk mensimulasikan lingkungan perusahaan yang menggunakan domain identitas kustom. Proses teknisnya dimulai dengan pembuatan *custom identity domain* baru melalui konsol OCI, di mana OCID (Oracle Cloud Identifier) yang dihasilkan kemudian dimasukkan sebagai variabel dalam *file* konfigurasi Terraform. Setelah verifikasi melalui terraform plan, perintah terraform apply dieksekusi, yang secara otomatis menciptakan 13 grup pengguna (IAM Groups) seperti auditor, security-admin, dan cost-admin, lengkap dengan kebijakan akses spesifik untuk membentuk fondasi *Role-Based Access Control* (RBAC) yang solid. Setelah fondasi identitas ini terbentuk, implementasi dilanjutkan dengan membangun arsitektur jaringan Hub & Spoke yang lebih kompleks untuk konektivitas eksternal. Setelah terraform apply berhasil dieksekusi, serangkaian

verifikasi dilakukan untuk memastikan semua komponen terpasang dengan benar. Validasi ini mencakup pemeriksaan visual topologi melalui OCI Network Visualizer, pengecekan DRG Attachments untuk memastikan konektivitas VCN Hub dan Spoke, verifikasi aturan perutean di DRG Route Tables, dan pengujian layanan OCI Bastion untuk menjamin akses *secure jump host* ke *instance* privat telah berfungsi.

3.3.1.6 Simulasi Cloud Migration Strategy

Proyek ini berfokus pada modernisasi lima aplikasi kunci dari infrastruktur pusat data *on-premise* ke *cloud* Amazon Web Services (AWS). Tujuan utama dari inisiatif ini adalah untuk memastikan sistem tidak ketinggalan zaman, sekaligus mencapai operasional yang lebih praktis, efisien, dan hemat biaya. Proyek ini mensimulasikan seluruh siklus hidup transformasi digital, mulai dari analisis tantangan bisnis, perancangan fondasi infrastruktur yang kokoh, eksekusi migrasi teknis, hingga analisis mendalam terhadap hasil operasional dan finansial pasca-migrasi.

A. Fase Perencanaan Strategis dan Perancangan Fondasi Arsitektur

Fase fundamental ini bertujuan untuk memastikan bahwa setiap keputusan teknis yang diambil selaras dengan tujuan bisnis, keamanan, dan kepatuhan regulasi yang dihadapi oleh Bank XYZ. Proses ini dimulai dengan analisis mendalam terhadap tantangan yang ada dan diakhiri dengan pembangunan fondasi *cloud* yang terstruktur dan aman.

1. **Tantangan Bisnis dan Pendorong Migrasi**
    a. Keterbatasan Infrastruktur Warisan (*Legacy Infrastructure*): Infrastruktur *on-premise* yang ada sangat bergantung pada perangkat keras yang mulai

menua. Hal ini menyebabkan keterbatasan skalabilitas yang signifikan, terutama dalam menangani lonjakan volume transaksi nasabah pada periode kritis seperti hari gajian atau selama kampanye promosi besar. Ketidakmampuan untuk melakukan *scale-out* secara dinamis menjadi penghambat utama kelincahan bisnis.

b. Tuntutan Kepatuhan Regulasi Perbankan yang Ketat: Sebagai institusi finansial, Bank XYZ diwajibkan untuk mematuhi standar regulasi yang sangat ketat dari Otoritas Jasa Keuangan (OJK) dan Bank Indonesia (BI). Salah satu tuntutan paling signifikan adalah terkait kedaulatan data (*data sovereignty*), yang mengharuskan semua data nasabah disimpan dan diproses di dalam yurisdiksi Indonesia.

c. Kurangnya Isolasi Antar Lingkungan: Ditemukan bahwa tidak ada pemisahan yang cukup kuat antara lingkungan pengembangan (*development*), pengujian (*testing*), dan produksi (*production*). Kurangnya isolasi ini meningkatkan risiko insiden di lingkungan produksi yang disebabkan oleh perubahan di lingkungan non-produksi, serta secara signifikan memperlambat siklus rilis fitur baru.

d. Beban Biaya Operasional dan Model Belanja Modal (CAPEX): Model operasional IT saat ini sangat bergantung pada belanja modal (CAPEX) yang tinggi untuk pengadaan dan pemeliharaan *server* serta perangkat keras jaringan. Model ini dianggap tidak efisien dan membebani anggaran, karena memerlukan investasi besar di muka untuk kapasitas yang mungkin tidak sepenuhnya terpakai.

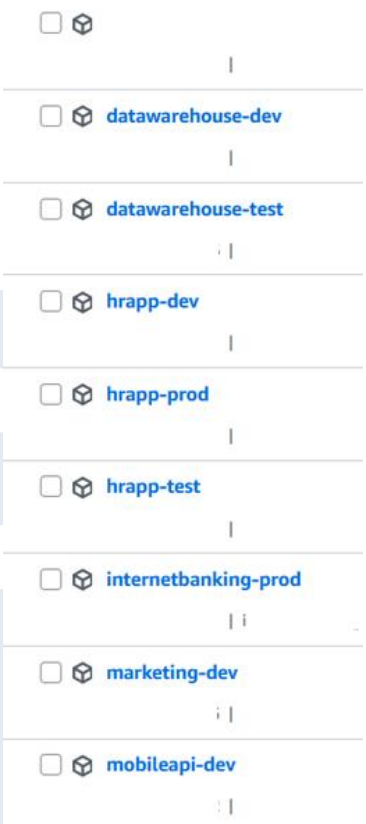
e. Kompleksitas Implementasi Keamanan Data Sensitif: Menerapkan praktik terbaik keamanan (*security best practices*) secara konsisten di seluruh infrastruktur *on-premise* merupakan tantangan yang kompleks dan memerlukan sumber daya yang besar untuk melindungi data nasabah yang sangat sensitif.

2. Perancangan Fondasi AWS Landing Zone

Dalam menjawab serangkaian tantangan tersebut, solusi fondasi yang dirancang dan diimplementasikan adalah AWS Landing Zone. Ini adalah sebuah cetak biru arsitektur yang menyediakan lingkungan *multi-akun* AWS yang aman, terkelola, dan patuh regulasi sejak awal .

a. Strategi Multi-Akun dengan AWS Organizations

Fondasi utama dibangun di atas layanan AWS Organizations, yang memungkinkan pengelolaan terpusat untuk banyak akun AWS. Pendekatan *multi-akun* ini dipilih karena memberikan tingkat isolasi dan keamanan tertinggi. AWS Organizations, yang memungkinkan pengelolaan terpusat untuk banyak akun AWS. Pendekatan *multi-akun* ini dipilih karena memberikan tingkat isolasi dan keamanan tertinggi.



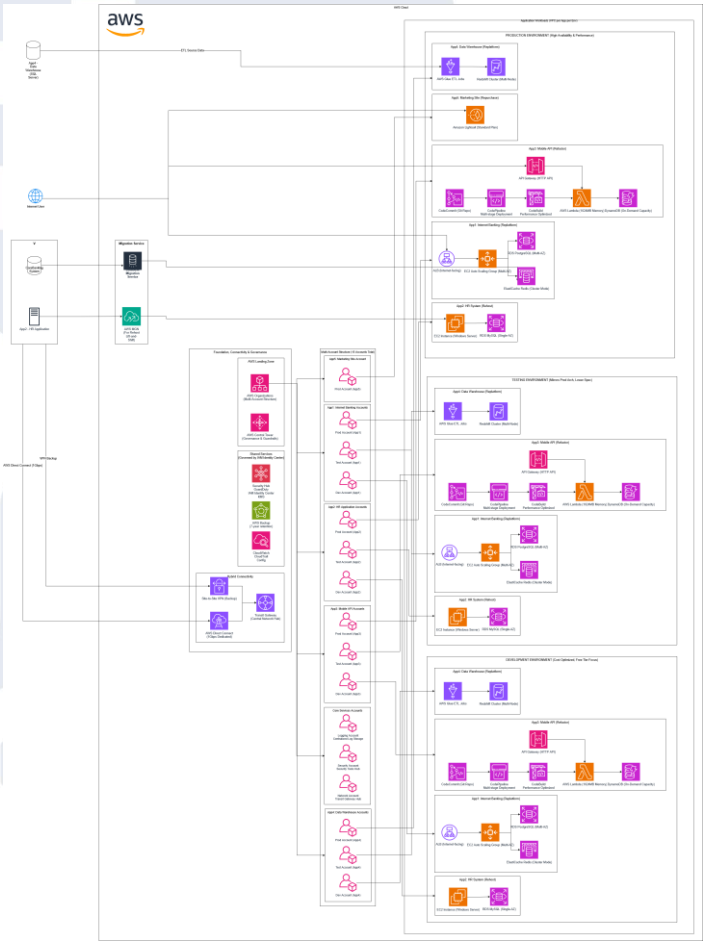
Gambar 3.19 Struktur Organisasi Multi-Akun Bank XYZ di AWS Organizations

Seperti yang ditunjukkan pada Gambar 3.19, sebuah struktur hierarkis yang telah diimplementasikan. Sebuah Akun Manajemen berada di puncak hierarki, berfungsi sebagai pusat administrasi dan penagihan. Di bawahnya, dibuat beberapa Unit Organisasi (OU) untuk mengelompokkan akun berdasarkan fungsi atau aplikasi. Sebagai contoh, terlihat adanya OU untuk datawarehouse, hrapp, internetbanking, marketing, dan mobileapi. Di dalam setiap OU aplikasi, dibuat akun-akun terpisah untuk setiap lingkungan—*development*, *testing*, dan *production*—seperti yang terlihat pada akun datawarehouse-dev dan datawarehouse-test, atau hrapp-dev, hrapp-prod, dan hrapp-test. Struktur ini secara

efektif mengisolasi beban kerja, menyederhanakan manajemen akses, dan memungkinkan penerapan kebijakan keamanan yang granular untuk setiap lingkungan.

b. Arsitektur Jaringan Terpusat Model Hub-and-Spoke

Dalam memastikan konektivitas yang aman dan terkelola, dirancang sebuah arsitektur jaringan dengan model Hub-and-Spoke yang canggih.



Gambar 3.20 Gambar 3.20 Diagram Arsitektur Jaringan Hub-and-Spoke di AWS

Seperti yang diilustrasikan secara detail pada Gambar 3.20 sebuah VPC (Virtual Private Cloud) Hub pusat dibangun untuk berfungsi sebagai gerbang utama untuk semua lalu lintas data. Di dalam Hub ini, ditempatkan komponen-komponen keamanan dan konektivitas bersama, seperti AWS Web Application Firewall (WAF) untuk inspeksi lalu lintas lapisan aplikasi dan NAT Gateway terpusat untuk menyediakan akses internet keluar yang aman bagi sumber daya di *subnet* privat. VPC-VPC lain yang menampung aplikasi (disebut VPC Spoke) terhubung ke Hub ini melalui AWS Transit Gateway. Penggunaan Transit Gateway secara signifikan menyederhanakan perutean jaringan dan memastikan bahwa semua komunikasi antar-VPC dapat diinspeksi dan dikontrol di satu titik pusat. Seluruh infrastruktur ini, tanpa terkecuali, di *deploy* di AWS Region Jakarta (ap-southeast-3) untuk memastikan kepatuhan 100% terhadap regulasi kedaulatan data yang ditetapkan oleh OJK dan BI.

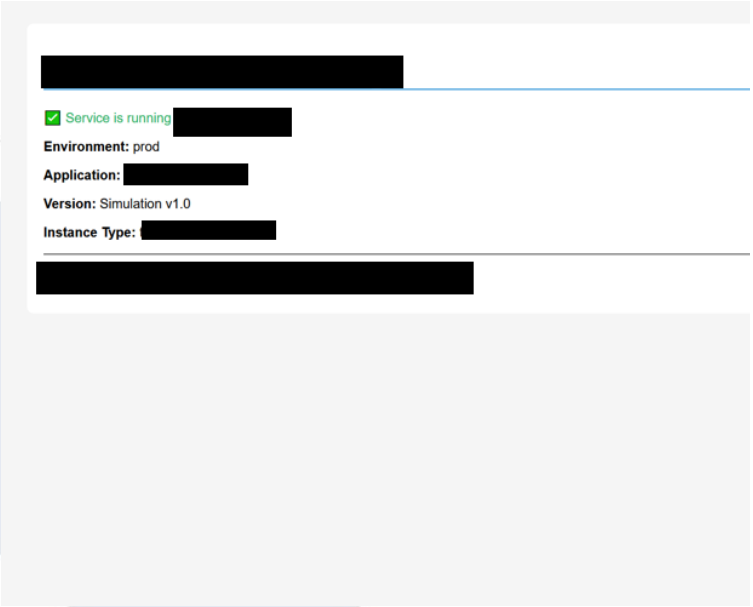
B. Eksekusi Migrasi Lima Aplikasi Inti

Dengan fondasi infrastruktur yang telah siap, proses eksekusi migrasi untuk lima aplikasi kunci dilakukan secara bertahap. Setiap aplikasi dianalisis secara individual untuk menentukan strategi migrasi yang paling sesuai dari kerangka kerja 7 R's, guna memaksimalkan nilai bisnis dan efisiensi teknis.

1. Platform Internet Banking (Strategi: Replatform)

Platform Internet Banking, sebagai layanan paling kritis yang berhadapan langsung dengan nasabah, dimigrasikan

menggunakan strategi Replatform. Strategi ini dipilih untuk memungkinkan aplikasi dimodifikasi secara minimal agar dapat memanfaatkan layanan *cloud* terkelola, khususnya pada lapisan *database*.



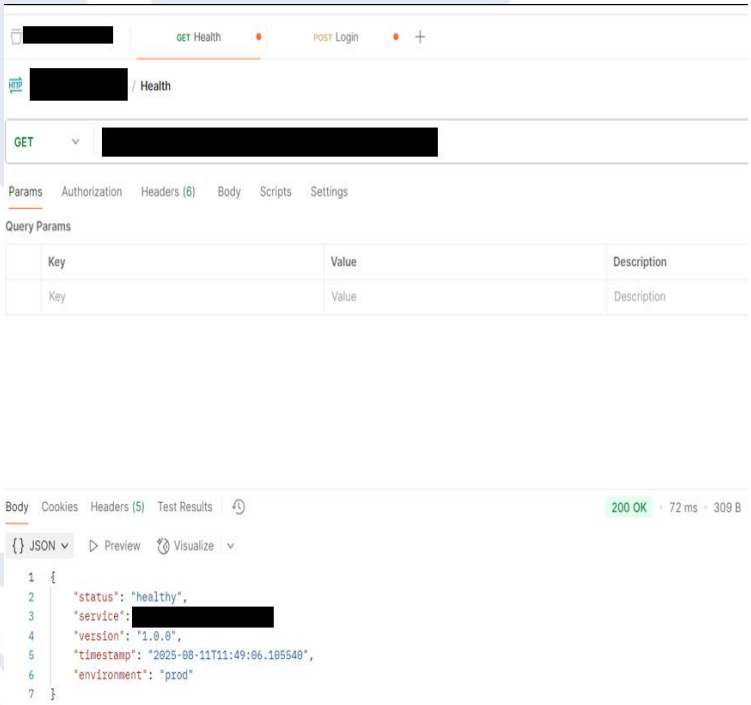
Gambar 3.21 Tampilan Platform Internet Banking yang Berjalan di AWS

Arsitektur target di AWS terdiri dari Application Load Balancer (ALB) yang mendistribusikan lalu lintas ke sekelompok EC2 instance di dalam Auto Scaling Group . Hal ini memastikan skalabilitas elastis, di mana jumlah *server* dapat bertambah atau berkurang secara otomatis sesuai dengan volume transaksi. *Database* Oracle *on-premise* yang sebelumnya memerlukan biaya lisensi tinggi dan manajemen manual, berhasil dimigrasikan ke Amazon RDS for PostgreSQL, sebuah layanan *database* terkelola penuh. Gambar 3.21 enunjukkan bukti bahwa aplikasi telah berhasil beroperasi penuh di lingkungan produksi AWS dan dapat diakses melalui *endpoint* ALB-nya. Pasca-migrasi, hasil pengujian performa menunjukkan waktu respons rata-rata

aplikasi adalah 1.2 detik, berhasil melampaui target yang ditetapkan di bawah 2 detik.

2. Layanan API Mobile Banking (Strategi: Refactor)

Dalam layanan API Mobile Banking yang menjadi motor inovasi digital, strategi yang lebih agresif yaitu Refactor diadopsi. Arsitektur aplikasi dirombak total dari model berbasis Refactor diadopsi. Arsitektur aplikasi dirombak total dari model berbasis *server* menjadi arsitektur *serverless* untuk memaksimalkan potensi elastisitas, efisiensi biaya, dan kecepatan pengembangan.



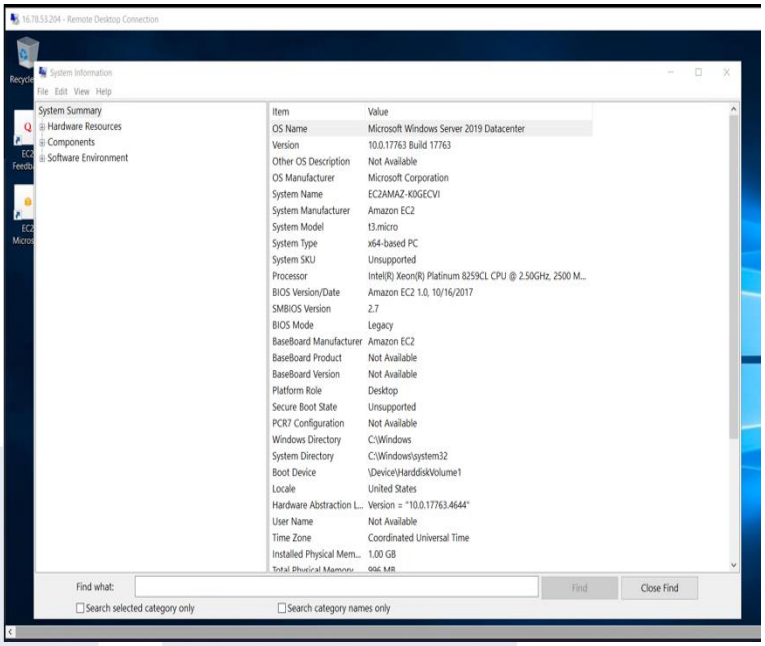
Gambar 3.22 Pengujian Endpoint API Mobile Banking Pasca-Migrasi

Arsitektur baru ini menggunakan Amazon API Gateway sebagai gerbang masuk yang aman dan terkelola untuk semua permintaan API, AWS Lambda sebagai mesin komputasi yang menjalankan logika bisnis tanpa perlu mengelola *server*, dan Amazon DynamoDB sebagai

database NoSQL berlatensi rendah. Gambar 3.22 menampilkan hasil pengujian langsung menggunakan Postman ke *endpoint* API yang telah di-*deploy*. Terlihat pada gambar tersebut, pengujian ke *endpoint* /health memberikan respons "healthy" dengan status 200 OK dan latensi sangat rendah yaitu 72 milidetik. Pengujian fungsionalitas login dengan mengirimkan *username* dan *password* juga memberikan respons "success". Hasil ini memvalidasi bahwa API tidak hanya berfungsi penuh tetapi juga sangat responsif, dengan latensi rata-rata tercatat sebesar 150 milidetik, jauh lebih cepat dari target performa <200 ms.

3. Sistem Manajemen SDM (Strategi: Rehost)

Aplikasi internal untuk manajemen SDM dimigrasikan menggunakan strategi tercepat dan paling minim risiko, yaitu Rehost (*lift-and-shift*). Aplikasi dan sistem operasinya dipindahkan "apa adanya" ke AWS untuk memastikan kelangsungan operasional bisnis dengan gangguan seminimal mungkin.

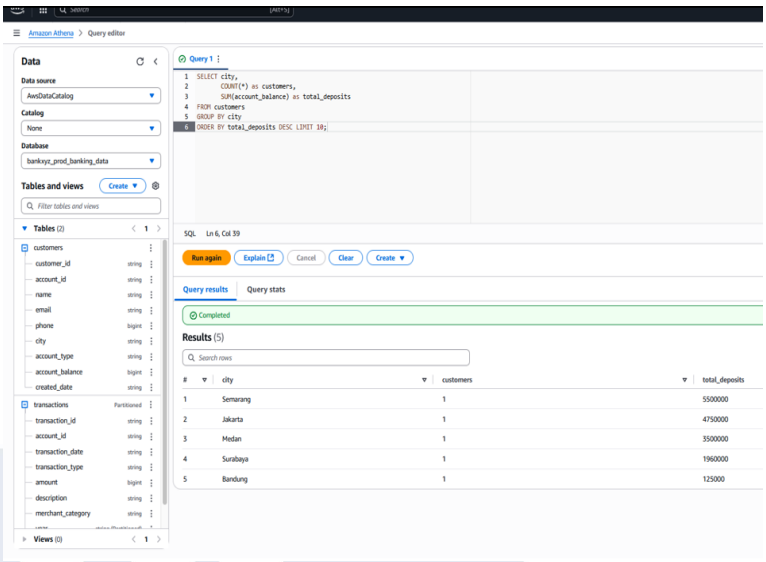


Gambar 3.23 Sesi RDP ke Server Aplikasi HR di Amazon EC2

Arsitektur targetnya adalah sebuah Amazon EC2 instance yang menjalankan sistem operasi Windows Server 2019 Datacenter. Gambar 3.23 memberikan bukti visual dari keberhasilan migrasi ini, menampilkan sebuah sesi *Remote Desktop Protocol* (RDP) yang aktif ke *instance* EC2 tersebut. Jendela *System Information* di dalam sesi RDP mengkonfirmasi bahwa aplikasi kini berjalan di atas infrastruktur AWS (System Manufacturer: Amazon EC2), menggunakan *instance* tipe t3.micro di *Availability Zone* ap-southeast-3a. *Database* aplikasi juga dipindahkan ke Amazon RDS for MySQL untuk meningkatkan keandalan.

4. Data Warehouse & Platform Analitik (Strategi: Replatform)

Platform analitik lawas dimodernisasi menjadi sebuah arsitektur *data lakehouse* yang lebih cepat dan skalabel menggunakan strategi Replatform.



Gambar 3.24 Kueri Analitik Menggunakan Amazon Athena pada Data di S3 Data Lake

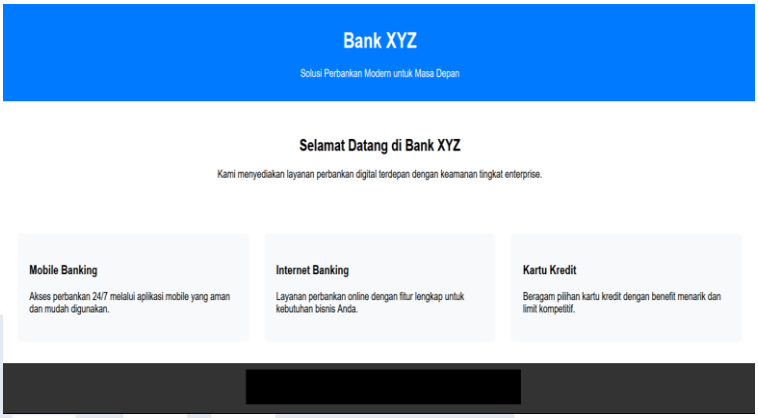
Arsitektur baru ini menggunakan Amazon S3 sebagai *data lake* terpusat, AWS Glue sebagai layanan ETL *serverless* untuk memproses dan mengkatalogkan data, dan Amazon Athena untuk menjalankan kueri SQL interaktif secara langsung pada data di S3.

Gambar 3.24 menunjukkan konsol Amazon Athena di mana sebuah kueri analitik (SELECT city, COUNT(*) as customers, SUM(account_balance) as total_deposits...) berhasil dieksekusi pada tabel customers yang datanya tersimpan di S3. Hasil kueri ditampilkan dalam waktu eksekusi yang sangat singkat, memvalidasi efisiensi arsitektur baru ini. Waktu eksekusi kueri analitik rata-rata adalah 5 detik, 50% lebih cepat dari target di bawah 10 detik.

5. Website Pemasaran (Strategi: Repurchase)

Website pemasaran korporat dimigrasikan dengan strategi Repurchase, yang dalam konteks ini berarti

membangun ulang di atas tumpukan teknologi standar yang lebih andal di *cloud*.

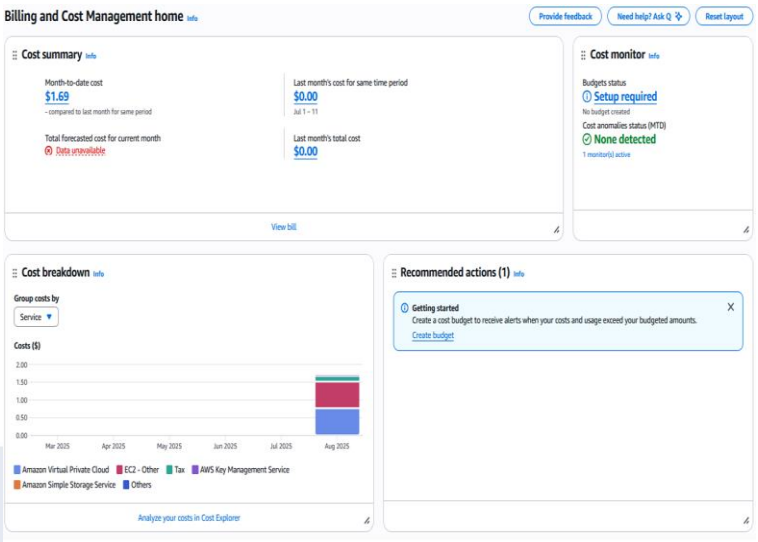


Gambar 3.25 Tampilan Website Pemasaran Bank XYZ Setelah Migrasi

Arsitekturnya terdiri dari Amazon EC2 untuk menjalankan *web server* Apache dan Amazon RDS for MySQL untuk mengelola *database* konten. Amazon EC2 untuk menjalankan *web server* Apache dan Amazon RDS for MySQL untuk mengelola *database* konten. Gambar 3.25 menampilkan halaman depan situs yang telah berhasil dimigrasikan dan dapat diakses secara publik melalui alamat IP-nya. Hasil pengujian menunjukkan waktu muat halaman rata-rata adalah 2.1 detik, secara konsisten berada di bawah target 3 detik yang direkomendasikan untuk pengalaman pengguna yang optimal.

C. Analisis Hasil dan Optimalisasi Pasca-Migrasi

Fase terakhir dari proyek ini adalah melakukan analisis mendalam terhadap hasil yang dicapai, terutama dari sisi keamanan, performa, dan dampak finansial.



Gambar 3.26 Dasbor Manajemen Biaya AWS untuk Bank XYZ

Salah satu keberhasilan terbesar dari proyek ini adalah realisasi penghematan biaya yang signifikan. Gambar 3.26 menunjukkan dasbor AWS Billing and Cost Management yang menampilkan biaya aktual *month-to-date* yang sangat rendah, yaitu hanya $1.69 . Angka ini membuktikan efektivitas dari strategi optimisasi biaya yang diterapkan. Secara keseluruhan, proyek ini berhasil mencapai reduksi biaya operasional infrastruktur sebesar 60-70% dibandingkan dengan model *on-premise*, dengan penghematan tahunan diproyeksikan mencapai $2,400 hingga $3,600. Penghematan ini dicapai melalui kombinasi beberapa strategi: penggunaan NAT Gateway bersama yang menghemat ~$135/bulan, adopsi arsitektur *serverless* yang menghilangkan biaya sumber daya *idle*, dan maksimalisasi penggunaan layanan dalam AWS Free Tier untuk lingkungan non-produksi . Sejak diluncurkan, tercatat nol insiden keamanan dan tingkat ketersediaan layanan (uptime) mencapai 99.9%, membuktikan bahwa efisiensi biaya yang

masif dapat dicapai tanpa mengorbankan pilar utama yaitu keamanan dan keandalan.

3.3.2 Kendala yang Ditemukan

Selama periode magang, diidentifikasi beberapa tantangan utama yang memengaruhi alur kerja dan proses pengembangan keahlian teknis:

1. Keterbatasan Lingkungan Praktik: Pada fase awal, tantangan utama adalah tidak tersedianya lingkungan *cloud* khusus (*sandbox*) untuk implementasi. Hal ini menyebabkan proses pembelajaran terbatas hanya pada teoretis melalui dokumentasi. Akibatnya, validasi kode *Infrastructure as Code* (IaC), simulasi *deployment*, dan pengembangan keterampilan *debugging* secara langsung tidak dapat dilakukan, sehingga menciptakan kesenjangan antara pengetahuan konseptual dan penerapan di dunia nyata.
2. Kurva Pembelajaran pada Teknologi Skala Perusahaan

Modul yang digunakan, seperti OCI Core Landing Zone, merupakan teknologi skala perusahaan (enterprise-grade) yang memiliki kurva pembelajaran yang curam.

a. Arsitektur Kompleks: Memahami konsep-konsep lanjutan seperti arsitektur Hub-and-Spoke, *Dynamic Routing Gateway* (DRG), dan kebijakan IAM yang granular memerlukan studi mendalam di luar dokumentasi dasar.
b. Dokumentasi yang Tersebar: Informasi yang dibutuhkan seringkali tersebar di berbagai sumber, mulai dari repositori GitHub untuk kode modul, dokumentasi resmi *platform cloud* untuk detail layanan, hingga forum komunitas untuk solusi atas *error* yang spesifik.

3.3.3 Solusi atas Kendala yang Ditemukan

Bagian ini berisi solusi atas kendala/kesulitan yang penulis temukan selama menjalani praktek kerja.

1. Penyediaan Akses ke Lingkungan *Sandbox* OCI: Sebagai solusi atas keterbatasan praktik, PT Infracom Technology menyediakan akses penuh ke lingkungan *sandbox* Oracle Cloud Infrastructure (OCI) internal perusahaan. Hal ini memungkinkan secara aman melakukan *deployment*, pengujian, dan *debugging* kode Terraform secara langsung. Akses ini menjadi kunci yang menjembatani kesenjangan antara teori dan praktik, serta mempercepat kurva pembelajaran secara signifikan.
2. Pendekatan Pembelajaran Terstruktur dan Kolaboratif

Dalam mengatasi kurva pembelajaran yang curam, diterapkan pendekatan pembelajaran yang lebih sistematis.

a. Bimbingan Terstruktur dan *Mentoring*: Sesi bimbingan reguler dengan *engineer* senior dan pembimbing lapangan dilakukan untuk membahas arsitektur kompleks dan memecahkan masalah teknis. *Mentoring* ini mempercepat transfer pengetahuan praktis yang tidak tertulis dalam dokumentasi.
b. Implementasi Bertahap (*Iterative*): Proyek dipecah menjadi beberapa tahap yang lebih kecil dan dapat dikelola. Daripada membangun seluruh *Landing Zone* secara bersamaan, implementasi dimulai dari komponen dasar seperti jaringan (VCN), dilanjutkan dengan IAM, dan seterusnya. Pendekatan iteratif ini membuat proses pembelajaran lebih mudah dicerna dan dikelola.